Applying machine learning to the prediction of defaults in loans

June 8, 2018

UNIVERSITY OF TWENTE.

Author:
Michiel Cornelissen BSc.
s1229532

Supervisors:
University of Twente
drs. ir. Toon de Bakker
dr. Mannes Poel

Accenture
Paul Weiss MSc.

University of Twente
Drienerlolaan 5
7522 NB Enschede

Accenture
Gustav Mahlerplein 90
1082 MA Amsterdam

Management summary

The goal of this thesis is to compare the predictive performance of several machine learning algorithms in their capability to predict defaults in loans. This is done to come up with valuable information about which algorithms are most suitable for this task. Such information is required before machine learning can be implemented in practice to predict defaults. To determine the performance, several algorithms that can be used to classify samples have been implemented. Those are used to predict the defaults and the performance of each of the algorithms is measured.

The loans used in this thesis come from two data sets of which it is known which loans went into default. The choice has been made to use two data sets in order to be able to determine if the same results are found for both sets. The first data set, from the UCI Machine Learning Repository (Yeh and Lien, 2009), consists of 30,000 Taiwanese credit lines. The second data set, after preparation, is comprised of 85,964 peer to peer loans that have been originated using the Prosper platform (Prosper, 2017).

The main question of this thesis is:

> *What is the predictive performance of machine learning when applied to the prediction of defaults in loans?*

Model performance

The Area Under the Curve has been used as the primary performance indicator. The measure ranges from 0.0 to 1.0, with the latter being a perfect predictor. A value of 0.5 indicates that a model performs equal to one that classifies by random guesses. Table 1 shows which algorithms have been used and what the corresponding performance is. Aside from the AUC, the precision and recall are also shown.

TABLE 1: Performance of the different algorithms when trained and validated on the original data set.

	Taiwan			Prosper		
Algorithm	AUC	Precision	Recall	AUC	Precision	Recall
Logistic regression	0.77	0.48	0.58	0.77	0.16	0.73
Neural network	0.78	0.45	0.62	0.81	0.17	0.77
Naive Bayes	0.74	0.40	0.63	0.66	0.10	0.77
k Nearest neighbors	0.76	0.44	0.61	0.75	0.15	0.66
Decision tree	0.76	0.45	0.60	0.77	0.15	0.78
Random forest	0.77	0.44	0.63	0.80	0.15	0.83
ADABoost	0.77	0.41	0.67	0.80	0.17	0.73
Gradient boosting	0.78	0.45	0.61	0.81	0.16	0.83
Support Vector Machine	0.76	0.47	0.59	-	-	-

In the Taiwan data set the AUC of all algorithms is similar. Naive Bayes has the lowest performance with an AUC of 0.74, the highest AUC, 0.78, is shared by two algorithms, Neural Network and Gradient Boosting. Aside from the Naive Bayes

algorithm, the performances are so close together that it is difficult to draw conclusions based solely on the AUC. The performances achieved with the Prosper data set show more variation. Naive Bayes again has the lowest performance. The highest performance is achieved by the same two algorithms, Neural Network and Gradient Boosting. When one takes the explainability into account, the view on which of the algorithms is most suitable changes. For the Taiwan data set, the Decision Tree has a decent AUC which is 0.02 below the highest value, in the Prosper data set it is 0.04 below the highest value. Clearly a choice between explainability and performance has to be made.

For the Prosper data set, it is interesting to see how good the models are in predicting which loans don't go into default. The best algorithms, Random Forest and Gradient Boosting, have a negative predictive value of 97,9% and even the worst algorithm, Naive Bayes, has 95,9%. This makes it interesting to use the algorithms not as default predictors, but to define a group of low risk loans.

Finally, both of the data sets have been split according to certain characteristics. The goal of such a modification is to find differences in performances on specific parts of the data set. With the Taiwan data set a positive effect was observed in two occasions. Some algorithms with the Prosper data set showed an increase up to 0.03. However, with both data sets a decrease in performance was observed in most situations. The last modification was winsorization of the numerical features. This led to no substantial changes.

Based on the above mentioned findings the following conclusions are drawn. Both Gradient Boosting and Neural Network seem to be the best performing algorithms. They both have the highest AUC for both of the data sets. The disadvantages of those algorithms is that they tend to become a black box , making it difficult to explain why a classification is made. Furthermore, an increase in performance can be achieved by splitting the data sets by some characteristics, but this should be examined case-by-case.

Resampling

The data sets used in this research project where imbalanced. The Taiwan data set had 22% of the samples in the minority class, the Prosper data set even less, only 8%. To remove the imbalance the following methods have been used: random undersampling, random oversampling, SMOTE (regular, borderline1 and borderline2) and ADASYN (2, 5 and 10 neighbors).

After using those methods on both data sets, it is concluded that resampling can have a positive effect, but it should not be applied without reservation. Especially the more complex methods, SMOTE and ADASYN, are practically never the best choice with these two data sets. Random undersampling is the algorithm that has been used the most in this research project, mainly because its positive effect on the performance, but also due to its effect of decreasing the computation times.

Preface

I proudly present you my master's thesis, which is the result of more than half a year of research. This thesis is written in order to graduate from the master Industrial Engineering and Management at the University of Twente. Meaning that this thesis marks the end of my life as a student, after seven years, which feel more like three of four.

The majority of this thesis has been written during my internship at Accenture. During this time I have been able to get to know Accenture and some of its people. From whom I appreciate their willingness to discuss the dry matter of machine learning while enjoying a cup of coffee. I want to especially thank my external supervisor, Paul Weiss, for the opportunity to write my thesis at Accenture which gave me the possibility to get to know the company from within and for his advice and comments on my thesis.

I also thank my supervisors from the university, Toon de Bakker and Mannes Poel for guiding me through the process of writing a thesis. Your advice and comments on my research were most useful and helped me to create the result you are reading now.

My parents have supported me throughout all my years as a student, for which I am most grateful and want to thank them. I also thank all my friends I met during my study in Enschede, for all the good times we had and will have in the future. Finally, I want to thank my girlfriend for her support and patience. Even when too much time went into writing my thesis.

Amsterdam, June 2018

Michiel Cornelissen

Contents

Chapter 1

Introduction

In this chapter the research is introduced to the reader. First Accenture is introduced, the company where this research project is performed. This is followed by a description of the project context, after which the problem is described and the research objective is given. Based on the research objective several questions are defined to which this research project will give an answer. Finally, this chapter is concluded with an outline of the rest of the report.

1.1 Organization

Accenture is a global management consulting firm with over 400,000 employees. The history of the company goes back to 1953, but it has been working under its own name since 1989 as Andersen Consulting. Since 2001 the current name, Accenture, is used. In that same year the company had its initial public offering at the New York Stock Exchange.

The company has divided its business into several categories: strategy, consulting, digital, technology and operations. This research project is carried out within the consulting branch and more specific the banking industry. This department is specialized in supporting banks with a broad range of services. To stay up to date with new technologies there is large interest in research focused at financial innovations.

1.2 Project context

One of the core functions of a bank is to give out loans to consumers and companies. For each loan, the bank is at risk of not receiving back the entire principal. The amount of risk usually has an influence on interest the bank will receive. For a lender it is valuable to be able to estimate the risk associated with each client, it can help a bank in two ways. First, it can be used in the loan origination process. This is the process consisting of all the steps a borrower and lender go through to process the application of a new loan. During the loan origination, the lender has to make a decision whether or not to accept the loan and on which terms. By being able to accurately estimate the risk associated with accepting the loan, the bank can make better decision on the terms, for example interest, and might also decline the application. The second situation where risk estimation is crucial, is in monitoring the already accepted loans. If a bank is capable of estimating which loans are likely to default, those loans can be handled with more attention. This can be done with the aim of increasing the recovered amount after a default, or preventing the loan from going into default at all. The accuracy of determining the risk of individual loans is crucial to the profit of a bank (Blöchlinger and Leippold, 2006). Therefore, financial institutions are always working on methods to improve their possibility to estimate risk.

Turing (1950) explored the topic of computing intelligence. In his article, he asked the question "can machines think?", by asking that question he was years ahead of his time. The term machine learning was first used in Samuel (1959) about learning a machine to play checkers. It took until the 90's before machine learning started to flourish as its own field of research (Langley, 1995). This came with a shift of paradigm: from achieving artificial intelligence towards tackling practical and solvable problems. Since then machine learning has slowly evolved into a more mature technology. In 2015 several tech giants open-sourced their machine learning tools, among which where Microsoft, Facebook and Google (Thomas, 2015; Chintala, 2015; Dean, 2015). With the widespread availability of tools, enterprises across industries start to experiment with machine learning. To be able to implement machine learning, it is necessary to have a thorough understanding of the available algorithms. Knowing which algorithm is most suitable for a task is a part of that understanding.

1.3 Problem description

As mentioned before, estimating the risk of a loan is an important task within banking. This makes it interesting to research the possible improvements that can be reached by applying machine learning. Several scientific papers have been written about the expected benefits of using machine learning in default prediction (Abellán and Mantas, 2014; Harris, 2015; Huang, Chen, and Wang, 2007). These show that machine learning can lead to a higher in accuracy default prediction, compared to conventional methods. However, most of these papers are limited in the number of algorithms compared. Moreover, the observation is that nearly all papers use a different data set, making the results difficult to compare. This creates an interesting opportunity to determine the performance of a broad range of algorithms on the same data sets.

1.4 Research objective

Now that the problem has been described, an objective for this research project is defined. The objective is to develop knowledge about the predictive performance of different machine learning algorithms when used to predict defaults in loans. This is done by implementing different machine learning algorithms which can be used to classify samples. These different algorithms will be used to predict the defaults on loans. The loans come from two data sets with loans of which it is known whether or not they went into default. The final part of the objective is to compare the performance of the used algorithms and to determine which is most suitable for this task.

To fulfill the research objective, several questions have to be answered. These questions are separated into a main question and sub questions. The sub questions must collectively form an answer to the main question.

1.4.1 Main question

The main question of this thesis is defined as:

What is the predictive performance of machine learning when applied to the prediction of defaults in loans?

1.4.2 Sub questions

Which machine learning algorithms are suitable for making binary classifications?

The first step of this research project is to determine what types of machine learning and which algorithms exist. Based on the findings from that analysis a selection of algorithms has to be made that will be used in this research project. This question will be answered using relevant literature.

What criteria should be used to objectively and accurately measure the performance of different machine learning algorithms?

The goal of this research project is to make a comparison of the performance of different machine learning algorithms. To be able to do so in a objective way, it is necessary to accurately measure the predictive performance. The measure that will be used in the experiments is based on a literature review.

What is the performance of the different machine learning algorithms when used to predict defaults?

The final subquestion is to actually determine the performance of the different algorithms. The answers of the previous questions are required to configure the algorithms and prepare the data. This answer will be answered by implementing the algorithms and using them in combination with the prepared data sets.

1.5 Report outline

The last section of the introduction contains an outline for the rest of this thesis. In the next chapter the Theoretical Framework is described. This consists of the background required to perform and understand the experiments. The main topics of the Theoretical Framework are a necessary background about machine learning, several methods of data preparation and a background about the different algorithms which can be used to classify data. The theoretical framework is followed by Chapter 3, Methodology. This chapter describes how the objective of this research will be achieved. In the next chapter the data is prepared to be used in machine learning, this is Chapter 4. After the data has been prepared the different models can be trained on the prepared data in Chapter 5. Here the settings of the models are determined and the performance is measured. Now all the results are known, conclusions can be drawn in Chapter 6. The conclusions will answer the main question of this research project. Finally, the paper is concluded with Chapter 7, Limitations and Further Research.

Chapter 2

Theoretical framework

The theoretical framework contains the necessary background to answer the research questions defined in the previous chapter. The chapter starts with a section about relevant research on the same topic, to provide a context for this thesis. A brief description of credit scoring will be given next. This subject is followed by a section about machine learning and goes in depth about the different forms of machine learning. This is followed by a section on the impact of an imbalance between classes in data sets, after which several performance measures are treated. To get to a reliable performance measure, cross-validation is required. It is therefore the next topic. After cross-validation it will be discussed how several models can be combined into so-called ensemble models. Finally the different algorithms used in this research project will be discussed in detail.

2.1 Current state of the literature

In this section of the Theoretical Framework some relevant literature will be described. The goal is to make the purpose and added value of this research project clear.

To find the beginning of artificial intelligence in scientific literature, one has to go back to 1950, the year in which Turing (1950) wrote his paper Computing Machinery and Intelligence. Since then, and particularly in recent years, the topic has gotten a lot of attention. According to Jordan and Mitchell (2015) machine learning has become the technology of choice within artificial intelligence to achieve practical solutions. They mention the rapid decrease in the cost of computational power and the availability of online data as two factors that have driven the rapid development of machine learning. As an important financial application of machine learning Jordan and Mitchell mention the detection of credit-card fraud. Their paper is concluded by mentioning it is necessary that society begins to consider how to maximize the benefits of machine learning.

This research project is not the first about using machine learning to predict defaults in loans. Using machine learning to score credit has been done since before 2000, for example Langley (1995) already mentioned it as a possible use case for machine learning. Since then a lot has happened in the field of machine learning, among other reasons due to the increasing computational power. In more recent years multiple scholars have performed research into the accuracy of machine learning when used for default prediction. However, most often the research is about a single new algorithm which is compared to a few benchmarks.

Several projects have been carried out to research the possibilities of machine learning in credit scoring (Abellán and Mantas, 2014; Harris, 2015; Huang, Chen, and Wang, 2007). These projects show that machine learning can lead to a high accuracy in credit scoring. However most of these projects are limited in the number of models compared and are based on different data sets which make the results incomparable. One of the early papers in which machine learning is applied to default prediction is about Support Vector Machines (Shin, Lee, and Kim, 2005), but no comparison is made. Alaraj, Abbod, and Hunaiti (2014) use a nearal network to make default prediction, but again no comparison is made with other techniques. Khandani, Kim, and Lo (2010) use machine learning to make default predictions and do make a comparison. However this is only between three algorithms, where in this research project a more broad comparison will be made.

Now that machine learning is slowly becoming a more mature technology that is starting to be used in practice another type of research is needed. Before making the choice on which algorithm should be implemented it is required to make a broad comparison between algorithms. For this comparison it is important that the same data set is used, otherwise the results cannot be compared.

Only summing up research projects with a positive conclusion would give a biased expectation of this project. Recently some critical reports have been published. According to a report by Gartner (2016) machine learning is currently on the top of a hype cycle and thus too high expectations exist. When a technology passes the top of the hype cycle, expectations will drop considerably. If machine learning moves to through the hype cycle as expected, it will get mainstream adoption in two to five years.

2.2 Credit risk

As mentioned before the goal of this research project is to determine if machine learning can be used to make better loan decisions. The reason for chasing this goal is to minimize the credit risk the bank is exposed to.

Credit risk arises from the possibility that borrowers, bond issuers, and counter parties in derivatives transactions may default (Hull, 2015).

In this project the focus is on the credit risk that arises from the possible defaults of borrowers. The formula for the expected loss from defaults is given in Equation (2.1). EAD is the expected exposure at the time of default, LGD is the fraction lost given default and PD is the probability of default.

$$\sum_i EAD_i \times LGD_i \times PD_i \tag{2.1}$$

When the customer of a bank is unable to repay a loan, it goes into default. The goal of the bank in such a situation is damage mitigation, it will try to minimize the amount that has to be written off due to the default. When accepting a customer for a loan the risk is determined. Part of this process is determining the expected amount that has to be written off, if the loan goes into default. The expected fraction that has to be written off, is the loss given default.

2.3 Machine learning

Machine learning is a field of computer science focused on giving computers the capabilities to learn. The goal of machine learning is to create algorithms that can learn and make predictions based on data and feedback. An important characteristic of machine learning is that it is not explicitly programmed to follow certain decision rules to create results. Instead it has the capability of creating those rules based on data and feedback.

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E (Mitchell, 1997).

By applying this definition to the goal of credit scoring it figures that information about historical loans and their default status is required to train a model into learning to predict defaults.

2.3.1 Features

In machine learning, a feature is a single measurable property of the phenomenon that is being studied. All the features describing a single entry usually form the feature vector. Features have one of the four following data types.

- **Nominal** features are labels without any quantitative value and the labels do not have a specific order. Examples of such a feature are gender or color.
- **Ordinal** features are labels without a quantitative value but with a specific order. An example is satisfaction (unhappy, neutral, happy).
- **Interval** features are numeric values of which the order is known and also the difference between the values. An example of such a label is the Celsius scale, the difference between the values is known. The problem with interval scales is that they do not have a true zero. This makes it impossible to calculate ratios, 10 °C is not twice as warm as 5 °C.
- **Ratio** features are similar to interval features but in addition have an absolute zero. This makes it possible to multiply and divide the values. Weight and length are examples of ratio features.

2.3.2 Learning methods

Machine learning problems are subdivided into different categories. The most used method of categorizing machine learning tasks is by learning method. In this section, the most used learning methods are described. Each category has a textual description, an example and a mathematical representation.

Supervised learning

The first method is supervised learning, this method is used when there is an input and a known output and the task is to learn the mapping form input to output (Alpaydın, 2010). This task can be described as to infer a function from label training data, which can be used to classify future unlabeled data.

This typically means that in a supervised learning scenario, a training set is given. This set consists of multiple cases, each containing features with a value and the resulting class. For example, a data set describing the color, trunk capacity and top speed of a few hundred cars and classification based on the car being a family car or not. A supervised learning algorithm uses the examples in the training set to infer a function relating the features to the car being a family car or not. The goal of inferring this function is to determine for future cars whether the car is a family car or not.

Given a set S, containing N training samples, s, $\{(x_1, y_1), ..., (x_N, y_N)\}$ with x_i being the feature vector of the i-th example and y_i being the corresponding label, the goal is to determine a function $g : X \to Y$. g is a function that maps the input space X (the features) to the output space Y (the labels). The performance of the function g is crucial in the performance of the machine learning model. It is usually assumed that the values in a feature vector x_i, are generated randomly and independently according to a fixed and unknown random distribution (Maimon and Rokach, 2005).

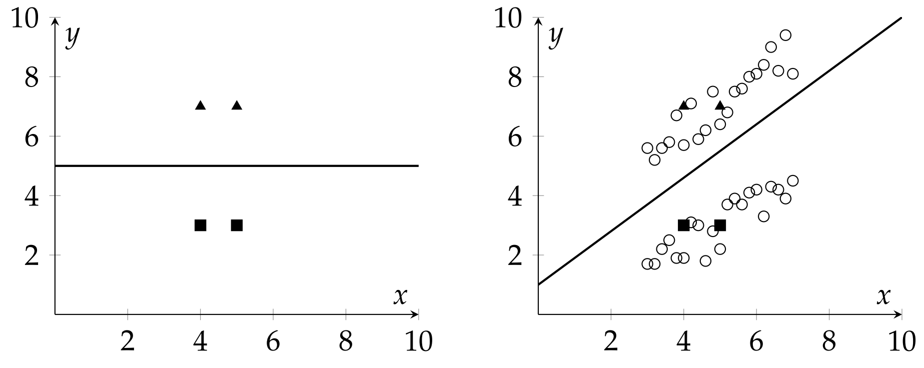
Semi-supervised learning

Semi-supervised learning considers the problem of classification when only a small subset of the observations have corresponding class labels (Kingma et al., 2014). Such a learning algorithm lies between supervised learning and unsupervised learning, the latter of which is discussed later in this section. At first it might seem strange that unlabeled data has any use and one might expect to discard the unlabeled data and regard the task as if being supervised learning. In Figure 2.1, the left image shows a situation with four labeled data points, the line shows a classifying border that is a good fit. The right image shows the same data points, but now unlabeled data is added. By analyzing the newly added unlabeled data, a pattern arises in the data. This pattern is only visible by taking the unlabeled data into account.

FIGURE 2.1: Graphs showing the possible added value of unlabeled data.

(A) Classification boundary based on labeled data only.

(B) Classification boundary based on labeled and unlabeled data.



Situations in which semi-supervised learning is practical, usually have a high cost of determining labels in the training set. An example of such a situation is image recognition in which lots of data is available but adding labels is manual work and thus expensive.

The following notations are based on Zhu, Ghahramani, and Lafferty (2003). In semi-supervised learning l labeled points are given, $(X_l, Y_l) = \{(x_1, y_1), ..., (x_l, y_l)\}$, and u unlabeled points are given, $X_u = \{x_{l+1}, ..., x_{l+u}\}$. In semi-supervised learning, on usually has $l \ll u$. Due to the complexity of semi-supervised learning further details are limited to the general idea. Which is to find functions $f : V \to R$, with V corresponding to the $l + u$ data points and to assign labels based on f. These functions f must agree with the labeled data X_l and should be smooth regarding the unlabeled data X_u.

Reinforcement learning

A learning algorithm that works by means of reinforcement does not receive any labels at first. Such a system learns by interacting with its environment by producing actions which affect the environment. The effect on the environment returns a reward or punishment. The goal of the algorithm is to produce actions in such a way that the reward is maximized or the punishment is minimized.

In recent times attention has been going out to cars autonomously driving a route. Learning a car to traverse such a route without violating any laws or causing accidents can be done by applying reinforcement learning. For example, an algorithm in a simulated environment must learn how to drive a car along a certain route without violating any traffic rules. At first the algorithm will simply perform random actions, by punishing traffic violations the algorithm will start to learn how to drive according to the rules. By repeating this process for many runs and rewarding the algorithm for reaching the end of the route it will slowly start to learn how to safely drive a car.

The notations used in this explanation are based on Maimon and Cohen (2009). Reinforcement learning is usually based on the Markov Decision Process, or MDP. Such a process contains all possible states S, actions A, rewards R and state-transitions P. State-transitions specify the resulting state of applying a certain action to the current state. The sets of states and actions can theoretically be both finite and infinite. The learning algorithm starts at time t in a certain environment $s_t \in S$ and reacts by selecting an action $a_t \in A$. This leads to the algorithm getting a reward r_t determined by the reward function $R(s_t, a_t)$. The result of selecting an action is a transition of the environment to a state s_{t+1} with probability $P(s_t, a_t, s_{t+1})$ determined by the state-transition function. The algorithm starts performing actions within this MDP without knowing anything about the reward function or the state-transition function. The goal of the algorithm is to find a policy that maximizes the achieved reward within the MDP.

Unsupervised learning

The last learning method discussed is unsupervised learning. In this method, the algorithm is trained using just an input set, no output, desired results or feedback is given. The algorithm must find structure in the data by itself. Unsupervised learning can be seen as finding patterns in data that are different from pure unstructured noise (Ghahramani, 2004).

A promising use of unsupervised learning is in the behavior-based detection of network security (Engel, 2017). Due to the amount of data generated it is impossible for a human to analyze all the data. An algorithm based on unsupervised learning could detect anomalies in the data without being learned what a breach looks like. When such an anomaly is detected, IT security could be notified.

2.3.3 Weak and strong learners

One of the ways to group similar machine learning models is by separating weak and strong learners. Weak learners are simple and fast trainable models that perform slightly better than random guessing (Freund and Schapire, 1997). Strong learners are all more complicated models.

2.4 Working with imbalanced data sets

In real-world data sets the number of 'interesting cases' is often small in comparison to the total number of instances. Consider for example a data set with loans and defaults, since in normal situations a small fraction of loans go into default the data set is imbalanced. This causes problems in training and evaluating machine learning models. Machine learning models can be evaluated by their predictive accuracy. In imbalanced data sets this measure is often misleading. Consider a data set with 99% of non-interesting (negative) instances, a model that classifies nothing as interesting (positive) will have a predictive accuracy of 99%. This accuracy is useless since it is unable to find any interesting case. Such a result is often seen with imbalanced data sets because the set contains too few interesting cases for the model to learn their characteristics. A method to mitigate this problem that received ample attention is over and under-sampling (Chawla, Japkowicz, and Kotcz, 2004).

2.4.1 Over-sampling by replication and under-sampling

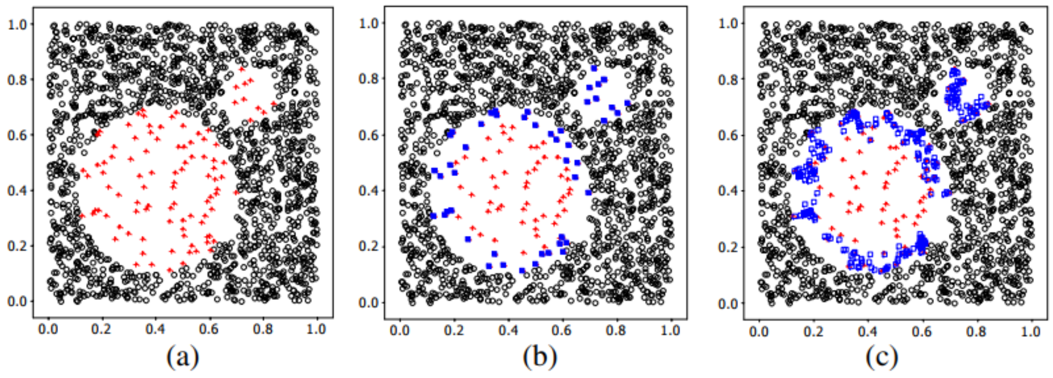
The most simple and straightforward method to balance imbalanced data sets is by using over-sampling by replication or under-sampling. The first of these is simply to replicate random samples of the under represented case until the data set is balanced. This results in a data set with multiple identical samples which might lead to specific and small decision regions which may cause over-fitting. Under-sampling is a method to balance the data set by ignoring a certain part of the over represented class. The issue with this method is that it leads to a loss of information and might cause under-fitting. It also decreases the size of the data set, which might result in a too small data set to still be able to train an algorithm.

2.4.2 Synthetic minority over-sampling technique

In the previous section, over-sampling by use of replication and under-sampling were introduced. Since that method replicates existing cases, the model trains on more but identical positive cases, this can cause over-fitting. To counter this issue Chawla et al. (2002) developed a technique to generate synthetic samples: synthetic minority over-sampling technique (SMOTE). Using synthetic samples, the classifier creates a larger and less specific decision area, this leads to better generalization of the decisions. Blagus and Lusa (2013) found that SMOTE does not always perform well on high dimensional data sets.

The method works by selecting the k nearest neighbors for each sample in the under represented class. Synthetic samples are generated on the line segments connecting neighbors. New samples are generated by calculating the difference between the feature vectors of a base sample and a neighbor. This difference is multiplied by a random number between 0 and 1, and is added to the feature vector of the base sample. The resulting vector is the feature vector of a newly generated sample. Depending on the required amount of over-sampling, neighbors are selected randomly from the k nearest neighbors. A graphical representation of the SMOTE process is given in Figure 2.2.

FIGURE 2.2: (a) The original distribution of data. (b) The borderline minority samples. (c) The borderline synthetic minority samples (Han, Wang, and Mao, 2005).



Borderline-SMOTE

Han, Wang, and Mao (2005) propose two alternatives on SMOTE in which only samples near the border between classes are over-sampled. They show that their proposed methods achieve a higher true positive rate and F-value. The philosophy of focusing on the borderline is that samples near the border are more apt to be misclassified and thus should get more attention.

The first proposed alternative is borderline-SMOTE1, it works as follows. Assume a set, S, is given containing training samples, $s_i = (x_i, y_i)$, with x_i being the feature vector of the i-th sample and y_i being the corresponding label. The set S is split, positive samples are a part of S_p and negative samples of S_n. For this explanation it is assumed the positive samples are under-represented. For all samples in the minority class, S_p, the k nearest neighbors are determined, these neighbors can belong to either class. The number of samples from the majority class among the nearest neighbors is denoted by k', for which $0 \leq k' \leq k$. If the number of majority class neighbors equals k for a sample, it is regarded as being noise. If the sample has more neighbors in the majority class as in the minority class ($k/2 < k' < k$), the sample is added to a set D (danger). The result will be a set $D \subseteq S$ with samples along the borderline of the classes. Finally the SMOTE algorithm is performed upon the set D as explained in the previous section.

The other proposed method is borderline-SMOTE2. In addition to the procedure for borderline-SMOTE1, a synthetic sample is calculated on the line segment between each sample in D and its nearest negative neighbor. In borderline-SMOTE1 the synthetic samples are generated by multiplying the difference between two samples with a random number between 0 and 1. For borderline-SMOTE2 the difference is multiplied with a random number between 0 and 0.5. This results in a synthetic sample closer to the positive sample.

2.4.3 Adaptive synthetic sampling

Adaptive synthetic sampling (ADASYN) is a method proposed by He et al. (2008) to balance data sets. The philosophy of ADASYN is to use a weighted distribution which gives higher weights to minority samples that are difficult to learn (close to the borderline). Samples with a large weight are used more often as base sample for generating a synthetic sample. Resulting in a greater focus on difficult regions which should lead to better predictions.

ADASYN is started by determining the number of synthetic samples that have to be generated, G. This is done by calculating the difference between N_n, the number of majority samples, and N_p, the number of minority samples, as shown in Equation 2.2. It follow that $N_n > N_p$ and thus $G > 0$. β is a parameter to specify the desired balance, with $\beta \in [0,\ 1]$. A value of 1 means a perfectly balanced data set.

$$G = \left(N_n - N_p\right) \times \beta \tag{2.2}$$

The next step is to calculate a measure for the learning difficulty for each minority sample. This measure is based on the number of samples in its close vicinity that are majority samples. For the calculation, the k nearest neighbors are taken for consideration. k is a predetermined model parameter. The number of neighbors that belong to the majority class is denoted as k'. The relative learning difficulty for sample i is calculated as given in Equation 2.3. Finally the relative difficulty for each minority sample is normalized in such a way that the sum of all difficulties equals one.

$$r_i = k'_i / k \tag{2.3}$$

Using the normalized difficulty measure, $\widehat{r_i}$, the number of synthetic samples to generate per specific minority sample can be determined:

$$g_i = \widehat{r_i} \times G \tag{2.4}$$

The samples are generated using the same method as in SMOTE.

2.5 Model performance measures

Since the goal of this research project is to compare several methods of machine learning, it is required to define criteria on which the methods are to be compared. In this section several statistics to assess the performance are discussed.

2.5.1 Confusion matrix

The first method to analyze the performance of a classification algorithm is by using a confusion matrix. It is a method to visualize the accuracy using a table, an example is shown in Table 2.1. For the purpose of explaining the confusion matrix a classifier that classifies instances as positive or negative is assumed. Four fields have to be calculated to fill in the matrix, true positive, false positive, true negative and false negative. These fields are simple to calculate. For example, true positive is the number of occurrences correctly classified as positive and false negative are the occurrences incorrectly classified as negative.

TABLE 2.1: Confusion matrix of a two-class problem.

	Predicted positive	**Predicted negative**
Positive (P)	True positive (TP)	False Negative (FN)
Negative (N)	False positive (FP)	True Negative (TN)

For the following example, assume a classifier that classifies every instance as positive. This would result in 100% of the actual positive instances being classified as positive. If the accuracy would be purely evaluated on the actual positive instances being classified as positive such an algorithm appears to be performing perfect. To prevent this situation, the confusion matrix can be used. Since the column with the negative predictions is empty, it is clear that the algorithm is not working,

2.5.2 Accuracy

Using the values calculated for the confusion matrix it is possible to calculate several interesting statistical measures. The first of which is the accuracy, shown in (2.5). The abbreviations used in this section are identical to those given in 2.1 The accuracy is used to calculate the fraction of total predictions that is correctly classified. A random classifier will get on average half of the classifications correctly. Values above 0.5 indicate the model has a higher accuracy as random guessing. A perfect prediction has accuracy 1.0.

$$\text{accuracy} = \frac{TP + TN}{P + N} \tag{2.5}$$

A drawback of using the accuracy to assess the performance of a classifier is the so-called accuracy paradox. This paradox states that a model with a higher accuracy may have a lower predictive power. To explain this paradox assume a situation in which insurance fraud has to be detected. Two different models are available, the performance of both models is given in Table 2.2. The model belonging to the left table detects 100 out of 150 cases of fraud and has an accuracy of:

$$\frac{100 + 9,700}{150 + 9,850} = 0.980$$

The model belonging to the right table, is not able to detect any fraudulent activity, it has no predictive power. Its accuracy is:

$$\frac{0 + 9,850}{150 + 9,850} = 0.985$$

Even though the second model has no predictive power, it has a higher accuracy. This disadvantage of accuracy is important to keep in mind when evaluating the performance of different models. To avoid this paradox several other statistics have been developed to quantify model performance.

TABLE 2.2: Two confusion matrices showing the accuracy paradox. The right example has a higher accuracy but is a worse predictor.

	Prediction	
	Positive	**Negative**
Positive	100	50
Negative	150	9,700

	Prediction	
	Positive	**Negative**
Positive	0	150
Negative	0	9,850

2.5.3 Precision and recall

Precision, or positive predictive value, is the fraction of positive classified instances that is true positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.6}$$

Recall, also called sensitivity or true positive rate, is the fraction of true positive instances that are classified as positive.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{TP}{P} \tag{2.7}$$

These two statistics are usually used together. Either both are given, or the statistics are combined in a different statistics, for example the F_1-score which is discussed later in this chapter.

2.5.4 F_1-score

As mentioned in the previous section, precision and recall are often combined. One of the statistics resulting from such a combination is the F_1-score. As can be seen in Equation 2.8, the F_1-score is equal to the harmonic mean of the recall and precision. A disadvantage of the F_1-score is that it does not take the true negatives into account.

$$F_1\text{-score} = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2.8}$$

2.5.5 Area under the curve

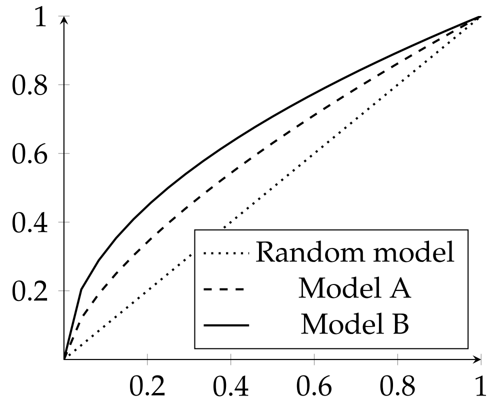
To understand the details of the area under the curve, it is required to first explain the receiver operating characteristics (ROC) curve. It is used for visualizing classifier performance and has long been used in signal detection theory to depict the trade off between true and false positive rates of classifiers (Fawcett, 2006).

An ROC curve is created by plotting the true positive rate (TP/P) against the false positive rate (FP/N) for different thresholds. Since classifiers calculate a score between 0.0 and 1.0, a threshold has to be chosen as border between positive and negative classifications. The calculated score, x, can be can be seen as being sampled from a continuous random distribution X. An instance is classified as positive if $x > T$, with T being the chosen threshold. Different thresholds will result in different true and false positive rates.

Figure 2.3 shows three examples of an ROC curve: a random model and two models with predictive capabilities. The ROC curve of a random model approaches the line stretching from $(0, 0)$ to $(0, 1)$. The reason behind this behavior is best explained with an example. Assume that a random fraction K is classified as positive, then a fraction K of the instances that should be classified as positive will be correctly classified, and the same fraction K of values that should be negative will be correctly classified as negative. For models that perform better than random guessing, the true positive rate will be higher than the false positive rate and thus the model will have a ROC curve above the diagonal.

The **area under the curve**, AUC, is a measure that tries to summarize the ROC curve in a single number. It is important to note that it is impossible to summarize the curve in a single number without loss of information. The name of the AUC is very accurate, it is the area under the ROC curve. For the ROC curve of the random model graphed in Figure 2.3, the AUC is exactly 0.5, for Model B the AUC is approximately 0.67. Models that are better than a random classifier have an AUC above 0.5, a perfect classifier has an AUC of 1.0.

FIGURE 2.3: Graph containing the ROC of a random classifier and two better performing classifiers.



2.6 Cross-validation

When training models to make predictions, it is needed to have a method for estimating how accurately the model will make predictions in practice. In cross-validation the data is split in a set used to train the model, the training set, and a set against which the models is tested, the test set. Many types of cross-validation rely on multiple iterations to reduce variability. In the following sections several methods are discussed.

2.6.1 Holdout method

The holdout method randomly splits the data set into two sets, d_0 and d_1, the training set and test set, respectively. The model is trained using d_0 and validated using d_1. Usually a majority of the samples is assigned to d_0. Typical splits in data mining applications range from 30:70 to 10:90 (Zhang, 2009). The disadvantage of this method is the usage of a single train/test split, this makes the method susceptible to random variations.

2.6.2 Repeated random sub-sampling validation

This method works by repeating the holdout method. Because of this repetition, it is also known as Monte Carlo cross-validation. In each replication the data set is randomly split into a training and test set. The results are averaged over all iterations. The disadvantage of this method is that some samples may never be used as validation whereas other may be selected multiple times.

2.6.3 *k*-fold cross-validation

In k-fold cross-validation the data set is shuffled and split into k equally sized sub sets. Of these k sets, a single set is retained to be used as test set, the remaining k–1 sets are used as training set. This process is repeated k times, with each of the sub sets being used as test set once. The performance is averaged over all iterations to get an accurate estimation of model performance. Rodríguez, Pérez, and Lozano (2010) used a sensitivity analysis to determine that k should usually be 5 or 10 when the method is used for error estimation.

2.7 Ensemble models

The basic idea of ensemble models is to combine several individual classifiers into a single composite classifier (Rokach, 2009). The goal of creating such a composite classifier is to obtain a model performing better than would be possible with a single classifier. Rokach (2009) has determined four factors to describe the differences between ensemble models.

– **Ensemble size** - The number of classifiers in the ensemble.

– **Combining method** - In an ensemble model each classifier makes a prediction. To come to a final classification the separate results have to be combined according to a certain method.

– **Diversity generator** - Combining classifiers into an ensemble only has an effect if the individual classifiers are not identical. Most methods to create diversity are based on differences in input data or differences in model design.

– **inter-classifiers relationship** - Ensemble models can be divided in sequential and concurrent models based on whether the individual models influence each other.

Several ensemble models are described in Section 2.8.

2.7.1 Combining method

The most simple form of combining classification is hard voting. This method allows each classifier to cast one vote, it assigns the instance to the class which received most votes (Ali and Pazzani, 1996). In case of a draw the instance is assigned randomly to one of the classes with the most votes.

In soft voting the individual classifiers all determine the probabilities of the instance belonging to each class. These probabilities are summed for each class and the instance is assigned to the highest probability class. This method limits the classifiers in the ensemble to those that base classifications on a probability.

2.7.2 Diversity generator

Diversity in an ensemble model can be generated in several ways. The first approach is to train each of the models in the ensemble on a different part of the data set. In most of these approaches the data is separated randomly, but other more structured approaches exist. Another approach is to use different algorithms, this method can be combined with randomly selected data. Many algorithms are used, for example one that changes the weight of the samples at each iteration based on the difficulty the algorithm has with classifying that sample.

2.7.3 Inter-classifiers relationship

As mentioned before ensemble models can be categorized based on if and how the individual classifiers have an influence on each other. Concurrent ensemble models have no interaction between the individual classifiers, each classifier is calculated independently. A major advantage of concurrent models is the high computational speed, because the classifiers are not influenced they can be calculated simultaneously. With modern multi-core processors this reduces the computation time compared to sequential calculating.

The other group consists of the sequential ensemble models, these all have some kind of interaction between individual classifiers. An important group of sequential models comes in the form of boosting models. These models work by repeatedly training a weak learner on various selections of training data. The data selection is based on the results of previously trained weak learners. After reaching a predetermined stop criterion all weak learners are grouped and can be used to make a classification.

2.8 Model descriptions

In this section several machine learning algorithms are discussed. Of these algorithms a description will be given in combination with a mathematical formulation.

2.8.1 Logistic Regression

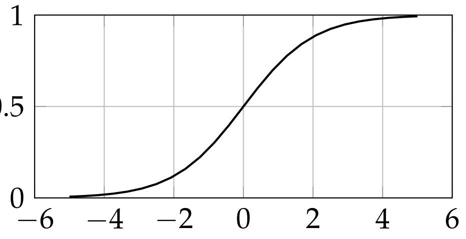
The first of the machine learning algorithms discussed, is Logistic Regression. It has been proposed by Cox (1958), making it one of the older machine learning algorithms. The primary idea of Logistic Regression is to use techniques developed for linear regression to model the probability of a sample belonging to a certain class. This is done using a linear predictor function, Equation 2.9, which is a linear combination of m feature values and $m+1$ regression coefficients.

$$f(i) = \beta_0 + \sum_{i=1}^{m} \beta_i x_i \tag{2.9}$$

Logistic regression is different from other forms of regression due to the way the linear predictor is linked to the probability of a certain outcome. It transforms the output of the linear predictor using the logit function, depicted in Figure 2.4, which is the natural log of the odds. An advantage of using the logit function, is that it takes any real value as input and returns a value between zero and one.

$$logit(p) = ln\left(\frac{p_i}{1-p_i}\right) = f(i) \tag{2.10}$$

FIGURE 2.4: The standard logistic function.



Using the above transformation of the linear predictor, the following equation for the probability of a positive sample can be determined.

$$p(x) = (1 + e^{\beta_0 + \sum_{i=1}^{m} \beta_i x_i})^{-1} \tag{2.11}$$

What remains now, is to describe a method that can be used to determine the coefficients. Unlike in linear regression, it is not possible to determine a closed form equation to determine the coefficients. Instead other methods like maximum likelihood estimation are used. In this method an iterative process is used during which in each iteration the coefficients are slightly changed to try to improve the maximum likelihood. In this research project two methods to fit the model are taken in to account. These methods will not be discussed in depth since it is not within the scope of this research project. The first method is Liblinear, it used a coordinate descent algorithm to find suitable values for the coefficients. The second method is saga which uses a stochastic average gradient descend. The second method usually is faster on large data sets.

In the loss function that is minimized it is usual to include a regularization term. Such a term is to penalize complex models and favor models which are simpler. With Logistic Regression two types of regularization are commonly used, L1 and L2. The first of these is a regularization that favors sparse models, or models where a large fraction of the coefficients is zero. L2 is used as regularization term when a sparse model is not suitable. When the data set contains highly correlated features, L1 should be used as regularization term. It picks a single of the correlated features and sets the coefficient of the other features to zero. L2 would simply shrink the coefficient of all correlated features. Usually a parameters is added to the algorithm which can be used to determine the strength of the regularization.

2.8.2 *k* Nearest Neighbors

Nearest Neighbors classification is a method that bases a classification on the *k* samples closest to the instance that has to be classified (Larose, 2005). This algorithm does not attempt to induce a model, it simply stores instances of the training data, making it a so-called lazy algorithm.

The basic Nearest Neighbors classification uses the same weight for each of the *k* selected neighbors. In some situations it might be better to differ the weight of the neighbors based on the distance to the sample that has to be classified. Meaning the weight will be inverse to the distance, so the closest samples will get the greatest weight.

Brute-force

The most basic computation method for the Nearest Neighbor classification is brute-force. This algorithm simply calculates the distances between all points in the data set and uses those to determine which points are closest. For small sample sizes this algorithm can return accurate result. Due to its naive nature, brute-force quickly becomes an unfeasible approach when sample size increases.

k-d tree

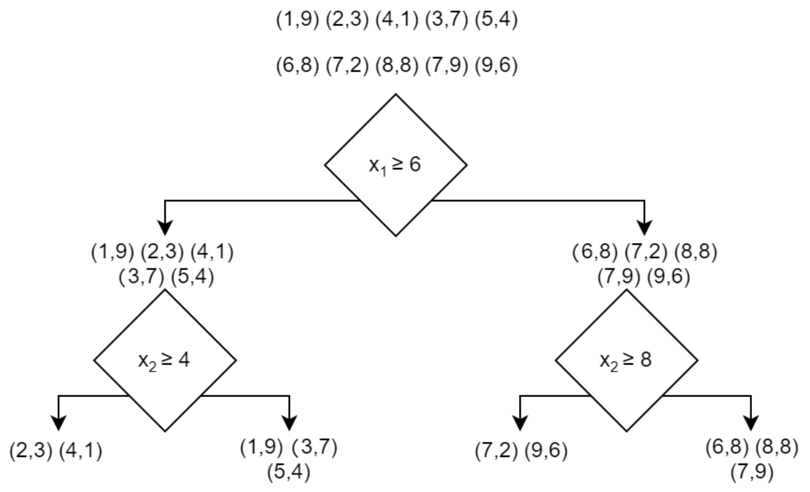
To counter the issue of the brute-force method being unfeasible for larger sample sizes, a more efficient method has been developed by Bentley (1975). This method used a decision tree to efficiently store distance information requiring less computations. Assume three points A, B and C, of these points A and B are very distant and C and B are close. From this information it follows that points A and C are also very distant.

A *k-d* tree is constructed by iterating over several steps. In each iteration a (not previously used) feature is selected at random, on which a decision will be based.

The median value of the selected feature is calculated, values larger than this median are separated from the smaller values. Now two branches have been created, each with approximately half of the samples. On both branches these steps are repeated. This process continues until the number of samples in a branch drops below a certain threshold. An example of the result of this process is shown in Figure 2.5.

After a tree has been generated, the approximately closest neighbors can easily be determined. All nodes of the tree are applied on the new instance, the branch where the new instance ends up in, contains samples that are close. For all of these samples the distance to the new instance is calculated to find nearest ones.

FIGURE 2.5: Schematic overview of a *k-d* tree.



Ball tree

In high dimensional space it becomes computationally expensive to create a *k-d* tree. In those situations it is computationally favorable to create a ball tree (Bhatia, 2010). Omohundro (1989) describe a ball tree as a binary tree of which each node represents a hypersphere called a ball. Each node of the tree splits the data into two disjoint sets, each set is contained by the smallest ball containing all points. The hyperspheres are allowed to cross, data points are assigned to the sphere of which the center is closest.

2.8.3 Naive Bayes

The Naive Bayes classifier is based on statistical theory, more specific on Bayes' theorem, Equation 2.12. Using Bayes' it is possible to calculate the probability that a certain hypothesis is true given observed evidence. The naive part comes from the fact that in Naive Bayes, it is assumed that all of the features are independent from each other.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.12}$$

A clear example of a use for Bayes' theorem can be found in drug testing. Suppose 0.5% of people are users of a certain drug and a drug test produces 99% true positives and 99% true negatives. Using Bayes' theorem it is possible to calculate the probability that a random person who tests positive is a drug user.

$$P(U|+) = \frac{P(+|U)P(U)}{P(+)}$$

$$P(U|+) = \frac{P(+|U)P(U)}{P(NU)P(+|NU) + P(U)P(+|U)}$$

$$P(U|+) = \frac{0.99 \times 0.005}{0.995 \times 0.01 + 0.005 \times 0.99} = 33.2\%$$

The above calculations show that the aforementioned probability equals 33.2%. Even though the intuitive answer would be 99%.

Since Bayes' theorem can be used to calculate the probability that something is true given evidence, it can be used to calculate the probability that a new sample belongs to a certain class given the evidence. Classification is done by calculating the probability that the sample belongs to a class, for each class. The sample is assigned to the class with the highest probability. In the above example $P(U|+) = 33.2\%$ so $P(NU|+) = 66.8\%$, therefore the drug tested person is assigned to the 'no drug user' class. In the calculations for the different classes the evidence, or sample, is identical, therefore it is possible to discard the denominator of Bayes' theorem and still make the same classification.

2.8.4 Decision Tree

A Decision Tree probably is one of the best known classifiers due to its logic structure. Decision Tree classifiers are extensively described by Rokach and Maimon (2009). A Decision Tree consists of connected nodes which form a rooted tree, meaning that the tree has a single root node as starting point. All following nodes have a single incoming edge, if the node also has outgoing edges it is called an internal node. Each of the internal nodes splits the data set according to a certain logic. In classification this split is usually based on the value of a certain feature. Nodes that do have incoming edges but no outgoing edged are called leaves. Leaves are assigned to a label based on which label is most appropriate. After a tree has been constructed, the classification is done by starting at the root node and following through the internal nodes until a leave has been reached

Constructing an optimal Decision Tree is only feasible for small problems due to computational requirements (Hancock et al., 1996). This results in the need for heuristic algorithms. In this research the CART algorithm will be used. A Decision Tree is trained on a feature set containing $X = x_i, ..., x_n$ and corresponding labels $Y = y_i, ..., y_n$. At each node m the relevant part of the set is represented by Q_m. The construction algorithm tries to find a split $\theta = (j, t_m)$, with feature j and threshold t_m, that splits Q into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ with a minimized impurity. Several measures can be used to calaculate the impurity of which Gini and Entropy are widely used. Equation 2.13 shows how the Gini is calculated, p_{mk} is the probability of a sample with label k being in node m and $I(y_i = k)$ is one if $y_1 = k$ and otherwise zero.

$$p_{mk} = \frac{1}{N_m} \sum_{i \in R_m} I(y_i = k)$$
$$H(X_m) = \sum_k p_{mk}(1 - p_{mk}) \tag{2.13}$$

By combining the weighted impurity of Q_{left} and Q_{right} a measure for the split is constructed, Equation 2.14. The goal is to find θ^* that minimizes this measure.

$$G(Q_m, \theta) = \frac{n_{left}}{N_m} H(Q_{left}(\theta)) + \frac{n_{right}}{N_m} H(Q_{right}(\theta)) \tag{2.14}$$

This process is executed in a recursive manner. After each iteration the process is repeated for Q_{left} and Q_{right} until a stop criterion is reached. This criterion can be a maximum depth or a minimum number of remaining samples.

2.8.5 Artificial Neural Network

An Artificial Neural Network is is a network of interconnected neurons. This section is based on Zhang (2009). Each neuron receives an input, processes these signals and produces an output signal. A Neural Network consists of several layers of neurons, see Figure 2.6 for a schematic overview. The leftmost layer consists of the input neurons, the rightmost neurons are the output neurons. The layers in between are called hidden layers. The neurons are connected in such a way that the output of one neuron is the input of all neurons in the next layer. The exceptions are the input and output neurons. Input neurons have no predecessor and are used as input for the network. Output neurons have no successor and function as network output. In classification problems with two classes only a single output neuron is used. Depending on the value of the output neuron the sample is assigned to either class. When more then two classes exist, each class has a corresponding output neuron and the sample is assigned to the output neuron with the highest value.

FIGURE 2.6: Schematic overview of a Neural Network.

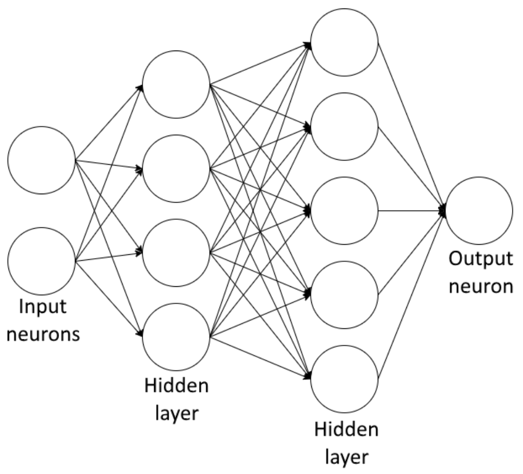


Figure 2.7 shows a schematic overview of how an artificial neurons operates. A neuron receives inputs x_{ij} with weights w_{ij}. With i being the neuron from which the signal is the output and j the neuron which receives the signal as input. The input signals are multiplied with their corresponding weights and summed. The weights

are usually given as a matrix W_l, containing all weights for layer l. The weights are changed repeatedly during the training process and do not have to sum to 1.

$$p_j = \sum_i w_{ij} x_{ij} \tag{2.15}$$

Since a neuron has to output a signal, the next step is to transform the calculated input using a transformation function. Such a function can be of any form, but most used are logistic sigmoid, hyperbolic tangent or rectified linear unit (Schmidhuber, 2015).

FIGURE 2.7: Schematic overview of a single artificial neuron.

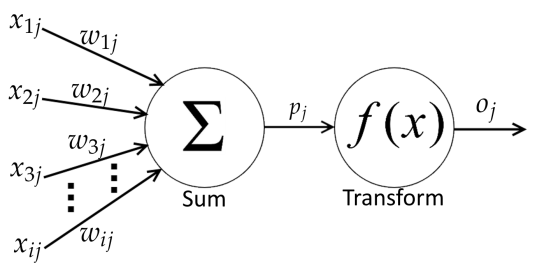
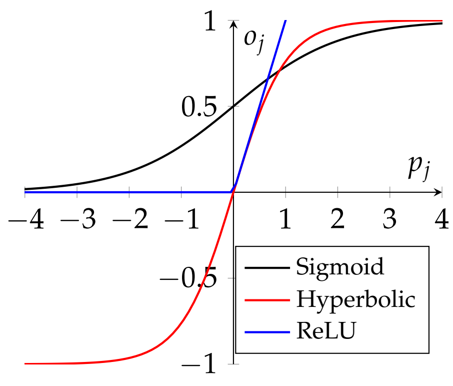


FIGURE 2.8: Three different activation functions.



The equation of the logistic sigmoid is given as Equation 2.16 and drawn in Figure 2.8 as a black line. The function has a minimum value of 0 and a maximum of 1. The equation of the logistic sigmoid is shown below. An advantage of using the sigmoid as activation function is its bounded nature. This causes the network itself to be bounded, if $p_j \gg 1$, the transformation function will never result in a value higher than 1. In many cases this is a desirable characteristic, because it prevents a single neuron from dominating the network. However, it also has a disadvantage, all input values have to be normalized ($\mu = 0, \sigma = 1$) in order to have a meaningful impact on the output.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

The second transformation function is the hyperbolic tangent, it is drawn as the red line in Figure 2.8 according to Equation 2.17. The hyperbolic tangent has a shape similar to the logistic sigmoid. The biggest difference are the bounds, where the logistic sigmoid returns values between 0 and 1, the hyperbolic tangent returns values between -1 and 1. Due to its bounds it also requires input values to be normalized. The two advantages of using the tangent over the logistic sigmoid arise from its symmetry across the x-axis (LeCun et al., 1998). The first advantage is faster convergence in comparison with the logistic sigmoid. The second advantage is that the output will on average be closer to zero due to the possibility of negative values. Since the output of a neuron is often the input for a next neuron having an average output of zero is preferable (remember how that input data is normalized).

$$f(x) = \tanh(x) \tag{2.17}$$

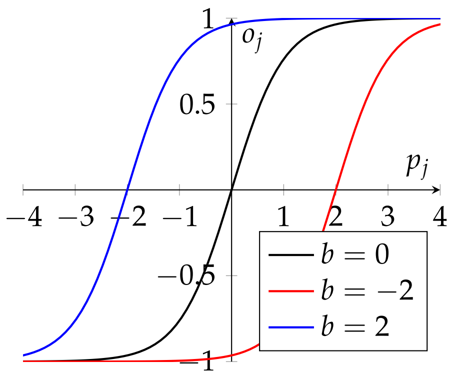
The last transformation function taken into account for this research is the rectified linear unit (ReLU). It is a function with a lot of recent attention, according to Ramachandran, Zoph, and Le (2017) it is the most used activation function as of 2017. The equation of the ReLU is given in Equation 2.18 and is graphed in blue in Figure 2.8. The ReLU has a completely different shape compared to the previous functions, it is simply the positive part of the input p_j. This activation function was first proposed by Hahnloser et al. (2000) based on biological motivations. According to Glorot, Bordes, and Bengio (2011), networks using the ReLU activation function outperform networks using the logistic simoid function.

$$f(x) = x^+ = \max(0, x) \tag{2.18}$$

As mentioned before in this section the input for the transformation is the weighted sum of the input signals p_j as shown in Equation 2.15. However, this is not necessarily true in most models. Usually a bias b_j is added to the input value. This bias shifts the activation function to the left or right, as can be seen in Figure 2.9. The bias is a value that can be changed during the learning process.

$$o_j = f(p_j + b_j) \tag{2.19}$$

FIGURE 2.9: Impact of three different bias values on the hyperolic tangent activation function.



Network training

Now that the components of a Neural Network are known, the mechanics to train the network have to be made clear. First a broad description of the training process is given and then the mathematical formulation is discussed.

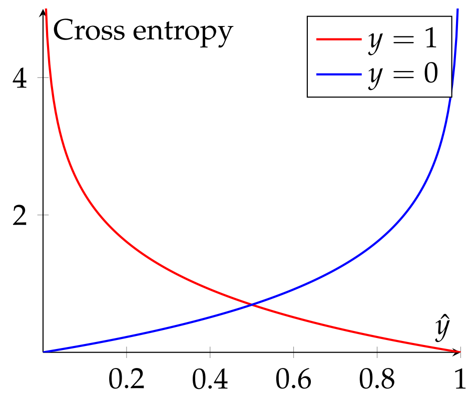
Training a network for classification is a form of supervised learning (2.3.2), the network is presented with samples and their known label. At first the weights w_{ij} of the signal traveling from neurons i to j are chosen at random. The network is then presented with the training samples and the resulting classifications are stored. The predicted classes are compared to the real classes, the difference is expressed in some measure for the error. The first order derivative of the error with respect to the weights is calculated. Based on this derivative the weights are changed in such a way that the error decreases fastest. These steps are repeated many times until the improvement drops below a certain threshold.

As mentioned in the previous paragraph a measure has to be used to express the difference between the real and predicted results. Such a measure is called a loss function. For classification problems a loss function based on cross-entropy is used. Cross-entropy is commonly used to quantify the difference between probability distributions. For binary distributions cross-entropy is expressed as:

$$-y \ln \hat{y} - (1-y)\ln(1-\hat{y}) \tag{2.20}$$

with $0 \leq \hat{y} \leq 1$ being the predicted value and $y \in \{0,1\}$ the real value. The cross-entropy tends to zero when the model makes a good prediction. For example assume $y = 1$ and $\hat{y} \approx 1$, the first term, $y \ln(\hat{y})$, approaches zero and the second term, $(1-y)\ln(1-\hat{y})$, equals zero, thus the cross-entropy is close to zero. The cross-entropy also approaches zero when $y = 0$ and $\hat{y} \approx 0$. When the prediction of the network gets less accurate, the cross-entropy increases. Figure 2.10 shows the cross-entropy for different values of y and $\hat{y}$.

FIGURE 2.10: Value of cross-entropy for different values of y and $\hat{y}$.



The loss-function used for training a Neural Network is the average cross-entropy over all training samples including a penalty for complex models.

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}[y \ln \hat{y} + (1-y)\ln(1-\hat{y})] + \alpha\|W\| \tag{2.21}$$

$\alpha\|W\|$ is a regularization term that penalizes complex models. The exact formulation of the regularization is not within the scope of this research and not required to understand the topic. The importance given to the complexity penalty is controlled by $\alpha > 0$. The justification of penalizing complex models can be found in Occam's Razor. The theorem states that when having two theories with identical outcomes, the simpler one is the better.

The next step in the training process is to change the weights and biases based on the loss calculated using Equation 2.21. Updating the parameters happens according to the following equation.

$$w_{i,j}^{new} = w_{i,j}^{old} + \eta\frac{\delta\mathcal{L}}{\delta w_{i,j}} \tag{2.22}$$

From the equation, the new weight is determined by updating the old weight with the partial derivative of the loss with respect to the weight under consideration. This method is called the gradient steepest descent, because it uses the derivatives to determine the direction of the weights that decreases the error most. At first it might seem strange that it is possible to calculate the derivative of the error. But since each neuron in the network transforms the input values using a predefined formula, the entire network can be expressed as a single formula using the following approach.

$$
\begin{aligned}
o_j &= f(p_j + b_j) \\
o_j &= f(\sum_i w_{ij} x_{ij} + b_j) \\
o_j &= f(\sum_i w_{ij} o_{j-1} + b_j) \\
o_j &= f(\sum_i w_{ij} f(p_{j-1} + b_{j-1}) + b_j)
\end{aligned}
$$

The exact calculation of the derivative is not within the scope of this research. The derivation can be very complex and depends on the activation function. To understand the general idea of the approach is enough to keep track of the research project.

The derivative is multiplied with a factor η before being added to the weight. This factor is called the learning rate and determines the size of the steps when updating the weights. A small learning rate might cause the optimization to get stuck in a local optimum and a high learning rate can result into overshooting. Usually a value between 0.0001 and 1 is chosen.

Equation 2.22 seems to have no variables representing the biases in the network, meaning the biases will not be updated. However, using a trick it is possible to treat the biases as weights in the training process. Substituting Equation 2.15 into 2.19 results in Equation 2.23. From this it is clear that the bias can be assumed to be the weight of a signal always equaling 1.

$$o_j = f(\sum_i (w_{ij} x_{ij}) + b_j)$$

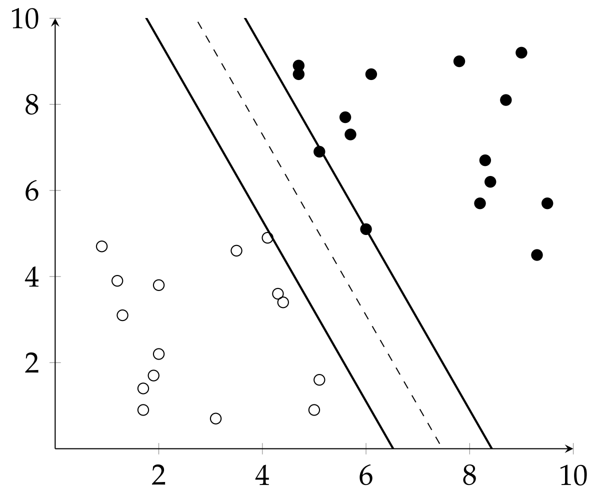
$$o_j = f(\sum_i (w_{ij} x_{ij}) + b_j \times 1) \tag{2.23}$$

After the biases and weights have been updated the same steps are repeated. Each iteration of the training process is called an epoch. When between two epochs the loss decreases less than a certain threshold, training stops.

2.8.6 Support Vector Machines

The next type of model discussed in this chapter is the Support Vector Machine, or SVM (Shmilovici, 2009). This model will only be discussed on a high level, it is one of the more complex models, the exact mathematical derivation falls not within the scope of this research project. SVM works by constructing a hyperplane in a high dimensional space. It is constructed in such a way that it splits the classes of the training samples in such a way that the distance to the closest sample of either class is maximized. The plane is used as a separation border for classifying new samples. A hyperplane is a plane whose dimension is one less than its ambient space. In other words, points in a three dimensional space are separated by a two dimensional plane. Points in a two dimensional space are separated by a one dimensional line. An example of the application of SVM is shown in Figure 2.11. As can be seen in the graph, the dashed line splits the classes in such a way that the distance to the closes sample of either class is maximized.

FIGURE 2.11: SVM generated maximum margin hyperplane in a two class two dimension problem.

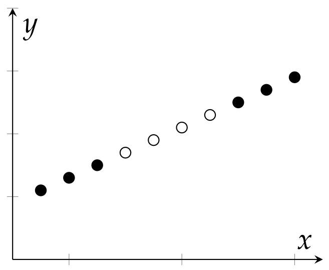


The above description clearly has a large flaw, what will happen when the samples are not linearly separable? Without deviating from linearity, the solution is to add a penalty for samples on the wrong side of the separation. This penalty is multiplied with a certain weight to determine the trade-off between the margin around the hyperplane and the samples on the wrong side. This weighted penalty is added to the hyperplane margin and finally minimized.

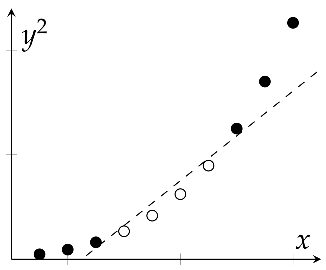
Later a more advanced method was created to separate samples using a non-linear classifier. This method is achieved by applying the so called 'kernel trick'. The general idea of the method is to map the samples to a higher dimensional space. An example showing the benefit of this transformation is shown in Figure 2.12. By mapping data from a $\{x, y\}$ space to three dimensions, $\{x, y, y^2\}$, it becomes possible to separate the two classes with a line. Each type of transformation is called a kernel, some examples are polynomial, Gaussian and hyperbolic.

FIGURE 2.12: Mapping data to a higher dimension to achieve linear separability.

(A) Two dimensional $\{x, y\}$ space.



(B) Two dimensions of the $\{x, y, y^2\}$ space.



2.8.7 Random Forest

Another popular ensemble model is the random forests classifier. The basic principle of this classifier is to train multiple Decision Trees and have those together make a classification (Breiman, 2001). Each of those trees is trained on a subset of the training data drawn with replacement. The training procedure is similar to how a normal Decision Tree is trained except for one difference. At each split in the tree a random selection of features is selected, from which the feature for the split is selected. Usually the square root of the number of available features is used for how many features have to be drawn (Hastie, Tibshirani, and Friedman, 2009). The reason for this random feature selection is to decrease the correlation between the individual trees.

Given a feature set $X = x_i, ..., x_n$ and corresponding labels $Y = y_i, ..., y_n$, for each tree in the random forest a random subset X_r and Y_r is drawn with replacement. To each of the sets of random samples a Decision Tree is fitted. At each split in the tree a random subset of features is selected on which the split can be based. For a classification with p features the most used number of features considered for a split is $\sqrt{p}$ or $\log_2(p)$. This tree construction process results in N separate Decision Trees which are combined in a single classifier. This can be done either by letting each classifier cast a vote or by averaging the probabilistic predictions.

2.8.8 AdaBoost

AdaBoost is a popular ensemble model developed by Freund and Schapire (1997). It is a sequential model that works by combining multiple weak learners into a strong learner. The algorithm works by adding a weak learner to the classifier in each iteration, until a certain criterion has been met, usually the amount of weak learners. The weak learners are trained using a weighted training set in which weights are based on the performance of the ensemble model so far. Misclassified training samples are assigned a greater weight, this weight is incorporated in the equation used to calculate the error of a weak learner. The result is that a greater importance is given to samples with a large weight. A weight is also given to each of the weak learners. This weight is based on the performance of the weak learner in such a way that it minimizes the training error of the ensemble model so far.

In theory an AdaBoost classifier can be constructed using any type of classifier that is able to calculate probabilities. However, AdaBoost is most often used with single level Decision Trees, called decision stumps.

A more formal definition of AdaBoost follows. The algorithm combines T weak learners into an ensemble. At each iteration $t = 1, 2, ..., T$ a new weak learner is added to the ensemble. These weak learners are trained using a set, containing m samples, in which each sample has a weight w_i. At the start of the algorithm, each sample is given an equal weight, $1/m$. The samples all have a label that is based on the true classification, it denoted as -1 or 1. Using this training set a weak learner h_t is trained of which the error ϵ_t is calculated.

$$\epsilon_t = \frac{\sum_{y_i \neq h_t(x)} w_i}{\sum w_i}$$

In each iteration a weight has to be determined for the newly generated classifier. It can be shown that the error of AdaBoost is minimized if the weight for the classifier h_t is defined as follows.

$$\alpha_t = 0.5 \ln \frac{1 - \epsilon_t}{\epsilon_t} \tag{2.24}$$

Aside from the weight for the weak learner, the weights for the samples also have to be updated in each iteration. This is done in such a way that misclassified samples receive a greater weight and thus more importance in following iterations.

$$w_i^{(t+1)} = w_i^{(t)} \times e^{-\alpha_t y_i h_t(x_i)}$$

The final model $A(x)$ is reached when the algorithm has run for T iterations. Each weak learner $h_t(x)$ makes a classification of -1 or 1, if the weighted sum of these classifications is positive then $A(x) = 1$, if the weighted sum is negative then $A(x) = -1$. The signum function returns 1 or -1 based on the input value being positive or negative.

$$A(x) = \text{signum}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2.25}$$

2.8.9 Gradient Boosting

Gradient Boosting is another classification algorithm based on an ensemble of Decision Trees, it is proposed by Friedman (1999). The difference with ADABoost is in the way the new trees are trained. In this algorithm each new tree is trained on the so called residual. This is the difference between the actual value and the value computed by the model. The goal is to get a smaller residual after each time a new tree is added, logically this should lead to a better prediction.

Assume that a feature set $X = x_i, ..., x_n$ and corresponding labels $Y = y_i, ..., y_n$ are given. The algorithm starts by training a single Decision Tree, this results in the model $F_m(x)$. As with the previous ensemble model this model is improved by adding an additional Decision Tree $h(x)$, resulting in the improved model $F_{m+1}(x)$.

$$F_{m+1}(x) = F_m(x) + h(x)$$

In the hypothetical situation with a perfect predictor, $F_{m+1}(x)$ will be equal to y:

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

and thus:

$$h(x) = y - F_m(x).$$

The goal of Gradient Boosting is to fit the additional tree, $h(x)$, to the residual $y - F_m(x)$. The idea is that adding predictors for the residual will decrease the residual and thus let the algorithm predict a value closer to the true value.

One of the problems of Gradient Boosting is that it is vulnerable to overfitting. Therefore a learning rate is usually added to the algorithm which shrinks the contribution of each next tree. This slows down the speeds with which the algorithm learns and makes it easier to let the learning stop before it starts to overfit.

Chapter 3

Methodology

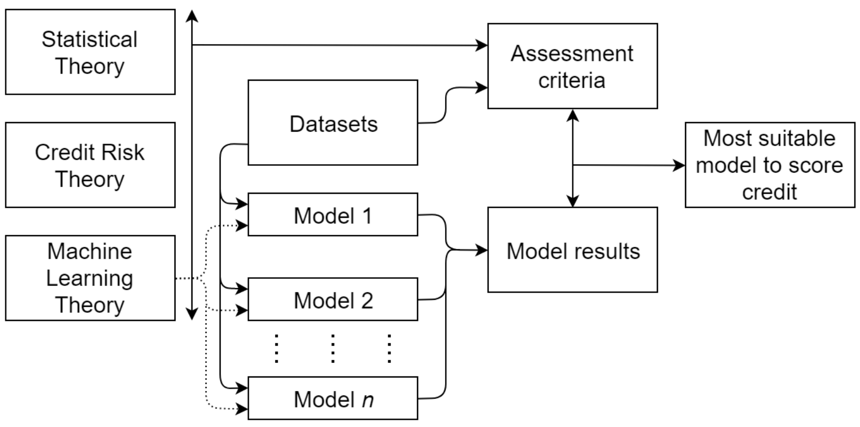
Now that the required theoretical background has been discussed, the next step is to describe in more detail how the objectives of this research project will be achieved. This experiment design consists of a combination of topics discussed in the previous chapter. The design is split into two stages, the first stage is the data preparation, the second stage is the model training and testing.

3.1 Research framework

In this section, the approach to achieve the research objectives will be discussed. This approach forms the research framework and is summarized as a visual representation in Figure 3.1.

The first step in the research framework is to gather and read current scientific literature on three topics. These topics are machine learning theory, credit risk theory and statistical theory. The machine learning theory is used to create different models that are capable of classifying credit based on the likelihood of default. These models are first confronted with a data set to train the models and then confronted with a data set to get the model results. All three sources of literature are used to create assessment criteria to objectively compare the performance of the models. Part of the assessment criteria is the performance of a benchmark method representing the currently used credit scoring method. For this benchmark logistic regression is used. The last step in the research framework is to draw conclusions based on the confrontation between the model results and the assessment criteria.

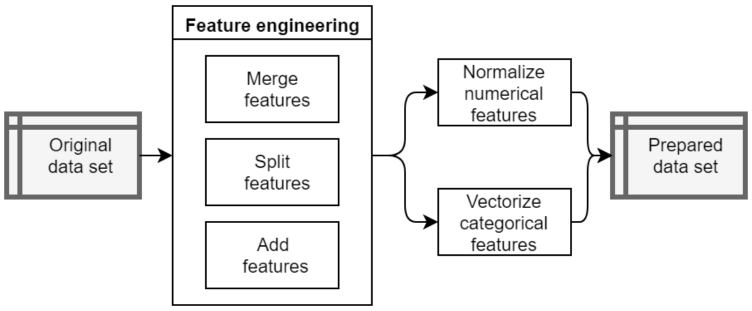
FIGURE 3.1: Graphical representation of the research framework.



3.2 Data preparation

As mentioned above, the first stage in conducting the experiment is to prepare the data for usage in machine learning. This stage is schematically shown in Figure 3.2. Logically, the data preparation starts with the original data set as retrieved from the source. To make the data set ready for machine learning three steps have to be performed.

FIGURE 3.2: Schematic overview of the data preparation.



The first step in the data preparation process is feature engineering. In this step the features will be analysed and changed to better represent the information in the data set. Feature engineering is not a static approach that is identical for each data set, it is a dynamic process that is highly dependent on the features in the original data set. It might consist of merging, splitting or adding features. An example could be to split a feature that contains information about two characteristics. In one of the data sets used in this research, a single feature contains information about late payments and about no use of available credit. It might be a good idea to represent the data more logically by splitting the feature. The effect of the engineered features should be tested by training and testing the algorithms on the altered data set. Another possible operation can be to divide to features to get a ratio, for example how close a balance is to its limit.

Feature engineering is followed by the vectorization of categorical features. Vectorizing is the process of making a separate feature for each of the possible values the original feature can take. The new columns are binary and thus can only take values of zero or one. The reason for vectorization is to make the feature easier to "understand" for machine learning algorithms. Assume for example a data set that includes a feature for the gender of clients. The feature has a value of 1 if the client is male and a value of 2 if female. Due to the nature of machine learning algorithms an order will be assumed in the values, the feature will be treated as being numerical. This clearly leads to a misrepresentation of the feature in the learning process. To counter this issue, the feature has to be vectorized into separate binary features. This process will result in two separate gender features, one for male and one for female.

The goal of normalization is to transform the values of a feature in such way that it has a mean of zero and a standard deviation of one. Normalization of the values is done using Equation 3.1, where z is the normalized value, X the original value, μ the average of the feature and σ the standard deviation of the feature.

$$z = \frac{X - \mu}{\sigma} \tag{3.1}$$

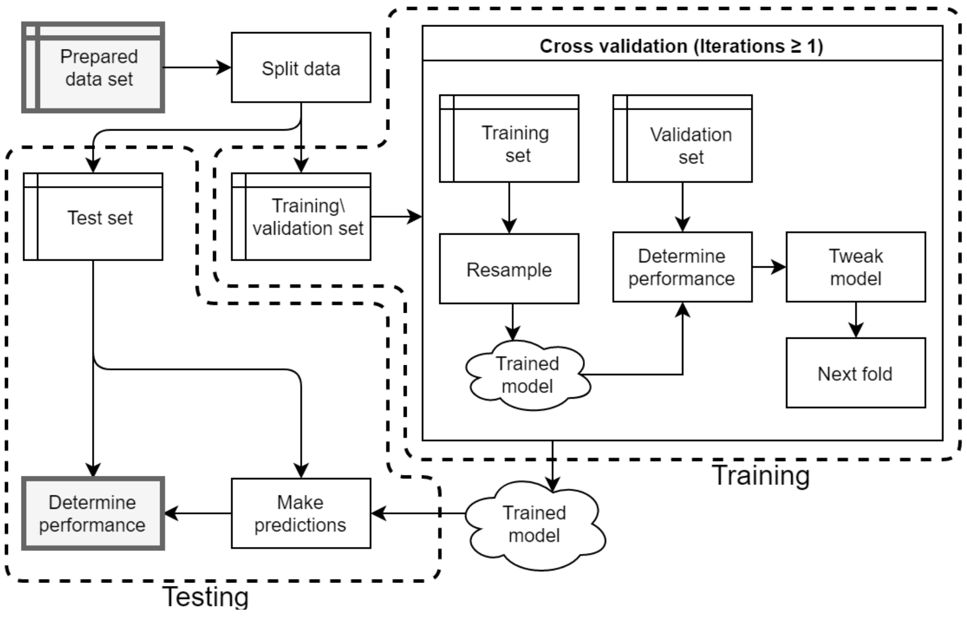
The reason for this transformation is to change all numerical features so that they have the same mean and standard deviation. If features are on a different scale, one might have a bigger influence than the other. This difference in influence does not necessarily occur, but on average machine learning models learn faster and more accurate with normalized features (Zhang and Qi, 2002). This transformation should also be applied to newly generated numerical features during feature engineering.

After these three steps have been completed, the data set is ready to be used for machine learning. The preparation of the different data sets is discussed in detail in the next chapter, Chapter 4. The effects on model performance are determined in that chapter to come to conclusions on the impact of data preparation.

3.3 Model training and testing

The next phase of the experiment is the training and testing phase. The data set constructed in the previous phase is used to train and test the different algorithms. An overview of this phase is schematically shown in Figure 3.3. As can be seen in the figure, this phase can roughly be split in two separate processes, training and testing.

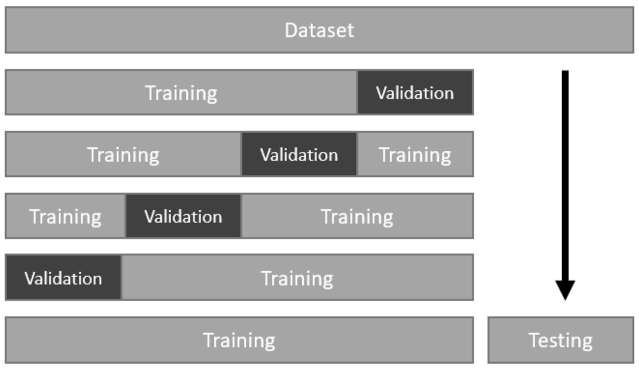
FIGURE 3.3: Schematic overview of the training and testing phase.



3.3.1 Cross-validation

Before either training or testing can start, the data set first has to be split in a training and testing set, which will respectively be used for training and testing the algorithms. The reason for this split is to test the final algorithm settings on an unseen data set, this is explained in more detail in Section 2.6. In this experiment a split of 25:75 will be used, with the latter being the training set. Figure 3.4 shows how a fraction of the initial data set is separated and only used in the last phase to test the model.

FIGURE 3.4: Schematic overview of how cross-validation is used.



3.3.2 Training

The training phase is more than simply training the models on the data set. It is an iterative process to update the model parameters with the goal to increase the predictive performance. The input of this phase is the training set created in the previous phase. In each iteration the parameters of the algorithm are changed and the performance is determined using k-fold cross-validation. This form of validation has been explained in detail in Section 2.6. In short, it is a repeated process where an algorithm is trained and validated repeatedly with the same settings. In each iteration a different part of the training set is used for training, the remainder of the training set is used for validation of the trained model. The performance of the model with the chosen parameter settings is the average of the performances achieved in each individual iteration. Figure 3.4 shows how k-fold cross-validation is applied to the data set without the test fraction of the data set. It are the steps between the first and last row.

The computations are done using an Intel Core i5-4690K CPU running at 3.50GHz. The algorithms are implemented using Scikit-learn (Pedregosa et al., 2011). This is a widely used machine learning library for the Python programming language.

After an iteration of the training is complete, the parameters will be changed and the performance will again be determined using k-fold cross-validation. This is repeated until a good understanding of the influence of the different settings is reached. To keep computation times realistic, it is not possible to calculate all possible combinations of all possible parameter values. Therefore the parameters will be analysed individually. After a good understanding of the individual parameters has been reached, combinations can be made by changing multiple parameters.

The following models will be analyzed in this experiment:

- Logistic Regression
- Neural Network
- Naive Bayes
- k Nearest Neighbors
- Decision Tree
- ADABoost
- Random Forest
- Gradient Boosting
- Support Vector Machine

3.3.3 Performance measure

In the process described in this chapter it is often required to accurately determine the predictive performance of an algorithm. In Section 2.5 several measures are discussed, a selection has to be made of which measure will be used in the experiments. The selected measure should provide an accurate indication of the model performance. It is important that the measure are not susceptible for problems like the accuracy paradox or imbalance. This results in the more simple measures to be excluded. For this project the AUC has been selected as the measure to express the performance. The theoretical background of the area under the curve is described in Section 2.5. Aside from using the AUC, in some situation like resampling and assessing the final tested performance, a confusion matrix will be used to make it possible for the reader to calculate other measures.

In machine learning, logistic regression is often used as a benchmark algorithm (Dumitrescu et al., 2018). The theoretical background of logistic regression is relatively simple and the output is understandable and explainable. It is also an algorithm with widespread usage. If a more complex model performs equal to logistic regression it is often seen as a lesser algorithm.

3.3.4 Model testing

The last step of the model training and testing phase is testing. In this step the final model performance is determined. The goal is to determine the performance using a part of the data set that has not yet been seen by the model. This makes it impossible for over fitting to result in a high performance measure. Since a part of the data set has been separated during the first phase of the experiment, that part can now be used. This step is shown in Figure 3.4 as the last row. The held back part of the data set is used to make predictions and the rest is used to train the model. Finally the results of this step will be used to draw conclusion about the performance of the different machine learning algorithms when used to predict defaults.

Chapter 4

Data description and preparation

In this research project two different data sets will be used. Each of the data sets will be discussed separately. For each set a short summary will be given and the different features are described. Next the data sets will be prepared as described in the previous chapter. After the data set has been prepared, the performance of the different resample methods is measured. The reason for using multiple data sets is to decrease the influence of the individual data sets. If a finding is supported by all two data sets it is more reliable.

4.1 Credit data - Taiwan

4.1.1 Data description

The first data set contains details of 30,000 Taiwanese credit lines (Yeh and Lien, 2009). The set is a combination of revolving credit and instalment credit. Features in this set can be split into two categories, personal and financial. The personal features consist of education, marriage, age and gender. The financial features are the credit limit, the amount paid per month and the amount on the bill statement per month. All individual features are concisely described in Table 4.1 and the types of the features in Table 4.2. A statistical summary with mean, standard deviation, percentiles and possible values is given in Table 4.3. A summary of the categorical features is given in Table 4.4 and 4.5, finally a correlation matrix for the numerical features is given in Table 4.8. It is important to note that this data set has no missing values.

TABLE 4.1: Description of the features in the Taiwan credit data set.

Feature	Description
LIMIT_BAL	Maximum of the credit given to the consumer.
GENDER	Gender of the consumer.
EDUCATION	Education level of the consumer.
MARRIAGE	Marital status of the consumer.
AGE	Age of the consumer.
PAY_1	Payment status in Sep 2005. No consumption, on time or how late.
...	...
PAY_6	Payment status in Apr 2005. No consumption, on time or how late.
BILL_AMNT1	Amount of bill statement in Sep 2005.
...	...
BILL_AMNT6	Amount of bill statement in Apr 2005.
PAY_AMNT1	Amount paid by consumer in Sep 2005.
...	...
PAY_AMNT6	Amount paid by consumer in Apr 2005.
DEFAULT	Whether or not the consumer went into default.

TABLE 4.2: Data types of the features in the Taiwan data set.

Feature	Type		Feature	Type	
LIMIT_BAL	Numerical	Ratio	BILL_AMT1	Numerical	Ratio
GENDER	Categorical	Nominal	...	...	...
EDUCATION	Categorical	Nominal	BILL_AMT6	Numerical	Ratio
MARRIAGE	Categorical	Nominal	PAY_AMT1	Numerical	Ratio
AGE	Numerical	Ratio	...	...	...
PAY_1	Categorical	Ordinal	PAY_AMT6	Numerical	Ratio
...	...	...	default	Categorical	Nominal
PAY_6	Categorical	Ordinal			

TABLE 4.3: Statistical summary of the numerical features in the Taiwan data set.

Feature	Mean	Standard deviation	min	25%	50%	75%	max
LIMIT_BAL	167,484	129,747	10,000	50,000	140,000	240,000	1,000,000
AGE	35.5	9.2	21	28	34	41	79
BILL_AMT1	51,223	73,635	-165,580	3,558	22,381	67,091	964,511
BILL_AMT2	49,179	71,173	-69,777	2,984	21,200	64,006	983,931
BILL_AMT3	47,013	69,349	-157,264	2,666	20,089	60,165	1,664,089
BILL_AMT4	43,263	64,333	-170,000	2,327	19,052	54,506	891,586
BILL_AMT5	40,311	60,797	-81,334	1,763	18,105	50,191	927,171
BILL_AMT6	38,872	59,554	-339,603	1,256	17,071	49,198	961,664
PAY_AMT1	5,664	16,563	0	1,000	2,100	5,006	873,552
PAY_AMT2	5,921	23,041	0	833	2,009	5,000	1,684,259
PAY_AMT3	5,226	17,607	0	390	1,800	4,505	896,040
PAY_AMT4	4,826	15,666	0	296	1,500	4,013	621,000
PAY_AMT5	4,799	15,278	0	253	1,500	4,032	426,529
PAY_AMT6	5,216	17,777	0	118	1,500	4,000	528,666

TABLE 4.4: Summary of the categorical features, gender, marriage and education in the Taiwan data set.

(A) Gender

Value	Meaning	n	%
1	Male	11,888	39.63%
2	Female	18,112	60.37%

(B) Marriage

Value	Meaning	n	%
0	Other	54	0.18%
1	Married	13,659	45.53%
2	Single	15,964	53.21%
3	Divorced	323	1.08%

(C) Education

Value	Meaning	n	%
0	Other	14	0.05%
1	Graduate school	10,585	35.28%
2	University	14,030	46.77%
3	High school	4,917	16.39%
4	Other	123	0.41%
5	Other	280	0.93%
6	Other	51	0.17%

TABLE 4.5: Summary of categorical features, PAY_1, ..., PAY_6.

		PAY_1		PAY_2		PAY_3		PAY_4		PAY_5		PAY_6	
Value	Meaning	n	%	n	%	n	%	n	%	n	%	n	%
-2	No consumption	2,759	9.20%	3,782	12.61%	4,085	13.62%	4,348	14.49%	4,546	15.15%	4,895	16.32%
-1	Paid in full	5,686	18.95%	6,050	20.17%	5,938	19.79%	6,687	18.96%	5,539	18.46%	5,740	19.13%
0	Revolving credit	14,737	49.12%	15,730	52.43%	15,764	52.55%	16,455	54.85%	16,947	56.49%	16,286	52.29%
1	One month late	3,688	12.29%	28	0.09%	4	0.01%	2	0.01%	0	0.00%	0	0.00%
2	Two months late	2,667	8.89%	3,927	13.09%	3,819	12.73%	3,159	10.53%	2,626	8.75%	2,766	9.22%
3	Three months late	322	1.07%	326	1.09%	240	0.80%	180	0.60%	178	0.59%	184	0.61%
4	Four months late	76	0.25%	99	0.33%	76	0.25%	69	0.23%	84	0.28%	49	0.16%
5	Five months late	26	0.09%	25	0.08%	21	0.07%	35	0.12%	17	0.06%	13	0.04%
6	Six months late	11	0.04%	12	0.04%	23	0.08%	5	0.02%	4	0.01%	19	0.06%
7	Seven months late	9	0.03%	20	0.07%	27	0.09%	85	0.19%	58	0.19%	46	0.15%
8	Eight months late	19	0.06%	1	0.00%	3	0.01%	2	0.01%	1	0.00%	2	0.01%

To get a better feeling of the data set, several features will be more extensively described and some interesting relations between features are researched. First of all the relation between the balance limit and several other features will be analyzed. Figure 4.1 shows how the balance limit differs between the different education categories. The figure clearly shows that a higher education level leads to a higher limit. This effect is expected since higher education usually also means a higher income. Furthermore it is remarkable how compact the values in the "other" category are distributed. This is probably caused by the small number of occurrences, only 1.56% of the feature belongs to this category. A similar plot is shown for the marital status in Figure 4.2. It also shows a clear difference between the different categories.

FIGURE 4.1: Boxplots showing the balance limit for the different education statuses.

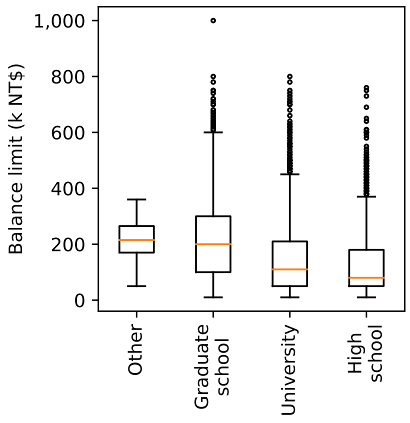
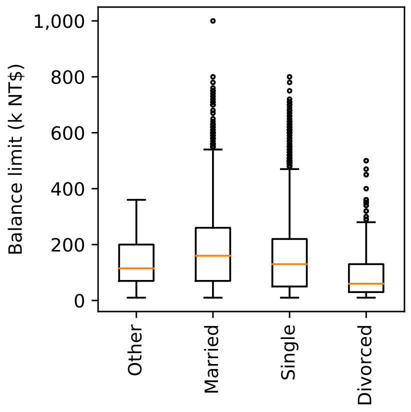


FIGURE 4.2: Boxplots showing the balance limit for the different marital statuses.



More information about both features is given in Table 4.6 and 4.7. Those two tables shows the average age and default rate for the different categories. It is interesting to see that for both features the "other" categories have a very low default rate. This leads to the probable conclusion that it contains high income or net worth individuals. The remaining categories also show some clear differences, clients in the graduate school category have a lower default rate. For the marital feature it is the divorced category which has different default rate, it is higher than the other categories. Furthermore, the age of the "single" category is lower.

TABLE 4.6: Average age and default rate for the education categories.

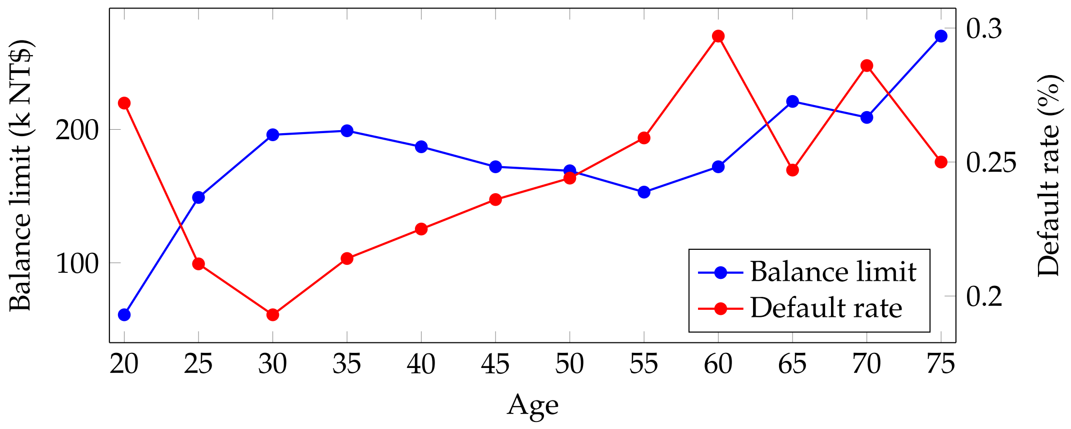
Education	Average age	DR
Other	36	7%
Graduate school	34	19%
University	35	24%
High school	40	25%

TABLE 4.7: Average age and default rate for the marital status categories.

Relationship	Average age	DR
Other	38	9%
Married	40	23%
single	31	21%
Divorced	43	26%

For a better understanding of the data set, the influence of the age of the client will be more thoroughly analyzed. The previous figures and tables already show differences in age between the different categories. To get a grip of the impact of age, all clients have been separated in groups of a similar age. Figure 4.3 shows the result of this analysis. From the figure it is clear that the group with the lowest age have a low balance limit and a high default rate. Up until the group starting at age 30, the limit increases and the default rate decreases. This is expected since income increases with age and thus the risk decreases. However, from 35 until 55 older clients pose a greater risk, this outcome was unexpected. The balance limit start increasing again from 55 onwards. In that section the trend of the default rate gets less clear due to the small amount of data points in those categories, only 3.51% of the clients is age 55 or older.

FIGURE 4.3: Average balance limit and default rate per age group, starting at the label value up to the next label.



4.1.2 Correlations

To further analyze the data set, the correlations between the different features have been calculated and are shown in Table 4.8. From the correlations given in the mentioned table, the correlations between the bill amount features are the highest. They range from 0.8 up to 0.95. Since the values of those features change respectively to the value in the previous period, the correlation with the previous and next period is the highest.

The limit of the balance also shows some strong positive correlations with the other features. This is logical, since a higher balance means that the borrower can take out larger loans. The balance limit also has a positive correlation with age. This is probably caused by people with a higher age on average having a higher income and thus can borrow larger amounts.

The payment amounts show less strong correlation than the bill amounts. This is probably caused by the fact that the payments amount are not relative to the amount of the previous period.

TABLE 4.8: Correlation between the different numerical features in the first data set.

	LIMIT_BAL	AGE	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	PAY_AMT6
LIMIT_BAL	-	0.14	0.29	0.28	0.28	0.29	0.30	0.29	0.20	0.18	0.21	0.20	0.22	0.22
AGE	0.14	-	0.06	0.05	0.05	0.05	0.05	0.05	0.03	0.02	0.03	0.02	0.02	0.02
BILL_AMT1	0.29	0.06	-	0.95	0.89	0.86	0.83	0.80	0.14	0.10	0.16	0.16	0.17	0.18
BILL_AMT2	0.28	0.05	0.95	-	0.93	0.89	0.86	0.83	0.28	0.10	0.15	0.15	0.16	0.17
BILL_AMT3	0.28	0.05	0.89	0.93	-	0.92	0.88	0.85	0.24	0.32	0.13	0.14	0.18	0.18
BILL_AMT4	0.29	0.05	0.86	0.89	0.92	-	0.94	0.90	0.23	0.21	0.30	0.13	0.16	0.18
BILL_AMT5	0.30	0.05	0.83	0.86	0.88	0.94	-	0.95	0.22	0.18	0.25	0.29	0.14	0.16
BILL_AMT6	0.29	0.05	0.80	0.83	0.85	0.90	0.95	-	0.20	0.17	0.23	0.25	0.31	0.12
PAY_AMT1	0.20	0.03	0.14	0.28	0.24	0.23	0.22	0.20	-	0.29	0.25	0.20	0.15	0.19
PAY_AMT2	0.18	0.02	0.10	0.10	0.32	0.21	0.18	0.17	0.29	-	0.24	0.18	0.18	0.16
PAY_AMT3	0.21	0.03	0.16	0.15	0.13	0.30	0.25	0.23	0.25	0.24	-	0.22	0.16	0.16
PAY_AMT4	0.20	0.02	0.16	0.15	0.14	0.13	0.29	0.25	0.20	0.18	0.22	-	0.15	0.16
PAY_AMT5	0.22	0.02	0.17	0.16	0.18	0.16	0.14	0.31	0.15	0.18	0.16	0.15	-	0.15
PAY_AMT6	0.22	0.02	0.18	0.17	0.18	0.18	0.16	0.12	0.19	0.16	0.16	0.16	0.15	-

4.1.3 Vectorization and normalization

In Chapter 3 the processes of vectorization and normalization are described. In the methodology this phase is said to be performed after feature engineering. This is done because new or changed features should also be vectorized and normalized. In this chapter the order of the phases is switched, this is done to determine whether or not the impact of the engineered features has to be analyzed with or without vectorization and normalization.

First of all it is required to determine the performance when the algorithms are trained and validated using the original data set. This is given in Table 4.9. The results achieved with the default settings of the algorithm on the default data set varies greatly between algorithms. It is clear that the four algorithms based on trees perform better.

TABLE 4.9: Performance of the different algorithms when trained and validated on the original data set.

Algorithm	AUC
Logistic Regression	0.65
Neural Network	0.62
Naive Bayes	0.67
k Nearest Neighbors	0.60
Decision Tree	0.73
Random Forest	0.73
ADABoost	0.75
Gradient Boosting	0.77
Support Vector Machine	0.51

To make this data set better usable for the machine learning, some preparations have to be performed. Two important preparation are vectorization and normalization. The first of these has to do with categorical features. Most machine learning models assume a certain order in the value of the features. When a categorical feature is used as input in its original form, the model is likely to have difficulties with distinguishing the gender. To solve this issue the feature is vectorized, as has been mentioned in the previous chapter. An example of the vectorization of the gender feature is given in Table 4.10. Vectorization is applied to all of the categorical features; gender, education, marriage and possible a later engineered feature. To determine the resulting number of columns after vectorization, it is necessary to know all different values the feature can have. As mentioned before the possible values are given in Appendix A.

TABLE 4.10: Vectorization of the gender feature.

Id	Gender
0	1
1	1
2	2
3	1
4	2

Id	Female	Male
0	0	1
1	0	1
2	1	0
3	0	1
4	1	0

Next to the vectorization of categorical features, numerical features also have to be prepared, this is done by normalization. It is the process of manipulating the values of a feature in such a way that the average will be zero and the standard deviation will be 1. The reason to normalize features is to transform them in such a way they all have the same mean and standard deviation. The goal is to minimize different levels of influence between features. A few samples before and after normalization are shown in Table 4.11.

TABLE 4.11: Normalization of the balance limit feature.

Id	Balance limit	Id	Balance limit
80	470,000	80	1.55
81	360,000	81	0.81
82	60,000	82	-1.2
83	400,000	83	1.08
84	50,000	84	-1.26
85	160,000	85	-0.53
86	360,000	86	0.81
87	160,000	87	-0.53
88	130,000	88	-0.73

Now that both vectorization and normalization have been applied to the original data set, it is possible to determine the impact on the performance. The results of training and validating the algorithms with the vectorized and normalized original data set are shown in Table 4.12. The AUC of all algorithms is higher or identical, so it is a good idea to sue normalization and vectorization. Especially the Neural Network and Support Vector Machine show a large increase in performance, the AUC increased respectively from 0.62 to 0.77 and from 0.51 to 0.73. Naive Bayes and k Nearest Neighbors both also show a decent increase in performance. The Decision Tree and algorithms that are an ensemble of Decision Trees show no increase in performance. This was expected, because decision trees do not rely on the scale of a feature.i

TABLE 4.12: Performance of the different algorithms when trained and validated on the original features that are vectorized and normalized.

Algorithm	AUC	Change
Logistic Regression	0.72	+0.07
Neural Network	0.77	+0.15
Naive Bayes	0.73	+0.06
k Nearest Neighbors	0.70	+0.10
Decision Tree	0.73	+0.00
Random Forest	0.73	+0.00
ADABoost	0.75	+0.00
Gradient Boosting	0.77	+0.00
Support Vector Machine	0.73	+0.22

4.1.4 Feature engineering

The next step in preparing the data set, is to see if the model performance can be further increased by using feature engineering. The goal of this process is to change the features in such a way that they are easier to "understand" for the algorithms. The results of the previous section show that vectorizing and normalizing have a positive influence on the performance. Therefore that performance will be used as baseline, the changes made through feature engineering will also be vectorized and normalized where possible.

In Appendix A, it is given that the education feature can take seven different values of which four mean "other". No information is available to distinguish between the four types of "other". To stay consistent with the other categorical features, 4, 5 and 6 will be changed to belong to category 0. To make sure this change does not result in information loss, all algorithms are tested with their default settings on the data set before and after the change. The results are given in Table 4.13. It is clear that grouping the "other" types of education has no large impact on the model performance. Since it makes the education feature more logical and it has no large impact, the "other" categories are indeed grouped together.

TABLE 4.13: Performance of the different algorithms when the different "other" education values are grouped.

Algorithm	AUC	Change
Logistic Regression	0.72	-
Neural Network	0.77	-
Naive Bayes	0.73	-
k Nearest Neighbors	0.70	-
Decision Tree	0.73	-
Random Forest	0.73	-
ADABoost	0.75	-
Gradient Boosting	0.77	-
Support Vector Machine	0.73	-

The group of PAY_n features might be interesting to adjust, these categorical features contain a lot of information. It shows whether the credit is revolving or instalment, paid on time or paid late and if there has been no consumption. Since all this information is mapped onto values ranging from -2 to 8, it might be difficult for some machine learning algorithms to extract the information. To make the data set more logical, three separate characteristics might be separated into different features. The current meaning of the different values the feature can take are given in Table 4.4. The first new feature is for the lateness of the payment, each of the values from 1 to 8 indicate that the payment is respectively one to eight months late. This can easily be mapped to a new feature, as shown in Table 4.14. The two other new features are also shown in the table, they are for consumption and for the credit type.

TABLE 4.14: Newly created features based on the PAY_n features.

(A) Lateness

Value	Meaning
0	On time
1	One month late
2	Two months late
3	Three months late
4	Four months late
5	Five months late
6	Six months late
7	Seven months late
8	Eight months late

(B) Consumption

Value	Meaning
0	No consumption
1	Consumption

(C) Revolving

Value	Meaning
0	Instalment
1	Revolving

Just as with education the effect of this operation will be analyzed by running the different algorithms in their default settings on the old and new data set. The results of the analysis are shown in Table 4.15. Splitting the feature has a large positive effect on Logistic Regression and a small positive effect on Naive Bayes and k Nearest Neighbors. The change has no negative effect on any of the algorithms. Therefore it is a good choice to apply this modification to the data set when using Logistic Regression, Naive Bayes, k Nearest Neighbors or Support Vector Machine.

TABLE 4.15: Performance of the different algorithms when PAY_n features are separated as shown in 4.14.

Algorithm	AUC	Change
Logistic Regression	0.76	+0.04
Neural Network	0.77	-
Naive Bayes	0.74	+0.01
k Nearest Neighbors	0.71	+0.01
Decision Tree	0.73	-
Random Forest	0.73	-
ADABoost	0.75	-
Gradient Boosting	0.77	-
Support Vector Machine	0.73	-

Another possibility for feature engineering is to use the ratio of two features. In this data set the candidates for creating ratios are the monetary values. Deciding which, is primarily based on logical reasoning and experimenting. For example the amount on the bill statement might have an influence on the likelihood of defaulting. But the bill statement (BILL_AMNTn) relative to the credit limit might have a bigger influence. This is also valid for the payment amount (PAY_AMNTn) feature relative to the limit. Some examples of calculating this ratio are shown in Table 4.16. It is logically that the table shows that samples closer to the balance limit have a higher ratio. In the previous section it was found that vectorization and normalization have a positive effect on model performance, therefore the calculated ratio's are also normalized.

TABLE 4.16: Calculating the ratio of two features.

Id	Balance limit	Bill amount
30	230,000	16,646
31	50,000	30,518
32	100,000	93,036
33	500,000	10,929
34	500,000	13,709
35	160,000	30,265

Id	Ratio
30	0.072
31	0.610
32	0.930
33	0.022
34	0.027
35	0.189

When ratio's are calculated a decision has to be made on whether or not to keep the original feature. To support that decision the performance will be determined for both situations, the results are shown in Table 4.17. The first impression of the results is that adding ratio's is not very influential with this data set. Only adding a ratio involving the bill statement has a positive performance on one of the algorithms, k Nearest Neighbors. All other ratio's and algorithms either show no change in performance of a decrease in performance. Therefore, the choice has been made to only add the ratio in the combination where it is beneficial.

TABLE 4.17: Performance when ratio's are calculated. Performance is with respect to the benchmark performance. Both ratio's have LIMIT_BAL as denominator.

	BILL_AMNT		PAY_AMNT		Both	
Original	Keep	Delete	Keep	Delete	Keep	Delete
Logistic Regression	-	-	-	-	-	-
Neural network	-	-	-	-	-	-
Naïve bayes	-0.01	-0.01	-	-	-0.01	-0.01
k Nearest neighbors	+0.01	+0.01	-	-	-	-
Decision tree	-	-	-	-0.01	-	-
Random forest	-	-	-	-	-	-
ADABoost	-	-	-	-	-	-
Gradient boosting	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
Support Vector Machine	-	-	-	-	-	-

4.1.5 Resampling

As mentioned in Chapter 2, data sets often have an imbalance in the number of samples per class. This problem is often present in data sets with default events, this is logically caused by the rareness of a default. In this data set the number of defaults is 6,636 from a total of 30,000 clients. This clearly is an imbalance in the data set but not so large that resampling is guaranteed to improve performance.

In Chapter 2, several methods have been discussed to balance a data set. It is necessary to decide which method is most suitable to be used in this data set. To make this decision, each of the resampling methods will be used on the training subset. The resulting training sets will be used to train and validate the different algorithms. Testing the machine learning algorithms in their default settings will give an indication of the impact of the different resampling algorithms. In total nine different methods of resampling will be tested:

- No resampling
- Random undersampling
- Random oversampling
- SMOTE
- SMOTE borderline1
- SMOTE borderline2
- ADASYN 2 neigbors
- ADASYN 5 neigbors
- ADASYN 10 neigbors

For this decision, the size of the data set is crucial. For small data sets undersampling usually is a bad choice because it decreases the size of the data set even further. Undersampling this data set will result in a set with 13,272 $_{(2\times 6{,}636)}$ samples. This is sufficient to use for machine learning, so it is possible to use undersampling. As described before, a disadvantage of undersampling is that some data is lost due to ignoring a large part of the majority samples. When using random oversampling that problem does not occur. Instead of removing samples from the majority class it adds the same minority samples multiple times until balance is restored in the data set. The resulting data set will consist of 46,728 $_{(2\times 23{,}364)}$ samples.

The results of applying the different resample methods are shown in Table 4.18. The results are given relative to the performance of the normalized and vectorized data set, the exact AUC values are given in Appendix B. From the results it is clear that in most cases resampling has a negative impact on the performance. Especially the more complex resample methods like SMOTE and ADASYN almost always have a negative impact. A few exception can be seen in the results. The first exception is the performance of Logistic Regression, all SMOTE and ADASYN variants improve its performance slightly. The second exception is the Decision Tree algorithm which performs better when random under or oversampling is used. During further experimenting all algorithms except the two with increased performance should be run on the data set without resampling. For the other two the corresponding algorithms should be used.

TABLE 4.18: The average AUC of 25 folds for different algorithms and resample methods.

Algorithm	None	Random undersampling	Random oversampling	Smote regular	Smote borderline1	Smote borderline2	ADASYN_2	ADASYN_5	ADASYN_10
Logistic Regression	-	-	-	+0.01	+0.01	+0.01	+0.01	+0.01	+0.01
Neural Network	-	-	-0.01	-0.02	-0.04	-0.03	-0.03	-0.03	-0.04
Naïve Bayes	-	-	-	-	-	-0.01	-	-	-
k Nearest Neighbors	-	-	-0.03	-0.02	-0.03	-0.03	-0.03	-0.03	-0.03
Decision Tree	-	+0.01	+0.01	-	-	-	-	-	-
Random Forest	-	-	-	-	-	-0.01	-0.01	-0.01	-0.01
ADABoost	-	-	-	-	-0.01	-0.02	-0.01	-0.01	-0.01
Gradient Boosting	-	-0.01	-	-0.01	-0.01	-0.01	-0.01	-0.01	-0.01
Support Vector Machine	-	+0.03	+0.03	+0.03	+0.03	+0.02	+0.03	+0.03	+0.03

4.1.6 Conclusions

Now all steps of the data preparation have been researched it is possible to determine the best approach for each individual algorithm. The first part of this section was about increasing the performance of the algorithms by vectorization and normalization. For all non tree based algorithms it resulted in a major performance increase, Neural Network performance increased the most, from 0.62 to 0.77. The tree based algorithms showed no change in performance at all.

The next step in the preparation was feature engineering, the process of changing the features to better represent the information it contains. The first change had to do with the education feature, it had four different values with meaning "other". It was not possible to prove an explanation for that difference, therefore they should be merged if it does not result in a major performance decrease. After running all algorithms on the changed data set no difference in performance occurred and thus the "other" values are merged together.

The next proposed change concerned the group of payment status features. These features each contain several different characteristics, they contain information about the lateness of payment, type of credit and whether or not credit has been used. To represent this information in a better way it has been proposed to split each of the payment status features in three separate features. This change has a positive influence on Logistic Regression, Naive Bayes and k Nearest Neighbors. The AUC of Logistic Regression increased by 0.04, the other two increased by 0.01. The other algorithms showed no change in performance. Since this change only has positive impacts it will indeed be applied to the data set.

The third proposed change is to calculate ratio's of several monetary features. Two different ratio's have been calculated, the first ratio is the bill statements over the balance limit and the second ratio is the payment amount over the balance limit. The results show that adding the ratio's has no large impact on performance. The only increase in performance is seen for k Nearest Neighbors when the ratio of the bill statements over the balance limit is added to the data set. In all other situation it has no influence or a negative one. Ratio's will only be added for the algorithm that showed an increase in performance.

Finally the impact of the different resample methods has been researched. The most important conclusion is that the more complex methods, SMOTE and ADASYN have almost no positive influence. Logistic Regression is the exception, it is the only algorithm that performs better when SMOTE or ADASYN is applied. The Decision Tree algorithm performs better when random under- or oversampling is applied.

4.2 Peer to peer lending - Prosper

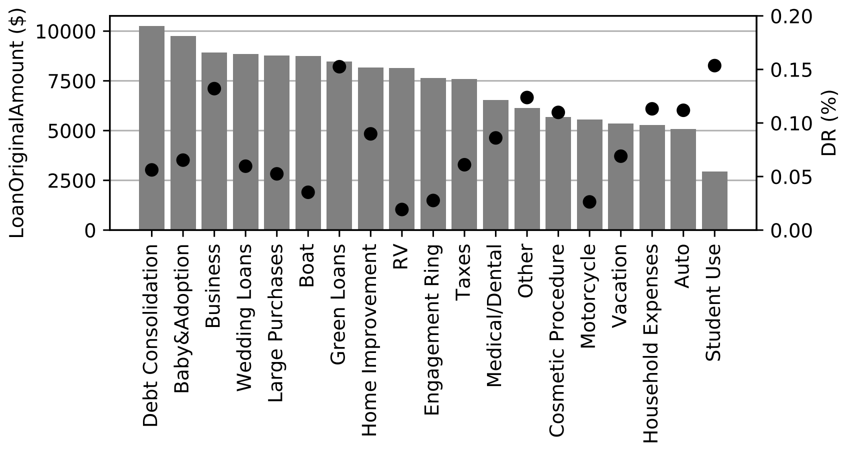
The second data set that will be used in this experiment contains peer2peer loans (Prosper, 2017). These type of loans are not given out by a bank but are arranged between private parties. Prosper is a platform that facilitates this process by bringing lenders and borrowers together. The platform also provides a wide range of measures to support the lend decision. Prosper publishes their data to provide the possibility for investors to create models they can use to decide which loan to accept. For this data set the data preparation will be discussed more concise because it is the second time this process is applied. The vectorization and normalization section will be left out for this data set. In the previous section and literature it was shown that is has a positive effect and thus will be applied to this data set.

4.2.1 Data description

The data set contains 81 features and thus is high dimensional. The table with the features, their descriptions and statistical summaries are due to their size presented in Appendix A Table A.1 up to Table A.6. Of the 81 features, 45 are used in the final data set, the majority of the removed features are taken out because of one of the following two reasons. Prosper implemented a new system to score loans within the timeframe of the used data set. This change came with several new features, which where blank for loans started before the new method. Since 84,848 out of the 113,932 loans use the new method, only those are used for the final data set. The table in Appendix A describing the features can be used to see which features where removed as a result of this split. The second major reason to remove features is data leakage.

Just as with the previous data set, several of the features will be analyzed to get a better understanding of the data set. The loans in the data set are split up into 21 categories, as can be seen at the ListingCategory feature. For each of the categories the average loan amount and the default rate has been determined. The details are graphed in Figure 4.4. The debt consolidation category has the highest average loan with a value above 10,000. Student use is the lowest category with just above 2,500. Even though student use has the lowest average value, it is the category with the highest default rate. Other high risk categories are green loans (15.3%), business (13.2%) and other (12.4%). The most safe categories are RV (1.9%), motorcycle (2.6%) and engagement ring (2.7%).

FIGURE 4.4: Graph showing the average loan amount per category on the left axis (gray) and the default rate on the right axis.



The next part that will be analyzed is the impact of the monthly income of the borrower. In Figure 4.5 the relation between the stated monthly income and the loan amount is given, dark areas indicate a high concentration of loans. To make the relation between the two features clear, a linear trend line has been added. It clearly shows that borrowers with a higher income will on average borrow a higher amount. Figure 4.6 shows the relation between the debt to income and the stated monthly income. This graph shows that higher income borrowers will on average take out a loan that is smaller in comparison with their income, even though they loan a higher amount on average.

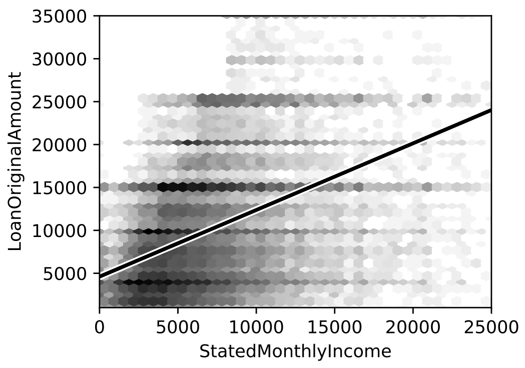


FIGURE 4.5: Heatmap showing the relation between the stated income and amount loaned.

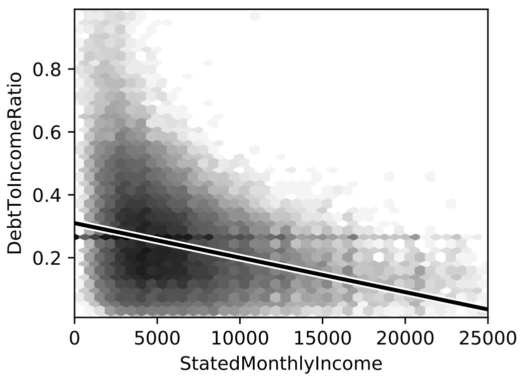


FIGURE 4.6: Heatmap showing the relation between the stated income and the debt to income ratio.

The last graph about the monthly income is Figure 4.7, it shows the default rate for different levels of income. It shows a relation between the features that is expected, borrowers with a higher income are less likely to default. However, this effect is only clearly visible up to a monthly income of $7,500. After that, a higher income does not lead to a lower default rate. The last feature related to income that is analyzed is IncomeVerifiable, this binary feature shows whether or not the stated income can be verified. The expectation is that this feature is important, because if the income can not be verified it might not have been filled in truthfully. Figure 4.8 shows that the default rate for non verifiable income loans is double that of loans with a verifiable income. From the figure it is also clear that the loans without a verified income are for lower amounts.

FIGURE 4.7: Default rate for different levels of income. Bins start at the axis value, up to the next axis value.

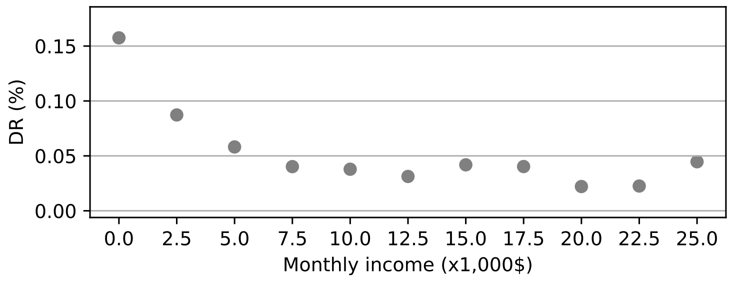
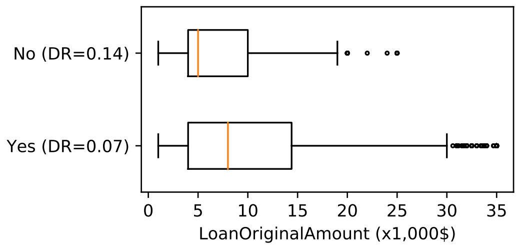


FIGURE 4.8: Comparison of the loans based on if the stated income is verifiable.



The next feature that will be analyzed is the IsBorrowerHomeOwner feature. This feature is either true of false depending on whether the borrower owns a house. Figure 4.9 shows two boxplots that describe the amount loaned by both groups. The default rate is also given for both groups. The figure makes it clear that home owners borrow on average a higher amount. It also shows that home owners have a slightly lower default rate. In Figure 4.10 the income of both groups is described, it clearly shows that home owner have a higher average income. Since it is known form the previous section that borrowers with a higher income loan more on average, it might also explain the differences in the loan amount between the home owners.

The last feature is the credit score obtained from a scoring agency. In the data set the scores are divided into bins with a width of 20. The lowest range is 600-620 and

FIGURE 4.9: Comparison of the loan amount based on if the borrower owns a home.

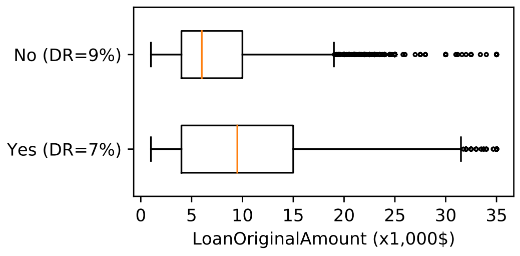
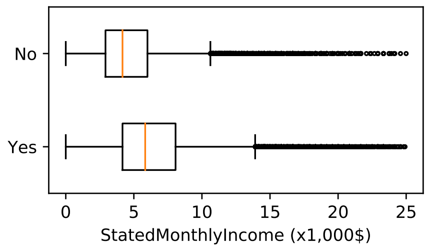
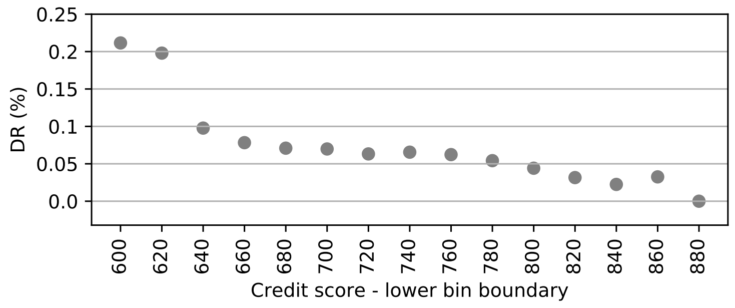


FIGURE 4.10: Comparison of the stated income based on if the borrower owns a home.



the highest range is 860-880. Figure 4.11 shows per bin what the default rate is for the loans in that bin. From the table it becomes clear that the credit score indeed has a large impact on the default rate.

FIGURE 4.11: Graph showing the default rate per credit score bin. Bins start at the axis value, up to the next value.



4.2.2 Correlations

Finally, the correlations between the different features will be analyzed. Due to the high number of features not all correlations will be given. Most of the higher correlation can be easily explained, for example the correlation between the 'TotalProsperLoans' and the 'OnTimeProsperPayments' or between the monthly payments and principal of the loan. Therefore, not all high correlation will be given, but only those with an interesting characteristic.

First the correlation between the homeownership and the other features will be analyzed, one expects that being a homeowner has a positive effect on the borrower. This is verified by looking at the correlations, the 'IsBorrowerHomeowner' has a correlation of 0.28 with the credit score and a -0.06 correlation with the number of delinquencies in the last 7 years. It is also interesting to see that it has a correlation of 0.29 with the number of trades, this is probably caused by homeowner being better in paying off their loans and thus can easier take out multiple loans.

The data set contains a feature that specifies how many friends have invested in a loan. One might expect that if more friends invest, someone is more likely to pay on time. However, the correlations show no such relation. The correlation between the number of friends that invested and the current delinquencies to be 0.00.

The correlations can also be used to verify the correctness of the creditscore. If the credit score is a good indicator of the creditworthiness of the borrower, it should for example have a negative correlation with the number of delinquencies and a positive correlation with the loan amount. Luckily, these expectations are confirmed by the correlations. The correlation between the credit score and number of current delinquencies is -0.16 and the correlation with the amount loaned is 0.28.

4.2.3 Missing and strange values

Where the Taiwan data set did not have a single missing value, this data set has several. In this section handling the missing data will be described. Luckily the majority of the missing values are from the period where the old scoring system was used. After the date on which the new system went live, the missing values are not present as much. The total number of missing values per feature is given in Table 4.19. After the changes discussed below, 86,964 samples remain.

TABLE 4.19: The number of missing values per incomplete feature.

Feature	Missing values
Occupation	1,333
EmploymentStatusDuration	19
DebtToIncomeRatio	7,307
TotalProsperLoans	65,128
TotalProsperPaymentsBilled	65,128
OnTimeProsperPayments	65,128
ProsperPaymentsLessThanOneMonthLate	65,128
ProsperPaymentsOneMonthPlusLate	65,128
ProsperPrincipalBorrowed	65,128
ProsperPrincipalOutstanding	65,128
ScorexChangeAtTimeOfListing	68,285

The first non usable values are found in the LoanStatus feature. This feature can take several different values, including 'Canceled'. Since canceled loans cannot go into default they cause noise in the data set. Only 25 loans are canceled, therefore they are all removed from the data set.

The 'Occupation' feature has a large number of blank values, in total 1,333. This value is after the samples from the old scoring system where removed. The missing values have been changed to be a part of the "Other" category since that is the most broad category.

The next feature that needs correction is 'EmploymentStatusDuration', it contains 7,625 blank values. The majority of these samples are from the period the old scoring system are used and thus are already removed, only 19 remain. As the name states, the feature gives the duration the current employment status has been the same. Since no information is available on how long the borrowers is in the same employment status, a neutral value will be used. Just as with the previous feature this neutral value is the mean of the non-blank values.

The feature with the most missing values is 'DebtToIncome', with 7,307 blanks. First it was attempted to reconstruct the feature from the given income and loan. However it did not result in the exact values shown in the 'DebtToIncome' feature. Therefore it has been decided that the blanks will be filled with a neutral value. The value that will be used is calculated by taking the mean of the samples that do have a value.

The 'TotalProsperLoans' feature is the first of the features with a very large number of missing values, 65,128. Luckily this can easily be explained. This feature gives the total number of Prosper loans the borrower previously had. If the borrower had no previous loans at Prosper, the feature is blank. This can easily be solved by changing the blank values to zero. This same situation is present at the features with the same number of missing values, this can be seen in Table 4.19. For those features the missing values will also be replaced with zeroes.

The 'ScorexChangeAtTimeOfListing' feature shows the difference in credit score between the listing of a previous loan and the listing of the current loan. If no previous loan has been taken out through Prosper, the value is blank. Clearly this situation is similar to the above features. For this feature the missing values will also be changed to zero.

4.2.4 Feature engineering

Just as with the previous data set, some features will be changed. This can either be because is necessary or it might improve predictive performance. In the previous data set quite a lot of feature engineering was applied. For this data set it will be limited. The data set is already very extensive and high dimensional. This limits the potential benefit of feature engineering and computation time should be taken into account.

The first feature to add is the 'default' feature. Where the previous data set already had such a feature, it has to be created in this data set, it will be based on the 'LoanStatus' feature. That feature can take several different values: chargedoff, completed, current, defaulted, finalPaymentInProgress and pastDue. To create the default feature, the samples that have the status chargedoff or defaulted are set to 1 and the other status are set to 0. The resulting data set has 6,350 defaults in a total of 84,964 loans.

Another small step that has to be carried out has to do with the boolean features. In the original data set these are given in text as 'true' or 'false'. However the implementation of the algorithm expects boolean features to be integers. Thus, the boolean features have to be mapped to ones and zeroes.

4.2.5 Resampling

The last step of the data preparation process is to determine the most suitable resampling method for each of the algorithms. Just as with the previous data set this is done by applying the resample methods and determining the performance. With the previous data set it was found that the more complex resample methods, SMOTE and ADASYN, do not perform well. Those more complex resample algorithms take more time to run, especially for the Prosper data set since it is a few times larger. Therefore, the choice has been made to apply one kind of SMOTE and one kind of ADASYN. The results of using the different resample methods is shown in Table 4.20.

TABLE 4.20: The average AUC of 25 folds for different algorithms and resample methods.

Algorithm	None	Random undersampling	Random oversampling	Smote regular	ADASYN_5
Logistic Regression	-	-	-	-	-
Neural Network	-	+0.01	-0.01	-0.02	-0.02
Naïve Bayes	-	-	-	-0.08	-0.09
k Nearest Neighbors	-	+0.09	-	+0.07	+0.07
Decision Tree	-	-0.01	-	-0.02	-0.03
Random Forest	-	+0.06	+0.03	+0.03	+0.02
ADABoost	-	+0.02	+0.01	+0.02	+0.02
Gradient Boosting	-	-	-	-0.01	-0.02

As mentioned before, one of the risks of working with an imbalanced data set is a high accuracy but low predictive performance. This is caused by the model disregarding the minority class and assigning every sample to the majority class. This leads to a seemingly high accuracy but the model is a bad predictor. To prevent this from happening the confusion matrices of the experiments are taken into account. In three cases the confussion matrix looks like the one shown in Table 4.21. Since none of the samples is classified as positive, it is clear that ADAboost, Decision Tree and Gradient Boosting require at least a form of resampling.

TABLE 4.21: Confusion matrix of ADABoost without any form of resampling.

	Predicted positive	Predicted negative
Positive	0	4,743
Negative	0	59,027

4.2.6 Conclusions

Now that a lot more is known about the data set and how it should be prepared, several conclusions about it can be drawn.

Where the first data set had no missing values, this data set did have those. Most of the missing values occurred in a part of the data set that was generated in a period with an older and different system. Since the the data set originated for 74.4% from the period with the new system, the samples generated by the old system can be removed without losing to much data. Also some smaller problem where found in the data, which where easily solved.

Since the second data set did not have a default feature, it had to be created. Adding this new feature has been done based on the 'LoanStatus' feature. Samples that have the status chargedoff or defaulted are regarded as defaulted.

The last subject which will be discussed in this section, is the different resample methods that are applied to the data set. In the first data set 22,1% of the samples results in a default, with this data set only 7,5%. This makes the usage of a resample technique more important. This was verified by the fact that several of the algorithms classified all samples as negative, if no resampling was performed. For none of the algorithms, the more complicated resample algorithms, SMOTE and ADASYN, improved the performance. This was also observed with the first data set. For all of the algorithms, random undersampling either improved the performance with the greatest amount or had no influence. Since that method reduced computation times, it can be used in combination with all of the algorithms.

Chapter 5

Model training

In this chapter, the parameters of the different machine learning algorithms will be determined. This is done in two sections, first the parameters will be determined for use with the first data set, then for use with the second data set. Note that all of the model training happens on the training set only. The testing set will only be used to determine the final performance.

5.1 Credit data - Taiwan

The parameters of all the algorithms will first determined in combination with the Taiwanese credit data set. In the previous chapter the data preparation has been examined, the results will be used here. For all of the algorithms the data set will be normalized and vectorized. Several different forms of feature engineering have been applied with different results per algorithms. The "other" education values will be grouped together for all algorithms, since it had no negative effect but makes the feature more logical. Separating the payment features and calculating ratio's will only be done if it has a positive effect. Finally, the used resample method is also based on the results found in the previous chapter. The exact data preparation is discussed with each of the algorithms.

5.1.1 Logistic regression

The first algorithm of which the influence of the model parameters are explored is Logistic Regression. In the previous chapter it is determined that the model performs best, on this data set, when the payment features are separated and the data set is resampled using SMOTE. The following parameters will be examined:

- **Regularization:** Two types of regularization are available, l1 and l2. The first of these tries to create a sparse classifier.
- **Solver:** Two solvers are taken into account, saga and liblinear. Both solvers are explained in Chapter 2.
- **C:** Regularization term, smaller values indicate more generalization.

First the regularization and the solver are examined together. The performance is determined for each of the four possible combinations, the result is shown in Table 5.1. The different settings have no influence on the performance, the AUC is steady at 0.76. Since sag takes the least amount of time, it will be used for the next experiments.

The last parameter that has to be analyzed is C. This variable is used to specify how strong the generalization is, smaller values indicate a stronger regularization.

TABLE 5.1: Performance and computation time of different solvers and regularizations.

	AUC	Computation time (MM:SS)
saga - l1	0.76	02:25
saga - l2	0.76	01:45
liblinear - l1	0.76	00:52
liblinear - l2	0.76	00:18

To determine a good value for C, the performance will be determined for values ranging from 0.1 up to 10.0, the default value is 1.0. These different values have been used in combination with liblinear and both a l1 and l2 regularization, there is no difference in performance. Thus the choice has been made to keep the value at 1.0 since it is the default value. A summary of the selected values for each of the parameters is given in Table 5.2.

TABLE 5.2: Settings selected for the Logistic Regression algorithm

Parameter	Selected value
Regularization	l2
Solver	liblinear
C	1.0

5.1.2 Artificial Neural Network

The next algorithm which will be analyzed is the Artificial Neural Network. In the previous chapter the different resample methods where tested on this data set in combination with Neural Networks. It was found that no resampling or random undersampling both show the best results. Since random undersampling decreases computation times, without impacting performance, it will be used here. It was also found that adding ratio's or separating the payment status features has no impact on the performance and it will therefore not be applied for this algorithm. The following parameters have to be analyzed for this algorithm.

- **Solver:** Two solvers are taken into account, sgd and adam. Both solvers are explained in Chapter 2.
- **Hidden layers:** The number and size of the layers between the input and output layers.
- **Activation function:** The shape of the function used to transform the input signals of the neurons.
- **Learning rate:** Size of the step with which weights are changed. It can be a constant or adaptive rate.
- **Regularization term:** A term that can be used to penalize complex models. This can be used to counter overfitting.
- **Tolerance:** If the decrease in loss during training is smaller than this threshold for two consecutive epochs, training is stopped.
- **Max iterations:** The maximum number of iterations to reach convergence.

For the Neural Network, determining the parameters will be split in two phases. First the parameters will be determined for the sgd solver, followed by the adam solver. For both solvers, first the shape of the hidden layers will be determined followed by the activation functions and finally the learning rate. The tolerance and maximum number of iterations will be determined simultaneously since it is required when training stops too early or too late.

Sgd solver

As mentioned above, the parameters for the Neural Network will first be determined in combination with the sgd solver. The first parameter which will be analyzed is the size and number of the hidden layers. To determine the shape of the hidden layers it is important to know the number of input nodes, for this data set that is 30. The number of nodes in each of the hidden layers is almost always chosen to be equal to or less than the number of input nodes. When multiple hidden layers are added to the model, the size usually decreases with each consecutive layer. Since the number of hidden layers has a large impact on the computation time of the training process, it will be added to the results. Several settings for the hidden layers are taken into account. These range from no hidden layers to three hidden layers. The impact on the performance of each of the settings is shown in Table 5.3 in combination with the different activation functions. The conclusion drawn form these results is that it is necessary to add a hidden layer. Regardless of the used activation function, the auc is a lot lower without hidden layer. This is caused by the fact that a Neural Network without hidden layers can only make linear separations. The results also shows that adding more than a single hidden layer does not lead to a performance increase. In the case of three hidden layers in combination with the logistic activation function it results in a low AUC of only 0.59. This is probably caused by overfitting. Furthermore, using a ReLu or tanh activation function leads to a slightly higher AUC in comparison with a logistic activation function. However, the difference is small that all activation functions will be taken into account in the next steps. Since the number of hidden layers has no influence, if at least a single hidden layer is added, a single hidden layer will be used based on the computation time. The size of the hidden layer is chosen to be 20.

TABLE 5.3: Neural Network performance for several different forms of hidden layers with different activation functions. All networks are trained with the sgd solver.

Nodes per layer			Logistic		ReLu		Tanh	
First	Second	Third	AUC	Time	AUC	Time	AUC	Time
30	20	10	0.59	04:00	0.77	05:39	0.77	06:05
30	20	-	0.76	14:38	0.77	05:46	0.77	06:21
30	10	-	0.76	18:13	0.77	04:57	0.77	05:52
20	10	-	0.76	15:16	0.77	03:54	0.77	05:29
30	-	-	0.76	08:12	0.77	03:35	0.77	05:51
20	-	-	0.76	09:01	0.77	02:55	0.77	04:54
10	-	-	0.76	08:34	0.77	02:36	0.77	04:21
-	-	-	0.72	00:15	0.72	00:18	0.72	00:17

The next parameter of the Neural Network that will be analyzed is the Learning rate. This rate determines the step size with which the loss is attempted to decrease. A too high learning rate can lead to overshooting which caused the training process to stop too early. The learning rate can be constant or adaptive. When the latter is used the learning rate is decreased each time the stopping criteria is reached instead of stopping the training process. This leads to the training process making big steps in the beginning and increasingly small steps later on. When the learning rate drops below a certain minimum value, training will stop. In this experiment an adaptive learning rate will begin at 0.1 and end at 0.0001. First the constant learning rate will be analyzed by selecting different values for it. Then the adaptive learning rate will be examined, this is done with a single high value for the learning rate since smaller values will be automatically passed during training.

The results of using a constant learning rate is shown in Table 5.4. Clearly the effect of the learning rate on the performance is very small. The small differences are probably caused by random variation. However, the required computation time is heavily influenced by the learning rate. It ranges from several seconds up to almost an hour. The results of using an adaptive learning rate are shown in Table 5.5. The performance is similar to that when using a constant learning factor. An advantage of using an adaptive learning rate is the short computation time.

TABLE 5.4: Neural Network performance for several different constant learning rates and several activation functions.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
0.1	0.77	00:19	0.76	00:12	0.76	00:19
0.01	0.77	01:15	0.77	00:32	0.77	00:45
0.001	0.76	08:10	0.77	02:57	0.77	04:35
0.0001	0.76	55:31	0.77	21:15	0.77	34:46

TABLE 5.5: Neural Network performance with an adaptive learning rate and several activation functions.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
0.1 -> 0.0001	0.77	00:49	0.77	00:50	0.76	01:02

The next parameter is the generalization term, or alpha as it is called in the used implementation. Higher values of this term will lead to a more general model and thus reduce overfitting. To determine the influence of the generalization term it has been varied from 0 up to 10. Values up to 1 had no influence on the performance whatsoever. A value of 10 leads to a decrease in performance. This is probably caused by the model being to general. Since lower values have no impact on the performance, the default value of 0.0001 will be selected. Since a higher generalization term leads to a more general model, it was expected that the deviation of performance between folds would decrease with higher generalization terms. However, no such effect is found in the data.

The final two parameters are the tolerance and the maximum number of iterations. Both of these parameters are used to determine when training should stop. These are most important when computation times are long since they can be used to determine the stopping moment. With this data set, computation times are limited to several minutes in most cases. Therefore, it has been chosen to keep the stopping criteria very loose. The threshold has been set to zero, so that training will continue until the loss decreases or the maximum number of iterations has been reached. The maximum iterations have been set to a value of 20,000 so that is will not easily be reached.

Adam solver

Just as with the sgd solver, the first parameter to be analyzed is the shape of the hidden layers. In Table 5.6 the same shapes are taken into account as with the sgd solver. From the table it becomes clear that it again has no added value to add more than a single layer. More layers do not increase performance yet lead to a longer computation time. Therefore, the choice has been made to add a single layer consisting of 20 nodes. A small difference in performance between the different activation functions is seen. Using a logistic function seems to lead to a higher performance, however the difference is quite small.

TABLE 5.6: Neural Network performance for several different forms of hidden layers with different activation functions. All networks are trained with the adam solver.

Nodes per layer			Logistic		ReLu		Tanh	
First	Second	Third	AUC	Time	AUC	Time	AUC	Time
30	20	10	0.77	01:42	0.76	01:19	0.76	01:33
30	20	-	0.77	01:29	0.76	01:09	0.76	01:34
30	10	-	0.77	01:25	0.77	01:02	0.76	01:18
20	10	-	0.78	01:24	0.77	00:53	0.77	01:03
30	-	-	0.78	01:34	0.77	00:40	0.77	01:14
20	-	-	0.78	01:32	0.77	00:35	0.77	00:55
10	-	-	0.78	01:28	0.77	00:33	0.77	00:49
-	-	-	0.72	00:08	0.72	00:10	0.72	00:10

The next parameters that will be analyzed is the learning rate. The same learning rates are taken into account as with the sgd solver. However, adam does not support adaptive learning rates. As expected the learning rate has a large influence on the computation time. With the sgd solver the relation between performance and learning rate was not very clear. With a logistic activation function a smaller learning rate even led to a decrease in performance. With the adam solver the relation between the learning rate and performance is more clear. A smaller learning rate increases performance at the cost of a longer computation time. From Table 5.7 it can be concluded that a logistic activation function in combination with a learning rate of 0.001 is a good choice. It has a high performance without increasing the computation time to much.

TABLE 5.7: Neural Network performance for several different constant learning rates and several activation functions.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
0.1	0.76	00:08	0.76	00:04	0.76	00:07
0.01	0.77	00:25	0.77	00:10	0.76	00:19
0.001	0.78	01:32	0.77	00:34	0.77	04:56
0.0001	0.78	09:51	0.77	03:38	0.77	06:33

The regularization term shows similar behavior as when used in combination with the sgd solver. It has no influence on the performance until very high values are used. Then the performance is negatively impacted by regularization.

The last two parameters are the maximum number of iterations and the tolerance. The approach to determine their values is identical as used with the sgd solver. Since computation times are still feasible, both parameters will be set very loose so that performance is increased as long as possible. The parameters are set to the same values as for the sgd solver, with maximum iterations being 20,000 and the tolerance set to zero.

Comparison

Now that the performance of both solvers has been analyzed, they can be compared to make a decision on which one to use. The most important measure is logically the performance. The difference between the algorithms is not large, nevertheless, adam has a slightly higher performance. Aside from the performance, adam also has a computation time that is shorter. The conclusion is that adam will be used a solver for this algorithm. The following settings have been selected for the Artificial Neural Network.

TABLE 5.8: Settings for the Artificial Neural Network algorithm

Parameter	Selected value
Solver	Adam
Hidden layers	Single layer with 20 nodes
Activation function	Logistic
Learning rate	0.001
Regularization term	0.0001
Tolerance	0
Max iterations	20,000

5.1.3 k Nearest Neighbors

k Nearest Neighbors is the next algorithm for which the parameters will be determined. This algorithm works by making a classification based on the classes of the closest samples, the nearest neighbors. In the previous chapter is has been determined that this algorithm should be used in combination with no resampling or random undersampling. Since random undersampling decreases computation times it will be used for this algorithm. It is also shown that splitting the payment status features and calculating the bill statement as ratio of the balance limit increases the performance. This algorithm has the following parameters that have to be determined.

– **Algorithm:** Three different algorithms are compared, $k - d$ Tree, Ball Tree and Brute Force.
– **Leaf size:** Number of samples per leaf, only used when the $k - d$ Tree or Ball Tree algorithm is used
– n **Neighbors:** The number of neighbors on which the classification is based.
– **Weights:** The weights of the individual n neighbors used for classification. These can be based on the distance of the neighbor or can all be the same, uniform.

The algorithm is the first parameter which is analyzed. Three different algorithms can be used to run k Nearest Neighbors, each of these algorithms is used with several values for n neighbors. This is done to examine if any interaction effects are present between the parameters. The results are shown in the table below, clearly the algorithms perform equally accurate. However the calculation times are vastly different, as shown in Table 5.10. The Ball Tree algorithm is several times faster than the other algorithms. This is related to the size of the data set, *kd* Tree and Ball Tree both construct a tree during the training process, Brute force does not do this and simply compares each sample. Constructing a tree saves time in large data sets, but clearly this set is small enough to make Brute Force the better choice. Since this algorithm does not create a tree, it is not necessary to determine the leaf size.

TABLE 5.9: Performance of different k Nearest Neighbors algorithms for several n.

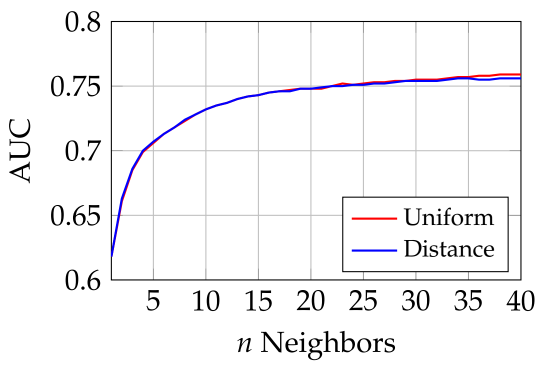
	kd Tree	Ball Tree	Brute Force
1	0.62	0.62	0.62
5	0.71	0.71	0.71
10	0.73	0.73	0.73
20	0.75	0.75	0.75

TABLE 5.10: Calculation time of different k Nearest Neighbors algorithms for several n.

	kd Tree	Ball Tree	Brute force
1	01:35.3	02:38.6	00:17.0
5	02:00.9	02:41.5	00:24.2
10	02:08.4	02:44.9	00:24.3
20	02:16.6	02:46.1	00:24.3

Now that the most suitable algorithm has been selected, the next parameter to be analysed is the number of neighbors taken into account. Since the computation time for this algorithm is very short, it is possible to calculate the performance for a lot of different values. In Figure 5.1 the results are shown for experiment ranging from $n = 1$ up to $n = 35$. The performance clearly increases when a high n is used up until a value of approximately 35, after which the performance increase is unsubstantial. From the analysis of the different algorithms it is known that computation time does not increase substantially for n greater than five. Therefore a value of 35 seems a good choice.

FIGURE 5.1: Impact of using different number of neighbors in combination with uniform or distance based weights.



The last parameter analyzed for this algorithm is the weight, or the importance of the different neighbors. Two different settings can be used, uniform or distance. If the first of these is chosen all n neighbors get an identical weight. When distance is used, the closest of the n neighbors gets the highest weight. With increasing distance, the weights of the neighbors decreases. The performance of both possibilities for different values of n is given in Figure 5.1. The performance of both weights is similar, therefore uniform will be used since it is the less complex option.

TABLE 5.11: Settings selected for the k Nearest Neighbors algorithm

Parameter	Selected value
Algorithm	Brute force
Leaf size	Not applicable
n Neighbors	35
Weights	Uniform

5.1.4 Naive Bayes

The Naive Bayes classifier has no parameters which can be analyzed. The only parameters it he type of distribution, this can be Gaussian, Multinomial and Bernoulli. However, Bernoulli requires binary features and is thus not applicable. Multinomial is only suitable for non-negative features which makes it not usable for this data set. Therefore the conclusion is that only Gaussian can be used.

5.1.5 Decision Tree

The next algorithm that is analyzed is the Decision Tree. The decision tree works by constructing a branching tree, at each branch the data set is split according to a feature value. If a branch does not split any further, it is called a leaf. Each leaf belongs to a certain class. In the previous chapter it has been determined that this algorithm works best with random undersampling or oversampling. Since oversampling increases computation times and undersampling decreases them, the choice has been made to apply random undersampling. It was also determined that performance is not increased by applying additional feature engineering.

The following parameters have to be determined for this algorithm.

– **Criterion:** What measure of impurity to use to base the split on.

– **Max depth:** The maximum depth of the constructed tree.

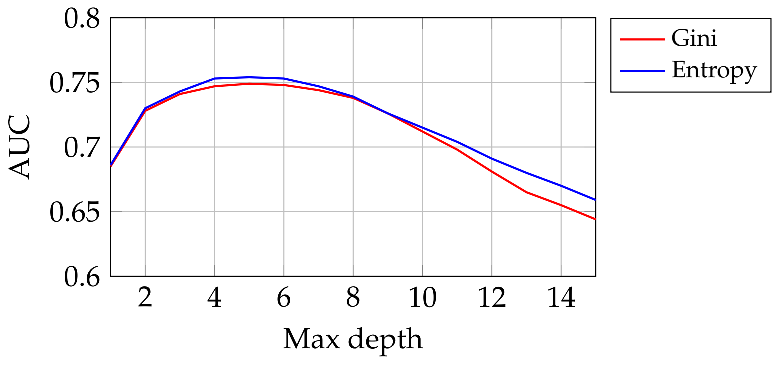
– n **Features:** The number of features taken into account at each split.

The first parameter that is analysed is the split criterion. This criterion is a value to express the impurity in a data set, at each split the goal is to minimize the impurity. For Decision Trees two criteria are widely used, Gini and Entropy. To determine the best suitable criterion for this data set, both have been used in an experiment. To get a better view on the performance it has been combined with the experiment for the max depth. The results are shown in Figure 5.2, the difference between the criteria is small, but entropy clearly is performing slightly better for almost all values of the maximum depth.

The second parameter of the Decision Tree algorithm is the max depth, it is used to specify the maximum size of the constructed tree. On first sight it might be expected that a deeper decision tree performance better than a less deep one. Figure 5.2 confirms this expectation, but only up to a max depth of five, higher values lead to a decrease in performance. This is caused by overfitting to the training set. A deep tree can fit itself to the exact data set, instead of to the underlying structure of the data.

The final parameter is the number of features taken into account at each split. The goal of decreasing the number of features to consider is to decrease overfitting. The features are randomly selected from all available features. Three different settings

FIGURE 5.2: Performance of the decision tree for different impurity criteria and max depth values.



are taken into account, n, $\sqrt{n}$ and $\log_2 n$. These three are used in combination with different max depth values to determine their performance. The results are shown in Figure 5.3. Due to the randomness of the feature selection process, the number of folds is increased to 50 in order to get a good estimation of the real performance. The expected decrease in overfitting does indeed return in the results. It starts to have a clear impact for a maximum depth of eight or greater instead of five. However, this does come at the cost of a lower maximum performance. Based on these finding the choice has been made to keep the number of available features at n. The required computation time with the selected parameters is 3 seconds.

FIGURE 5.3: Performance of the decision tree for different number of features and max depth values.

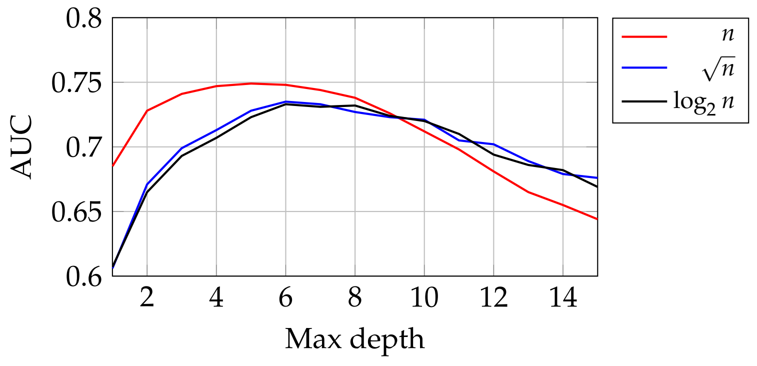


TABLE 5.12: Settings selected for the Decision Tree algorithm

Parameter	Selected value
Criterion	Entropy
Max Depth	5
n Features	n

5.1.6 Random Forest

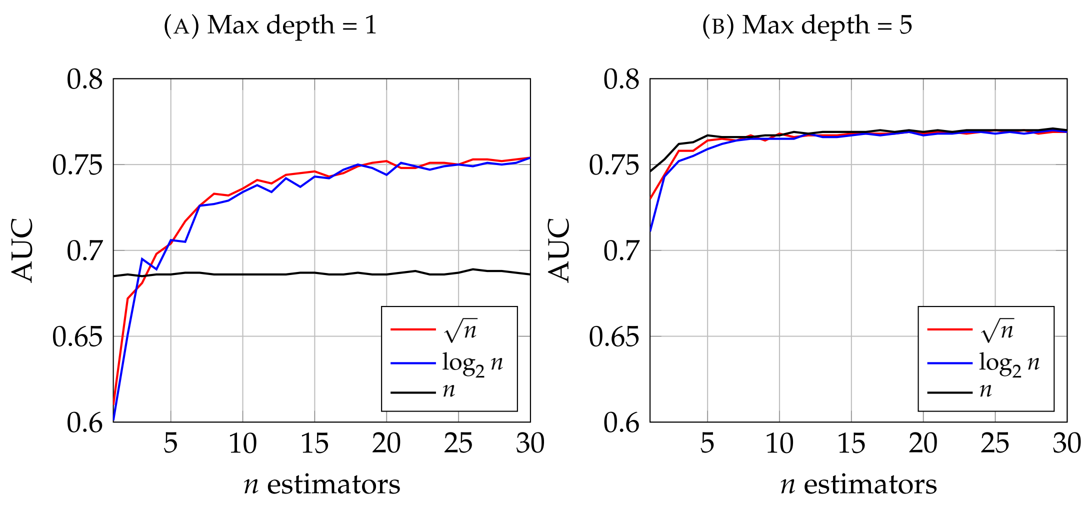
The Random Forest classifier is another ensemble model constructed from several individual Decision Trees. In this algorithm each of the individual trees is trained on a random subset of the training data, drawn with replacement. At each of the splits a subset of the features can be selected out of which one is selected to base the split on. After the training process the output of all individual trees is combined to determine the classification. In the previous chapter it has been determined that no resample method increases performance. Random undersampling will be used since it decreases computation times without negatively impacting algorithm performance. Any feature engineering aside from the default changes has no positive impact and will not be applied.

The following parameters will be analyzed for the Random Forest classifier.

– **Max depth:** The maximum number of layers the individual trees can contain.
– *n* **Estimators:** The number of individual trees out of which the classifier is constructed.
– **Max features:** The number of features considered at each split.

The first parameter for the Random Forest algorithm that will be determined is the maximum depth of the individual trees. For the maximum depth two strategies are taken into account. The first strategy is to use decision stumps, which are trees with a depth of one. When using those trees, the number of estimators has to be high. The second option is to use trees with a maximum depth of five, this is the value determined for the Decision Tree classifier. Using this value will lead to a lower number of individual estimators and a better performance per tree. Figure 5.4 shows the performance of both strategies for different values of *n* estimators. As expected, performance increases when the number of estimators increases. Performance increases fast with trees of depth five and slower with trees of depth one. The exception is when trees with a depth of one are used in combination with *n* features available at each split. This causes the performance to never increase. Which is probably caused by the same split being made in each of the decision stumps in the ensemble.

FIGURE 5.4: Performance of the Random Forest classifier for different numbers of estimators.



Using the aforementioned figure, the two strategies can be compared and for each a suitable number of estimators can be selected. When decision stumps are used, the performance increases until the ensemble consists of approximately twenty trees. As mentioned before, having n features available for the split leads to a constant low performance. The other two possibilities, $\sqrt{n}$ and $\log_2 n$, show no substantial difference. When trees with a depth of five are used, the performance increases faster. Performance stops increasing after approximately ten estimators, depending on the number of features available for a split. Performance increases fastest when all n features are available. The maximum performance reached with the deep trees is higher than is reached when using decision stumps.

To verify that a maximum depth of five is a good value for the deeper tree, several other values will be examined. In Figure 5.5, maximum depth values of 1 up to 30 are used in combination with five and ten estimators. It clearly shows a decrease in performance when the maximum depth goes beyond a certain threshold. Just as with the Decision Tree classifier, this is caused by overfitting. The problem decreases when the number of estimators increases. However, this does lead to an increase in computation time and still does not perform better than maximum depth values around five. Therefore it is a good choice to keep the maximum depth of the individual trees at five. The computation time with the selected parameters is 8 seconds.

FIGURE 5.5: Performance of the Random Forest classifier for different max depth values.

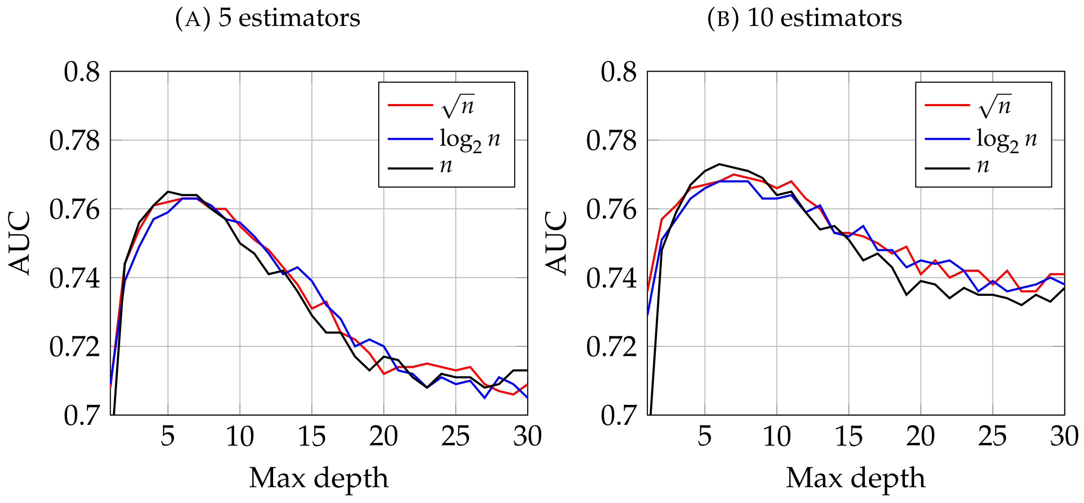


TABLE 5.13: Settings selected for the Random Forest algorithm

Parameter	Selected value
Max depth	5
n Estimators	10
Max features	n

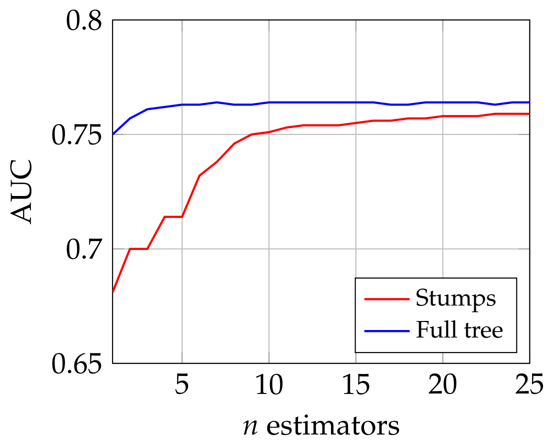
5.1.7 ADABoost

The ADABoost algorithm is an ensemble model consisting of multiple Decision Trees. The classifier is constructed by adding a new tree per iteration until a predetermined number of trees is reached. Each of the trees is trained using a weighted training set. In the weighted set, the samples that are most often incorrectly classified are given the highest weight. The trees are most often chosen to be decision stumps but can also be deeper trees. In the previous chapter is has been determined that resampling does not lead to a performance increase for ADABoost. Random undersampling has no influence on the performance but does decrease computation times, therefore it will be used for this algorithm. It was also determined that additonial feature engineering does not lead to an increased performance, thus only the default changes will be used. The following parameters will be taken into account to determine the ADABoost performance.

- **Max depth:** The maximum number of layers the individual trees can contain.
- *n* **Estimators:** The number of individual trees out of which the classifier is constructed.
- **Learning rate:** The speed with which the weight of each next tree decreases.
- **Criterion:** The criterion used to determine the split for branches.

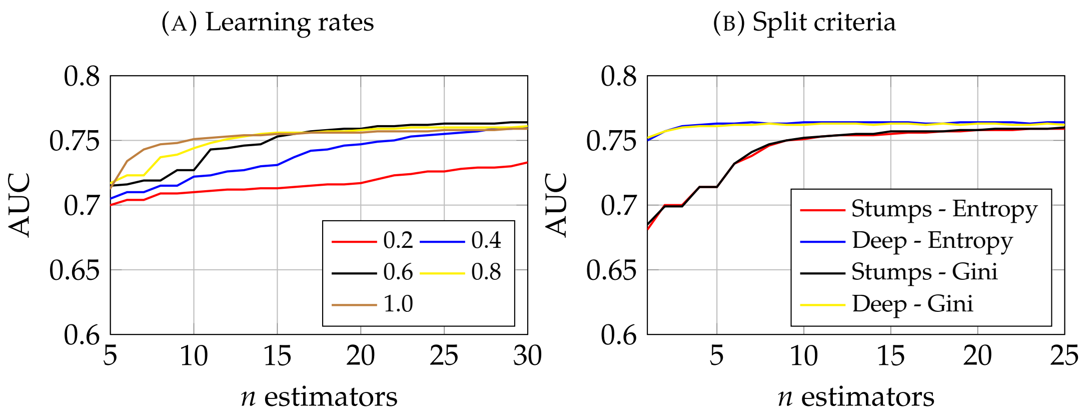
The first parameter is the number of estimators contained in the ADABoost classifier. The effect of the number of estimators is highly dependent on the settings of the individual trees. Therefore this test will be performed with two different individual classifiers. Since ADABoost is often used in combination with decision stumps, trees with a depth of one will be used. The second type of tree is the one with the settings found in the above section about Decision Trees, a depth of five. The results of this analysis are shown in Figure 5.6. Clearly the type of tree has a large impact on the performance of the model, especially for a small number of estimators. When the individual trees are chosen to be more elaborate, the ADABoost algorithm leads to a small performance increase. This is logical since each of the individual trees has a good predictive capability. For the decision stumps, ADABoost leads to a larger increase in performance. Up until ten estimators the performance increases sharply and even thereafter the performance keeps increasing slowly. Concluding, if the more complex tree is used five estimators is sufficient, for decision stumps the number of estimators should be increased to at least 25.

FIGURE 5.6: Performance of the decision tree for different number of features and max depth values.



The next parameter under consideration is the learning rate. This settings determines how fast the weight of the next tree decreases. The learning rate is closely related to the number of estimators. A high number of estimators does not work well in combination with a high learning rate because the later trees have practically no impact on the prediction. Therefore the learning rate shall also be analysed together with the number of estimators. Figure 5.7a shows the resulting performance for different learning rates. The most important conclusion from this figure is that a higher learning rate leads to a faster increasing performance. However the different settings all approach approximately the same final performance. The performance of the algorithm with a learning rate of 0.6 seems to be the highest, but it is questionable if this difference is significant.

FIGURE 5.7: Relation between n estimators and two parameters.



The last parameter for the ADABoost algorithm is the criterion. Just as with the Decision Tree algorithm, two criteria are taken into account, entropy and gini. Figure 5.7b shows the results of the comparison between the criteria. Since the plots of the different criteria are almost overlapping there is no substantial difference in performance. The computation time of 25 folds with the selected settings is 33 seconds.

TABLE 5.14: Settings selected for the ADABoost algorithm

Parameter	Selected value
Max depth	5
n Estimators	10
Learning rate	0.6
Criterion	Entropy

5.1.8 Gradient Boosting

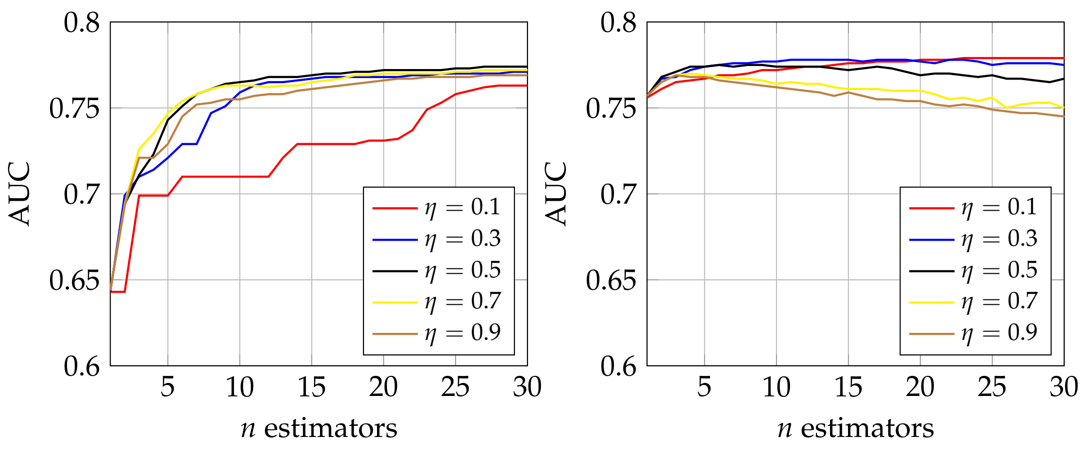
The last algorithm that is analyzed is Gradient Boosting. Just as the previous two algorithms this is an ensemble of multiple Decision Trees. It differs from the other ensemble models in the way that the trees are constructed. Instead of doing this by introducing random variation or weighting samples, it determines a gradient of the loss function and used that to generate supporting trees. In the previous chapter it has been determined that this algorithm can be best used with random oversampling or no resampling. Since random oversampling increases the computation time, no resampling will be used. It was also found that additional feature engineering has no positive effect, thus only the default engineering will be applied. Below the parameters are given that will be analyzed. One might expect the loss function to be one of the parameters that has to be determined. Gradient Boosting can be used in combination with two loss function, Entropy and Deviance. When Gradient Boosting is used with an exponential loss function, the algorithm is identical to ADABoost. Therefore, a Deviance loss function will be used.

– **Learning rate:** The speed with which the impact of each next tree decreases.
– *n* **Estimators:** The number of individual trees out of which the classifier is constructed.
– **Max depth:** The maximum number of layers the individual trees can contain.
– **Criterion:** The criterion used to determine the split for branches.

The learning rate is the first parameter that will be analysed. Figure 5.8 shows the performance of Gradient Boosting for several learning rates and number of estimators. The left graph shows the performance when Decision Stumps are used, the right graph for when the selected depth (five) of the Decision Tree classifier is used. From the graph about the Decision Stumps, it can be concluded that the learning rate has a small influence, only for very small values the performance stays low. An exception occurs when using a learning rate of 0.1, it shows an interesting pattern. It has several horizontal sections, between which the performance jumps up very sudden. The influence of the learning rate in combination with deeper trees is more interesting. In the right graph it can be seen that the performance actually decreases when a learning rate of 0.5 or higher is used. Thus for the larger trees it is a good choice to use a learning rate of 0.1. The highest AUC is slightly higher when trees with a depth of five are used.

FIGURE 5.8: Performance of the Gradient Boosting classifier for different numbers of estimators.

(A) Max depth = 1 (B) Max depth = 5



The last parameter for the Gradient Boosting algorithm is the Criterion used to determine a good split in the trees. Three different criteria are available, mean squared error, Friedman improved mean squared error and mean absolute error. During the comparison it became clear that mean absolute error is not a realistic option due to the high computation time. In Figure 5.9 the other two criteria are compared. Clearly the difference between the criteria is negligible. Therefore the default value will be used, which is the Friedman improved mean squared error. With the selected parameters, the required computation time for 25 folds is 42 seconds.

FIGURE 5.9: Impact of using different number of neighbors in combination with uniform or distance based weights.

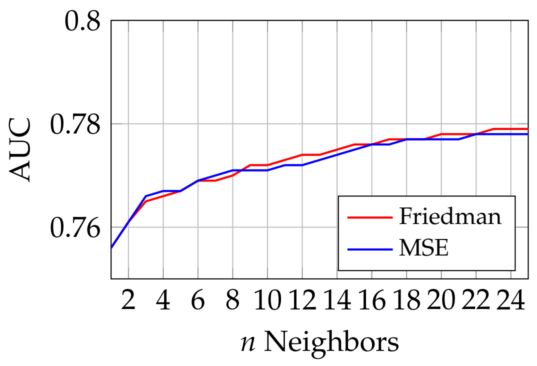


TABLE 5.15: Settings selected for the Gradient Boosting algorithm

Parameter	Selected value
Learning rate	0.1
n Estimators	20
Max depth	5
Criterion	Friedman improved mean squared error

5.1.9 Support Vector Machine

The last algorithm for which the parameters have to be determined is the Support Vector Machine. In the previous chapter it has been determined that a Support Vector Machine requires a form of resampling, which method makes no large difference. The choice has been made to use random undersampling since it decreases computation times. It has also been determined that doing additional feature engineering has no beneficial effect. For this algorithm, the following parameters have to be determined.

- **Kernel:** The type of kernel to use in the algorithm.
- **Error term (C):** Size of the penalty given to samples on the wrong side of the decision boundary.
- **Gamma:** Measure to specify the similarity between data points. Influences the standard deviation used in the kernel.
- **Degree:** The degree of the polynomial, when a polynomial kernel is used.

The first parameter which will be analyzed is the kernel which will be used. The available kernels are: linear, rbf and poly. Table 5.16 shows the performance of the different kernels when used with the corresponding default values. Rbf and poly have a similar performance, using a linear kernel results in a lower performance. Before a choice can be made on which kernel is most suitable, the parameters of the kernels have to be determined.

TABLE 5.16: Performance of Support Vector Machine in combination with different kernels and default settings.

Kernel	Performance
linear	0.71
rbf	0.75
poly	0.75

First the linear kernel, which has a single parameter that has to be set, the error term. Values from 0.1 up to 100 are used to determine the performance. The results are given in Table 5.17 and show that the performance is the same for all settings.

The next kernel to be analyzed is rbf. It has two parameters which can be changed, the error term and gamma. The same values of the error term are taken into account as with the previous kernel. For gamma values ranging from 0.001 up to 10 are used. The results are shown in Table 5.18. It is clear that a too small or too high value of gamma, leads to a decrease in performance. A value of 0.01 seems to lead to the highest performance. This highest value is reached in combination with an error term of 10 or 100.

TABLE 5.17: Performance of a Support Vector Machine with a linear kernel and varying error terms.

Performance			
C=0.1	C=1	C=10	C=100
0.71	0.71	0.71	0.71

TABLE 5.18: Performance of a Support Vector Machine with a rbf kernel and varying error terms and gamma values.

	Performance			
Gamma	C=0.1	C=1	C=10	C=100
0.001	0.69	0.73	0.74	0.74
0.01	0.74	0.75	0.76	0.76
0.1	0.74	0.75	0.71	0.71
1	0.61	0.66	0.64	0.64
10	0.47	0.55	0.54	0.54

The last of the kernels for which the parameters will be determined is the polynomial kernel. This algorithm has three parameters, the firs two are gamma and error term, just as with the rbf kernel. The third parameters is the degree of the polynomial. A degree of 1 leads to linear kernel and is not taken into account. The highest degree taken into account is three, higher values will probably lead to overfitting. The results are shown in Table 5.19. With those results, the computation time for 25 folds is 54 seconds.

TABLE 5.19: Performance of a Support Vector Machine with a polynomial kernel and varying degrees and error terms.

		Performance			
Degree	Gamma	C=0.1	C=1	C=10	C=100
2	0.01	0.70	0.74	0.74	0.74
2	0.1	0.74	0.74	0.73	0.72
2	1	0.73	0.72	-	-
3	0.01	0.67	0.73	0.74	0.75
3	0.1	0.74	0.73	0.63	-
3	1	0.56	-	-	-
4	0.01	0.54	0.67	0.71	0.73
4	0.1	0.71	0.57	0.53	-
4	1	0.51	-	-	-

TABLE 5.20: Settings selected for the Support Vector Machine algorithm

Parameter	Selected value
Kernel	rbf
Error term	10
Gamma	0.01
Degree	Not applicable

5.1.10 Tested result

The final step in determining the performance of the different algorithm is to verify the performance with the test data set. As mentioned in Chapter 3, 25% of the data set has been kept separated until now. This is done to be able to verify the models on data that was not accessible before. First the algorithms are trained using the 75% that has been used until now. The trained algorithm is then applied to the test data set, and the performance is determined. The table below shows the results of testing the models. To be able to make a more in depth comparison between the algorithms, the precision and recall of all algorithms is also added. In Appendix B, Table B.3, the confusion matrices of the results are given.

TABLE 5.21: Performance of the different algorithms when determined using the test set.

Algorithm	AUC	Precision	Recall
Logistic regression	0.77	0.48	0.58
Neural network	0.78	0.45	0.62
Naive Bayes	0.74	0.40	0.63
k Nearest neighbors	0.76	0.44	0.61
Decision tree	0.76	0.45	0.60
Random forest	0.77	0.44	0.63
ADABoost	0.77	0.41	0.67
Gradient boosting	0.78	0.45	0.61
Support Vector Machine	0.76	0.47	0.59

5.2 Peer to peer lending - Prosper

For the second data set the parameters of the algorithms have to be determined separately from the first data set. Since each data set is different it might be beneficial to use other settings. However, since this is the second iteration of determining the parameters it will be discussed more concise. In graphs showing the result of running an experiment, the results of the same experiment with the previous data set will be shown as a dashed line. This is done to be able to see the differences and similarities between the data sets. Please note that Naive Bayes has been left out since it has no parameters that can be changed, see Section 5.1.4 for more details.

5.2.1 Logistic regression

Just as with the previous data set, Logistic Regression is the first algorithm which will be analyzed. In the previous chapter the different resample methods have been applied to this data set. From that experiment it was concluded that the choice of resample method has no impact on the performance of the machine learning algorithm. Therefore the choice has been made to use random undersampling since it decreases computation times.

– **Regularization:** Two types of regularization are available, l1 and l2. The first of these tries to create a sparse classifier.
– **Solver:** Two solvers are taken into account, sag and saga. Both solvers are explained in Chapter 2.
– **Regularization term (C):** smaller values indicate more generalization.

The first parameters which will be analyzed are the solver and regularization. Since both have two possible values, four experiments have to be ran. The results are shown in Table 5.22. The table shows a similar pattern as with the previous data set, the performance difference between the two solvers and regularization terms is minimal, but the computation time shows a large difference. Thus, based on the computation time, liblinear will be used in combination with L1.

The next parameters is the regularization term, it will be analyzed from 0.1 up to 10.0. Those values will be used in a combination with Liblinear and L1 regularization. The results are shown in Table 5.23, just as with the previous data set, this variable has almost no impact on the performance and thus the default value of 1.0 will be used. Only when a high value of 10.0 is used, does the performance decrease a small amount.

TABLE 5.22: Performance and computation time of different solvers and regularizations.

	AUC	Computation time
saga - l1	0.78	02:02
saga - l2	0.77	01:16
liblinear - l1	0.78	00:12
liblinear - l2	0.78	00:14

TABLE 5.23: Performance for different values of the regularization term C.

C	AUC
0.1	0.78
0.5	0.78
1.0	0.78
2.0	0.78
5.0	0.78
10.0	0.77

TABLE 5.24: Settings selected for the Logistic Regression algorithm

Parameter	Selected value
Regularization	L1
Solver	Liblinear
Regularization term	1.0

5.2.2 Artificial Neural Network

The next algorithm for which the parameters will be determined is the Artificial Neural Network. Preparing the data for a Neural Network can best be done using random undersampling as resample method. For this algorithm, the following parameters have to be determined.

– **Solver:** Two solvers are taken into account, sgd and adam. Both solvers are explained in Chapter 2.
– **Hidden layers:** The number and size of the layers between the input and output layers.
– **Activation function:** The shape of the function used to transform the input signals of the neurons.
– **Learning rate:** Size of the step with which weights are changed. It can be a constant or adaptive rate.
– **Regularization term:** A term that can be used to penalize complex models. This can be used to counter overfitting.
– **Tolerance:** If the decrease in loss during training is smaller than this threshold for two consecutive epochs, training is stopped.
– **Max iterations:** The maximum number of iterations to reach convergence.

The parameters will be examined in the same order as has been used with the previous data set. First the parameters will be determined for use with the sgd solver followed by the parameters for the adam solver. Fist the shape of the hidden layers and the different activation functions will be analyzed, then the learning rate, followed by the regularization term and finally the tolerance and maximum iterations.

Sgd solver

The first parameter that will be analysed is the shape of the hidden layer. As discussed with the previous data set, the shape usually depends on the number of input nodes. With this data set the model has 182 input nodes and is clearly high dimensional. The number of nodes in a hidden layer is usually chosen to be smaller than the input layer, or a previous hidden layer. Due to the size of the data set computation time where long with at least 15 minutes per experiment. With the many experiments that have to be run this caused the total computation time to become unfeasible. Therefore, the choice has been made to increase the learning rate from the default value of 0.001 up to 0.01. The different shapes of the hidden layer that are taken into account and the corresponding performances are shown in Table 5.25. From the results it can be concluded that adding more than a single hidden layer has no positive effect on the performance or computation times. From the three settings with a single hidden layer, the one with thirty nodes seems to lead to the highest performance, but the difference is small. For now the Logistic activation function leads to the highest performance, but this might change with the learning rate.

TABLE 5.25: Neural Network performance for several different forms of hidden layers with different activation functions. All networks are trained with the Sgd solver.

Nodes per layer			Logistic		ReLu		Tanh	
First	Second	Third	AUC	Time	AUC	Time	AUC	Time
120	60	30	0.63	01:08	0.76	04:21	0.76	40:52
120	60	-	0.76	06:20	0.77	07:01	0.75	32:21
60	30	-	0.79	09:33	0.76	03:52	0.75	13:36
120	-	-	0.77	06:27	0.77	12:53	0.76	51:13
60	-	-	0.79	13:55	0.78	05:10	0.75	18:23
30	-	-	0.80	08:48	0.79	02:38	0.77	04:36
-	-	-	0.77	00:24	0.77	00:27	0.77	00:24

The next parameters of the Neural Network that will be determined is the learning rate. In Table 5.26, the performance is shown for several values of the learning rate and the different activation functions. The table shows that the learning rate has a large influence on the computation time required, but also on the performance. Smaller values for the learning rate lead to a higher performance at the cost of an increased computation time. An adaptive learning rate has also been taken into account, as shown in Table 5.27. The results show that it has no beneficial effect to make the learning rate adaptive. A constant learning rate of 0.01 will be selected.

TABLE 5.26: Neural Network performance for several different learning rates.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
1	0.76	01:02	0.76	00:28	0.72	00:43
0.1	0.76	03:22	0.78	00:51	0.74	00:51
0.01	0.80	08:48	0.79	02:38	0.77	05:36

Now a learning rate has been selected, alpha will be determined for use with the sgd solver. The results of using alpha values ranging from 0.001 to 1 are shown in Table 5.28. When a ReLu or tanh activation function is used, performance increases

TABLE 5.27: Neural Network performance with an adaptive learning rate and several activation functions.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
1 -> 0.0001	0.75	18:24	0.76	14:05	0.73	19:41
0.1 -> 0.0001	0.76	07:12	0.78	06:48	0.76	01:02

with higher values for alpha. This effect is reversed when a logistic activation function is used. The best performance is achieved when ReLu is used in combination with an alpha of 1, this results in an AUC of 0.81 and also a low computation time of 1:35. It is interesting to see that alpha has a larger influence on the computation time when a logistic activation function is used in comparison with the other activation functions.

TABLE 5.28: Neural Network performance for several different alpha values

	Logistic		ReLu		Tanh	
Alpha	AUC	Time	AUC	Time	AUC	Time
1	0.77	01:22	0.81	01:35	0.80	01:43
0.1	0.80	04:23	0.80	02:17	0.79	03:34
0.01	0.80	08:36	0.80	02:43	0.77	05:03
0.001	0.80	08:48	0.79	02:38	0.77	04:36

Finally, the maximum iterations and tolerance have to be determined. These parameters are used to determine when the algorithm should stop training. When they are chosen to strict, it will result in the performance not being as high a it can be. Since using a small decrease in learning rate lead to feasible computation times it was not necessary to define a strict threshold or low number of maximum iterations. The choice has been made to set the threshold to zero, this leads to the algorithm continuing training until the loss does not decrease for two consecutive epochs. The maximum number of iterations has been set to 20,000, a value which will almost never be reached.

Adam solver

Applying the algorithm with its default settings, except for the learning rate and varying the hidden layer shape results in the performance given in Table 5.29. These results are remarkable since the model without any hidden layers seems to perform best regardless of the chosen activation function. This should only happen if the data set is linearly separable, which is unlikely in a high dimensional data set. One of the possible reasons for this behavior is overfitting, in Neural Network this can be countered by increasing the regularization term. By varying the regularization term it was found that it indeed has the expected impact on the performance. The highest performance is found with a term of 0.1, the results are shown in Table 5.30. This confirms that the model is overfitted to the training set.

With the generalization term increased, the results look very different. Now the experiments where a hidden layer is added clearly show a higher performance, as is expected with data sets that are not linearly separable. Just as with the previous data set, the results show that it is not necessary to add more than a single hidden layer. Adding more layers does not increase performance, using three hidden layers even

TABLE 5.29: Neural Network performance for different forms of hidden layers and activation functions. These networks are trained with the adam solver and use a regularization term of 0.001.

Nodes per layer			Logistic		ReLu		Tanh	
First	Second	Third	AUC	Time	AUC	Time	AUC	Time
120	60	30	0.75	03:11	0.76	02:21	0.75	02:23
120	60	-	0.76	03:47	0.76	01:45	0.74	03:55
60	30	-	0.74	01:50	0.75	01:06	0.74	01:50
120	-	-	0.77	03:57	0.76	01:36	0.74	03:37
60	-	-	0.75	02:47	0.75	01:11	0.74	02:08
30	-	-	0.74	02:33	0.76	00:56	0.72	01:37
-	-	-	0.77	00:09	0.77	00:09	0.77	00:09

leads to the same performance as no hidden layers. Using a single hidden layers with thirty nodes, leads to the highest performance and the lowest computation time of the single hidden layer variants. Thus the choice has been made to use a single hidden layer with thirty nodes.

TABLE 5.30: Neural Network performance for different forms of hidden layers and activation functions. These networks are trained with the adam solver and use a regularization term of 0.1.

Nodes per layer			Logistic		ReLu		Tanh	
First	Second	Third	AUC	Time	AUC	Time	AUC	Time
120	60	30	0.77	01:07	0.77	02:28	0.77	03:08
120	60	-	0.79	01:40	0.77	02:04	0.77	02:17
60	30	-	0.80	01:23	0.77	01:10	0.77	01:24
120	-	-	0.79	01:48	0.79	01:22	0.77	01:46
60	-	-	0.80	01:14	0.79	00:50	0.77	01:04
30	-	-	0.80	00:50	0.79	00:35	0.78	00:41
-	-	-	0.77	00:09	0.77	00:09	0.77	00:07

Table 5.31 shows the impact of changing the learning rate. As seen with the previous experiments concerning the learning rate, it again has a large impact on the computation time. When a learning rate of 1 is used, computation times are very short, but the performance is very low. Decreasing the learning rate to 0.1 still results in a short computation time, but a higher performance. Using a learning rate greater than 0.01, does not lead to an increased performs, only to a longer computation time. Therefore, a learning rate of 0.01 will be selected.

TABLE 5.31: Neural Network performance for several different learning rates.

	Logistic		ReLu		Tanh	
Learning rate	AUC	Time	AUC	Time	AUC	Time
1	0.68	00:15	0.60	00:12	0.67	00:07
0.1	0.78	00:15	0.77	00:14	0.76	00:07
0.01	0.80	00:47	0.79	00:32	0.78	00:42
0.001	0.80	03:50	0.79	02:51	0.78	03:25

The maximum number of iterations and the tolerance are set to the same values as used with the sgd solver.

Comparison

Now that the performance for both of the solvers has been determined, they can be compared. The core of this comparison is the performance and the computation time. Using the sgd solver, the highest performance is 0.81. This is achieved by using a single 30 node hidden layer, a ReLu activation function with a learning rate of 0.01 and an alpha of 1. The computation time is feasible with 01:35. The best performance with the adam solver is 0.80. This is performance is reached with a single hidden layer consisting of 30 nodes and a logistic activation function. The learning rate is set to 0.01 and alpha is set to 0.1.

The performance of the sgd solver is higher than the performance of the adam solver. However, the computation time when using adam is lower. The difference in computation time is small, 00:47 for the adam solver and 1:35 with the sgd solver. Based on the performance, the choice has been made to select the sgd solver.

TABLE 5.32: Settings selected for the Artificial Neural Network algorithm

Parameter	Selected value
Solver	Sgd
Hidden layers	Single layer with 30 nodes
Activation function	Relu
Learning rate	0.01
Regularization term	1
Tolerance	0
Max iterations	20,000

5.2.3 k Nearest Neighbors

k Nearest Neighbors is the next algorithm that will be experimented with. In the previous chapter it has been determined that this algorithm in combination with this data set returns the best results when used with random undersampling. For this algorithm, the following parameters have to be determined.

- **Algorithm:** Three different algorithms are compared, $k - d$ Tree, Ball Tree and Brute Force.
- **Leaf size:** Number of samples per leaf, only used when the $k - d$ Tree or Ball Tree algorithm is used
- k **Neighbors:** The number of neighbors on which the classification is based.
- **Weights:** The weights of the individual n neighbors used for classification. These can be based on the distance of the neighbor or can all be the same, uniform.

The first parameter to analyze is the algorithm used to construct the tree. Three different algorithms can be used $k - d$ Tree, Ball Tree and Brute Force. For each of these the performance is determined for several values of k, the computation time is also taken into account. Applying Nearest Neighbor with different algorithms to this data set shows the same results as with the previous data set. The performance is similar, but the computation time differs. The brute force construction algorithm is substantially faster and thus chosen to be used from now on. This results in the leaf size not being necessary to determine.

TABLE 5.33: Performance of different k Nearest Neighbors algorithms for several *n*.

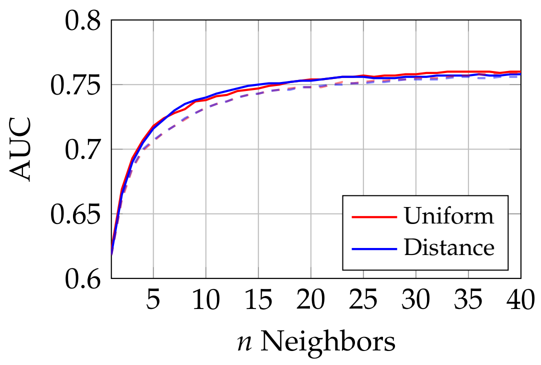
	kd Tree	Ball Tree	Brute Force
1	0.62	0.62	0.62
5	0.72	0.72	0.72
10	0.74	0.74	0.74
20	0.75	0.75	0.75

TABLE 5.34: Calculation time of different k Nearest Neighbors algorithms for several *n*.

	kd Tree	Ball Tree	Brute force
1	05:34.4	04:28.8	00:45.6
5	05:46.9	04:30.4	00:52.8
10	05:52.2	04:28.7	00:52.7
20	05:54.8	04:30.9	00:24.3

The next step in determining the parameters for the k Nearest Neighbors algorithm is to determine a suitable number of neighbors taken into account during classification. In Figure 5.10 values from 1 up to 40 are displayed. The performance clearly increases with the number of neighbors. The increase becomes less when the number of neighbors gets larger, at approximately 25 neighbors, the increase in performance becomes negligible.

FIGURE 5.10: Impact of using different number of neighbors in combination with uniform or distance based weights.



The last parameters of this algorithm is the weights with two possible settings, uniform and distance. When uniform is chosen, each of the *n* neighbors are given an equal weight. If distance weights are used, the weight of each of the *n* neighbors is based on the distance from the sample that has to be classified. The figure used for the number of neighbors, shows the performance for both of the types of weight. The difference in performance between the two settings is negligible. Therefore the choice has been made to use uniform weights, since it is the less complex setting.

TABLE 5.35

Setting	Selected value
Algorithm	Brute force
Leaf size	Not applicable
n Neighbors	25
Weights	Uniform

5.2.4 Decision Tree

The next algorithm for which the parameters will be determined is the Decision Tree. During the resample experiment it was determined that this algorithm performs the highest when random oversampling is used. For this algorithm the following settings are taken into account.

– **Criterion:** What measure of impurity to use to base the split on.
– **Max depth:** The maximum depth of the constructed tree.
– *n* **Features:** The number of features taken into account at each split.

The first experiments are to determine a good value for the maximum depth of the decision tree. For the previous data set it was found that a too deep tree will lead to overfitting. The expectation is that this problem will be less visible for this data set since the number of features is substantially higher. Figure 5.11 shows the results of the experiments. Clearly overfitting does still become a problem for deep trees, however it does occur at a relatively large depth. With the entropy criterion overfitting only start happening at a max depth of nine. When overfitting start deteriorating the predictive performance, the trees constructed with the entropy criterion show less deterioration. Since performance is otherwise equal, entropy will be selected as the criterion.

FIGURE 5.11: Performance of the decision tree for different impurity criteria and max depth values.

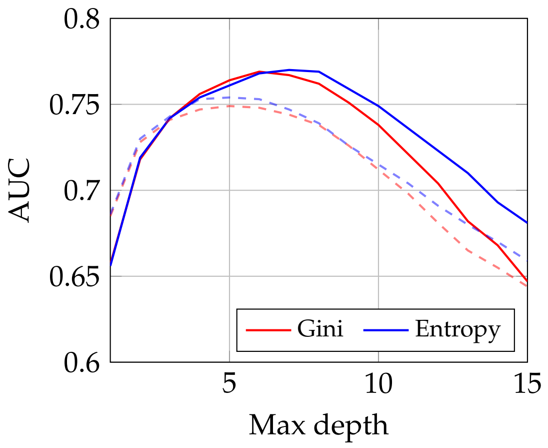
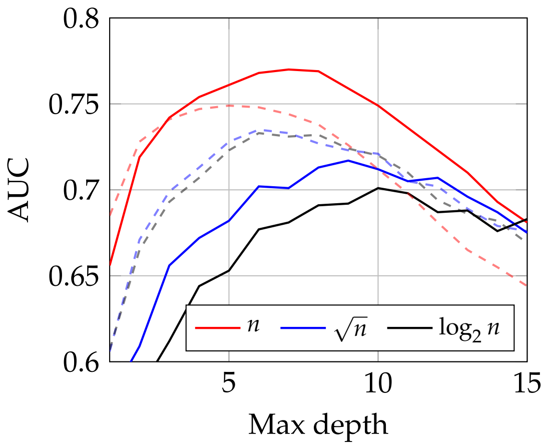


FIGURE 5.12: Performance of the Decision Tree algorithm for different number of features and max depth values. All have been constructed with the entropy criterion.



Now that the criterion is determined, the maximum depth of the trees has to be chosen. Since the choice of the maximum depth depends on the number of feature available per split, they will be determined together. Figure 5.12 shows the performance of Decision Trees with varying maximum depth and available features. The results are very similar to those seen with the previous data set. Making *n* features available at each split leads to the highest performance, but it also leads to the greatest influence of overfitting.The other two possibilities do not come close to the performance of having all *n* features available. Therefore the choice is made to select *n* features in combination with a maximum depth of 7. The required computation time for 25 folds with the selected parameters is 01:06.

5.2.5 Random Forest

Random Forest is the next algorithm which will be analyzed in combination with the second data set. In Chapter 4 this algorithm has been used in combination with

TABLE 5.36: Parameters selected for the Decision Tree algorithm.

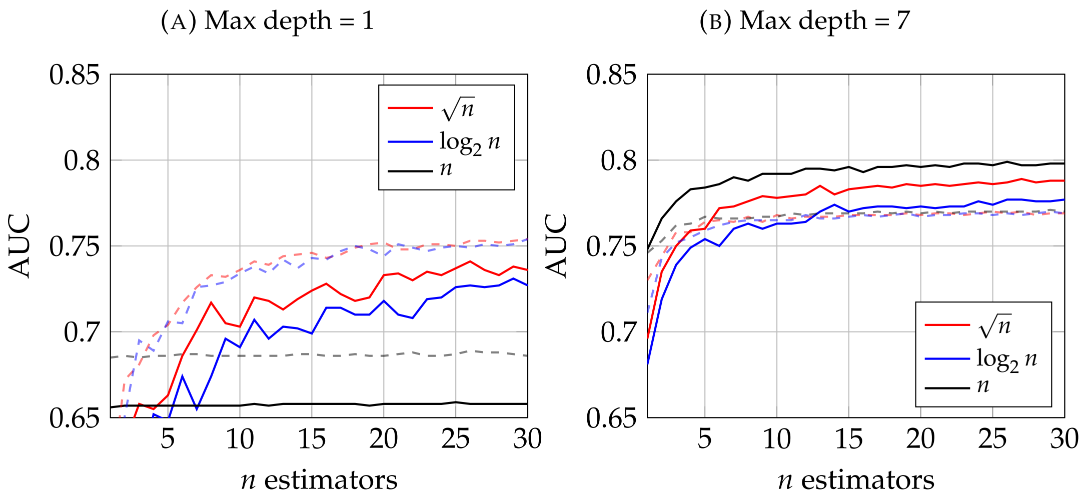
Parameter	Selected value
Criterion	Entropy
Max Depth	7
n Features	n

several resample methods. From those experiments it was concluded that Random Forest has to be used in combination with random undersampling. The following parameters have to be determined for this algorithm

– **Max depth:** The maximum number of layers the individual trees can contain.
– n **Estimators:** The number of individual trees out of which the classifier is constructed.
– **Max features:** The number of features considered at each split.

The first Random Forest experiment with the Prosper data set is to determine the influence of the maximum depth of the trees. Just as with the previous data set, two strategies for the maximum depth will be used. The first strategy is to use decision stumps and try to get a high performs by adding many trees. The second strategy uses the maximum depth determined as the best value for a single Decision Tree. The goal is to make each individual tree a good predictor. Figure 5.13 shows the performance of both strategies for different values for the n estimators parameter. The performance when using decision stumps in combination with n features available for a split, leads to a constant low performance. When not all features are available for a split, the performance does show the expected increase when the number of estimators is increased. The highest performance is approximately 0.74 which is reached with 25 estimators and $\sqrt{n}$ features available for a split. The performance when using the deeper trees with a depth of 7 is higher. The performance reaches approximately 0.80, when 20 estimators are used in combination with n features available per split.

FIGURE 5.13: Performance of the Random Forest classifier for different numbers of estimators.



To be sure that seven is a good value for the maximum depth, several other values will be examined. in combination with 20 estimators and n features available. In

Figure 5.14 the performance is shown for a maximum depth of 1 up to 30. The figure shows how the performance increases at first when the maximum depth increases. After the depth has reached approximately 12, the performance starts to deteriorate due to overfitting. The maximum depth selected up to now is seven, from the figure it can be concluded that performance does increase with a higher maximum depth. A value of 10 is more suitable. The required computation time for 25 folds with the selected parameters is 01:18.

FIGURE 5.14: Performance of the Random Forest classifier for different max depth values and 20 estimators.

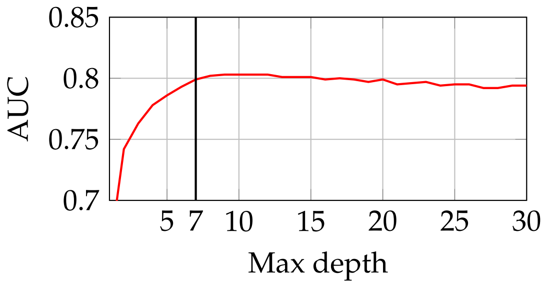


TABLE 5.37: Settings selected for the Random Forest algorithm

Parameter	Selected value
Max Depth	10
n Estimators	20
n Features	*n*

5.2.6 ADABoost

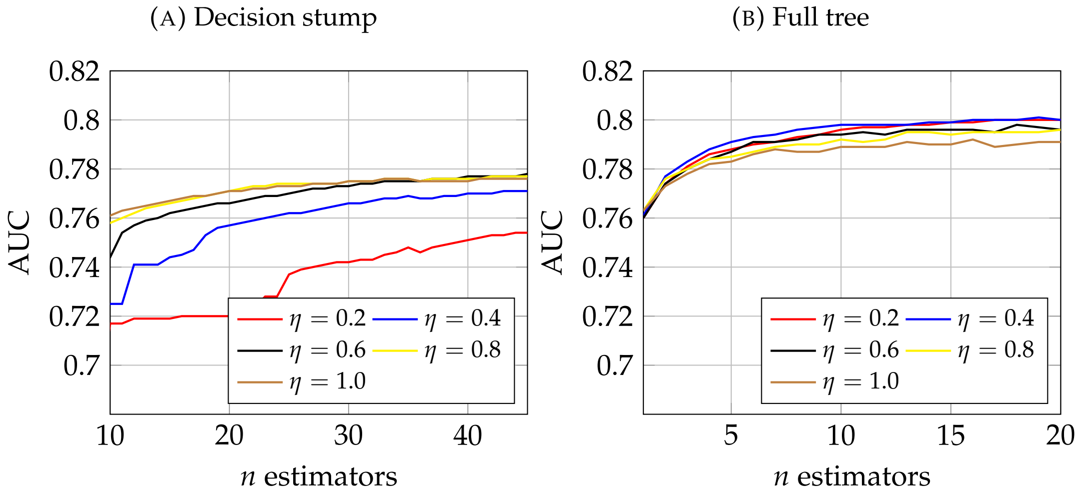
The next algorithm also is also an ensemble of Decision Trees, ADABoost. In the previous chapter the impact of the different resample methods has been determined. It was found that random undersampling is a suitable method for use with this algorithm. In this section the following parameters will be analyzed.

- *n* **Estimators:** The number of individual trees out of which the classifier is constructed.
- **Learning rate:** The speed with which the impact of each next tree decreases.
- **Max depth:** The maximum number of layers the individual trees can contain.
- **Criterion:** The criterion used to determine the split for branches.

The first three of the parameters mentioned above will be analyzed together. For the maximum depth of the trees, two values are taken into account. The first maximum depth take into account is one, this results in using Decision Stumps. The second value is seven, the optimal depth determined for the Decision Tree algorithm. Figure 5.15a shows the performance when Decision Stumps are used in combination with several learning rates and number of estimators. Figure 5.15b shows the same information but for ADABoost based on full tress. Both graphs clearly show that performance increases with the number of estimators. Performance increases slower when the number of estimators gets larger. When using Decision Stumps the AUC approaches 0.78, a learning rate of 0.8 or 1.0 seems to make the performance increase the fastest. For those two learning rates, the highest performance is reached at approximately 40 estimators. When using trees with a depth of seven, the results are different. Now the higher learning rates actually lead to lower performance. When using a learning rate of 0.2 or 0.4, an AUC of 0.80 is reached. Based on these numbers, the choice has been made to use trees with a depth of seven, 15 estimators and a learning rate of 0.4.

The last parameter that has to be determined is the criterion on which the split is based. To determine the criterion, both possibilities will be used in combination

FIGURE 5.15: Performance of the ADABoost algorithm for different learning rates and different numbers of estimators.



with the above determined values for the other parameters. The results are shown in Figure 5.16. From this graph the conclusion can be drawn that entropy leads to a higher AUC across a wide range of n estimators. The required computation time for 25 folds with the selected parameters is 01:15.

FIGURE 5.16: Influence of the criterion on the performance of the ADABoost algorithm.

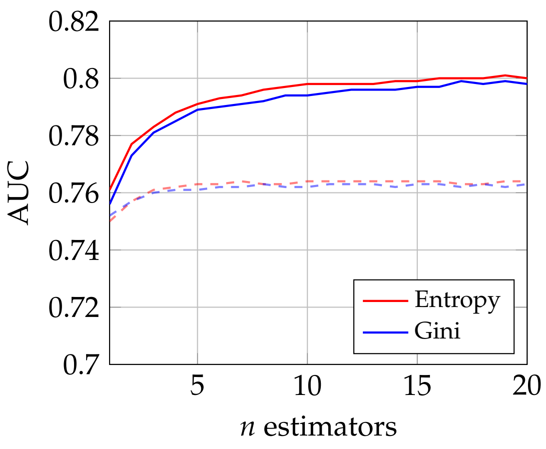


TABLE 5.38: Settings selected for the ADABoost algorithm

Parameter	Selected value
n Estimators	15
Learning rate	0.4
Max depth	7
Criterion	Entropy

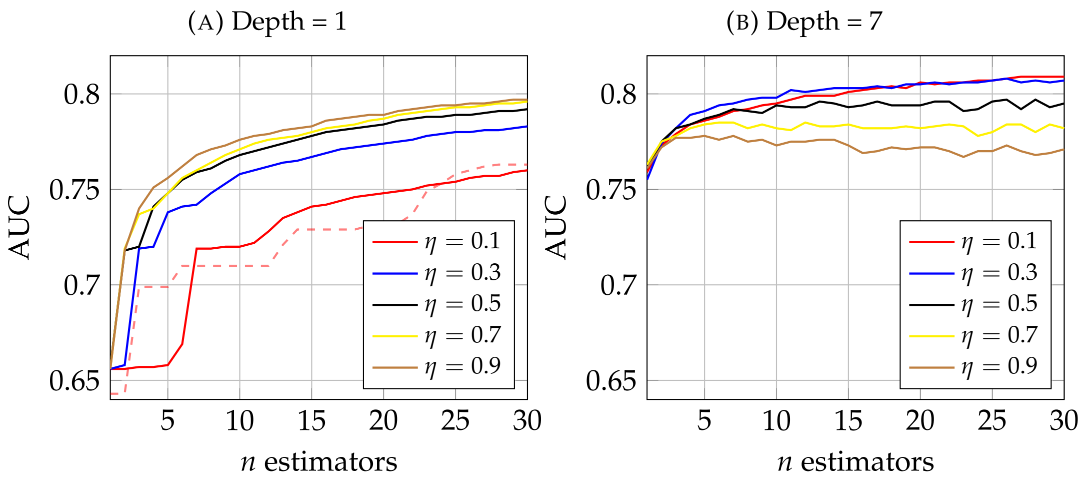
5.2.7 Gradient Boosting

The last of the algorithm for which the parameters have to be determined is the Gradient Boosting classifier.

- **Learning rate:** The speed with which the impact of each next tree decreases.
- ***n* Estimators:** The number of individual trees out of which the classifier is constructed.
- **Max depth:** The maximum number of layers the individual trees can contain.
- **Criterion:** The criterion used to determine the split for branches.

The first parameter to be analysed is the learning rate. The performance of the algorithm will be determined for several different learning rates in combination with two possible values for the maximum depth. The first value is one, which leads to the usage of Decision Stumps. The second value is seven, which is the maximum depth selected for the Decision Tree classifier. In Figure 5.17 the performance is plotted against the number of estimators. The left graphs shows the performance when Decision Stumps is used, the right graph when deeper trees are used. For Decision Stumps it is clear that a higher learning rate leads to a higher performance. The lowest learning rate shows a similar pattern as with the previous data set, the performance shows several flat section between which it suddenly jumps up. For Decision Stumps a learning rate of 0.7 or 0.9 is a good choice. Looking at the figure for the trees with a maximum depth of seven, the influence of the learning rate is reversed. For the deeper trees a larger learning rate leads to a lower performance. A learning rate of 0.1 is selected to be used in combination with deeper trees.

FIGURE 5.17: Performance of the Gradient Boosting classifier for different numbers of estimators.



The next parameter that has to be determined for the Gradient Boosting algorithm is the criterion. Three possible criteria can be used, mean squared error, Friedman improved mean squared error and the mean absolute error. Using all three of the possibilities resulted in the same performances. Therefore the choice has been made to use the default settings, Friedman improved mean squared error. The required computation time for 25 folds with the selected parameters is 01:53.

TABLE 5.39: Settings selected for the Gradient Boosting algorithm

Parameter	Selected value
Learning rate	0.1
n Estimators	20
Max depth	7
Criterion	Friedman improved mean squared error

5.2.8 Tested result

Now that the parameters have been determined for each of the algorithm, the final performance can be determined. To do so, each of the algorithms will be used to predict the class of the samples in the test set. The models used to make these predictions are trained using the training set. The result is that samples are classified that have never been used before. The results are given in Table 5.40. For the reader to be able to assess the performance in more detail, the confusion matrix of each test is shown in Appendix B, Table B.4.

TABLE 5.40: Performance of the different algorithms when determined using the test set.

Algorithm	AUC	Precision	Recall
Logistic regression	0.77	0.16	0.73
Neural network	0.81	0.17	0.77
Naive Bayes	0.66	0.10	0.77
k Nearest neighbors	0.75	0.15	0.66
Decision tree	0.77	0.15	0.78
Random forest	0.80	0.15	0.83
ADABoost	0.80	0.17	0.73
Gradient boosting	0.81	0.16	0.83

5.3 Data influences

Now that it is known which parameters are suitable for each of the algorithms, some further experimenting will be done. The experiments in this section will mainly be about separating the data set before using it with the algorithms. Such a separation could for example be based on the education level. The goal of these experiments is to determine if any differences between the groups exist which influence the algorithm performance. This separation is only applied to the testing set. This leads to the algorithms being fully trained on the training set, and used to classify only a subset of the testing set.

Aside from separating the data set, winsorizing will also be experimented with. In this section not all of the algorithms will be used. Since in the previous sections of this chapter the performance of the different algorithms has been determined, only the ones with a relatively high performance will be used here. Just as in the rest of the report, the data set with the Taiwanese credit lines will be used first, followed by the second data set.

5.3.1 Credit data - Taiwan

The first feature which will be used to split the data set with the Taiwanese credit lines is the 'education' feature. As described in Chapter 4, this feature can take four different values, 'graduate school', 'high school', 'university' and 'other'. In Table 5.41, some info is shown about the different categories. Both the 'other' and 'high school' categories have a low amount of samples.

The results of applying the selected algorithms to the different education categories are given in Table 5.42. They clearly show that the 'other' category performs worst of the different categories. This is probably caused by the low number of samples in that category. It can also be the result of the 'other' category containing hard to classify samples. The 'high school' category also seems to perform less than 'graduate school' and 'university'. However, it is difficult to determine the exact cause. The category has 16.39% of the samples, which should be sufficient. Another possible reason is that the category simply has less structure, which leads to accurate

TABLE 5.41: Possible values for the 'education' feature in the first data set and the number of occurrences.

Category	n	%
Other	468	1.56%
Graduate school	10,585	35.28%
University	14,030	46.77%
High school	4,917	16.39%

predictions being difficult. The largest two categories, 'graduate school' and 'university', both show a good performance similar to when all samples are used together. 'university' does perform slightly better, this might be caused by the samples in this category being easier to predict. However, it might also be due to random variation.

TABLE 5.42: Performance of the selected algorithms, when only the samples belonging to the stated education category are used.

Value	LR	NN	DT	GB	RF
Original	0.77	0.78	0.76	0.78	0.77
Other	0.72	0.70	0.68	0.68	0.70
Graduate school	0.76	0.77	0.75	0.78	0.77
University	0.77	0.79	0.76	0.78	0.78
High school	0.74	0.76	0.74	0.76	0.75

The next feature which will be analyzed is the 'marriage' feature. This feature can take four different values, 'other', 'married', 'single' and 'divorced'. In Table 5.43 some information is shown about the different categories. Just as with the 'education' feature, this feature also has some categories with a very small number of samples. The 'other' category is the smallest with just 54 samples. Since the number of samples is so low, it is difficult to determine the performance of classifying those samples and thus it will not be taken into account. The second category with a small number of samples is the 'divorced' category, with 323 samples.

TABLE 5.43: Possible values for the 'marriage' feature in the first data set and the number of occurrences.

Meaning	n	%
Other	54	0.18%
Married	13,659	45.53%
Single	15,964	53.21%
Divorced	323	1.08%

The results of separating the samples based on the marital status is shown in Table 5.44. The 'married' and 'single' categories show almost no deviation in performance, only the Decision Tree algorithm has a change of 0.01. The 'divorced' category does show a large change in performance, across all of the algorithms the AUC is a lot lower. This is probably caused by the size of the category, but it might also be caused by the category having less structure.

The last modification that will be made, is to winsorize the data set. By applying winsorization to a data set, its extreme values are limited to a certain value. The limit is usually chosen as a certain percentile. The effect of winsorizing the 'balance limit' feature can be seen in Figure 5.18. Both 98% and 90% winsorization will be

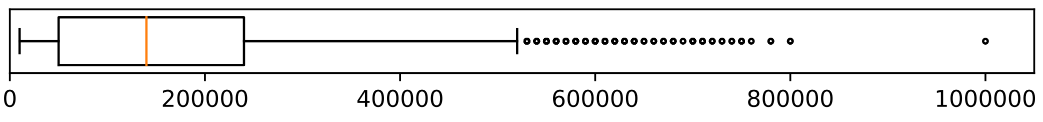
TABLE 5.44: Performance of the selected algorithms, when only the samples belonging to the stated marriage status category are used.

Value	LR	NN	DT	GB	RF
Original	0.77	0.78	0.76	0.78	0.77
Married	0.77	0.78	0.75	0.78	0.77
Single	0.77	0.78	0.76	0.78	0.77
Divorced	0.73	0.73	0.72	0.77	0.72

used on all of the numerical features in the data set. 98% winsorization means that the smallest 1% and largest 1% of the values will be changed to the value of the 1st and 99th percentile respectively. The results of doing so are shown in Table 5.45. As only very small differences in performs occur, it can be concluded that winsorization has a very limited influence on this data set.

FIGURE 5.18: Effect on the 'balance limit' feature when winsorization is applied at the 1st and 99th percentile.

(A) Original



(B) Winsorized

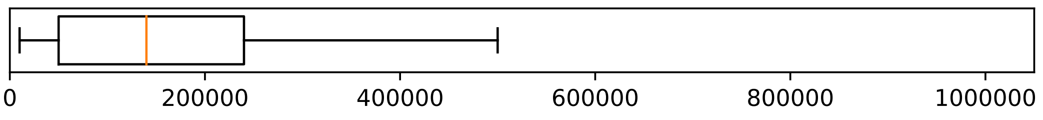


TABLE 5.45: Possible values for the 'education' feature in the first data set and the number of occurrences.

Value	LR	NN	DT	GB	RF
Original	0.77	0.78	0.76	0.78	0.77
Winsorized (98%)	0.77	0.77	0.75	0.78	0.77
Winsorized (90%)	0.77	0.77	0.75	0.78	0.77

5.3.2 Peer to peer lending - Prosper

The first feature which will be used to split the second data set is the 'ListingCategory'. This feature is used to specify the reason for which the loan is taken out. Twenty different categories exist, of which a few have the majority of the samples. To prevent the creation of very small data sets, only the categories with more than 2,000 samples will be used. Five of the twenty categories meet that criterion. In Table 5.46 the different categories, their meaning and the number of samples in each of the categories is given.

The performance of each of the separate categories is given in Table 5.47. It is interesting to see the high performance when only category one is used. With a Neural Network or Gradient Boosting algorithm an AUC of 0.83 is achieved, the highest value seen so far. All the other categories show a lower performance than when the entire data set is used. This can be caused by the size of the category, however since they all have at least 2,000 samples, this is unlikely. It would be more logical if the difference in performance is caused by the structure of the samples in the different categories.

TABLE 5.46: Amount of samples in the selected listing categories and their meaning.

Category	Meaning	n
1	Debt consolidation	53,246
2	Home improvement	6,812
3	Business	5,315
6	Auto	2,244
7	Other	9,242

TABLE 5.47: Performance of the selected algorithms when used in combination with only homeowner or only not homeowners.

Category	LR	NN	DT	GB
Original	0.77	0.81	0.77	0.81
1	0.80	0.83	0.80	0.83
2	0.73	0.72	0.66	0.70
3	0.70	0.75	0.67	0.74
6	0.69	0.73	0.62	0.69
7	0.70	0.73	0.69	0.74

The next feature on which a split of the data is based is the term of the loan. Three different terms occur in the data set, 12, 36 and 60 months. Of these categories the group of 12 months is the smallest with only 1,614 samples. 58,806 loans have a term of 36 months and 24,544 a term of 60 months.

The results of using the selected models on the separate terms is shown in Table 5.49. The category with loans of a term of 12 months has the lowest performance for all the tested algorithms. Again the category with the lowest number of samples has the worst performance, since this is a returning observation, the size probably has a large influence. From the other two categories, the loans with a term of 36 months perform best with a performance equal to or slightly lower than the original performance.

TABLE 5.48: Amount of samples per term length category.

Term	n
12 months	1,614
36 months	58,806
60 months	24,544

TABLE 5.49: Performance of the selected algorithms when used in combination with only homeowner or only not homeowners.

Term	LR	NN	DT	GB
Original	0.77	0.81	0.77	0.81
12	0.67	0.66	0.62	0.55
36	0.77	0.80	0.77	0.80
60	0.75	0.78	0.73	0.78

The next change in the data set is made based on if the borrower is a homeowner. This is a binary feature, so it is either true of false. Of all the loans, 40,065 are taken out by homeowners. The remaining 44,899 belong to borrowers that do not own a home.

Just as with the previous splits, the selected algorithms have been used in combination with both of the newly created data sets. The result of doing so is given in Table 5.51. Logistic Regression shows no difference in performance at all. The other algorithms only shows minor differences. The differences in performance are such that it cannot be concluded that one of the categories leads to a better performance.

The last modification that will be analyzed is to winsorize the data set. As mentioned before, this is the process of limiting extreme values to a certain percentile.

TABLE 5.50: Amount of samples in the homeowner and not homeowner category.

Homeowner	n
Yes	40,065
No	44,899

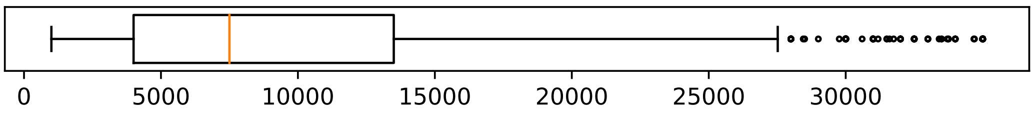
TABLE 5.51: Performance of the selected algorithms when used in combination with only homeowner or only not homeowners.

Homeowner	LR	NN	DT	GB
Original	0.77	0.81	0.77	0.81
Yes	0.77	0.80	0.76	0.80
No	0.78	0.81	0.76	0.81

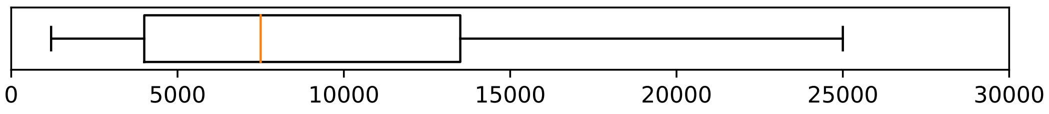
Two different limits will be used for the extreme values. The first test will limit values to be between the 1st and 99th percentile. The second winsorization limits values below the 5th and above the 95th percentile. An example of applying winsorization to the 'LoanOriginalAmount' is shown in Figure 5.19.

FIGURE 5.19: Effect on the 'LoanOriginalAmount' feature when winsorization is applied at the 1st and 99th percentile.

(A) Original



(B) Winsorized



The results of applying winsorization are given in Table 5.52. They show that it probably has a small positive influence on Logistic Regression, the AUC increases from 0.77 to 0.78. The other algorithms are not positively influenced, some show a small decrease in performance.

TABLE 5.52: Performance of the selected algorithm when used with a winsorized data set.

Homeowner	LR	NN	DT	GB
Original	0.77	0.81	0.77	0.81
Winsorized (98%)	0.78	0.81	0.77	0.80
Winsorized (90%)	0.78	0.80	0.77	0.81

Chapter 6

Conclusions

This research project has shed a light on the performance of different machine learning algorithms when used to predict defaults. This is done by applying the models to two data sets consisting of loans, and trying to predict which of the loans will default. The first data set consists of 30,000 Taiwanese credit lines. The second data set is comprised of 85,964 (after preparation) peer to peer loans that have been originated using the Prosper platform.

Model performance

The primary part of these conclusions is about the performance of the different machine learning algorithms used in this research project. This is where the main question will be answered by examining the performances, which are given in Table 6.1. In this research project, the Area Under the Curve has been used as the primary indicator. To make a better comparison possible, the precision and recall have also been added to the table with performances. Logistic Regression is used as a benchmark model due to its widespread usage, fast computation and descent results.

TABLE 6.1: Performance of the different algorithms when trained and validated on the original data set.

	Taiwan			Prosper		
Algorithm	AUC	Precision	Recall	AUC	Precision	Recall
Logistic regression	0.77	0.48	0.58	0.77	0.16	0.73
Neural network	0.78	0.45	0.62	0.81	0.17	0.77
Naive Bayes	0.74	0.40	0.63	0.66	0.10	0.77
k Nearest neighbors	0.76	0.44	0.61	0.75	0.15	0.66
Decision tree	0.76	0.45	0.60	0.77	0.15	0.78
Random forest	0.77	0.44	0.63	0.80	0.15	0.83
ADABoost	0.77	0.41	0.67	0.80	0.17	0.73
Gradient boosting	0.78	0.45	0.61	0.81	0.16	0.83
Support Vector Machine	0.76	0.47	0.59	-	-	-

In the first data set the performances are all quite similar. Naive Bayes has the lowest performance with an AUC of 0.74, the highest AUC of 0.78 is shared by two algorithms, Neural Network and Gradient Boosting. Aside from the Naive Bayes algorithm, the performances are so close together that it is difficult to draw conclusions based solely on the AUC.

The performances achieved with the second data set show more variation. Naive Bayes has again the lowest performance with an AUC of 0.66. The highest performance is also achieved by the same two algorithms, Neural Network and Gradient Boosting. The difference between those two best performing algorithms and Logistic Regression is 0.04, a lot more as with the previous data set. Relative to the other data set, precision is low and recall is high for each of the algorithms. The highest precision is achieved by the Neural Network and ADABoost algorithms. The highest recall by Random Forest and Gradient Boosting.

When one takes the explainability into account, the view on which of the algorithms is most suitable changes. From the algorithms used in this research project, Decision Trees are the most explainable. Decision Trees can be easily graphed and their classifications explained. For the Taiwan data set, the Decision Tree has a decent AUC which is 0.02 below the highest value, in the Prosper data set it is 0.04 below the highest value. Clearly a choice between explainability and performance has to be made. Depending on the situation it might be worth it to settle for a lower performance and be able to exactly explain how a decision is made. This might for example be required when working with a regulator.

Looking at the precision and recall instead of the AUC changes some of the findings. When the goal of the algorithm is maximize recall, ADABoost is the best choice for the Taiwan data set. However, it has one of the lower recalls in the Prosper data set. For that data set it is a better choice to use Random Forest or Gradient Boosting instead. Precision is highest with Logistic Regression on the Taiwan data set. With the Prosper data set it is highest for Neural Network and ADABoost.

For the Prosper data set, it is interesting to see how good the models are in predicting which loans don't go into default. The best algorithms, Random Forest and Gradient Boosting, have a negative predictive value of 97,9% and even the worst algorithm, Naive Bayes, has 95,9%. This makes it interesting to use the algorithms not as default predictors, but to define a group of low risk loans.

Finally, both of the data sets have been split according to certain characteristics, for example the education level of the borrower. The goal of such a modification is to find differences in performances on specific parts of the data set. With the Taiwan data set it was found that nearly every applied split lead to a decrease in performance, in two cases the performance improved by 0.01. The Prosper data set did show a more positive result, by splitting it by the listing category of the loan, the performance of one of the new data sets increased with up to 0.03, an AUC of 0.83 was reached. All other splits did not lead to an increase in performance. The last modification was winsorization of the numerical features. This led to a decrease in performance with the Taiwan data set and small increase of the performance of Logistic Regression with the second data set.

Based on the above mentioned findings the following conclusions are drawn. Both Gradient Boosting and Neural Network are the best performing algorithms based. They both have the highest AUC for both of the data sets. The disadvantages of those algorithms is that they tend to become a black box, making it difficult to explain why a classification is made.

Resampling

The data sets used in this research project where imbalanced, less than half of the loans went into default. The Taiwan data set had 22% of the samples in the minority class, the Prosper data set even less, only 8%. This can pose a problem for machine learning and thus different methods to remove the imbalance have been applied and analyzed. The following methods have been examined: random undersampling, random oversampling, SMOTE (regular, borderline1 and borderline2) and ADASYN (2, 5 and 10 neighbors).

For the Taiwan data set, the influence of applying resampling to the data set was small. Only the Support Vector Machine algorithm had a substantial increase in performance. The other algorithms had either no substantial change or a decrease in performance. Especially the more complex methods, SMOTE and ADASYN, had a negative impact more often than not. These results of resampling or not great, this is probably caused by the fact that the data set does not have a large imbalance.

The results of applying resampling to the Prosper data set are quite different. With this data set in its original balance, some of the algorithms would classify all samples as not defaulting. By doing so they would seem to perform well due the high accuracy of 92%. However, it is not what is expected from a predictive model. Therefore, resampling was more important than with the Taiwan data set. After applying the different methods, random undersampling, random oversampling, SMOTE (regular) and ADASYN (5 neighbors), the performance of some of the algorithms increased substantially. The largest increase was reached with a combination of k Nearest Neighbors and random undersampling, its AUC increased with 0.09. Each of the resample methods lead to the algorithm no longer classifying all sample to a single class. Random undersampling has the beneficial effect to decrease computation times and this is an interesting method. Random oversampling on the other hand, can be used when a data set is relatively small.

After using different resample methods on both data set, it can be concluded that it can have a positive effect, but it should not be applied without reservation. Especially the more complex methods, SMOTE and ADASYN, are practically never the best choice in these two data sets. Random undersampling is the algorithm that has been used the most in this research project, mainly because its positive effect on the performance, but also due to its effect on the computation times.

Chapter 7

Limitations and further research

As is customary in a scientific project, the limitations will be discussed and several possibilities for interesting further research will be discussed.

The first limitation is caused by the method used to determine the parameters of the different machine learning algorithms. A step-by-step process has been used, in most cases, to determine the parameters. The disadvantage of using such a method, is that interaction effects between the parameters are not fully taken into account. Not taking those effects into accounts means that the performance might be lower in comparison with a training process in which they are take into account.

Another limitation is the result of the in some cases long computation times. Especially the Support Vector Machine and the Neural Network tend to become indefeasible with certain configurations in combination with the hardware used in this research project. During the model training with the Taiwan data set, the Support Vector Machine algorithms has not been used with the settings originally planned. Nine of the experiments took so long that they where stopped. This has an influence on how likely a better performing combination of parameters could have been found. Since the Prosper data set is substantially larger than the Taiwan data set and computation time for Support Vector Machines is at least quadratic, the computation time was even more of a problem. After running the algorithm for several hours in a low computation time setting, no noticeable progress was made and thus the choice has been made to not take Support Vector Machines into account with the Prosper data set.

Recently, a lot of discussion is happening about the ethics behind machine learning. This discussion is closely related with a lot of the algorithms working like a 'black-box'. It is hard to determine why and how a certain decision is made. This makes it prone to making unwanted decision, for example based on ethnicity. This is a subject that has to be thoroughly researched before this type of model can be implemented.

Now that the limitations have been discussed, the recommendations for further research are discussed. The first recommendation is to look into the the possibility of combining different algorithms together. This does happen for example in the Random Forest classifier. But this recommendation is based on more high level combinations. In this research project, the Neural Network and Gradient Boosting classifier are two of the top performing algorithms. Is it possible to further increase the classifying accuracy by combining those algorithms? Further research can be conducted into combining different types of algorithm into an ensemble.

Another interesting recommendation is to look into the explainability of the different algorithms. In this thesis it has been mentioned that Decision Tree might be favorable due to its decision being explainable. For most of the other used algorithms, it is more difficult to explain why a classification is made. This has several disadvantages, for example that a regulator will not approve a 'black-box' algorithm or that it is not possible to tell a client why its loan is declined. Further research into the algorithms might result in variations that have a better explainability or methods that make the current models more intuitive to understand.

Bibliography

Abellán, Joaquín and Carlos J. Mantas (June 2014). "Improving experimental studies about ensembles of classifiers for bankruptcy prediction and credit scoring". In: *Expert Systems with Applications* 41.8, pp. 3825–3830. ISSN: 0957-4174. DOI: `10.1016/j.eswa.2013.12.003`. URL: `http://www.sciencedirect.com/science/article/pii/S0957417413009676` (visited on 11/23/2017).

Alaraj, Maher, Maysam Abbod, and Ziad Hunaiti (Jan. 2014). "Evaluating Consumer Loans Using Neural Networks Ensembles". en. In: International Institute of Engineers. ISBN: 978-93-82242-63-5. DOI: `10.15242/IIE.E0114084`. URL: `http://iieng.org/siteadmin/upload/6437E0114084.pdf` (visited on 04/05/2018).

Ali, Kamal M. and Michael J. Pazzani (1996). "Error reduction through learning multiple descriptions". In: *Machine Learning* 24.3, pp. 173–202.

Alpaydın, Ethem (2010). *Introduction to machine learning*. eng. 2. ed. Adaptive computation and machine learning. OCLC: 699516236. Cambridge, Mass.: MIT Press. ISBN: 978-0-262-01243-0.

Bentley, Jon Louis (Sept. 1975). "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9, pp. 509–517. ISSN: 0001-0782. DOI: `10.1145/361002.361007`. URL: `http://doi.acm.org/10.1145/361002.361007` (visited on 01/30/2018).

Bhatia, Nitin (2010). "Survey of nearest neighbor techniques". In: *arXiv preprint arXiv:1007.0085*.

Blagus, Rok and Lara Lusa (Mar. 2013). "SMOTE for high-dimensional class-imbalanced data". In: *BMC Bioinformatics* 14, p. 106. ISSN: 1471-2105. DOI: `10.1186/1471-2105-14-106`. URL: `https://doi.org/10.1186/1471-2105-14-106` (visited on 01/02/2018).

Blöchlinger, Andreas and Markus Leippold (2006). "Economic benefit of powerful credit scoring". In: *Journal of Banking & Finance* 30.3, pp. 851–873.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pp. 5–32.

Chawla, Nitesh V., Nathalie Japkowicz, and Aleksander Kotcz (2004). "Special issue on learning from imbalanced data sets". In: *ACM Sigkdd Explorations Newsletter* 6.1, pp. 1–6.

Chawla, Nitesh V. et al. (2002). "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16, pp. 321–357.

Chintala, Soumith (Jan. 2015). *FAIR open sources deep-learning modules for Torch.* (Visited on 05/04/2018).

Cox, D. R. (1958). "The Regression Analysis of Binary Sequences". en. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 20. URL: `http://www.jstor.org/stable/2983890`.

Dean, Jeff (Nov. 2015). *TensorFlow - Google's latest machine learning system, open sourced for everyone*. (Visited on 05/04/2018).

Dumitrescu, Elena et al. (2018). "Machine Learning for Credit Scoring: Improving Logistic Regression with Non Linear Decision Tree Effects". PhD thesis. Paris Nanterre University, University of Orleans.

Engel, Giora (Nov. 2017). *3 Flavors of Machine Learning: Who, What & Where*. URL: `https://www.darkreading.com/threat-intelligence/3-flavors-of-`

machine-learning--who-what-and-where/a/d-id/1324278 (visited on 11/21/2017).

Fawcett, Tom (June 2006). "An introduction to ROC analysis". en. In: *Pattern Recognition Letters* 27.8, pp. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010. URL: http://linkinghub.elsevier.com/retrieve/pii/S016786550500303X (visited on 12/12/2017).

Freund, Yoav and Robert E Schapire (Aug. 1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1, pp. 119–139. ISSN: 0022-0000. DOI: 10.1006/jcss.1997.1504. URL: http://www.sciencedirect.com/science/article/pii/S002200009791504X (visited on 01/30/2018).

Friedman, Jerome H (1999). "Greedy Function Approximation: a Gradient Boosting Machine". en. In: *The Annals of Statistics*, p. 44.

Gartner (Aug. 2016). *Gartner's 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage*. URL: https://www.gartner.com/newsroom/id/3412017 (visited on 11/28/2017).

Ghahramani, Zoubin (2004). "Unsupervised Learning". en. In: *Advanced Lectures on Machine Learning*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 72–112. ISBN: 978-3-540-23122-6 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_5. URL: https://link.springer.com/chapter/10.1007/978-3-540-28650-9_5 (visited on 11/21/2017).

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Hahnloser, Richard H. R. et al. (June 2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". En. In: *Nature* 405.6789, p. 947. ISSN: 1476-4687. DOI: 10.1038/35016072. URL: https://www.nature.com/articles/35016072 (visited on 01/10/2018).

Han, Hui, Wen-Yuan Wang, and Bing-Huan Mao (2005). "Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning". In: *Advances in intelligent computing*, pp. 878–887.

Hancock, Thomas et al. (1996). "Lower bounds on learning decision lists and trees". In: *Information and Computation* 126.2, pp. 114–122.

Harris, Terry (Feb. 2015). "Credit scoring using the clustered support vector machine". en. In: *Expert Systems with Applications* 42.2, pp. 741–750. ISSN: 09574174. DOI: 10.1016/j.eswa.2014.08.029. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957417414005119 (visited on 11/23/2017).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning - Data Mining, Inference and prediction*. en. Springer. URL: //www.springer.com/us/book/9780387848570 (visited on 02/01/2018).

He, Haibo et al. (2008). "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, pp. 1322–1328.

Huang, Cheng-Lung, Mu-Chen Chen, and Chieh-Jen Wang (Nov. 2007). "Credit scoring with a data mining approach based on support vector machines". en. In: *Expert Systems with Applications* 33.4, pp. 847–856. ISSN: 09574174. DOI: 10.1016/j.eswa.2006.07.007. URL: http://linkinghub.elsevier.com/retrieve/pii/S095741740600217X (visited on 11/23/2017).

Hull, John (2015). *Risk management and financial institutions*. eng. Fourth edition. Wiley finance series. Hoboken, New Jersey: Wiley. ISBN: 978-1-118-95594-9 978-1-118-95596-3 978-1-118-95595-6.

Jordan, M. I. and T. M. Mitchell (July 2015). "Machine learning: Trends, perspectives, and prospects". en. In: *Science* 349.6245, pp. 255–260. ISSN: 0036-8075, 1095-9203. DOI: `10.1126/science.aaa8415`. URL: `http://science.sciencemag.org/content/349/6245/255` (visited on 11/23/2017).

Khandani, Amir E., Adlar J. Kim, and Andrew W. Lo (Nov. 2010). "Consumer credit-risk models via machine-learning algorithms". en. In: *Journal of Banking & Finance* 34.11, pp. 2767–2787. ISSN: 03784266. DOI: `10.1016/j.jbankfin.2010.06.001`. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0378426610002372` (visited on 04/05/2018).

Kingma, Diederik P. et al. (2014). "Semi-supervised learning with deep generative models". In: *Advances in Neural Information Processing Systems*, pp. 3581–3589.

Langley, Pat (1995). "Applications of Machine Learning and Rule Induction". en. In: p. 20.

Larose, Daniel T. (2005). *Discovering knowledge in data: an introduction to data mining*. eng. OCLC: 265634768. Hoboken, NJ: Wiley-Interscience. ISBN: 978-0-471-66657-8.

LeCun, Yann et al. (1998). "Efficient BackProp". en. In: *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 9–50. ISBN: 978-3-540-65311-0 978-3-540-49430-0. DOI: `10.1007/3-540-49430-8_2`. URL: `https://link.springer.com/chapter/10.1007/3-540-49430-8_2` (visited on 01/09/2018).

Maimon, Oded and Shahar Cohen (2009). "A Review of Reinforcement Learning Methods". en. In: *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, pp. 401–417. ISBN: 978-0-387-09822-7 978-0-387-09823-4. DOI: `10.1007/978-0-387-09823-4_20`. URL: `https://link.springer.com/chapter/10.1007/978-0-387-09823-4_20` (visited on 11/21/2017).

Maimon, Oded and Lior Rokach (2005). "Introduction to supervised methods". In: *Data Mining and Knowledge Discovery Handbook*. Springer, pp. 149–164.

Mitchell, Tom M. (Mar. 1997). *Machine learning*. eng. International ed., [Reprint.] McGraw-Hill series in computer science. OCLC: 846270128. New York, NY: McGraw-Hill. ISBN: 978-0-07-042807-2.

Omohundro, Stephen M. (1989). *Five balltree construction algorithms*. International Computer Science Institute Berkeley.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12.Oct, pp. 2825–2830.

Prosper (Nov. 2017). *Download Analysis Data - Prosper*. en. URL: `https://www.prosper.com/invest/download.aspx` (visited on 11/27/2017).

Ramachandran, Prajit, Barret Zoph, and Quoc Le (2017). "Searching for Activation Functions". en. In: URL: `https://research.google.com/pubs/pub46503.html` (visited on 01/10/2018).

Rodríguez, Juan, Aritz Pérez, and J.A. Lozano (Apr. 2010). *Sensitivity Analysis of k-Fold Cross Validation in Prediction Error Estimation*. Vol. 32.

Rokach, Lior (2009). "Ensemble Methods in Supervised Learning". en. In: *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, pp. 959–979. ISBN: 978-0-387-09822-7 978-0-387-09823-4. DOI: `10.1007/978-0-387-09823-4_50`. URL: `https://link.springer.com/chapter/10.1007/978-0-387-09823-4_50` (visited on 01/30/2018).

Rokach, Lior and Oded Maimon (2009). "Classification Trees". en. In: *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, pp. 149–174. ISBN: 978-0-387-09822-7 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_9. URL: https://link.springer.com/chapter/10.1007/978-0-387-09823-4_9 (visited on 11/21/2017).

Samuel, Arthur L (1959). "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3, pp. 210–229.

Schmidhuber, Juergen (Jan. 2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61. arXiv: 1404.7828, pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://arxiv.org/abs/1404.7828 (visited on 11/20/2017).

Shin, Kyung-Shik, Taik Soo Lee, and Hyun-jung Kim (Jan. 2005). "An application of support vector machines in bankruptcy prediction model". en. In: *Expert Systems with Applications* 28.1, pp. 127–135. ISSN: 09574174. DOI: 10.1016/j.eswa.2004.08.009. URL: http://linkinghub.elsevier.com/retrieve/pii/S095741740400096X (visited on 04/05/2018).

Shmilovici, Armin (2009). "Support vector machines". In: *Data mining and knowledge discovery handbook*. Springer, pp. 231–247.

Thomas, George (Nov. 2015). *Microsoft open sources Distributed Machine Learning Toolkit for more efficient big data research*. (Visited on 05/04/2018).

Turing, Alan M. (1950). "Computing machinery and intelligence". In: *Mind* 59.236, pp. 433–460.

Yeh, I-Cheng and Che-hui Lien (Mar. 2009). "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients". en. In: *Expert Systems with Applications* 36.2, pp. 2473–2480. ISSN: 09574174. DOI: 10.1016/j.eswa.2007.12.020. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957417407006719 (visited on 12/19/2017).

Zhang, G. Peter (2009). "Neural Networks For Data Mining". en. In: *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, pp. 419–444. ISBN: 978-0-387-09822-7 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_21. URL: https://link.springer.com/chapter/10.1007/978-0-387-09823-4_21 (visited on 12/21/2017).

Zhang, G Peter and Min Qi (2002). "Predicting consumer retail sales using neural networks". In: *Neural Networks in Business: Techniques and Applications, Smith, K. and Gupta, J. eds. Hershey: Idea Group Publishing*, pp. 26–40.

Zhu, Xiaojin, Zoubin Ghahramani, and John D. Lafferty (2003). "Semi-supervised learning using gaussian fields and harmonic functions". In: *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919.

Appendix A

Prosper data set summary

TABLE A.1: Summary of the categorical 'EmploymentStatus' feature.

Meaning	n	Percentage
Employed	67,306	79.22%
Full-time	8,030	9.45%
Not employed	648	0.76%
Other	3,805	4.48%
Part-time	262	0.31%
Retired	371	0.44%
Self-employed	4,542	5.35%

TABLE A.2: Summary of the categorical 'ListingCategory' feature.

	Meaning	n	Percentage
1	Debt Consolidation	53,246	62.67%
2	Home Improvement	6,812	21.48%
3	Business	5,315	21.34%
4	Personal Loan	0	0.00%
5	Student Use	280	1.43%
6	Auto	2,244	11.62%
7	Other	9,242	54.15%
8	Baby and Adoption	199	2.54%
9	Boat	85	1.11%
10	Cosmetic Procedure	91	1.21%
11	Engagement Ring	217	2.91%
12	Green Loans	59	0.82%
13	Household Expenses	1,996	27.82%
14	Large Purchases	876	16.92%
15	Medical/Dental	1,522	35.38%
16	Motorcycle	304	10.94%
17	RV	52	2.10%
18	Taxes	885	36.51%
19	Vacation	768	49.90%
20	Wedding Loans	771	100.00%

TABLE A.3: Summary of the categorical 'BorrowerState' feature.

State	n	Percentage	State	n	Percentage	State	n	Percentage
0	166	0.20%	16	888	1.05%	32	3375	3.97%
1	1187	1.40%	17	842	0.99%	33	733	0.86%
2	768	0.90%	18	1832	2.16%	34	1215	1.43%
3	1359	1.60%	19	2239	2.64%	35	2679	3.15%
4	10777	12.68%	20	2625	3.09%	36	410	0.48%
5	1734	2.04%	21	1721	2.03%	37	997	1.17%
6	1498	1.76%	22	1797	2.12%	38	189	0.22%
7	330	0.39%	23	675	0.79%	39	1540	1.81%
8	268	0.32%	24	221	0.26%	40	5652	6.65%
9	5408	6.37%	25	2442	2.87%	41	521	0.61%
10	3352	3.95%	26	556	0.65%	42	2779	3.27%
11	343	0.40%	27	443	0.52%	43	172	0.20%
12	403	0.47%	28	2728	3.21%	44	2159	2.54%
13	4270	5.03%	29	331	0.39%	45	1522	1.79%
14	1654	1.95%	30	1024	1.21%	46	310	0.36%
15	855	1.01%	31	5852	6.89%	47	123	0.14%

TABLE A.4: Summary of the categorical 'Occupation' feature.

Meaning	n	Percentage	Meaning	n	Percentage
Accountant/CPA	2,577	3.03%	Nurse's Aide	382	0.45%
Administrative As...	2,713	3.19%	Other	22,674	26.69%
Analyst	2,743	3.23%	Pharmacist	225	0.26%
Architect	149	0.18%	Pilot - Private/Com...	154	0.18%
Attorney	866	1.02%	Police Officer/Corr...	1,278	1.50%
Biologist	95	0.11%	Postal Service	486	0.57%
Bus Driver	251	0.30%	Principal	262	0.31%
Car Dealer	143	0.17%	Professional	10,561	12.43%
Chemist	109	0.13%	Professor	452	0.53%
Civil Service	1,141	1.34%	Psychologist	118	0.14%
Clergy	157	0.18%	Realtor	253	0.30%
Clerical	2,116	2.49%	Religious	93	0.11%
Computer Progr...	3,245	3.82%	Retail Management	2,003	2.36%
Construction	1,326	1.56%	Sales - Commission	2,350	2.77%
Dentist	56	0.07%	Sales - Retail	2,032	2.39%
Doctor	393	0.46%	Scientist	294	0.35%
Engineer - Chemical	176	0.21%	Skilled Labor	2,183	2.57%
Engineer - Electrical	900	1.06%	Social Worker	575	0.68%
Engineer - Mechanical	1,138	1.34%	Student - Freshman	17	0.02%
Executive	3,472	4.09%	Student - Graduate	114	0.13%
Fireman	320	0.38%	Student - Junior	27	0.03%
Flight Attendant	87	0.10%	Student - Senior	71	0.08%
Food Service	839	0.99%	Student - Sophomore	16	0.02%
Food Service Man...	1,008	1.19%	Community College	10	0.01%
Homemaker	57	0.07%	Technical School	2	0.00%
Investor	201	0.24%	Teacher	2,893	3.40%
Judge	22	0.03%	Teacher's Aide	200	0.24%
Laborer	1,217	1.43%	Tradesman - Carpenter	85	0.10%
Landscaping	172	0.20%	Tradesman - Electrician	386	0.45%
Medical Technician	891	1.05%	Tradesman - Mechanic	797	0.94%
Military Enlisted	825	0.97%	Tradesman - Plumber	74	0.09%
Military Officer	253	0.30%	Truck Driver	1,369	1.61%
Nurse (LPN)	415	0.49%	Waiter/Waitress	294	0.35%
Nurse (RN)	2,161	2.54%			

TABLE A.5: Statistical summary of the features in the Prosper data set, that are used for machine learning.

Feature	Mean	Standard deviation	min	25%	50%	75%	max
Term	42.48	11.64	12	36	36	60	60
EmploymentStatusDuration	103	97.04	0	30	74	147	755
IsBorrowerHomeowner	0.53	0.5	0	0	1	1	1
CreditScoreRangeLower	699.41	47.11	600	660	700	720	880
CreditScoreRangeUpper	718.41	47.11	619	679	719	739	899
CurrentCreditLines	10.51	5.32	0	7	10	13	59
OpenCreditLines	9.53	4.93	0	6	9	12	54
TotalCreditLinespast7years	27.65	13.26	2	18	26	35	125
OpenRevolvingAccounts	7.39	4.52	0	4	7	10	50
OpenRevolvingMonthlyPayment	430.53	425.74	0	156	311	563	13,765
InquiriesLast6Months	0.96	1.4	0	0	0	1	27
TotalInquiries	4.29	3.83	0	2	3	6	78
CurrentDelinquencies	0.32	1.11	0	0	0	0	51
AmountDelinquent	950.19	7,415.7	0	0	0	0	463,881
DelinquenciesLast7Years	3.66	9.35	0	0	0	2	99
PublicRecordsLast10Years	0.28	0.65	0	0	0	0	38
PublicRecordsLast12Months	0.01	0.13	0	0	0	0	20
RevolvingCreditBalance	17,931.86	31,351.77	0	3,822	9,321	20,334	999,165
BankcardUtilization	0.56	0.3	0	0.33	0.6	0.83	2.5
AvailableBankcardCredit	11,405.43	18,611.33	0	1,148	4,579	13,919	498,374

Feature	Mean	Standard deviation	min	25%	50%	75%	max
TotalTrades	23.92	11.61	1	15	23	31	122
TradesNeverDelinquent	0.91	0.12	0.08	0.85	0.95	1	1
TradesOpenedLast6Months	0.73	0.99	0	0	0	1	20
DebtToIncomeRatio	0.26	0.3	0	0.16	0.24	0.31	10.01
IncomeVerifiable	0.91	0.28	0	1	1	1	1
StatedMonthlyIncome	5,930.14	8,235.35	0	3,433.33	5,000	7,083.33	1,750,002.92
TotalProsperLoans	0.34	0.73	0	0	0	0	8
TotalProsperPaymentsBilled	5.68	14.03	0	0	0	0	141
OnTimeProsperPayments	5.51	13.67	0	0	0	0	141
ProsperPaymentsLessThanOneMonthLate	0.15	1.27	0	0	0	0	42
ProsperPaymentsOneMonthPlusLate	0.01	0.28	0	0	0	0	21
ProsperPrincipalBorrowed	2043.5	5,193.7	0	0	0	0	72,499
ProsperPrincipalOutstanding	680.31	2,214.26	0	0	0	0	23,450.95
ScorexChangeAtTimeOfListing	-0.89	22.53	-209	0	0	0	286
LoanOriginalAmount	9,076.71	6,287.17	1,000	4,000	7,500	13,500	35,000
MonthlyLoanPayment	291.73	186.67	0	157.18	251.76	388.28	2,251.51
PercentFunded	1	0.02	0.7	1	1	1	1.01
Recommendations	0.02	0.19	0	0	0	0	19
InvestmentFromFriendsCount	0.01	0.11	0	0	0	0	9
InvestmentFromFriendsAmount	4.39	118.38	0	0	0	0	11,000
Investors	68.41	95.3	1	1	32	97	1,189

TABLE A.6: Description of the original features in the Prosper data set.

Variable	Description
ListingKey	Unique key for each listing.
ListingNumber	Unique public key for each listing.
ListingCreationDate	The date the listing was created.
CreditGrade	The Credit rating that was assigned at the time the listing went live. (Only pre-2009)
Term	The length of the loan expressed in months.
LoanStatus	The current status of the loan.
ClosedDate	Date on which the loan is closed.
BorrowerAPR	The Borrower's Annual Percentage Rate (APR) for the loan.
BorrowerRate	The Borrower's interest rate for this loan.
LenderYield	The Lender yield on the loan, equal to the interest rate on the loan less the servicing fee.
EstimatedEffectiveYield	Estimated effective yield. (Only after 2009)
EstimatedLoss	Estimated loss is the estimated principal loss on charge-offs. Applicable for loans originated after July 2009.
EstimatedReturn	Estimated return is the difference between the Estimated Effective Yield and the Estimated Loss Rate. (Only after 2009)
ProsperRating (numeric)	The Prosper Rating assigned at the time the listing was created. (Only after 2009)
ProsperRating (Alpha)	The Prosper Rating assigned at the time the listing was created. (Only after 2009)
ProsperScore	A custom risk score built using historical Prosper data. (Only after 2009)
ListingCategory	The category of the listing that the borrower selected when posting their listing.
BorrowerState	The two letter abbreviation of the state of the address of the borrower at the time the Listing was created.
Occupation	The Occupation selected by the Borrower at the time they created the listing.
EmploymentStatus	The employment status of the borrower at the time they posted the listing.
EmploymentStatusDuration	The length in months of the employment status at the time the listing was created.
IsBorrowerHomeowner	A Borrower is a homeowner if they have a mortgage or provide documentation confirming they are a homeowner.
CurrentlyInGroup	Specifies whether or not the Borrower was in a group at the time the listing was created.
GroupKey	The Key of the group in which the Borrower is a member of.
DateCreditPulled	The date the credit profile was pulled.
CreditScoreRangeLower	The lower value representing the range of the borrower's credit score as provided by a consumer credit rating agency.
CreditScoreRangeUpper	The upper value representing the range of the borrower's credit score as provided by a consumer credit rating agency.
FirstRecordedCreditLine	The date the first credit line was opened.
CurrentCreditLines	Number of current credit lines at the time the credit profile was pulled.

Variable	Description
OpenCreditLines	Number of open credit lines at the time the credit profile was pulled.
TotalCreditLinespast7years	Number of credit lines in the past seven years at the time the credit profile was pulled.
OpenRevolvingAccounts	Number of open revolving accounts at the time the credit profile was pulled.
OpenRevolvingMonthlyPayment	Monthly payment on revolving accounts at the time the credit profile was pulled.
InquiriesLast6Months	Number of inquiries in the past six months at the time the credit profile was pulled.
TotalInquiries	Total number of inquiries at the time the credit profile was pulled.
CurrentDelinquencies	Number of accounts delinquent at the time the credit profile was pulled.
AmountDelinquent	Dollars delinquent at the time the credit profile was pulled.
DelinquenciesLast7Years	Number of delinquencies in the past 7 years at the time the credit profile was pulled.
PublicRecordsLast10Years	Number of public records in the past 10 years at the time the credit profile was pulled.
PublicRecordsLast12Months	Number of public records in the past 12 months at the time the credit profile was pulled.
RevolvingCreditBalance	Dollars of revolving credit at the time the credit profile was pulled.
BankcardUtilization	The percentage of available revolving credit that is utilized at the time the credit profile was pulled.
AvailableBankcardCredit	The total available credit via bank card at the time the credit profile was pulled.
TotalTrades	Number of trade lines ever opened at the time the credit profile was pulled.
TradesNeverDelinquent	Number of trades that have never been delinquent at the time the credit profile was pulled.
TradesOpenedLast6Months	Number of trades opened in the last 6 months at the time the credit profile was pulled.
DebtToIncomeRatio	The debt to income ratio of the borrower at the time the credit profile was pulled. This value is capped at 10.01
IncomeRange	The income range of the borrower at the time the listing was created.
IncomeVerifiable	The borrower indicated they have the required documentation to support their income.
StatedMonthlyIncome	The monthly income the borrower stated at the time the listing was created.
LoanKey	Unique key for each loan. This is the same key that is used in the API.
TotalProsperLoans	Number of Prosper loans the borrower has at the time they created this listing.
TotalProsperPaymentsBilled	Number of on time payments the borrower made on Prosper loans at the time they created this listing.
OnTimeProsperPayments	Number of on time payments the borrower had made on Prosper loans at the time they created this listing.
ProsperPaymentsLessThanOneMonthLate	Number of payments the borrower made on Prosper loans that were less than one month late.
ProsperPaymentsOneMonthPlusLate	Number of payments the borrower made on previous loans that were greater than one month late.
ProsperPrincipalBorrowed	Total principal borrowed on Prosper loans at the time the listing was created.
ProsperPrincipalOutstanding	Principal outstanding on Prosper loans at the time the listing was created.
ScorexChangeAtTimeOfListing	Credit score change relative to the last Prosper loan at the time the credit profile was pulled.

Variable	Description
LoanCurrentDaysDelinquent	The number of days delinquent.
LoanFirstDefaultedCycleNumber	The cycle the loan was charged off. If the loan has not charged off the value will be null.
LoanMonthsSinceOrigination	Number of months since the loan originated.
LoanNumber	Unique numeric value associated with the loan.
LoanOriginalAmount	The origination amount of the loan.
LoanOriginationDate	The date the loan was originated.
LoanOriginationQuarter	The quarter in which the loan was originated.
MemberKey	The unique key that is associated with the borrower.
MonthlyLoanPayment	The scheduled monthly loan payment.
LP_CustomerPayments	Pre charge-off cumulative gross payments made by the borrower on the loan.
LP_CustomerPrincipalPayments	Pre charge-off cumulative principal payments made by the borrower on the loan.
LP_InterestandFees	Pre charge-off cumulative interest and fees paid by the borrower.
LP_ServiceFees	Cumulative service fees paid by the investors who have invested in the loan.
LP_CollectionFees	Cumulative collection fees paid by the investors who have invested in the loan.
LP_GrossPrincipalLoss	The gross charged off amount of the loan.
LP_NetPrincipalLoss	The principal that remains uncollected after any recoveries.
LP_NonPrincipalRecoverypayments	The interest and fee component of any recovery payments.
PercentFunded	Percent the listing was funded.
Recommendations	Number of recommendations the borrower had at the time the listing was created.
InvestmentFromFriendsCount	Number of friends that made an investment in the loan.
InvestmentFromFriendsAmount	Dollar amount of investments that were made by friends.
Investors	The number of investors that funded the loan.

Appendix B

Experiment results

Credit data - Taiwan

TABLE B.1: Performance of two different ratio's, both have LIMIT_BAL as denominator. These ratio's are applied to the Taiwan data set.

	BILL_AMNT		PAY_AMNT		Both	
Original	Keep	Delete	Keep	Delete	Keep	Delete
Logistic Regression	0.72	0.72	0.72	0.72	0.72	0.72
Neural network	0.77	0.77	0.77	0.77	0.77	0.77
Naïve bayes	0.72	0.72	0.73	0.73	0.72	0.72
k Nearest neighbors	0.71	0.71	0.70	0.70	0.70	0.70
Decision tree	0.73	0.73	0.73	0.72	0.73	0.73
Random forest	0.73	0.73	0.73	0.73	0.73	0.73
ADABoost	0.75	0.75	0.75	0.75	0.75	0.75
Gradient boosting	0.76	0.76	0.76	0.76	0.76	0.76

TABLE B.2: The average AUC of 25 folds for different algorithms and resample methods when used on the Taiwan data set.

Algorithm	None	Random undersampling	Random oversampling	Smote regular	Smote borderline1	Smote borderline2	ADASYN_2	ADASYN_5	ADASYN_10
Logistic Regression	0.72	0.72	0.72	0.73	0.73	0.73	0.73	0.73	0.73
Neural Network	0.77	0.77	0.76	0.74	0.73	0.74	0.74	0.74	0.73
Naïve Bayes	0.73	0.73	0.73	0.73	0.73	0.72	0.73	0.73	0.73
k Nearest Neighbors	0.71	0.71	0.68	0.69	0.68	0.68	0.68	0.68	0.68
Decision Tree	0.73	0.74	0.74	0.74	0.73	0.73	0.73	0.73	0.73
Random Forest	0.73	0.73	0.73	0.72	0.73	0.72	0.72	0.72	0.72
ADABoost	0.75	0.75	0.75	0.75	0.74	0.73	0.74	0.74	0.74
Gradient Boosting	0.77	0.76	0.77	0.76	0.76	0.76	0.76	0.76	0.76

TABLE B.3: Confusion matrices of applying the algorithms on the Taiwan test set with the in Chapter 5 determined parameters.

(A) Logistic Regression

	Predicted positive	Predicted negative
Positive	921	670
Negative	1,007	4,902

(B) Neural Network

	Predicted positive	Predicted negative
Positive	986	605
Negative	1,193	4,716

(C) Naïve Bayes

	Predicted positive	Predicted negative
Positive	995	596
Negative	1,465	4,444

(D) k Nearest Neighbors

	Predicted positive	Predicted negative
Positive	960	631
Negative	1,236	4,673

(E) Decision Tree

	Predicted positive	Predicted negative
Positive	953	638
Negative	1,156	4,753

(F) Random Forest

	Predicted positive	Predicted negative
Positive	1,008	583
Negative	1,274	4,635

(G) ADABoost

	Predicted positive	Predicted negative
Positive	1,067	524
Negative	1,512	4,397

(H) Gradient Boosting

	Predicted positive	Predicted negative
Positive	973	618
Negative	1,180	4,729

Peer2peer loans - Prosper

TABLE B.4: Confusion matrices of applying the algorithms on the Prosper test set with the in Chapter 5 determined parameters.

(A) Logistic Regression

	Predicted positive	Predicted negative
Positive	1,141	426
Negative	6,188	13,486

(B) Neural Network

	Predicted positive	Predicted negative
Positive	1,203	364
Negative	5,747	13,927

(C) Naïve Bayes

	Predicted positive	Predicted negative
Positive	1,213	354
Negative	11,113	8,561

(D) k Nearest Neighbors

	Predicted positive	Predicted negative
Positive	1,024	525
Negative	5,871	13,803

(E) Decision Tree

	Predicted positive	Predicted negative
Positive	1,223	344
Negative	6,956	12,718

(F) Random Forest

	Predicted positive	Predicted negative
Positive	1,304	263
Negative	7,412	12,262

(G) ADABoost

	Predicted positive	Predicted negative
Positive	1,148	419
Negative	5,699	13,975

(H) Gradient Boosting

	Predicted positive	Predicted negative
Positive	1,292	275
Negative	6,852	12,822