Handelshøyskolen BI

GRA 19703 Master Thesis

Thesis Master of Science 100% - W

Predefinert informasjon

Startdato:	09-01-2023 09:00 CET	**Termin:**	202310
Sluttdato:	03-07-2023 12:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	T		
Flowkode:	202310||11184||IN00||W||T		
Intern sensor:	(Anonymisert)		

Deltaker

Navn:	Alexander Noreddin Habibi

Informasjon fra deltaker

Tittel *:	Predicting Default Loans using Machine Learning
Navn på veileder *:	Emil Stoltenberg

Inneholder besvarelsen konfidensielt materiale?:	Nei	**Kan besvarelsen offentliggjøres?:**	Ja

Gruppe

Gruppenavn:	(Anonymisert)
Gruppenummer:	287
Andre medlemmer i gruppen:	Deltakeren har innlevert i en enkeltmannsgruppe

Handelshøyskolen BI

Oslo, Fall 2023

Predicting Default Loans using Machine Learning

How effective are machine learning algorithms in predicting default loans?

Alexander Habibi

Supervisor: Emil Stoltenberg

Master Thesis

Major in Business Analytics

Handelshøyskolen BI

Acknowledgments

I now present to you my master's thesis. This thesis marks the end of my master's degree in Business Analytics at BI Handelshøyskolen, Oslo.

Working on this thesis has been demanding but also exciting and educational. Throughout this semester, I have learned how to process and model machine learning algorithms to predict default in loans. These methods have been, and are still in high development, which makes the subject more attractive and challenging, which suited me perfectly.

I want to express my gratitude and appreciation to my supervisor for this master's thesis Emil Stoltenberg. Your guidance has been valuable, and you have been helpful when needed and have pushed me through the semester.

I also would like to thank my father, brother, and girlfriend, who is always there for me and is a pillar of support.

Abstract

This thesis aims to determine whether machine learning algorithms effectively predict default loans and which machine learning algorithms are better performers than others. The research started by gathering information about which machine learning methods there are, implementing and processing their algorithms, and determining their performances.

For this thesis, I have used a dataset from a Taiwanese credit lending company consisting of 30,000 credit lenders, whereas the defaulters of this dataset are known. The choice has been made to train, test and validate six different machine learning algorithms, determine their performances, and gather helpful information on whether they are accurate in their predictions or flawed.

The main research question for this thesis is:

How effective are machine learning algorithms in predicting defaults in loans?

Some model performance measures have been used to determine the machine learning algorithms' performances. The Area Under the Curve has been set as a primary model performance measure. In this classifier measure, a score between 0 and 1 is calculated. While a classifier with 1 AUC is the perfect model, a classifier with 0.5 AUC is as good as a random guessing one. There are also three other measurements to determine the final models: Recall, Precision, and Accuracy. The table below showcases the performance of the six algorithms after training and validation on the original dataset.

Table 1. Summary of the performance of all the different algorithms when trained and validated on the original dataset.

Algorithm	AUC	Recall	Precision	Accuracy
Logistic Regression	0.762	0.34	0.67	0.816
Neural Network	0.790	0.34	0.68	0.820
k Nearest Neighbors	0.760	0.33	0.63	0.815
Decision Tree	0.784	0.38	0.60	0.819
Random Forest	0.766	0.30	0.67	0.815
XGBoost	0.783	0.37	0.64	0.822

From the performances, it has been determined that machine learning algorithms are highly effective in predicting default in loans. The average accuracy between the algorithms is 81.7%, and AUC is relatively high. The highest-performing algorithms are the Neural Network and the Decision Tree, having an AUC score of 0.790 and 0.784, respectively.

Contents

1. Introduction

1.1 Project Context

This master thesis aims to explore and analyze the potential of alternative and diverse machine learning approaches to accurately estimate the default risk on individual customer credit loans/mortgages. A bank's primary function is to loan money to individuals and businesses, but in doing so, they risk not being repaid the total amount of the principal. Accurate predictions are essential for any credit lending company, as they must strike a delicate balance between preventing losses from customers who will default and ensuring they are reasonable, thereby missing potential income. In recent years, corporations have been able to access more data about customer behavior than ever before, prompting the consideration of alternative machine learning techniques to the traditional logistic regression model. Logistic regression has long been favored for its predictive accuracy and interpretability. Still, this new wealth of information begs the question of whether other approaches could yield even better results.

So, what is modeling the probability of default or credit scoring? A simple explanation is that statistical models are used to transform relevant data into numerical measures that guide credit decisions. (Anderson 2007). As mentioned, credit scoring has been used for centuries. Still, advances in computing power have enabled further development of more subjective credit score techniques and taking a higher advantage of the now more significant amount of available data. There are a few different ways of credit scoring, and the most valuable and informative way to distinguish between them is to separate them into application scores and behavior scores. Behavior score deals more with predicting or scoring current customers and their likelihood of defaulting. It is used to guide decision-making, such as evaluating risk and over-limit management. On the other hand, application scores are used for newly customer appliances and for their likelihood to be profitable customers. It is used when checking the history of the new applicant. The borrower's income, previous history with other banks, loan size, and such. (SAS 2019). This thesis will focus on creating models for predicting default probability based on an existing dataset of customers who already have received loans, in other words, applicant scores.

Credit scoring tools are based on statistical and operational research techniques and are some of the most successful and profitable applications of statistical theory over the last 20 years. These predictive model techniques can be separated into parametric and non-parametric models. While the common factor of parametric models is that particular critical assumptions are made when they are used in relation to the data, non-parametric models require few, if any assumptions at all. (Anderson 2007). As mentioned, banks commonly employ logistic regression models when calculating the likelihood of default, which is a parametric model.

While machine learning models are primarily associated with non-parametric models and are increasingly considered and used by financial institutions, these models have some drawbacks. The two disadvantages are a tendency to overfit and the lack of transparency. However, all modern machine learning tools allow the data to be split into training sets, validation, and testing splits, reducing the tendency to overfit. A strength of the parametric logistic regression is that the model is highly interpretable, which is why it is used in such a high grade by the banks, as the regulators require them to provide evidence of its interpretability. However, some non-parametric machine learning models have high interpretability, such as the decision tree models.

1.2 Problem Description

An exciting approach when experimenting with credit risks and default loans is looking at the possibility of improvements for the bank by applying modern machine learning techniques. Several scientific papers are written about expected benefits in default prediction when using machine learning. Still, equality between most of them is the limitation of the number of algorithms compared. These papers show that machine learning may lead to higher accuracy in predicting default. However, two problems might arise; one must be cautious in comparing results from different papers, as error rates might be defined differently. Secondly, nearly all papers use different data sets, making comparison of results difficult. This presents an opportunity to evaluate the effectiveness of various algorithms using the same datasets.

1.3 Research Objective

Having identified and described a problem, the objective of this research project can be established. The aim of this thesis is to gain a deeper understanding of the predictive

performance of various machine learning algorithms when applied to loan default prediction. The research will involve implementing multiple machine learning algorithms for classifying samples and using them to predict defaults on loans. The performance of these algorithms will be compared and evaluated to determine the most suitable for this specific task. The dataset used is based on credit lenders in Taiwan, and in this research project, the lenders who went into default are known. The last part of the objective is comparing the algorithms' performances and determining which are most suitable for predicting default loans.

For this thesis to reach its objective, some research questions must be answered. I have taken the liberty to separate these into one main research question and some sub-questions. I believe and hope that the sub-questions will answer the main question.

1.3.1 Main Research Question

The main research question is defined as:

How effective are machine learning algorithms in predicting default in loans?

1.3.2 Sub-Questions

What are the appropriate machine learning algorithms for classification?

Firstly, I must determine which machine learning methods to use. When researching and discovering what machine learning and algorithms exists, I will have made out a selection of which to use in this research project.

How do I accurately measure the different machine learning algorithms performance?

As mentioned, evaluating and comparing of the different algorithm's effectiveness is a goal for this research project. If I want to do it objectively, I must accurately measure the predictive performance.

Which machine learning algorithm has the highest performance?

Lastly, I want to determine the performance of the different methods used and determine which performed better than the other. This last question is set to be the last sub-question but ultimately is the second main question for this research project.

To answer the main question and determine the efficiency and which machine learning method is preferably the best, all sub-questions are required to be answered. This leads to highly prepared datasets, which leads to the algorithm's possibility to configure high performances.

2. Literature review

Machine learning has not become a standalone practice for credit scoring and granting loans. Various credit scoring methods have been employed depending on the type of loans and the type of customer. The FICO score was first introduced by the Fair Isaac Corporation (FICO) in the United States in the late ‘50s and early ‘60s, and by the late ‘80s, FICO scores were first introduced. It quickly became the industry standard for assessing credit risk, and by 2000, over 100 billion FICO scores were sold. The FICO score is calculated by evaluating five factors which is payment history, amounts owned, length of credit history, new credit, and types of credit in use. The weighting of each factor is different, and the outcome is a credit score that typically ranges from 350 to 850, with a higher score indicating a lower credit risk and a lower score indicating a higher risk. (FICO, 2022) The score range is defined as:

300-579: Poor

580-669: Fair

670-739: Good

740-799: Very Good

800-850: Exceptional

Research on predicting default loans or credit scoring using machine learning is not new and can be dated back to before the year 2000. Langley & Simon (1995) is an example of the already mentioned application of machine learning and making credit decisions. Since the mid- ‘90s, a lot has happened with machine learning, especially as new technology has emerged. More and more research is being performed on predicting defaults and credit scoring using machine learning.

Crook, Edelman & Thomas (2007) provide a representation of different publications investigating predictive modeling with machine learning techniques. (a list of these will be included in the reference list.) They provide ten different research where they look at the relative predictive accuracy of different classifiers using credit application data and the percentage correctly classified (Crook et al. Table 2, 2007). When comparing the various machine learning methods, it is notably the neural network that is found to

perform best. Notably, these comparisons only include the relevant techniques, Logistic Regression, Decision trees, k-nearest neighbors, and neural networks. As well as a short list of comparisons, one also must be cautious when comparing results between different study papers because of the error rates in different classifiers and data usage.

Hand & Henley (1996) did a similar study on "statistical classification methods in consumer credit scoring'' and used neural networks and logistic regression to predict credit scoring. In this study, they concluded that both methods were good performers but also concluded that there are no best overall methods, and the method used is highly dependent on the dataset used. Hand (2006) further suggests that the distinctions in predicting capabilities among the classifiers may be overstated and that one classifier that works well on one data set may not work as well on new data. He also suggests that a classifier's aim should be profit maximizing. This means using a profit matrix may yield different results compared with the more commonly used ROC AUC statistic in research.

As mentioned in the problem description, several papers have been written on the possibilities of using machine learning in default predictions. Two examples are Alaraj, Abbod & Hunaiti (2014) and Khandani, Kim & Lo (2010). Many of these state that machine learning can lead to high accuracy of default predictions, but as stated, they are limited in the number of compared models, and using different data sets to evaluate results, which again makes the results incomparable. In their 2014 study, Alaraj et al. utilized a neural network for default prediction, but never made a comparison to other techniques. On the other hand, in Kahandi et al. study from 2010, they applied machine learning algorithms to predict defaults and compare them. However, their comparison was limited to only three algorithms. In contrast, this research project aims to comprehensively compare various methods I will use.

3. Theory/ Classification

3.1 Definition of credit risk

As mentioned before, the goal of this research project is to assess whether machine learning can be used to make better decisions on loans. The reasoning for chasing this goal is to minimize the credit risk the credit lender is exposed to.

Credit risk is the risk of loss that a business or individual may incur due to a borrower defaulting on payments owed. Credit risk arises when a borrower expects to use future cash flows to pay a current debt. It is the risk that a borrower will not pay a loan or other debt obligation or that the debt obligation will not be paid in full or on time. As in any financial business, the lender's purpose, is to minimize costs and maximize revenue. The primary goal of credit lending businesses is to maximize loan volume and interest revenue while minimizing defaults on loans. (Brock & Eichler, 2022)

3.2 Definition of Default

The Basel Framework is a comprehensive set of guidelines established by the Basel Committee on Banking Supervision (BCBS) for the regulation of banks. The BCBS, a global standard-setting body, has been adopted by its members, who have agreed to fully implement these standards and apply them to internationally active banks in their respective jurisdictions. (BIS, 2023). The Basel Committee has released three accords, referred to as The Basel Accords I, II, and III. With the probability of default modeling, the first and most important consideration to make when defining defaults is the regulatory requirements, and it is stated under the second accord that a default is considered to have occurred when either or both of the two following events have taken place:

(1) "The bank considers that the obligor is unlikely to pay its credit obligations to the banking group in full, without recourse by the bank to actions such as realizing security (if held)"

(2) "The obligor is past due more than 90 days on any material credit obligation to the banking group. Overdrafts will be considered as being past due once the customer has breached an advised limit or been advised of a limit smaller than current outstanding" (BIS, 2023)

3.3 Machine Learning

Machine learning is a field of computer science focused on giving computers the capabilities to learn. The development of algorithms and statistical models enables computer systems to learn and improve from data without being explicitly programmed. The main goal of machine learning is creating algorithms and training computer systems to analyze and interpret patterns in data automatically, further make decisions and predictions, and based on input received, adapt its behavior.

Machine learning is a branch of Artificial Intelligence, broadly defined as a machine's capability to imitate intelligent human behavior. Artificial intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems. (Sara Brown, 2021)

Tom Mitchell, a computer scientist and professor at Carnegie Mellon University and a prominent figure that has contributed significantly to the field of machine learning, defines machine learning as:

"*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)

We can interpret Tom Mitchell's definition of Machine learning and apply it to the goal of credit scoring: A credit scoring model requires information on historical credit data (E) to train a model into learning to predict creditworthiness (T) and the performance measure (P) if the performance of predicting creditworthiness improves with the analysis of historical credit data.

3.3.2 Learning Methods

This section describes the three primary categories machine learning models fall into. The three categories are often referred to as machine learning problems: supervised, semi-supervised, and unsupervised machine learning.

- Supervised learning

We define supervised learning by its use of labeled datasets to train algorithms to predict outcomes accurately or classify data. Supervised learning are methods that attempt to discover the relationship between a target attribute (dependent variable) and input attributes (independent variable) (Maimon & Rokach, 2010). When input data is fed to the model, it adjusts its weight until fitted appropriately. This happens as part of a cross-validation process to ensure that the model avoids under or overfitting (IBM, n.d.).

There are many benefits to using supervised machine learning, and it can help organizations solve various real-world problems at scale. An example is classifying and separating spam mail from another folder in your inbox. Supervised machine learning uses a trained set of many cases consisting of features with value and resulting class. Another example is a dataset describing the color, top speed, and capacity of the trunks of a certain number of cars and the classification of the cars being a family car or not. The supervised machine learning algorithm then uses the data to infer functions relating the features of the car being a family car or not. (Alpaydin, 2010)

Reinforcement machine learning is very similar to supervised learning, but the difference is that the reinforcement machine learning algorithms are not trained using sample data. This type of model learns as it goes, by trial and error, receiving feedback in the form of rewards or penalties based on its actions. So when a sequence of successful outcomes occurs, these will be reinforced to develop the best recommendation for a specific problem. (Alpaydin, 2010)

- Unsupervised learning

Unsupervised machine learning uses the algorithms to analyze and cluster unlabeled datasets. So, the algorithm is trained using just an input set, and desired results or feedback is given. The algorithm then finds the structure in the data by itself. Due to the data generated being so high, humans will not be able to analyze all the data, so an algorithm based on unsupervised learning finds hidden patterns or data groupings without the need of human intervention (IBM n.d.). This method is excellent for exploratory data analysis, and examples where it can be utilized are with customer segmentation and sales strategies. Using unsupervised learning, a seller/company can detect sales patterns and image recognitions, and behavior-based network security detection.

- Semi-supervised learning

Semi-supervised learning considers the classification problem only when smaller labeled subsets of the observation have corresponding class labels. When training, it uses the smaller subset to guide classification and feature extraction from a large and unlabeled dataset. It uses labeled data to data to ground predictions and unlabeled data to learn shapes of the larger data distributions. Semi-supervised machine learning lies between the two, supervised and unsupervised learning, and using semi-supervised learning; one can achieve results with only fractions of the labeled data, which can save valuable time and money.

3.4 Model Performance Measures

In this next section of the thesis, we will look at and discuss different statistics to assess the performance of the different models. When evaluating a model's performance, there are many different approaches and methods to consider, and each might differ in performance depending on the problem. As the goal of this thesis is to compare and find which methods of machine learning perform better, and a criterion on which methods used to compare them is required.

3.4.1 Confusion Matrix

Confusion matrix is a method to visualize the accuracy using a table. To easily explain the confusion matrix, we assume a classifier that classifies instances as positive and negative. We then have four fields to be calculated in the matrix: true positive, true negative, false positive, and false negative. In a binary classification problem, a confusion matrix is typically a 2x2 table with four cells representing different prediction outcomes:

Table 3.1 Confusion Matrix

	Predicted Positive	**Predicted Negative**
Positive	True Positive (Tp)	False Negative (Fn)
Negative	False Positive (Fp)	True Negative (Tn)

True Positive shows the models correctly predicted positive instances as positive.
True Negative shows the models correctly predicted negative instances as negative.
False Positive shows the models incorrectly predicted negative instances as positive.
False Negative shows the models incorrectly predicted positive instances as negative.

There are several advantages that the confusion matrix offers. Firstly, it is efficient to use and requires little computational power. Secondly, because of its simplicity, it is easy not just for professionals to understand but also for individuals with different backgrounds. Additionally, in my case, utilizing the confusion matrix along with accuracy, can help with the verification of the model's effectiveness in predicting default and non-defaulters. Which provides a comprehensive assessment of the model's predictive performance across different classes.

3.4.2 Accuracy

The possibility of calculating other statistical measures occurs when using the values found from a confusion matrix. The first one we will look at is the accuracy. A confusion matrix provides a clear and structured representation of the model's prediction and can help in decision-making based on the specific requirements of the classification task. To find the accuracy of the correct percentage of predictions made by the model, we can use the following formula:

$$Accuracy = \frac{Tp + Tn}{Tp + Tn + Fn + Fp}$$

In accuracy, a random classifier will get, on average, half of the classification correctly, which means that values above 0.5 indicate that the model has a higher accuracy as random guessing. A perfect prediction has an accuracy of 1.0.

The accuracy paradox is a phenomenon referring to an accuracy being high but misleading when evaluating the performance of a predictive model. The accuracy paradox states that a model with overall high accuracy, in reality, performs poorly in predictions. Imagine having two models that predict bankruptcy.

Below, in Table 3.2, the confusion matrices for the two models are showcased. The first model, let's call it A, predicts 200 out of 500 cases of bankruptcy, and the accuracy of the model is as follows:

$$\frac{200 + 8.300}{500 + 8.600} = 0.934$$

Model B, however, does not have the ability to detect any bankruptcy at all, and the accuracy of the model is as follows:

$$\frac{0 + 8.500}{500 + 8.500} = 0.944$$

From the performances, model B has the highest accuracy, even though it does not have any predictive powers. This disadvantage, the accuracy paradox, is very important to have in mind, not to determine and evaluate models that might score highly but, in reality, have less predictive powers.

Table 3.2 Confusion Matrices of Model A and B

Model A

	Predicted	
	Positive	**Negative**
Positive	200	100
Negative	500	8300

Model B

	Predicted	
	Positive	**Negative**
Positive	0	500
Negative	0	8500

It is crucial to be aware of the accuracy paradox when evaluating model performance. To only rely on accuracy as the sole evaluation metric might be misleading. Several other statistics have been developed to quantify model performance, and to gain a more comprehensive understanding of model effectiveness, some of them will be analyzed. The first ones we will look at are Precision and Recall.

3.4.3 Precision and Recall

Precision: It is a performance metric that measures the proportion of correctly predicted instances of positive instances and indicates how reliable the model is when it identifies positive instances. A high precision indicates a low rate of false positives. It is measures with:

$$Precision = \frac{Tp}{Tp + Fp}$$

Recall: Is a metric that measures the proportion of correctly predicted positive instances out of all actual positive instances. A high recall indicates a low rate of false negatives. It is measured with:

$$Recall = \frac{Tp}{Tp + Fn}$$

Both metrics provide valuable insights into different aspects of model performance, and they are often used together. They are often given together or combined in making another statistic, called F1-score.

3.4.4 F1-Score

The F1-score is, as mentioned, a combined metric between precision and recall, making a balanced measure of a model's performance. The F1-score is equal to the harmonic mean of the two, providing an evaluation of the model's ability to correctly predict positive instances while minimizing the false negatives and false positives. Naturally, a negative with the F1-score is that it does not consider true negatives. A high f1-score indicates better overall model performance and is measured by:

$$F1 - Score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

3.4.5 Receiver operating characteristics Area under the curve (ROC AUC)

A receiver operating characteristics (ROC) graph is a technique for visualizing classifiers based on their performance and has long been used in signal detection theory to depict the tradeoff between hit rates and false alarm rates of classifiers (Fawcett, 2006)

The ROC graph is created by plotting the true positive rate against the false positive rate. The true positive rate, also called hit rate and as earlier talked about, recall, is calculated by taking the positives correctly classified divided by total positives. The false positive rate, also called false alarm rate, is calculated by dividing the negatives incorrectly classified with total negatives (Fawcett, 2006).

$$Tp\ Rate = \frac{Tp}{P}$$

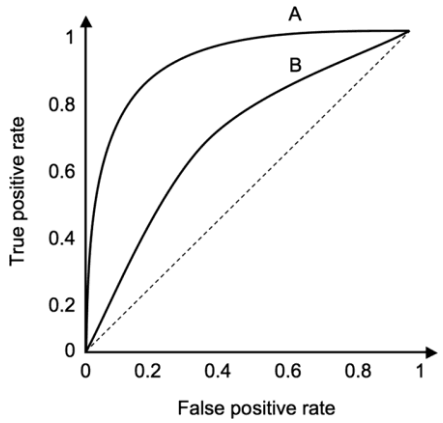
$$Fp\ Rate = \frac{Fp}{N}$$

In classifiers, scores between 0 and 1 are calculated, and a threshold must be selected to determine the boundary between positive and negative classifications. The score, denoted as x, can be considered a sample from a continuous random distribution X. An instance

is classified as positive when x > T, where T represents the chosen threshold. Varying the threshold will give different rates of true positives and false positives.

Figure 3.1 provides three examples of a ROC curve, with one model being random and two models having predictive capabilities. In the two models (A) and (B), which perform better than the random guessing one, the true positive rate will be higher than the false positive rate, and the ROC curve will be above the stippled, diagonal random model. This is easier explained with an example. Let's say we want to classify some instances as either positive or negative. When choosing a random fraction, R, as our threshold, then the R proportion of the instances that are true positives will be identified as positive, and the same for R proportion of instances that are true negatives will be identified as negatives. This means that the chosen threshold decides how accurately the positive and negative instances are classified.

Figure 3.1: One graph containing the ROC of three random models, with one random and two better performing.



An important summary statistic is the Area Under the ROC curve (AUC). As the name implies, this is simply the area under a classifier's curve expressed as a fraction of the unit square. Its value ranges from 0 to 1. Though a ROC curve provides more information than its area, the AUC is useful when a single number is needed to summarize performance or when nothing is known about the operating conditions (Fawcett & Provost, 2015).

For a random classifier, the AUC is precisely 0.5, while models that are better than a random classifier will have an AUC above 0.5, with the perfect classifier having an AUC score of 1. If the AUC is below 0.5, the model performs worse than a random guesser and

most probably is not able to handle the data information properly. In short, the AUC is a measure that summarizes the overall performance of a classifier model by measuring the area under the ROC curve with one single number. (Hullermeier & Vanderlooy, 2008)

3.5 Cross-Validation

When training a model to make predictions, it needs an estimate of how accurate it will predict in practice. Cross-validation is a technique to assess the performance and generalization ability of a model. In cross-validation, the available data is partitioned into subsets, also called training sets, used to train the model and to a set where the model is tested and evaluated, also called test sets or validation sets. This practice is repeated several times, usually using different partitions, relying on multiple iterations to reduce variability and to obtain more reliable estimations of the model's performance.

3.5.1 k-fold Cross Validation

In k-fold cross validation the data is divided into k equally sized subsets. Of all the k subsets, only one is retained to be used as a test set, while all k-1 remaining sets are used as training sets. This process is repeated k times, with each subsets being used as a test or validation set once. The results of the performance are then averaged over all iterations to obtain an accurate estimation of the model performance. K-fold are widely used and are one of the more commonly used cross-validation methods. However, the k-fold cross-validation has an upwards bias, which can be biased when applied to a large dataset. This can be negligible in leave-one-out cross-validation, but it sometimes cannot be neglected in 5-fold or 10-fold cross-validation, which is the favorite from a computational standpoint. (Fushiki Tadayoshi, 2011)

3.5.2 Holdout Method

The Holdout validation method splits the dataset into two sets, one training set, and one test set. The model is trained using the training set and validated using the test set. The majority of data is assigned to the training set, and splitting is typically ranged from 30:70 to 10:90. The holdout method is much simpler and less intensive than the k-fold cross-validation, making it suitable when using large datasets. However, because this method uses a single train/test split, it may yield high variance in the performance estimate,

especially in smaller datasets, making the method susceptible to random variations (Zhang, 2009).

3.5.3 Repeated random sub-sampling validation

The repeated random sub-sampling or also known as Monte Carlo cross validation works just like the holdout method but differs in that it splits the dataset into training and test sets multiple times. In each repetition, the dataset is split into random training and test sets. The resulting performance is then averaged over all the iterations (Berrar, 2019). The negatives with this method are that some samples of the data may never be used as validation, as well as some might be selected multiple times. This method can also result in higher variance compared to the k-fold cross-validation, because each random partition of the data might yield different subsets of the data.

3.6 Model Description

In this chapter of the thesis, I will look at the machine learning algorithms that are most common for credit scoring, as well as the algorithms I will use in this thesis. All the chosen models are non-parametric except the logistic regression model. I will in this chapter give an explanation of the classification techniques, and present some parameters to be tuned and determined.

3.6.1 Logistic Regression

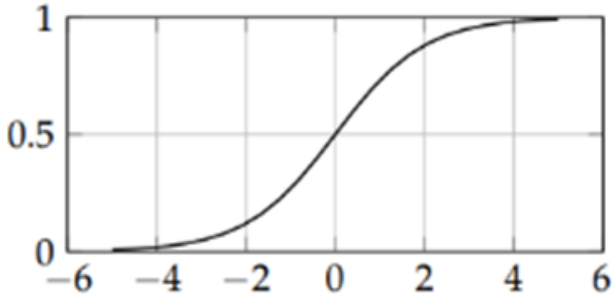
Logistic regression models the probabilities for classification problems with two possible outcomes. It is an extension of the linear regression model for classification problems. The problem when using the linear regression is that there are no upper or lower limits for the response variable. The linear model extrapolates and gives values below zero and higher than one. (Christoph Molnar, 2022). In this case, there would be a problem if a client of the said bank had somewhat of an extreme balance value; the value predicted could result in being outside of [0, 1]. The main goal is to predict the probability of default, which would create a problem, as predicting probabilities would have to be in range between 1 and 0.

The logistic function is given by:

$$p(x) = \frac{e^{\beta_1+\beta_1*X_1+\cdots+\beta_k*X_k}}{e^{\beta_1+\beta_1*X_1+\cdots+\beta_k*X_k}+1}$$

Where p is the probability that a case is in a particular category, e is the base of natural logarithms, $\beta 0$ is the constant of the equation, and bk are the coefficients of the predictor variables.

Figure 3.2: Standard Logistic Regression



Logistic regression uses maximum likelihood to estimate parameters in the model, and by transforming the logistic function given, we can find the odds for any given data points. The form of the logistic regression equation is:

$$Log\left(\frac{\mathrm{p}(\bar{\mathrm{X}})}{1-\mathrm{p}(\bar{\mathrm{X}})}\right) = \beta_0 + \beta_1 * X_1 + \cdots + \beta_k * X_k$$

We find the log odds from the parameters on the right side of the formula. For our case, the maximum likelihood estimates the parameters of each observation and it will predict a value closer to 0 for those who most likely would not default and closer to 1 for those likely to default.

Lastly, the parameters that will be analyzed and determined for the Logistic regression are the solver, regularization term, and regularization. The solver parameters are the one that determines the coefficients. This method is an iterative process in which, in each iteration, the coefficients change in a way, so the maximum likelihood is improved. The solver parameter consists of two methods to fit the model. These models are Liblinear and Saga. The liblinear is used for the coordination of descent algorithms to find the utmost suitable values for the coefficient. The Saga solver, however, uses a stochastic average gradient descent and is usually faster when applied to larger datasets.

The regularization term, often called C, is a term that penalizes complex algorithms and is favorable for simpler models. Further, there are two types of regularization methods commonly used, which are called L1 and L2. The L1 is favorable for sparse models, and models with a large fraction of coefficients equal to zero. The L1 is the better choice if the dataset consists of highly correlated features. The L2 is used when a sparse model is not suitable. While the L1 suits better for correlated features, the L2 simply shrinks the coefficients of all correlated features.

3.6.2 Neural Network

The core neural network algorithm is the neuron (a unit). Many neurons arranged in an interconnected structure make up a neural network, with each neuron linking to the inputs and outputs of other neurons. Neural networks can be visually represented as neurons distributed between layers, where the first and the last layers play an important role. The first layer, called the input layer, picks up the features from each data example processed by the network. The last layer, called the output layer, releases the results. They receive input, compute a weighted value, sum them, and use an activation function to evaluate the result, which transforms it in a non-linear way (Mueller & Massaron, 2021).

Figure 3.3: Simplified Neural Network Illustration.

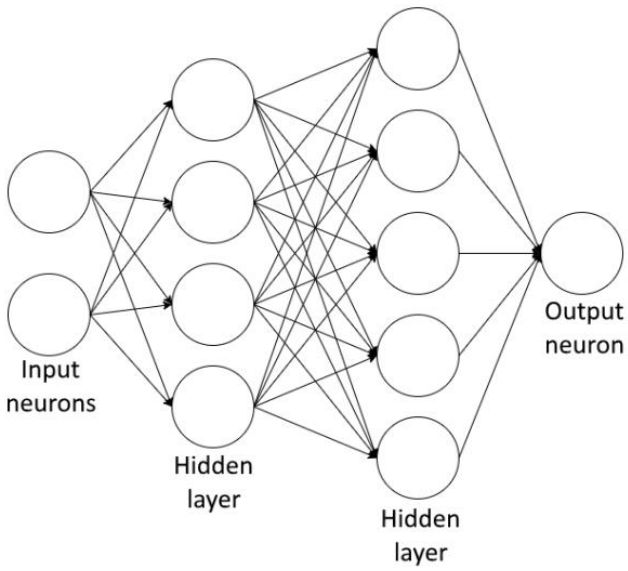


Figure 3.3 above is an example of a schematic overview of a neural network. The two nodes on the beginning left are the input nodes, and our predictors are at this stage. The

last nodes on the right are the output nodes, and the nodes in the middle are the hidden layers. (Du & Swamy, 2013)

Typically, one, two, or three layers are used, with neurons split equally between them. The number of neurons and layers is a part of the parameters that must be specified before training. Even though neural networks are much used in practice for the modeling of probability for defaulting, due to the great recognition of patterns, this machine learning technique is one of the most prone to overfitting.

For Neural Network algorithms, it is important to tune the parameters well to avoid overfitting. However, there is another problem endured if not tuning the parameters well, and that is the Black Box problem. Concerns about the neural network and other machine learning algorithms and their trustworthiness are increasing. With automation of routine decisions joined with highly complex information architecture and the usage of algorithmic systems of deep learning, which can remain hidden from human comprehension. This problem is commonly called the Black box problem. (Eschenback, 2021)

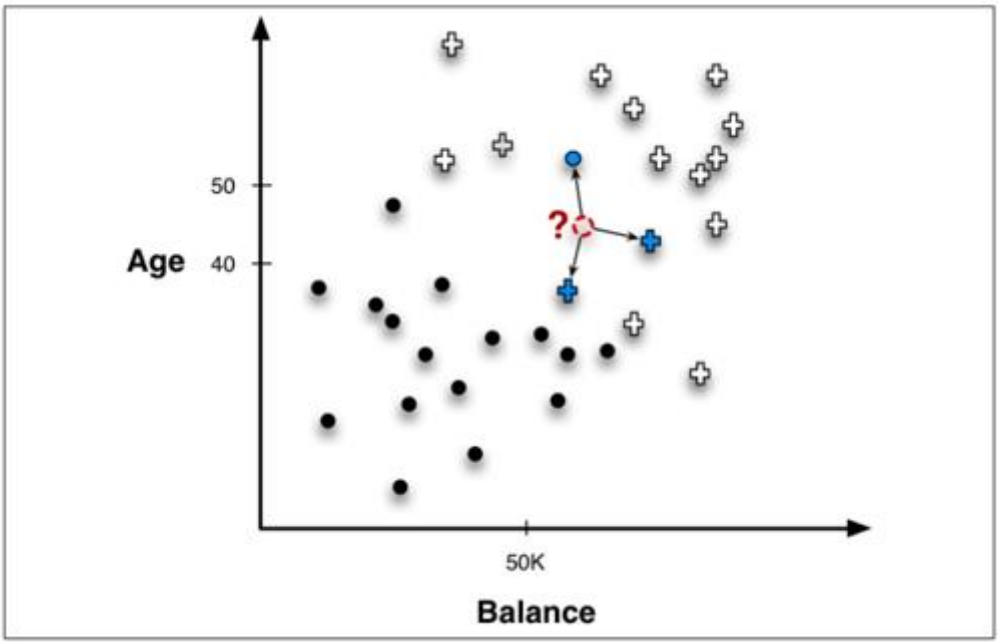
The parameters to be tuned and determined in Neural Network algorithms are usually the Solver, Hidden layers, Activation functions, Learning rates, Tolerance, Regularization term, and Maximum Iterations. The Hidden layers are the size and number of layers in between the input and output layers. The Activation function is the function shape which is used to transform the signals of the input of the neurons. Just as in the case of the Logistic regression model, the regularization term's goal is to penalize complex models, mainly used for countering overfitting the model. Learning rates are the step size where weights are changed. The last two parameters, tolerance, and maximum iterations are parameters working to stop the algorithm. Tolerance stops the training if the decrease in loss during training is smaller than the threshold, and the maximum number of iterations is the number of iterations to reach convergence. Not being highly careful when tuning and determining these parameters will most likely end in flaws in the algorithm.

3.6.3 k Nearest Neighbor

The k Nearest Neighbor classification method base a classification on the k samples closest to the instance that has to be classified. What defines a near neighbor is part of the parameters, and to give an example; we define a number of neighbors of a test

subject to be five. Our test subject's pattern closely resembles three of the five neighbors, and the new observation is then assigned to those three. Their responses are yes, yes, and no. If we then take a majority vote on these values, we predict yes for our test subject. We further assign a score to it, as scores give more information than just a yes or a no-decision. If we then score the yes decision to be = 1 and average the scores from our test subjects' neighbors, we average the score for David to be 2/3. In an actual experiment, using more than three nearest neighbors to compute the probability estimates strengthens the estimates. (Provost & Fawcett. 2013).

Figure 3.4: Example from Provost & Fawcett. 2013)

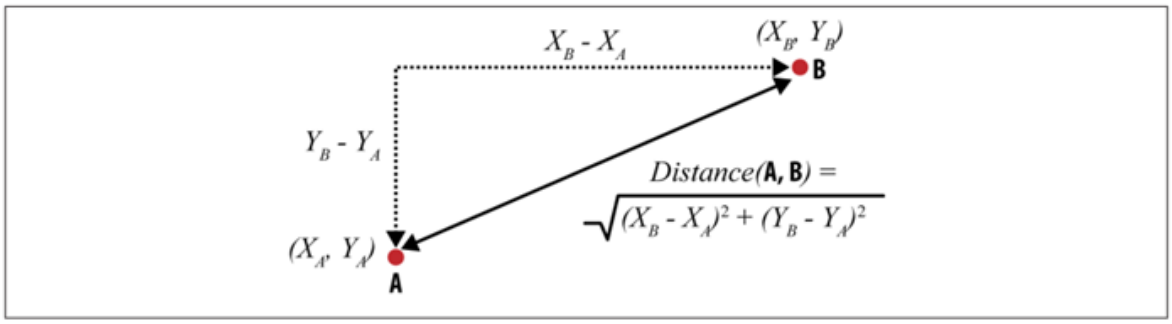


Our test subject is labeled with a question mark and would, in this case, be classified as a + because its nearest neighbors weighted average is a + as well. The downside of using this method is that it is a ''lazy'' algorithm. Meaning because the method bases a classification on the k-samples closest to the instance that has to be classified, the method does not induce a model, it just stores instances of the training data, making it biased again depending on the k-nearest neighbors. The smaller the sample size, the more effective and accurate this method would be.

Lastly, the distance to a neighbor is defined as the Euclidean distance. The Euclidean distance is one of the most common geometric ways to measure distance, and with this,

we can compute the overall distance by computing the distances of the individual features in our setting. (Provost & Fawcett 2013) The Euclidean distance is defined by:

Figure 3.5: The Educlian Distance



There are three commonly used computation methods for the k Nearest Neighbor algorithm: Brute Force, k-d Tree, and Ball Tree. The first one, Brute Force, is the simplest computation method for this algorithm. The brute force method calculates the distances between the points in the data and uses the distance to determine which ones are the closest. However, this method is usually better for smaller datasets and is prone to be infeasible when datasets increase in size.

The second computation method, the k-d tree, is a more efficient method for larger datasets compared to the brute force method. The k-d tree divides the data into two parts, the right node and the left node, where either of the nodes is searched according to query records. (Bhatia, 2010) the k-d tree effectively uses a decision tree to store information on distance, requiring fewer computations. An easy explanatory example is having three points, where the first point (1) is far away from the second point (2), but point (2) and three (3) are close. The k-d tree uses this information to determine that point (1) then also is far away from point (3).

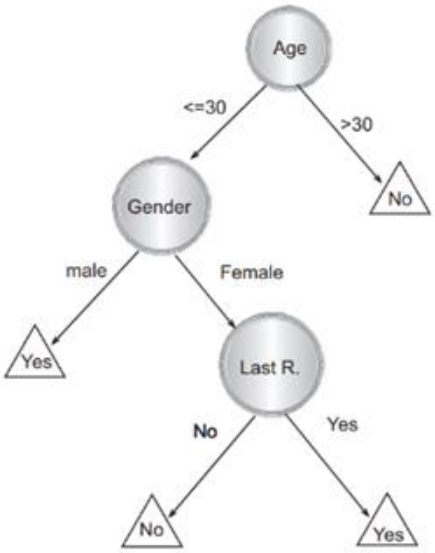
The last computation method, the ball tree method, is favorable to use when the k-d tree is difficult to create due to high dimensional space. (Bhatia, 2010) Each node in this method splits the data into two further sets, with each set then contained by the smallest ‘ball’, containing all the points. The points are then assigned to the sphere where the center is closest.

3.6.4 Decision Trees

A decision tree is one of the best-known classifiers due to its logical structure. A decision tree consists of nodes connecting, forming a tree with one single root node as a starting point. This node is called the "root" node and has no incoming edges. All nodes following the root node have a single incoming edge; if a node also has outgoing edges, that node is called an internal or test node. The internal node splits the data set into two or more subspaces according to a certain logic. Lastly, we have nodes called leaves, also known as terminal or decision nodes. Leaf nodes only have incoming edges, do not split like the internal nodes, and are assigned to a label based on which label is most appropriate. The classification process of a decision tree begins at the root node and continues by traversing through the internal nodes until a leaf is reached. (Maimon & Rokach pg. 149, 2010).

The figure below gives an example of a simple decision tree. (Maimon & Rokach pg. 150, 2010).

Figure 3.4: Simple Decision Tree



The decision tree incorporates both nominal and numeric attributes. Each node is labeled with the attribute it tests, and the tree's branches are labeled with its corresponding values. When using decision trees, impurity is measured to determine the split, and some parameters must be specified. The Gini Index and Entropy are the two most popular. The Gini index advantage is that it favors the production of pure over impure descendant nodes. When all possible candidate splits have been generated for one variable, the

procedure is repeated for another variable. From the maximum set of possible single variable splits, the split with the largest purity is applied to generate a new partition. The Gini Index corresponds to the variance of the outcome and ranges from 1 to 0. The lower the impurity value, the more accurately each observation can be classified into the appropriate split. (Kattan & Cowen, 2009)

The Gini Index is defined by:

$$The\ Gini\ Index = \sum_{j=1}^{c} p_j^2$$

Further parameters for the Decision tree are the determination of maximum depth, which is used to specify the maximum size of the constructed tree. A high-dimensional, deeper tree tends to have higher performance rates than less deep trees. Also, the number of features used at each split is a parameter usually determined for the decision tree. This parameter decreases the number of features used at each split, which decreases the chance of overfitting.

3.6.5 Random Forest

Random forest is, as its name suggests, also a ‘’tree’ like method. The difference between the decision tree and the random forests method is that the random forests are a classifier consisting of a collection of tree-structured classifiers. i.e., the random forests classifier consists of multiple trained decision trees that together make a classification. This technique is called Bootstrap Aggregation, or ‘’Bagging’’ for short. The bagging technique is to reduce the variance by averaging multiple samples, enhance accuracy when using random features and give ongoing estimates of the generalization error of the combined ensemble of trees, as well as estimates for the strength and correlation. (L. Breiman 2001)

The training procedure is similar to how a normal decision tree is trained, except that at each split in a tree, a random selection of features is selected, and from there, the future for the split is selected. The point of the random selection feature is to decrease the correlation between all the individual trees. Further, at least three parameters must be specified, and those are the number of features to consider at each node, the number of

samples to select out of which the algorithm is constructed, and lastly, the maximum depth/maximum number of layers each individual tree can contain.

3.6.6 XGBoost

XGBoost Is used widely by data scientists to achieve state-of-the-art results and is a scalable machine learning system for tree boosting. The impact of the system has been widely recognized in a number of machine learning and data mining challenges. The machine learning algorithm uses Gradient Boosting based on the second order from Friedmann et al. (Chen & Guestrin, 2016.)

The Gradient Boosting technique utilized an ensemble of tree methodology to generate a series of Decision Trees based on a given data sample. The output of the model is the sum of predictions from each individual tree. To prevent any overfitting, XGBoost implements a regularized objective model, which is given by this specified formula:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

Where: $\Omega(f) = \gamma T + \frac{1}{2}\gamma\|\omega\|^2$

The differentiable convex loss function l measures the difference between the prediction $\hat{y}_i$ and the target y_i. At the same time, the regularization term Ω penalizes the complexity of the model (e.g., the regression tree functions). The regularization term helps to smooth the final learned weights, thus preventing overfitting. (Chen & Guestrin, 2016.) The XGBoost uses a multiple set of parameters, and further parameters to be determined, which not mentioned are the Maximum depth, n number of Estimators, weight and Learning rates.

4.0 Methodology

After the theoretical background covered in the last chapter, this new chapter of the thesis will consist of the methodology, which indicates the description of how the objectives of this research project will be realized. The design of the research project is split into two phases, with phase one being data preparation and the second being model training and model testing.

4.1 Research Framework

In this next section of the thesis, I will discuss the approach to achieving the research objective.

The first step in this research framework is gathering and thoroughly reviewing the latest scientific literature on three specific subjects. The three subjects are machine learning theory, credit risk theory, and statistical theory. The purpose of reviewing machine learning theory is to develop and create various models that effectively can classify credit by assessing the likelihood of default. These models will first be trained using a dataset and then evaluated using another dataset to obtain the performing results. Combining all three literature sources, I can further create the assessment criteria to compare the models.

4.2 Data Preparation

As mentioned, the first stage is to prepare the data for machine learning. The data preparations start with one original dataset and for this dataset to be ready for machine learning, three steps must be performed. These steps are feature engineering, normalizing numerical features, and vectorizing categorical features.

In the first step in data preparation, feature engineering, the features will be analyzed and changed in the dataset, so the information is better represented. Future engineering is a dynamic process that highly depends on the futures in the original dataset and is, therefore, identical for each dataset. It can consist of adding, merging, and splitting features. An easy example of this is splitting a feature with two characteristics. By splitting the feature, the data might be more logically represented. It is then important to

assess the effectiveness of the engineered features through training and testing the algorithms on the modified dataset.

The second step in data preparation is the vectorization of categorical features. This step consists of making a separate feature for each value that the original feature consists of. These new features in the column are binary, taking only values of zero and one. Doing vectorizing makes the machine learning algorithms ‘’understand’’ the feature easier. For example, for this thesis, in the dataset, there is one feature representing sex, 1 if male and 2 if female. In machine learning algorithms nature, this order will be assumed in the values, and the feature will be treated as being numerical. This will lead to misinterpretation of the feature in the learning process. To counter this, the feature must be vectorized into two new binary features. Making two separate gender features, binary, one for male and one for female.

The third and last step in data preparation is the normalization of numerical features. Normalization is transforming the values of the feature in a way that it has the mean of zero, and a standard deviation of one. The way of doing normalization of the values is represented in the equation below. The z is the normalized value, X is the original value, μ is the average of the feature, and σ is the standard deviation of the feature.

$$z = \frac{x - \mu}{\sigma}$$

The point of this transformation is to change all numerical features so that the mean and standard deviation is the same. This is further because one feature might have a higher influence than another based on their difference in scale. While the difference in influence does not always happen, the average machine learning models learn faster and more accurately with normalized features. As a last note, this type of transformation should also be applied to all newly generated numerical features during feature engineering. (Qi & Zhang, 2002).

4.3 Model Training and Testing

The next stage in the approach to reach the objectives of this experiment is the training and testing phase. The dataset that has been conducted from the previous stage is now used to train and test different algorithms. This phase is roughly split into two separate processes, which naturally are training and testing.

Before either of the processes of training or testing can start, the dataset has to be split into training and testing sets. The reasoning for this cross-validating is to test the final algorithm settings on an unseen dataset. In this research experiment, a split of 30:70 is used, with 70% being the training set. Figure 4.1 below shows the 30% dataset being separated to only be used for testing the model at the last phase.

Figure 4.1: k-fold cross-validation

4.3.1 Training

The training phase is more complicated than first intended and involves more than simply training the models on the dataset. The process is iterative, aimed at updating the model parameters, intending to enhance the predictive performance. The input in this phase is the training set generated in the previous phase. During each iteration, the algorithm's parameters are changed and modified, and by using k-fold cross-validation, the performance is determined. In essence, the k-fold cross-validation entails a repeated process of training and validating an algorithm with the same settings. In each iteration, a different part of the training set is used for training, and the remaining part is used for validating the trained model. The average of the performances achieved in each iteration is then the model's performance.

Once an interaction of the training is complete, the parameters are changed and adjusted, and the performance is evaluated through k-fold cross-validation. This process is repeated until a comprehensive understanding of the influence of the various settings is reached. Due to computational limitations and to keep computation time realistic, it is impractical to calculate all the possible combinations of all possible parameter values. Because of this, the parameters will be individually analyzed. When reaching a good understanding

of the individual parameters, the allowance of combinations is there, by changing multiple parameters.

In this research project, all computations have been utilized using my own computer, Acer Aspire A315-42. Processor: AMD Ryzen 3 3200U with Radeon Vega Mobile Gfx running at 2.60GHz. More importantly, the algorithms have been implemented using Python learning.

4.3.2 Model Performance Measure

It is often required to accurately determine the predictive performance of an algorithm. Discussions of various model measures have been completed, and a selection of which measures to use in this research experiment must be made. The chosen measure should provide accurate indications of model performance while also being susceptible to problems such as the accuracy paradox or class imbalance, and simpler measures prone to these problems are naturally excluded. For this thesis and research experiment, the measure to express the performance of choice is the Area Under the Curve (AUC). The AUC has been theorized in Chapter 3. Additionally, in certain situations, such as resampling and assessing the final tested performance, a confusion matrix might be utilized. This will allow the reader to calculate alternative measures if desired.

4.3.3 Model Testing

The final step is the testing phase. This step aims to evaluate the final model's performance using a portion of the dataset that the model has not previously seen. This ensures that the performance measure is not inflated due to overfitting. The portion of the dataset that was separated during the initial phase of the experiment can now be utilized for testing. This is represented as the last row in Figure 4.1 above, where the withheld portion of the dataset is employed for making predictions, while the rest of the data is used for model training. The results and outcomes from doing such analysis will then be used to conjure a conclusion about the performances of the different machine learning algorithms when used to predict defaults.

5. Data Description and Preparation

5.1 Data Description

The dataset is publicly available from multiple sources and collected from two. The two are Kaggle and UCI Machine Learning Repository. This dataset was based on Taiwanese credit card customers in 2004. The dataset contains details of 30,000 Taiwanese credit lenders, where 6,636, or roughly 22% of the customers, are actual defaulters. There are 25 variables in this dataset, including the response variables that indicate default or non-default. The dataset's features can be split into two categories: personal and financial. While the personal category consists of sex, education, marriage, and age, The financial category consists of the amount paid per month and the amount on the billing statement per month, as well as the credit limit. It is important to note that this data set has no missing values in the dataset.

Below some Tables provide further information. In the first table, Table 5.1, all features are described. Table 5.1 Separates the feature types. Table 5.3 provides a statistical summary of all numerical features in the dataset, consisting of feature means, standard deviations, and minimum and maximum values. Tables 5.4 and 5.5 summarize the financial and personal categories, respectively.

Table 5.1: Dataset Feature Description

Features	Description
ID	The ID of each client
LIMIT_BAL	Client's Maximum given credit
SEX	Gender of the client
EDUCATION	Level of Education of the client
MARRIAGE	Marital status of the client
AGE	Age of client in years
PAY_1	Payment status in September 2005
…	…
PAY_6	Payment status in April 2005
BILL_AMT1	Amount of bill statement in September 2005
…	…
BILL_AMT6	Amount of bill statement in April 2005
PAY_AMT1	Amount paid by September 2005
…	…
PAY_AMT6	Amount paid by April 2005
DEFAULT	Whether the client defaulted of not

Table 5.2: Feature Types

Numerical Features	Categorical Features
LIMIT_BAL	GENDER
AGE	EDUCATION
BILL_AMT1 ... BILL_AMT6	MARRIAGE
PAY_AMT1 … PAY_AMT6	PAY_1 … PAY_6
	DEFAULT

Table 5.3: Statistical summary of numerical features.

Feature	Mean	Std	Min	25%	50%	75%	Max
LIMIT_BAL	167,484	129,747	10,000	50,000	140,000	240,000	1,000,000
AGE	35	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

Summary of the Categorical Features

Table 5.4: Summary of the financial category. PAY_1, PAY_2 and PAY_3 Features.

		PAY_1		PAY_2		PAY_3	
Value	Description	n	%	n	%	n	%
-2	No Consumption	2,759	9.20%	3,782	12.61%	4,085	13.62%
-1	Paid In Full	5,686	18.95%	6,050	20.17%	5,938	19.79%
0	Revolving Credit	14,737	49.12%	15,730	52.43%	15,764	52.55%
1	One Month Late	3,688	12.29%	28	0.09%	4	0.01%
2	Two Months Late	2,667	8.89%	3,927	13.09%	3,819	12.73%
3	Three Months Late	322	1.07%	326	1.09%	240	0.80%
4	Four Months Late	76	0.25%	99	0.33%	76	0.25%
5	Five Months Late	26	0.09%	25	0.08%	21	0.07%
6	Six Months Late	11	0.04%	12	0.04%	23	0.08%
7	Seven Months Late	9	0.03%	20	0.07%	27	0.09%
8	Eight Months Late	19	0.06%	1	0.00%	3	0.01%

Table 5.5: Summary of the financial category. PAY_4, PAY_5 and PAY_6 Features.

		PAY_4		PAY_5		PAY_6	
Value	Description	n	%	n	%	n	%
-2	No Consumption	4,348	14.49%	4,546	15.15%	4,895	16.32%
-1	Paid In Full	6,687	18.96%	5,539	18.46%	5,740	19.13%
0	Revolving Credit	16,455	54.85%	16,947	56.49%	16,286	52.29%
1	One Month Late	2	0.01%	0	0.00%	0	0.00%
2	Two Months Late	3,159	10.53%	2,626	8.75%	2,766	9.22%
3	Three Months Late	180	0.60%	178	0.59%	184	0.61%
4	Four Months Late	69	0.23%	84	0.28%	49	0.16%
5	Five Months Late	35	0.12%	17	0.06%	13	0.04%
6	Six Months Late	5	0.02%	4	0.01%	19	0.06%
7	Seven Months Late	85	0.19%	58	0.19%	46	0.15%
8	Eight Months Late	2	0.01%	1	0.00%	2	0.01%

Summary of the Categorical Personal Categories.

Table 5.6: Gender

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

Table 5.7: Education

Value	Description	n	%
0	Other	14	0.05%
1	Graduate School	10,585	32.28%
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 5.8: Marriage

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

To better understand the dataset, several features will be looked at and more extensively described, and some relations between features will be shown, as they seem interesting.

First, we will look at the average age and default rate for different categories. Below, three tables are displayed. Table 5.9 considers the average age and default rate for marital status, and Table 5.10 considers the average age and default rate for the educational categories. Lastly Table 5.11 considers both genders average age and default rate. The first notable and interesting notation is that for both features that consist of "other," the

"other" category has a relatively low default rate. One conclusion from this might be that these people probably have a high income or net worth.

Further, we see a clear difference between the "single" individuals and the "divorced" individuals, where the divorced have a higher default rate than the single ones. Interestingly, individuals who have finished graduation school have a lower default rate than those who have gone through high school and university. Lastly, when looking at the age difference between males and females, there are two years apart in average age, which is considered to be almost equal to each other, and there is only a 2.5% higher default rate for women.

Table 5.9: Average Age and Default Rate for Marital status

Marital Status	Average Age	Default Rate
Other	38	9%
Married	40	23%
Single	31	21%
Divorced	43	26%

5.10: Average Age and Default Rate for the Educational categories

Educational Status	Average Age	Default Rate
Other	36	7%
Graduate School	34	19%
University	35	24%
High School	40	25%

Table 5.11: Average Age and Default Rate for Male and Female

Gender	Average Age	Default Rate
Male	37	8%
Female	35	10.5%

5.2 Correlations

Calculation the correlation between the features is the next step in analyzing the dataset. Table 5.9 below shows the calculated correlation between all numerical features, and we see from the table that the correlations between the bill amount features are highest, ranging from 0.8 to 0.95. The correlations are high between the previous and next periods. This is because the values of those features change respectively to the value in the next period.

The correlations between the payment amounts are lower than the bill amount. The reasoning for this is probably the fact that the payments amount and the amount of the previous period are unrelated.

Lastly, the balance limit also shows strong positive correlations between the other features. This does make sense, as a higher balance limit provides the opportunity to take out larger loans. Age's highest positive correlation between the other features is between the balance limit feature, and the most probable cause for this is that older people, on average, have a higher income, meaning they have higher loans as well.

Table 5.12 Correlations between all numerical features.

	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	095	-	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	00.2	0.10	010	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	-

5.3 Vectorization and Normalization

In an earlier chapter, vectorization and normalization have been described. It was then stated that this phase will be performed after the future engineering stage. This is because new and/or changed features also, if necessary, have to be vectorized and normalized. However, in this phase, the order of the two has been switched, and the reason for this is to see and determine whether the impact of the engineered features affects the models, as well as if the engineered features must be analyzed with or without vectorization and normalization.

In this phase of analyzing the dataset, the first step, is to determine the performance of all the algorithms when the algorithms are trained and validated using an unchanged original dataset. The results are given in the table below, and there is a high variety between all the models. The algorithms that are based on trees perform better than the others, with XGBoost and Random Forest having the highest AUC at 0.76.

Table 5.13 Performance of the different model algorithms when trained and validated on the original dataset.

Model Algorithms	AUC
Logistic Regression	0.64
Neural Network	0.65
Decision Tree	0.62
k Nearest Neighbor	0.60
Random Forests	0.76
XGBoost	0.76

Now that the original dataset has been trained and validated, and the performance without doing any changes to it is gathered, the next step is preparing the dataset in order for the machine learning algorithms to better their performance. The first thing to prepare is the two preparation techniques of vectorization and normalization. The first one I will tackle is vectorization, dealing with categorial features. For example, when certain categorical features are used as input in its original form, such as in the original dataset, the model might have difficulty distinguishing the gender. This is due to machine learning algorithms assuming a certain order in the feature values, and vectorizing the gender feature will solve this problem. For this experiment, there will be vectorization of the

categorical features, such as gender, education, and marriage. Below, three tables is provided, highlighting vectorization of the features.

Table 5.14: Gender Feature Vectorized

ID	Gender	Male	Female
46	2	0	1
47	2	0	1
48	1	1	0
49	1	1	0
50	1	1	0

Table 5.15: Education Feature Vectorized

ID	Education	Other	Graduate School	University	High School
46	1	0	1	0	0
47	4	0	0	0	0
48	2	1	0	1	0
49	1	0	1	0	0
50	3	0	0	0	1

Table 5.16: Marriage Feature Vectorized

ID	Marriage	Married	Single	Divorced	Other
36	1	1	0	0	0
37	2	0	1	0	0
38	2	0	1	0	0
39	2	0	1	0	0
40	2	0	1	0	0

Now that categorical features have been vectorized, numerical features can be prepared by normalization. By normalizing the features, we manipulate the values in the features so that the average will be zero and the standard deviation will be one. We transform them

so they all have the same mean and standard deviation. As mentioned in Chapter 4, minimizing the different levels of influences between the features is the goal.

Table 5.17: Balance Limit Feature Normalized

ID	LIMIT_BAL	Balance Limit
46	20,000	-1.137
47	150,000	-1.135
48	380,000	1.638
49	20,000	-1.138
50	20,000	-0.751

Table 5.18: Pay Amount 1 Feature Normalized

ID	PAY_AMT1	Pay Amount 1
46	3,000	-0.160
47	1,013	-0.280
48	21,540	-0.958
49	1,318	-0.262
50	0	-0.341

After applying vectorization and normalization to the original dataset, it is now possible to determine the impact on the performance compared to the performance without. Below, Table 5.12, display the results of training and validating the model algorithms with the vectorized and normalized features of the original dataset.

Table 5.19 Showcasing Performance after vectorizing and normalizing.

Model Algorithms	AUC	Change
Logistic Regression	0.72	+0.08
Neural Network	0.78	+0.13
Decision Tree	0.62	+0.00
k Nearest Neighbor	0.70	+0.10
Random Forest	0.76	+0.00
XGBoost	0.76	+0.00

The results show that the AUC of all algorithms is either the same or higher, providing information that pursuing vectorization and normalization for all algorithms is a good idea. The highest increase in performance is Neural Network, with the AUC increasing from 0.65 to 0.78. The reason for no change in the Decision Tree, Random Forest, and

XGBoost is that these types of algorithms, based on trees, do not rely on the scale of a feature.

5.4 Feature Engineering

After vectorizing and normalizing, the next step in preparing the dataset, is further tweaking and feature engineering, which will enhance performance further. Just as before, the goal is to change possible features in such a way that the algorithms "understand" the dataset better and easier, so it can produce better results.

In the education feature, there are seven different value possibilities, whereas four of these are labeled as non, other, and unknown. There is zero explanation to what these mean and no possible way to distinguish between them. The result then, is combining all these values to belong to the 0 category, or other. My hypothesis is that this change has little to no impact on all the models results, and doing this keeps the consistency to all categorical features.

The most interesting change done to one of the features was the PAY_n features. These categorical features contain much information. They contain, referring to Table 5.5, information regarding whether the credit is revolving or installment, if there has been any consumption, and if the credit is paid on time or late. In total, 11 different values are mapped onto this feature, which, in many cases of machine learning algorithms, is difficult to read and understand. This applies especially to the values being negative. To make the data easier for the algorithms to read, three new and separate characteristics have been installed into new features.

Table 5.20: Consumption Table

Value	Description
0	No Consumption
1	Consumption

Table 5.21: Revolving

Value	Description
0	Instalment
1	Revolving

5.22: How Late

Value	Description
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

In the table above, an explanation of the new features and their values are provided. The first two features are for consumption and credit type, extracted from the PAY_n values, and are made binary [1, 0]. The last feature provides information on a client's lateness, ranging from 0 - 8, with 0 being on time and 1 through 8 depending on how many months the clients are behind. This change, in splitting the features, has no negative effect on any of the results of the algorithms, compared to the results from the vectorization and normalization phase, and is reason enough to keep the changes.

Additionally, as there are no missing or extreme values in the dataset, I started by removing the “ID” feature, as it has no predictive powers and is just a range of identification for the clients. The second little change was changing PAY_0 to PAY_1 for the feature to stay fit with BILL_AMT1 and PAY_AMT1. I also changed the response variable “default.payment.next.month” to Default, for personal preferences.

6. Model Training

In this chapter, the machine learning algorithms parameters will be determined. The results from the prepared dataset have been inspected and will be used in this section. There have been many different forms of feature engineering applied to all different algorithms. First, all datasets for all algorithms will be vectorized and normalized. The new features based on PAY_n and the combined values for "others" in education will also apply to all algorithms. This makes the datasets more logical, has no negative effect on any algorithms, and provides better and positive results. The exact data preparation is further discussed with each of the algorithms.

6.1 Logistic Regression

For Logistic regression, there are two regularizations available, l1 and l2. There are two types of solvers, which are Saga and Liblinear. As well as one regularization term, C.

I start by examining the regularization and solver together, and the performance of the four possible combinations is determined here, showcased in Table 6.1.

Table 6.1.1 The Performance of the different Regularizations and Solvers.

Regularization	Solver	AUC
Liblinear	l2	0.726
Liblinear	l2	0.726
Saga	l1	0.726
Saga	l2	0.726

The resulting AUC scores for all four combinations have no difference, leading to an AUC score of 0.762. However, the time difference for the models to run is slightly different. Although there were only small differences, the model running fastest was the one with liblinear and l2, which determines the use of them further.

The determination of the Regularization term is set to the default value of 1.0, as testing a range from 0,1 to 5 does not impact any results when testing with the regularizations and solvers.

Table 6.1.2 Parameters selected for Logistic Regression

Parameter	Selected
Regularization	l2
Solver	Liblinear
C	1.0

6.2 Neural Network

The “easiest” way to determine the parameters for neural networks is by splitting the process into two phases. These phases are based on the two possible solvers one can use: adam and sgd. The parameters that will be examined and determined in the two phases are the number and size of the layers between the output and input layers, called Hidden layers. The function shape used to transform the input signal of the neurons is called the Activation function. A constant or adaptive rate of the step size where weight is changed is called the Learning rate. The Regularization term, is used to penalize complex models, countering potential over-fitting. Lastly, Max iterations are the maximum number of iterations to reach convergence.

I will start by determining the parameters for the adam solver, then for the sgd solver. In both stages, I determine the hidden layers, followed by the activation function and learning rate. Throughout, the maximum number of iterations will be determined.

6.2.1 Adam Solver

As mentioned, the first parameter I will analyze in this stage is the hidden layers. Since there are 30 features in the dataset the number of input nodes is equal after preparations. This is usually a standard, having the number of nodes in each hidden layer be the same as the input nodes. With each hidden layer, the computation time increases and will be considered when determining the parameters. To analyze closer, some different hidden layers are considered, ranging from one to three hidden layers with a range of 10 to 30 nodes per layer. In the table below, I showcase the performance of the different hidden layers, in combination with the three activation functions, named Logistic, ReLu, and Tanh.

Table 6.2.1 The Performance of the different hidden layers with the activation functions. All models are trained with the Adam solver.

Nodes per Layer			Logistic	ReLu	Tanh
First Layer	Second Layer	Third Layer	AUC	AUC	AUC
30	20	10	0.780	0.742	0.751
30	20	-	0.771	0.755	0.759
30	10	-	0.768	0.758	0.760
20	10	-	0.799	0.765	0.761
30	-	-	0.780	0.774	0.764
20	-	-	0.781	0.770	0.759
10	-	-	0.788	0.778	0.765
-	-	-	0.712	0.721	0.732

Adding no hidden layers has the lowest AUC, concluding that it is necessary to add hidden layers for all activation functions. This is easy to understand, as neural networks can only make linear separations without hidden layers. When analyzing the results, we can also point out that adding more layers does not necessarily lead to higher AUC results but only adds computational time. Further, very little separates the AUC scores between the activation functions, but using the logistic activation function seems to perform slightly better, especially when only using one layer with ten nodes (0,79 AUC when rounded up).

The next step in this neural network parameter analysis is determining the Learning rate. A too high learning rate might cause overshooting, causing the training process to stop before intended. As mentioned, the learning rate can either be constant or adaptive, and when adaptive, the learning rate decreases each time the stopping criteria are reached instead of the training process stopping. Then, when the learning rate reaches a certain value, the training process stops. The two tables below showcase the performance of the model when using the different constant learning rates, as well as the performance of an adaptive learning rate, ranging from 0.1 to 0.0001.

Table 6.2.2 The Performance on different constant learning rates and the activation functions.

	Logistic	ReLu	Tanh
Learning Rate	AUC	AUC	AUC
0.1	0.758	0.754	0.760
0.01	0.768	0.767	0.768
0.001	0.791	0.777	0.772
0.0001	0.742	0.738	0.750

Table 6.2.3 The Performance on different adaptive learning rates and the activation functions.

	Logistic	ReLu	Tanh
Learning Rate	AUC	AUC	AUC
0.1 to 0.0001	0.771	0.762	0.769

From the results, the AUC performance is very close to each other, and the performance effects are minimal and probably caused by random variation. Yet again, the Logistic activation function performs better, and the best AUC results were performed when the constant learning rate was set as 0.001.

The next step is determining the generalization term. A more general model is led by a higher value of the generalization term, and thus higher value leads to reduced overfitting. To determine the generalization term, the term has been tested with a variety from 0 to 10. When testing values between 0 and 1, there were no influences at all. While the model becomes more general as the generalization term increases, the generalization term has been decided to be set as 0.0001. This is because 0.0001 is the standard default value. Because there were no influences between 0 and 1, there is no reason for having it higher than 1, risking the model being too general.

The last parameter, the Maximum iterations, works closely with another unmentioned parameter, called Tolerance. Both parameters work in the determining when the training should stop and are more important when computation time is high. For this dataset, the computation times are very low, taking only a couple of minutes in some cases, and therefore, these parameters are set as loose. The tolerance is set to zero, meaning the training will only stop when the maximum of iterations has been reached. To not have the iterations easily reached, the maximum iterations have been set to 20,000.

6.2.2 Sgd Solver

Below, in Table 6.2.4, the same shapes as before are considered, just as with the adam solver. From the table, we see, just as before, that there is no reason to add more than one hidden layer. The performances are not as strong as with the adam solver, but the results conclude that no more than one hidden layer is enough, preferably 10, as found earlier. In contrast to the results with the adam solver, we see that the usage of the logistic activation function provides a slightly lesser result than the other two.

Table 6.2.4 The Performance of the different hidden layers with the activation functions. All models are trained with the Sgd solver.

	Nodes per Layer		Logistic	ReLu	Tanh
First Layer	Second Layer	Third Layer	AUC	AUC	AUC
30	20	10	0.662	0.775	0.754
30	20	-	0.728	0.767	0.771
30	10	-	0.734	0.771	0.765
20	10	-	0.727	0.763	0.762
30	-	-	0.731	0.763	0.764
20	-	-	0.733	0.766	0.768
10	-	-	0.724	0.745	0.757
-	-	-	0.704	0.709	0.708

Next in line to be analyzed are the learning rate parameters. Just as with the adam solver, the same rates are considered, 0.1 to 0.0001. The main differences noticed when analyzing the learning rates in using sgd solver were the computing time and the AUC scores tended to decrease with decreased learning rates. The computing time, however performed better when using the adaptive learning rate.

Table 6.2.5 The Performance on different constant learning rates and the activation functions

	Logistic	ReLu	Tanh
Learning Rate	AUC	AUC	AUC
0.1	0.772	0.764	0.769
0.01	0.769	0.778	0.765
0.001	0.717	0.748	0.766
0.0001	0.676	0.729	0.714

Table 6.2.6 The Performance on different adaptive learning rates and the activation functions.

	Logistic	ReLu	Tanh
Learning Rate	AUC	AUC	AUC
0.1 to 0.0001	0.724	0.763	0.749

When analyzing the last three parameters, regularization term, maximum iterations, and tolerance, the values compared to the adam solver are similar, if not identical. The behavior of the regularization term was just as the adam solver. It only affected the result negatively when higher values were appointed. The same reasoning applies here for the last two parameters as it did with the adam solver. The tolerance was set to 0, and the maximum iterations were set to 20,000.

6.2.3 Comparing Adam and Sgd

Now that both solvers have been analyzed, comparing, and deciding which parameters to use is in order. The differences between the two algorithms, either with the adam or sgd solver, are small. However, the adam solver edges the sgd out and performs slightly better. The highest performance AUC was 0.79, using the adam solver, being the highest performer between the two. The adam solver had a clear relation with the learning rate, while the sgd solver was rather unclear, even having decreasing performance with lower learning rates. One last point; even though all algorithms were relatively fast to compute, the adam solver edges the sgd solver out with a usual 50% shorter computing time. Thus, a conclusion of using the adam solver has been made. The following selection of parameters for the neural network:

Table 6.2.7 Parameters selected for Neural Network

Parameters	Selected
Solver	Adam
Hidden Layers	One Layer with 10 Nodes
Activation Function	Logistic
Learning Rate	0.001
Regularization Term	0.0001
Tolerance	0
Maximum Iterations	20,000

6.3 Decision Trees

The Decision tree works by constructing a branching tree, splitting the data at each branch according to a feature's value. When a branch no longer splits, it is called a leaf, containing a certain class. To evaluate the best performance of the decision tree, the additional feature engineering, as mentioned in the last chapter, has been applied. Further, three parameters are analyzed to determine the performance. These three are the Criterion, Max Depth, and n Features. The Criterion is what measures of impurity to use when basing the split of branches. Max depth is the maximum depth of the constructed tree, and n Features are the number of features considered for each split.

The first two parameters are combined in an experiment to determine their values for the best performance of the decision tree. This is due to easily showcasing the results of both parameters in a single plot. Firstly, the criterion, which is a value that expresses dataset's impurity, has the goal of each split to minimize the impurity. There are two criterions generally used for decision trees, which are already mentioned in Chapter 2, the Gini index and Entropy. Secondly, the maximum depth is used to specify the maximum size of the tree constructed. It is usual to expect a higher performance with a deeper decision tree.

Figure 6.3.1: The Performance of both Criterions with different Maximum Depth

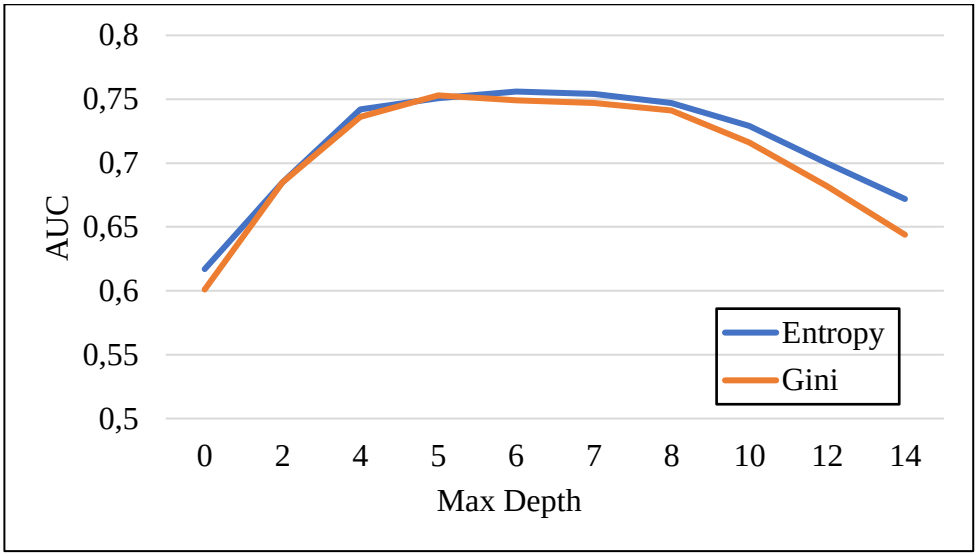


Figure 6.3.1 above shows the usual expectancy of higher performance with a deeper decision tree only applies to a certain degree of maximum depth. Having a higher max

depth between 5 and 6 decreases the overall performance. This is due to the training set overfitting. Deeper trees will fit itself to the exact dataset instead of the underlying structure of the data in the set. Further examination shows that the entropy criterion performs slightly better than the Gini index, with the best performance being with the value 6 of maximum depth.

The last parameter to analyze is the n features or the number of features considered at each split. The point is to decrease the number of features to prevent overfitting. The features are randomly selected from all available features in the dataset. To test the n features, three different values of features are tested, together with maximum depth again, to determine the parameter that performs better. The three different values of features are the n number of features, the square root of n, and log2 of n. The results are showcased in the table/plot below. Here we can see that the n numbers of features perform better than the others until we reach approximately 7 numbers of maximum depth. As before, the expected decrease in performance with higher maximum depths still applies, and we start to see a decrease in performance after 6 maximum depths as before.

Table 6.3.2: The Performance of the three different n features with different Maximum Depth

	n	Sqrt n	Log2 n
Max Depth	AUC		
0	0.619	0.607	0.618
2	0.693	0.696	0.692
4	0.738	0.741	0.737
6	0.756	0.749	0.751
7	0.754	0.745	0.749
8	0.752	0.745	0.747
9	0.747	0.733	0.750
10	0.737	0.725	0.725
12	0.709	0.713	0.703
14	0.672	0.664	0.659
16	0.640	0.642	0.660

From the results of the two analyses, these are the resulting parameters determined to the decision tree algorithm.

Table 6.3.3: Parameters selected for Decision Tree

Parameters	Selected
Maximum Depth	6
Criterion	Entropy
n Features	n

6.4 Random Forest

The random forest model is constructed from many individual decision trees. Each of the individual trees are trained on a random subset of the training data. During each split, a subset of the features is considered, on which the selected feature base the next selection to base the split on. After the training process, the classification is then determined by the output of all individual trees and combines them. The parameters to determine for the Random Forest algorithm are Maximum depth, Maximum features, and n Estimators. The Maximum depth and Maximum features are the number of layers the individual trees can contain and the number of features to consider at each split. The n Estimator is the number of individual trees from which the classifier is constructed.

The maximum depth parameter will be the first determined for the random forest algorithm. There will be two different methods to decide the maximum depth. The first one is the decision stump, which uses trees that consist of a maximum of one depth. This method involves having the number of estimators higher for the result to get better. The second method involves using the same maximum depth determined by in the decision tree algorithm. This method will lead to fewer estimators with higher performance. Tables 6.4.1 and 6.4.2 below displays the performance of both methods, using the three maximum features, n, n squared, and Log2 n.

Figure 6.4.1: The Performances of Maximum Depth = 1, using the three different Maximum Features, in relation to n Estimators.

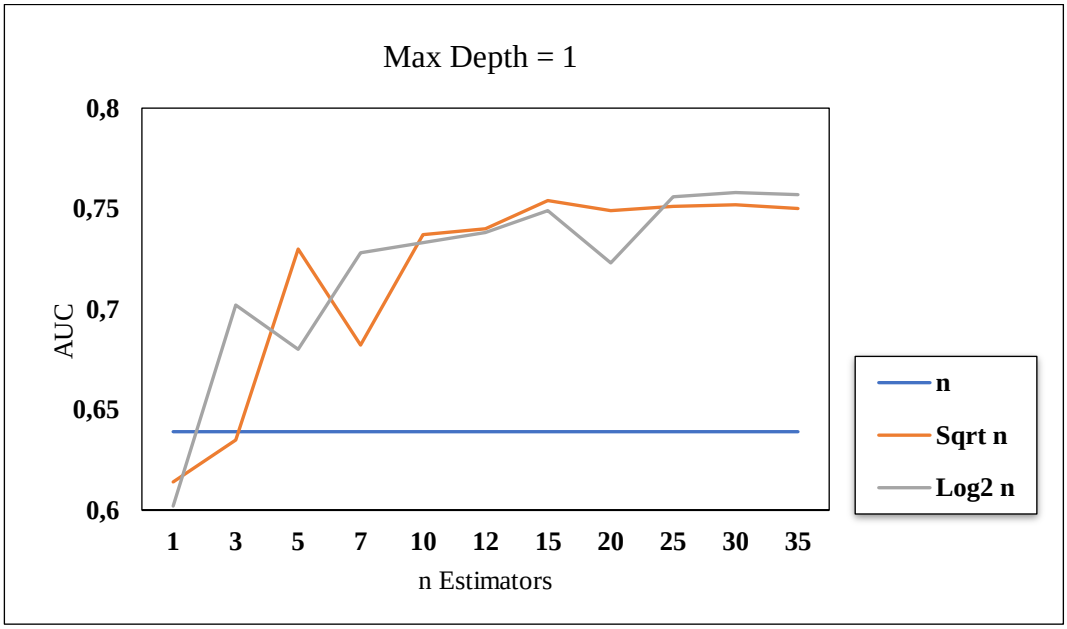
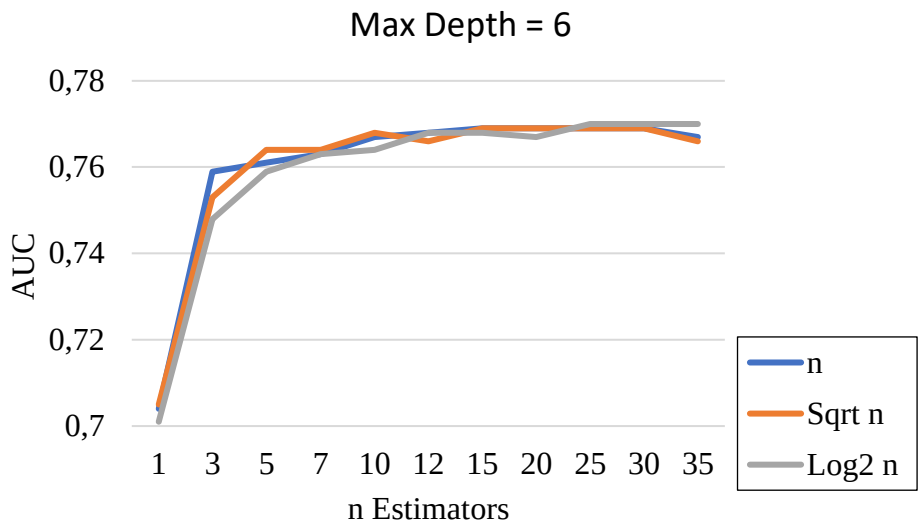


Figure 6.4.2: The Performances of Maximum Depth = 6, using the three different Maximum Features, in relation to n Estimators.



The result shows that both experiments' performances increased fast from the estimator value between 0 and 10. Beyond the estimator value 10, both experiments' performances stayed consistent at approx. The highest performance value for that same experiment. This indicates that determining the value of estimators at 10 might be a good idea. Further,

as mentioned, the performance increases faster when the estimators are lower, when maximum depth equals 6, confirming my initial thought.

When the maximum depth is one, the performance increase is slower, and one exception is, however, when trees with the maximum depth of one is used in combination with n features available at each branch-split. This is shown in Table 6.4.1, where the AUC equals 0.639 in every case of estimator value. This is caused by the same split being made in each of the decision stumps in the ensembled tree. Finally, there is a clear "winner" of the two, whereas the algorithm with maximum depth of 6 has a significantly higher performance.

Lastly, to verify that the maximum depth of 6 and the estimator of 10 are the right choice for the random forest, reversing the analysis just done has been made. The tables below showcase the performance of Estimator equals 5 and 10 using the same three different maximum features in relation to a maximum depth between 1 and 35.

Figure 6.4.3: The Performance of Estimator = 5, using the three different Maximum Features, in relation to Max Depth

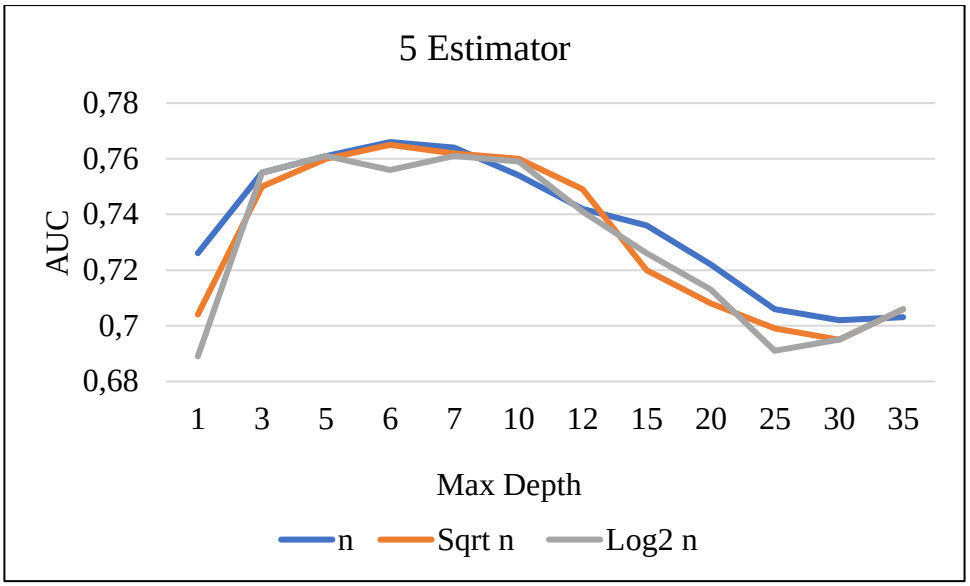
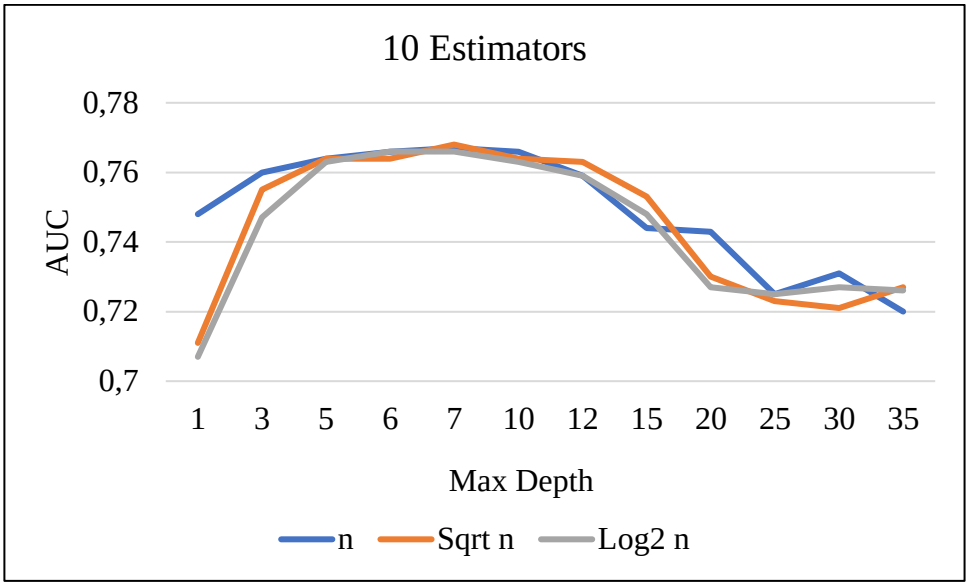


Figure 6.4.4: The Performance of Estimator = 10, using the three different Maximum Features, in relation to Max Depth



From the results, the same conclusion as with the decision tree can be made. Overfitting causes the performances to decrease with higher maximum depth reaching a certain value and the performance decrease with a higher number of estimators. Lastly, the highest performing value is when the maximum features are n and the maximum depth is approx. 6. This, therefore, verifies that having a maximum depth of 6, just as the decision tree, and the estimator value of 10, is a good choice.

From the results of the analysis, these are the resulting parameters that have been determined to the random forest algorithm.

Table 6.4.1: Parameters selected for Random Forest

Parameters	Selected
Maximum Features	n
n Estimators	10
Maximum Depth	6

6.5 k Nearest Neighbors

The next algorithm to determine the parameters is the k Nearest Neighbors model. The k Nearest Neighbors algorithm makes a classification based on the classes of the closest

sample, or as the name implies, the nearest neighbors. The parameters to be determined are Algorithms, Leaf Size, n Neighbors, and Weights. There are three algorithms to be analyzed and compared: Brute Force, k - d Tree, and Ball Tree. The Leaf size is the number of samples per leaf, only used for the k - d Tree and Ball Tree algorithms. n Neighbors, is the number of neighbors in the algorithm, and Weights which are the weights, or importance of the individual neighbors used for the classification.

For each calculated parameter, the n Neighbors will be combined.

The first parameters to analyze and determine are the three different algorithms that can be used for the k nearest neighbor algorithm. To analyze the difference between these, each of these has been calculated using different values for n neighbors to evaluate if the algorithms performance effects differ with different values for n neighbors.

Table 6.4.1: The Performance of the three Algorithms, in combination with different n Neighbors.

n Neighbors	Brute Force AUC	k-d Tree AUC	Ball Tree AUC
1	0.605	0.604	0.600
5	0.703	0.695	0.703
10	0.730	0.740	0.730
15	0.744	0.742	0.745
20	0.746	0.744	0.746
25	0.751	0.747	0.743
30	0.753	0.746	0.758
35	0.760	0.758	0.759
40	0.760	0.755	0.748
45	0.755	0.755	0.758
50	0.755	0.751	0.752

From the table above, the performances of the three different algorithms are very similar. All algorithms perform better with higher n Neighbors, with the value of n Neighbors of 35 being at the top of performances, for all three algorithms. The Brute Force algorithm edges it out compared to the other two, with a +0.02 and +0.01 better performance result, respectively. To further strengthen the Brute Force algorithm, its computation time compared to the other two, was considerably shorter. While the Brute Force used, on average approx. 10 seconds to run, the other two used at least a minute to compute. The reason the Brute Force is so much faster is because k-d Tree and Ball Tree both construct

a tree during the training process, while the Brute Force does not, and only compares each sample. Having larger datasets increases computation time, and while this dataset isn't the biggest, the Brute Force algorithm still performs better.

Lastly, the Weights parameter is analyzed and determined. The two weight parameters work in such a way that the n Neighbors get an identical weight if the Uniform is chosen, and if the Distance is chosen, the closest n Neighbor gets the highest weight. Table 6.4.2, showcase the performances of the difference of the two weight parameters, in combination with different n Neighbors. The result shows that there is very little that separates the two. However, the Uniform has been chosen for this experiment. This is because the uniform performances just about edges the distance out, and the uniform is more commonly used, due to the distance being more complex.

Table 6.4.2 The Performance of Weights, in combination with different n Neighbors.

Distance Weights n Neighbors	Distance Weights AUC	Uniform Weights AUC
1	0.611	0.600
5	0.698	0.699
10	0.721	0.727
15	0.745	0.744
20	0.745	0.737
25	0.744	0.755
30	0.748	0.745
35	0.748	0.751
40	0.757	0.759
45	0.751	0.752
50	0.759	0.759

Notably, since the Brute Force has been chosen, there will be no need to determine the leaf size, as the Brute Force algorithm is, as mentioned, not a tree-based algorithm.

The parameters chosen for the k Nearest Neighbors are presented in Table 6.4.3.

Table 6.4.3 Parameters selected for k Nearest Neighbors

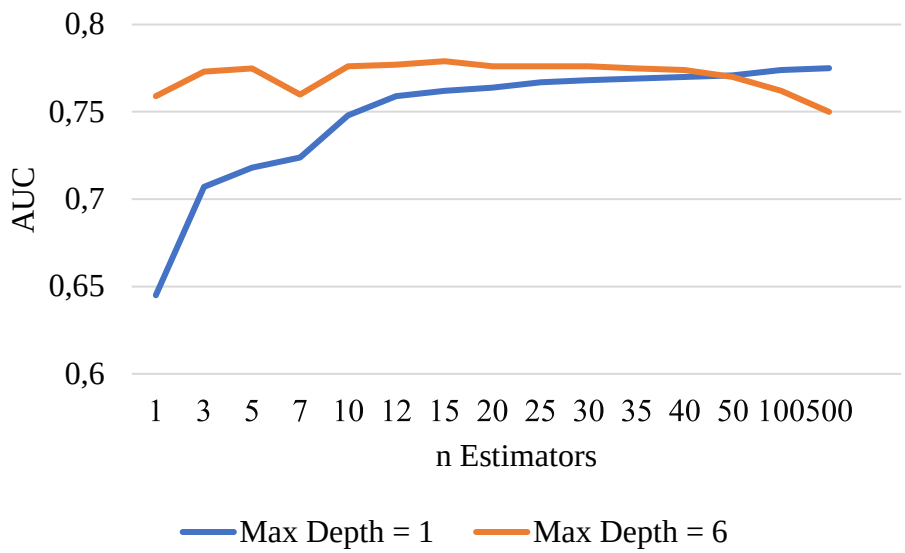
Parameters	Selected
Algorithm	Brute Force
n Neighbors	35
Weights	Uniform
Leaf Size	Not Applied

6.6 XGBoost

XGBoost is an advanced implementation of a gradient boosting algorithm, a sophisticated algorithm that uses parallel computation, where multiple decision trees are parallel trained to determine the final performance. This algorithm uses a multiple set of parameters, and the parameters to be determined in this experiment are the following: Maximum Depth, n Estimators Weights, Learning Rate and Regularization Term. While all these parameters have been used and explained in this chapter earlier, I will retain from explaining them again.

The first parameter I will analyze and determine is the maximum depth. This will be done in a similar way as in the random forest phase. I will use the same two methods, where the first method was using a decision stump, and the second method using the same amount of maximum depth as determined in the decision tree algorithm. Below, the resulting performances using maximum depth equal to 1 and 6 are showcased. Table 6.6.1 provides an easy conclusion of using the full tree rather than the decision stump. Even though the decision stump seems to climb with increased estimators, the AUC stops increasing when 500+ estimators are reached. The maximum AUC the decision stump computed was 0.77, whereas the algorithm with the maximum depth of 6 reached a performance of 0.78 with 15 valued as the estimator.

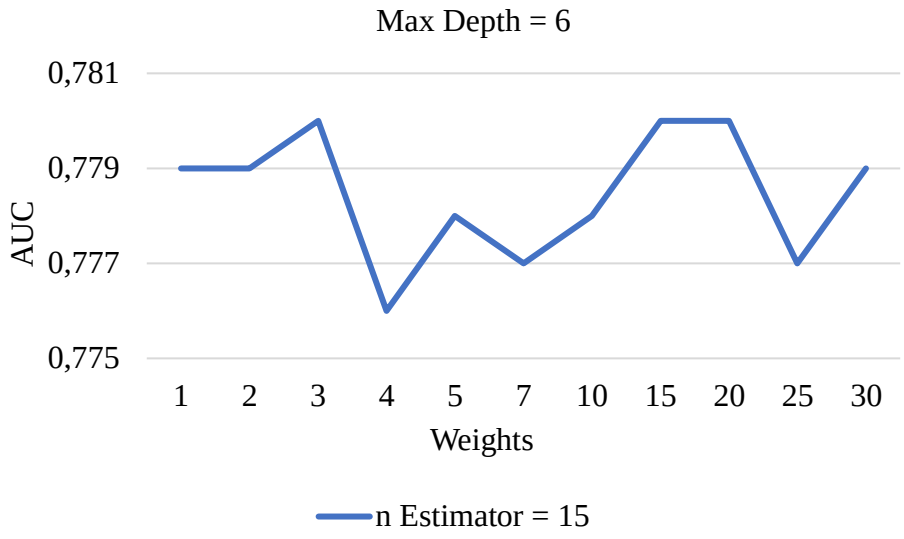
Figure 6.6.1: The Performances of using different n Estimators in combination with Maximum Depth = 1 and 6.



The following parameter to determine is the weight parameter. This parameter defines the minimum sum of weights of all observations required in a branch and controls over-fitting. Here it is about finding the right balance. Even though weight is controls over-fitting, a high value of weight can cause under-fitting. In Table 6.6.2, the performances of the algorithm consisting of different values of weights, combined with a n Estimator of 15, are showcased.

As the default value of weight is 1, and the results showcase the highest AUC being 0.78 at two different weights, the lowest weight has been considered, which is 3.

Table 6.6.2 The Performance of using different weights, in combination with Maximum Depth = 1 and n Estimator = 15



Max Depth = 6 Weight	n Estimator = 15 AUC
1	0.779
2	0.779
3	0.780
4	0.776
5	0.778
7	0.777
10	0.778
15	0.780
20	0.780
25	0.777
30	0.779

The last two parameters to be determined are the Learning Rates and the Regularization Term. These two are determined by simply adding learning rates and regularization terms to the algorithm and finding the values that provide the highest performances. From Table 6.3.1 it is concluded that using a Learning Rate of 0.1 provides the highest performance. Further, from Tables 6.6.2 and 6.6.3, it is concluded that a regularization term of 0.1 provides the highest performances.

Table 6.6.1: The Performances of using different values of Learning Rates

Learning Rate	AUC
0.1	0.777
0.01	0.772
0.001	0.759
0.0001	0.759

Table 6.6.2: The Performances of using different values of Regularization Terms

Regularization Term	AUC
0.1	0.778
0.01	0.777
0.001	0.777
0.0001	0.777

Table 6.6.3: Further inspection on the Regularization Term. Regularization Term closer to the Optimum value = 0.1

Regularization Term	AUC
0.05	0.777
0.08	0.778
0.1	0.778
0.12	0.778
0.15	0.778

The parameters that have been selected for the XGBoost are as follows.

Table 6.6.4 Parameters selected for XGBoost.

Parameters	Selected
Maximum Depth	6
n Estimators	15
Weights	3
Learning Rate	0.1
Regularization Term	0.1

6.7 Tested Results

The last step in determining the different algorithms performances is verifying the performances with the test set. The 70% of the data that has now been trained is applied to the 30% test set, and the performance is then determined for all algorithms. This is essentially the result performance of the different algorithms. In the table below the performances of testing the models are showcased.

Table 6.7.1 The performances of the algorithms when determined using the test set

Algorithm	AUC
Logistic Regression	0.762
Neural Network	0.790
k Nearest Neighbors	0.760
Decision Tree	0.784
Random Forest	0.766
XGBoost	0.783

6.8 Data Influences and Feature Importance

Now that the parameters for all algorithms have been determined, further experimentation of which data and variables have most influence and are the most important. This section's experiments will mostly consist of testing the separated personal categorical features, marriage and education. This is done to see whether there are any differences between the groups and if the differences influence the algorithm performance. Further, the feature importance will be checked for each algorithm. This will be done to examine if there are any inconsistencies or significant differences between the algorithms.

6.8.1 Marital Features

Firstly, referring back to Table 5.5 from the data description, we see the summary of the marital status. There are four possible values, being “Other”, “Married”, “single” and “Divorced”. Whereas roughly 98.7% of clients are either married or single. Of the two categories that roughly contain 1.3% of the data, the “Other” category only has 54 samples, being the smallest category. Containing such a small number of values makes the determination of influence difficult. Hence I will not take this category into consideration when experimenting. With Divorce being the category containing the second lowest value, with only 323 samples, my hypothesis is that the algorithm's performance will stay lower for this category compared with the other two.

Table 6.8.1 Summary of the Marital category.

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

Below, the resulting performances of the algorithms are showcased when the samples only belonging to the stated marital status are used. Overall, the performances are just below the original algorithm performances and stay relatively consistent throughout all categories. Even though some algorithms perform lesser with the Divorced category, the differences are very small, not confirming nor denying my hypothesis. The reason for the difference between the original and the categories being small might be for the simple reason that the algorithm features importance, and whether the marital features are not as important overall. This will be analyzed further at a later stage.

Table 6.8.2 Performances of the algorithms, when the samples only belonging to the stated marital status is used.

Algorithm	Original	Married	Single	Divorced
Logistic Regression	0.762	0.759	0.759	0.759
Neural Network	0.790	0.778	0.777	0.769
k Nearest Neighbors	0.760	0.746	0.745	0.745
Decision Tree	0.784	0.753	0.753	0.753
Random Forest	0.766	0.764	0.754	0.763
XGBoost	0.783	0.776	0.776	0.776

6.8.2 Educational Features

Referring back to Table 5.5 from the data description, the summary of the educational category status, there are 4 different possible values. These being “Other”, “Graduate School”, “University” and “High School”. The “other” category only consists of 1.56%, or 468 clients, of the dataset, and just as with the marital category, this category is hypothetically set to produce the lowest overall performances throughout all algorithms. The two highest valued categories are the university and graduate school, consisting of 46.77% and 35.28% of the dataset, respectively. The “high School” category, however, consists of the remaining 16.39% of the dataset, which should be a significant amount and should produce decent performances throughout all algorithms.

Below, the resulting performances of all the algorithms are displayed, when the samples only belonging to the stated educational status are used. The results are almost as conclusive as with the marital category. The performances are just below the original algorithms. The only difference here is that no feature that stands out in being different.

They are all relatively producing the same performances, just below the original algorithm. This again, might be due to the algorithms feature importance.

Table 6.8.3 Performances of the algorithms, when the samples only belonging to the stated educational status is used.

Algorithm	Original	Graduate School	University	High School	Other
Logistic Regression	0.762	0.759	0.758	0.759	0.760
Neural Network	0.790	0.782	0.778	0.778	0.778
k Nearest Neighbors	0.760	0.739	0.738	0.741	0.742
Decision Tree	0.784	0.754	0.753	0.754	0.754
Random Forest	0.766	0.764	0.764	0.763	0.764
XGBoost	0.783	0.776	0.776	0.776	0.776

6.8.3 Feature Importance

The last analysis I will endure in this chapter is the feature importance of all algorithms. This implies checking which features each algorithm found as most important and lesser important. Doing this will verify whether the personal categorical features are highly important. My hypothesis is that this method will provide an explanation that both the marital and educational features are not as relied on.

This is simply done by importing relevant packages and applying certain python code(Permutation_Importance) to each algorithm. The permutation importance reveals which features the algorithm relies most upon in their predictions. An example of a feature importance code has been added to the appendix. The results are displayed in Table 6.8.1 below.

Table 6.8.4 Feature Importance, top three of each algorithm

Algorithm	Most Important	Second Most	Third Most
Logistic Regression	Limit Balance	Pay Amount 2	Bill Amount 1
Neural Network	Pay Amount 2	Pay Amount 1	Limit Balance
k Nearest Neighbors	Limit Balance	Pay Amount 1	Bill Amount 1
Decision Tree	Pay Amount 1	Pay Amount 2	Limit Balance
Random Forest	Pay Amount 1	Pay Amount 2	Limit Balance
XGBoost	Limit Balance	Bill Amount 1	Pay Amount 2

From the results, there is a clear selection of which features are most important and most relied upon. The four most important features are ‘Limit Balance’, ‘Pay Amount 1’ and

‘Pay Amount 2’, and ‘Bill Amount 1’, in no particular order. This proved my hypothesis to be right; both educational and marital features are not among any top three features for any algorithms, which proves why there are so few differences in performance for each subfeature in each category. However, this does not conclude that both categorical features should be left out of the original algorithms, referring to the original algorithms performing better with the features than without.

7.0 Conclusions

This research project's main goal was to provide the performances of six different machine learning algorithms when they are used to predict default loans. This research has been finalized using a dataset containing 30,000 Taiwanese credit lenders. The main question for this thesis was: “How effective are machine learning algorithms in predicting default in loans?”, which this conclusion mainly will answer. Throughout the research, the Area Under the Curve has been used as the primary source of performances, but to make better conclusions and comparisons between the algorithms, Recall, Precision, and Accuracy have been added for the possibility of assessing the performances in more detail. In addition, the confusion matrix for each algorithm has been added to the appendix. Table 7.1 showcases all the above-mentioned performances of all the algorithms, when trained and validated on the original dataset.

Table 7.1 Summary of the performance of all the different algorithms when trained and validated on the original dataset.

Algorithm	AUC	Recall	Precision	Accuracy
Logistic Regression	0.762	0.34	0.67	0.816
Neural Network	0.790	0.34	0.68	0.820
k Nearest Neighbors	0.760	0.33	0.63	0.815
Decision Tree	0.784	0.38	0.60	0.819
Random Forest	0.766	0.30	0.67	0.815
XGBoost	0.783	0.37	0.64	0.822

From the results, the performances are relatively similar. The AUC scores span between the algorithms where only 0.03. The k Nearest Neighbor performed lowest with an AUC of 0.760, while the Neural Network had the highest AUC of 0.790. Even though concluding that Neural Network performs best out of all the algorithms, all performances are so closely resembling that drawing this conclusion based on AUC, without taking further research into account, might be difficult.

There are more important measures to consider when concluding between algorithms, such as recall, precision, and accuracy. When looking at the recall and precision scores between the algorithm, some changes in performances are made. When the goal is to reach the highest recall, the Decision Tree has the highest score, with a recall score of 0.38, however, the Decision Tree has the lowest precision score among all algorithms. With Decision Tree having the lowest performing precision (0.60), the Random Forest

algorithm has the second highest, providing a precision score of 0.67, but, yet again, a high scorer of precision has the lowest recall score (0.30). To separate and compare the two, looking at the accuracy score might shed a light on which performance should be preferred. Both the Decision Tree and Random Forest algorithms have similar accuracy, with only +0.04 higher accuracy separating the Decision Tree from the Random Forest, with the latter having an accuracy of 0.815.

From the results, when looking at the accuracy, there is very little that separates the algorithms. The highest scorer, however, is the XGBoost with an accuracy of 0.822, and the second highest having yet again a high score, the Neural Network. Interestingly, the Decision Tree only lies 0.001 behind the Neural Network, having a strong accuracy score as well. The algorithms with the lowest accuracy score are the Random Forest and the k Nearest Neighbors, sharing an accuracy score of 0.815. While all the algorithms have a decent accuracy score, all being in a range of 0.1 accuracy score between each other, further examination has been made.

Explainability must be considered, which changes the view a little on which algorithms perform best. The Decision Tree, for example, is one of the most explainable algorithms. The classifications are explainable and can easily be graphed and plotted. However, when looking at algorithms and explainability, the choice must be made whether explainability 'beats' performances. For the sake of the Decision Tree being highly explainable, an example of the Decision Tree is made. The Decision Tree which again, has a high explainability, has a high AUC score, only being 0.06 behind the highest AUC value. Further, the Decision Tree has the highest Recall, but the lowest Precision, as well as a decent Accuracy score. Concluding whether to use a highly explainable algorithm or a performance-based algorithm all depend on the situation.

Based on the analysis of the findings, I can now put an end to the main question, machine learning algorithms are highly effective in predicting default loans. Overall, the models have on average 81.7% accuracy, which is good, with only 18.3% being misclassification. I have, in this case, concluded that both the Neural Network and the Decision Tree are the best algorithms for predicting default. Both of these algorithms share the first and second place for the AUC score. While the Decision Tree has the highest Recall score, the Neural Network has the highest Precision score. They both share high Accuracy scores as well. The most common disadvantage of these two algorithms is that they both

are prone to overfitting, as well as the Neural Network tends to become a ‘black box’, due to its challenges to interpret and understand.

8.0 Further Research

In a bank's management, simulation plays a big part in decision-making. Simulation, predicting default, serves as a primary factor for accurately assessing a bank's resilience against economic shocks, such as bankruptcy. Using fine-tuned machine learning algorithms will benefit banks in credit risk management. However, implementing machine learning is heavily detained by the GDPR. This is due to the protection of the customers and is strictly following the development and deployment of AI and Machine learning. GDPR art 22 states, "The data subject shall have the right not to be subject to a decision based solely on automated processing", which means that no customers can be determined as high risk without human intervention.

Many different machine learning algorithms are biased or working like a 'black box'. This new possible further research aims at the possibility of removing this black box paradox. The algorithms are prone to decision making based on such features like gender and ethnicity. It is difficult to know exactly how and why these machine learning algorithms do this but being able to reduce or even remove this flaw, will increase performances, as well as increase credibility to use machine learning algorithms more in depth. Just as mentioned, this is one huge reason for the detainment of the GDPR, which solely stops decision making based on automated processing, or machine learning, in this case.

There might be many possibilities in making the algorithms non bias, and better performing. Research on combining multiple algorithms is limited, and researching the possibilities in making new algorithms, sound interesting. Algorithms with higher performances and fewer flaws. A though might be combining the Neural Network and Decision Trees, which are the two top performances in this experiment.

References

Abellan, Joaquin & Mantas, J. Carlos. (2014) “Improving experimental studies about ensembles of classifiers for bankruptcy prediction and credit scoring.” In: *Expert Systems with Applications (*p. 3825-3830)

Alaraj, Maher. Abbod, Maysam. & Hunaiti, Ziad. (2014) “Evaluating consumer loans using neural networks ensembles.” In: *International Institute of Engineers*

Alpaydin, Ethem (2010) “Introduction to Machine Learning, Second Edition”

Anderson, R. (2007) “The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation.”

Berrar, Daniel (2019) “Cross-Validation”

Bentley, J. Louis. (1975) “Multidimensional Binary Search Trees Used for Associative Searching.”

Bhatia, Nitin. (2010) “Survey of nearest neighbor techniques.”

BIS, (n.d.) “Background to the Basel Framework.” Retrieved from: https://www.bis.org/baselframework/background.htm

BIS, (2023) “Basel Committee on Banking Supervision” In: *The Basel Framework.*

Breiman, Leo. (2001) “Random Forests.” In: *Machine Learning*

Brock, Thomas, & Eichler, Ryan (2022) “Credit Risk: Definition, Role of Rating, and Examples.”

Brown, Sara (2021) “Machine Learning, Explained.” From: MIT Sloan School of Management, Cambridge.

Brownlee, Jason (2020) “How to Calculate Feature Importance with Python”

Ceballos, Frank (2019) “Scikit-Learn Decision Trees Explained”

Chen, Tianqi, & Guestrin, Carlos (2016) “XGBoost: A Scalable Tree Boosting System.” In: *International Conference of Knowledge Discovery and Datamining* (p. 785-794)

Crook, N. Johnatan, Edelman, B. David., & Lyn, C. Thomas, (2007) “Recent developments in consumer credit risk assessment.” In: *European Journal of Operational Research* (p. 1447-1465)

Du, Ke-Lin. & Swamy, M. N. S. (2014) “Neural Networks and Statistical Learning.”

Fawcett, Tom (2006) “An introduction to ROC analysis” In: *Pattern Recognition Letter* (p. 861-874)

FICO, (2022) “Frequently Asked Questions About FICO Scores.”

Fushiki, Tadayoshi (2011) “Estimation of prediction error by using K-fold cross-validation”

Hand, J. David, & Henley, E. William (1996) “Statistical Classification Methods in Consumer Credit Scoring: a Review.” In: *Journal of the Royal Statistical Society*

Hand, J. David (2006) “Classifier Technology and the Illusion of Progress.”

Hullermeier, Eyke & Vanderlooy, Stijn (2008) “A critical analysis of variants of the AUC”

Ibm (n.d.) “What is machine learning”

Kattan, W. Michael & Cowen, E. Mark (2009) “Encyclopedia of Medical Decision Making.” (p. 326)

Khandani, E. Amir, Adlar, J. Kim, & Lo, W. Andrew. (2010) “Consumer Credit Risk Models via Machine Learning Algorithms.” In: *Journal of Banking & Finance* (p. 2767-2787)

Langley, Pat & Simon, A. Herbert. (1995) “Application of Machine Learning and Rule Introduction.”

Larose, Daniel T. (2005). Discovering knowledge in data: an introduction to data mining

Maimon, Oded, & Rokach, Lior. (2010) “Introduction to supervised methods” In: *Data Mining and Knowledge Discovery Handbook, second edition.* (p. 149-164)

Mitchell, M. Tom. (1997) “Machine learning”

Molnar, Christoph. (2021) “Interpretable Machine Learning. A Guide for Making Black Box Models Explainable”

Mueller, J. Paul, & Massaron, Luca. (2016) “Machine Learning for Dummies. 2nd Edition”

Provost, Foster & Fawcett, Tom. (2013) “Data Science for Business.”

Violante, Andre, SAS, (2019) “Building credit scorecards using SAS and Python.”

Zhang, G. Peter (2009) “Neural Networks for data Mining”. In: *Data Mining and Knowledge Handbook*, (p. 419-444)

Zhang, G. Peter & Min Qi (2002) “Predicting consumer retail sales using neural networks” In *Neural Networks in Business*, (p. 26-40)

A List of Ten, From Crook et. al.:

Baesens, Bart. (2003) PhD thesis; “Developing Intelligent Systems for Credit Scoring Using Machine Learning Techniques”

Boyle, M., Crook, J.N., Hamilton, R., & Thomas, L.C. (1992) “Methods for credit scoring applied to slow payers.”

Desai, V.S., Conway, D.G., Crook, J.N., & Overstreet, G.A. (1997) “Credit scoring models in the credit union environment using neural networks and genetic algorithms.

Henley, W.E. (1995) “Statistical Aspects of Credit Scoring. Ph.D. thesis, Open University.”

Lee, T. S., Chiu, C.C., Lu, C.J., & Chen, I.F. (2002) “Credit scoring using the hybrid neural discriminant technique.”

Malhotra, R., & Malhotra, D.K. (2003) “Evaluating consumer loans using neural networks.”

Ong, C.S., Huang, J.J., & Tzeng, G.H. (2005) “Building credit scoring systems using genetic programming.”

Srinivisan, V., & Kim, Y.H. (1987) “Credit granting a comparative analysis of classificatory procedures.”

West, D. (2000) “Neural network credit scoring models.”

Yobas, M.B., Crook, J.N., & Ross, P. (2000) “Credit scoring using neural and evolutionary techniques.”

Appendix

Confusion Matrices

Logistic Regression

	Predictive Positive	Predicted Negative
Positive	669	337
Negative	1291	6703

Neural Network

	Predicted Positive	Predicted Negative
Positive	697	321
Negative	1298	6684

k Nearest Neighbor

	Predicted Positive	Predicted Negative
Positive	700	347
Negative	1268	6685

Decision Tree

	Predicted Positive	Predicted Negative
Positive	732	500
Negative	1216	6552

Random Forest

	Predicted Positive	Predicted Negative
Positive	593	296
Negative	1267	6744

XGBoost

	Predicted Positive	Predicted Negative
Positive	718	378
Negative	1230	6674

Table 6.3.1: The Performance of both Criterions with different Maximum Depth

	Gini	Entropy
Max Depth	AUC	AUC
0	0.601	0.617
2	0.685	0.685
4	0.736	0.742
5	0.753	0.751
6	0.749	0.756
7	0.747	0.754
8	0.741	0.747
10	0.716	0.729
12	0.682	0.701
14	0.644	0.672

Table 6.4.1: The Performances of Maximum Depth = 1, using the three different Maximum Features, in relation to n Estimators.

Maximum Depth = 1			
		AUC	
n Estimators	n	Sqrt n	Log2 n
1	0.639	0.614	0.602
3	0.639	0.635	0.702
5	0.639	0.730	0.680
7	0.639	0.682	0.728
10	0.639	0.737	0.733
12	0.639	0.740	0.738
15	0.639	0.754	0.749
20	0.639	0.749	0.723
25	0.639	0.751	0.756
30	0.639	0.752	0.758
35	0.639	0.750	0.757

Table 6.4.2: The Performances of Maximum Depth = 1, using the three different Maximum Features, in relation to n Estimators.

Maximum Depth = 6			
		AUC	
n Estimators	n	Sqrt n	Log2 n
1	0.704	0.705	0.701
3	0.759	0.753	0.748
5	0.761	0.764	0.759
7	0.763	0.764	0.763
10	0.767	0.768	0.764
12	0.768	0.766	0.768
15	0.769	0.769	0.768
20	0.769	0.769	0.767
25	0.769	0.769	0.770
30	0.769	0.769	0.770
35	0.767	0.766	0.770

Table 6.4.3 The Performance of Estimator = 5, using the three different Maximum Features, in relation to Max Depth

5 Estimators			
	n	Sqrt n	Log2 n
Maximum Depth	AUC	AUC	AUC
1	0.726	0.704	0.689
3	0.755	0.750	0.755
5	0.761	0.760	0.761
6	0.766	0.765	0.756
7	0.764	0.762	0.761
10	0.754	0.760	0.759
12	0.742	0.749	0.741
15	0.736	0.720	0.726
20	0.722	0.708	0.713
25	0.706	0.699	0.691
30	0.702	0.695	0.695
35	0.703	0.706	0.706

Table 6.4.4 The Performance of Estimator = 5, using the three different Maximum Features, in relation to Max Depth

10 Estimators			
		AUC	
Maximum Depth	n	Sqrt n	Log2 n
1	0.748	0.711	0.707
3	0.760	0.755	0.747
5	0.764	0.764	0.763
6	0.766	0.764	0.766
7	0.767	0.768	0.766
10	0.766	0.764	0.763
12	0.759	0.763	0.759
15	0.744	0.753	0.748
20	0.743	0.730	0.727
25	0.725	0.723	0.725
30	0.731	0.721	0.727
35	0.720	0.727	0.726

Table 6.6.1: The Performances of using different n Estimators in combination with Maximum Depth = 1 and 6.

	Maximum Depth = 1	Maximum Depth = 6
n Estimators	AUC	AUC
1	0.645	0.759
3	0.707	0.773
5	0.718	0.775
7	0.724	0.776
10	0.748	0.776
12	0.759	0.777
15	0.762	0.779
20	0.764	0.776
25	0.767	0.776
30	0.768	0.776
35	0.769	0.775
40	0.770	0.774
50	0.771	0.770
100	0.774	0.762
500	0.775	0.750
1000	0.774	0.745

Table 6.6.2 The Performance of using different weights, in combination with Maximum Depth = 1 and n Estimator = 15

Maximum Depth = 6 Weight	n Estimator = 15 AUC
1	0.779
2	0.779
3	0.780
4	0.776
5	0.778
7	0.777
10	0.778
15	0.780
20	0.780
25	0.777
30	0.779

An Example of Permutation Feature Importance code in Python (Brownlee, J. 2020)

```
# permutation feature importance with knn for regression
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.inspection import permutation_importance
from matplotlib import pyplot

# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)

# define the model
model = KNeighborsRegressor()

# fit the model
model.fit(X, y)

# perform permutation importance
results = permutation_importance(model, X, y, scoring='neg_mean_squared_error')

# get importance
importance = results.importances_mean

# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```