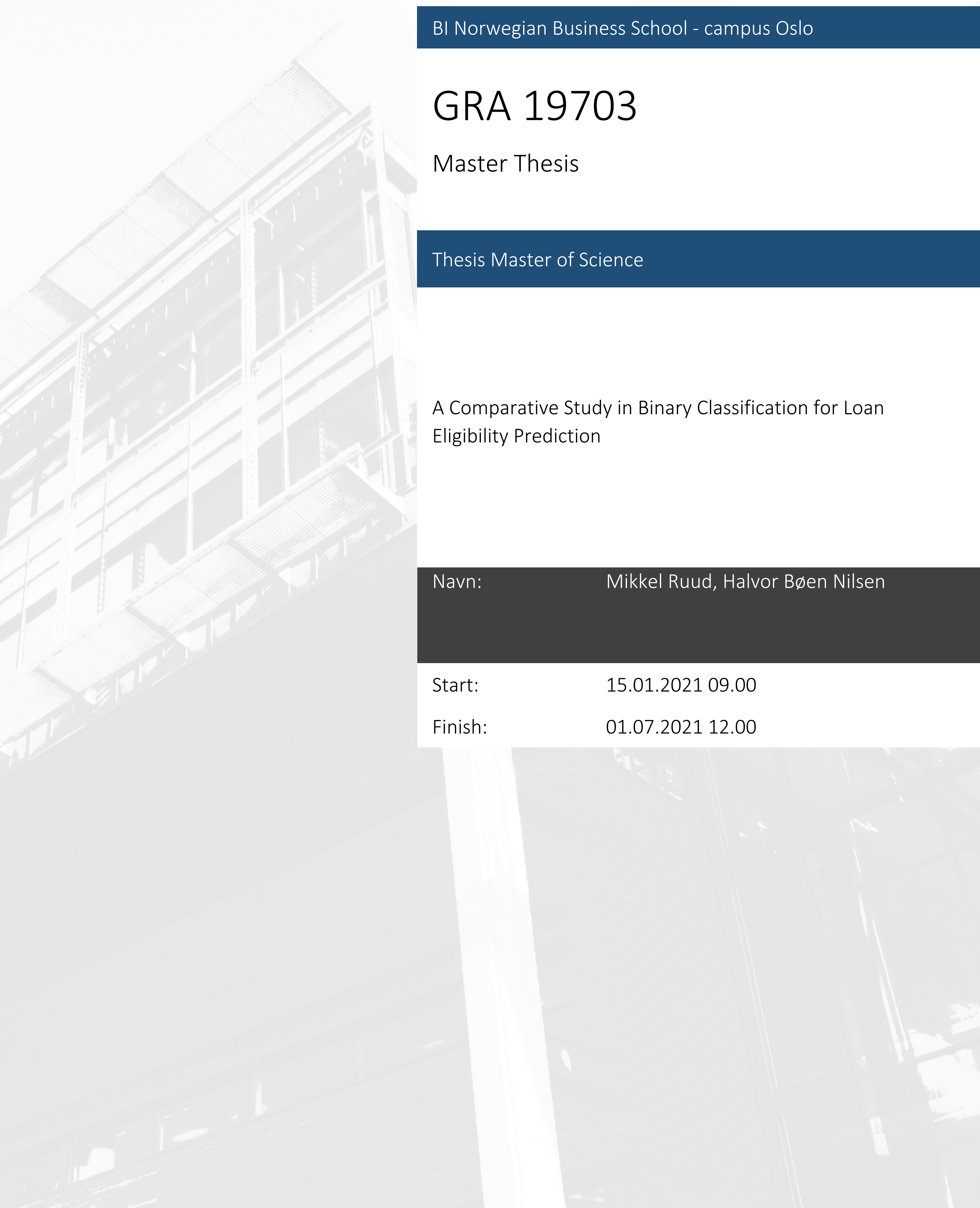


BI Norwegian Business School - campus Oslo

GRA 19703

Master Thesis

Thesis Master of Science

A Comparative Study in Binary Classification for Loan Eligibility Prediction

Navn:	Mikkel Ruud, Halvor Bøen Nilsen
Start:	15.01.2021 09.00
Finish:	01.07.2021 12.00

Abstract

In this thesis, loan eligibility prediction is explored and analyzed by investigating five different prediction models. The goal for the prediction models is to accurately predict whether a bank loan is approved or disapproved. The dataset utilized for this thesis is retrieved from kaggel, and is referred to as “bank loan data”. It contains about 100.000 rows of various loan applications, with the predictor variable available. Throughout this thesis, we will investigate five supervised learning algorithms, more specifically, Support Vector Machine, K-Nearest Neighbors, Decision Tree, Logistic Regression, and Stochastic Gradient Descent. The results are evaluated using various performance measures, and is compared to similar research within the same topic.

Acknowledgments

We have truly enjoyed working on this thesis throughout this semester. Both contributors have cooperated well, while building on each other's strengths and weaknesses. We would like to thank our supervisor Mass Soldal Lund, who has helped and guided us through this thesis. In the end, it's been a great experience, and we wouldn't trade it for anything.

Table of Contents

Contents

Abstract i

Acknowledgments ii

List of Figures v

List of Tables vi

1. Introduction 1

1.1 Motivation for research 3

1.2 Research question 3

1.3 Thesis structure 4

2 Research methodology 5

2.1 Selection of algorithms 5

2.2 Performance measures 6

2.3 Data collection 8

2.4 Pre-Processing: 9

2.4.1 Data cleaning: 10

2.4.1.1 NaN values in “loan_repaid”: 10

2.4.1.2 Feature exclusions: 10

2.4.2 Multicollinearity 11

2.4.3 Feature engineering: 13

2.4.4 Feature scaling 13

2.4.5 Imbalanced class 14

2.4.6 Data splitting 15

3 Method section 16

3.1 Supervised learning 16

3.2 Common notations 17

3.3 Technique for Optimizing Hyperparameters 17

3.4 Support Vector machine 18

3.4.1 Hyperparameters tuning 19

3.4.2 Results 21

3.5 Logistic regression 23

3.5.1 Hyperparameters tuning 24

3.5.2 Results 26

3.6 K-Nearest Neighbors 28

3.6.1 Hyperparameters tuning 29

3.6.2 Results 31
3.7 Decision Tree Classifier 32
3.7.1 Pruning 34
3.7.2 Hyperparameters tuning 34
3.7.3 Results 35
3.8 Stochastic gradient descent 37
3.8.1 Hyperparameters tuning 38
3.8.2 Results 39
4 Comparing Results and Additional Findings 41
5 Discussion and Conclusion 43
5.1 Related Work 43
5.2 Summary 45
5.2 Impact on the banking industry 46
5.3 Further research 47
6 Bibliography 49

List of Figures

Figure 1 Structure for the thesis 4
Figure 2 Confusion matrix 7
Figure 3 Feature description 8
Figure 4 VIF score original dataset 11
Figure 5 VIF score after removing features dataset 12
Figure 6 Count of target variable 15
Figure 7 Optimal Hyperplane 19
Figure 8 Confusion matrix on test dataset 23
Figure 9 Illustration of logistic regression 23
Figure 10 Confusion matrix from test dataset 28
Figure 11 Illustration on how the KNN algorithm work 28
Figure 12 Overfitting using small number of neighbors 30
Figure 13 Confusion matrix test dataset 32
Figure 14 Illustrated a decision tree with one split 33
Figure 15 Confusion matrix test dataset 37
Figure 16 Illustrates the movement of the gradient through the iterations 38
Figure 17 Confusion matrix on test dataset 41

List of Tables

Table 1 Selection of algorithms 6
Table 2 Feature Importance 11
Table 3 Feature engineering 13
Table 4 SVM Hyperparameters 20
Table 5 Performance score on validation dataset without tuning 21
Table 6 Performance score on validation dataset with tuning 22
Table 7 Performance score on test dataset with tuning 22
Table 8 Logistical regression Hyperparameters 25
Table 9 Performance scores on validation dataset without tuning 26
Table 10 Performance score on validation dataset with tuning 26
Table 11 Performance score on test dataset 27
Table 12 Result from the GridsearchCV 27
Table 13 KNN Hyperparameters 30
Table 14 Performance score on validation dataset without tuning 31
Table 15 Performance score on validation dataset with tuning 31
Table 16 Performance score on test dataset 32
Table 17 Decision tree Hyperparameters 35
Table 18 Performance score on validation dataset without tuning 35
Table 19 Performance score on validation dataset with tuning 36
Table 20 Performance score on test dataset 36
Table 21 SGD Hyperparameters 39
Table 22 Performance score on validation dataset without tuning 39
Table 23 Performance score on validation dataset with tuning 40
Table 24 Performance score on test dataset 40
Table 25 Accuracy from the algorithms 41
Table 26 Accuracy comparison of previous research 45

1. Introduction

It is fairly common that financial banks around the world perform various internal tasks manually. This may be very time consuming and can yield high cost in the long run. However, these types of applications and data are heavily number based, which means that it’s also prone to Machine Learning. More often than not, banks utilize manual labor when it comes to loan eligibility i.e approval/disapproval. This is a process that is very eligible for Machine Learning and automatization, because of its numeric nature. We wanted to investigate this topic further, and dive deep into the models and algorithms that surround this specific subject.

The loan application process has had a drastic change in the last decades. For many years, people who needed a loan had to first go to a bank, speak with a loan officer about possible credit options, fill out an application, spend days looking for all required paperwork, and then wait for the approval (Hope, 2020). The approval could take weeks since the verification was primarily done manually. Nowadays, borrowers don’t need to have personal contact with a bank representative to get a loan. It all starts online, where borrowers can fill out an application and get immediate approval. The loan approvals are generally calculated with the metrics loan-to-value ratio and/or debt-to-income ratio, where each country has different ratios of acceptance. With the advancement in technology, predictive analytics have become more important and accessible. Predictive analytics uses algorithms, data, and machine learning techniques to analyze the likelihood of a future outcome based on historical data, which can help banks identify patterns of characteristics that are likely to default (SAS, n.d.).

With the rise of big data, the improvements in data availability, technology, data analysis, and customer relationship management skyrocket. Businesses can now save their consumer data in their data warehouses and do their data analysis in-house, thanks to advancements in technology and lower computing power costs. Data in massive amounts can offer invaluable insights and a competitive edge if the right technological and organizational resources support them (Côrte-Real et al., 2017). Most organizations are currently collecting and storing data that can be used in statistical software to help them better understand their consumer base and forecast future behavior. For example, with access to modern analysis tools, banks can move from the generally loan-to-value ratio debt-to-income ratio calculation to predicting loan defaults ahead of time.

If these banks could acquire a reliable and stable machine learning model to correctly predict whether a bank loan application should be granted or not, there is potentially a lot of time and money to be saved in the long run. If this process is automated, banks could focus their workforce on other important aspects within the company. This would ultimately enhance the productivity and efficiency within the company. In recent years, Machine Learning (ML) and automation has become a new standard for both small businesses and corporations, who want to develop further into the world of technology.

Our goal is to create and compare various ML models and algorithms connected to this automated process, and describe the benefits and disadvantages connected to the various combinations. We will first compare our individual results to each other for binary classification in loan eligibility, before comparing it to previous research within the same research topic.

This thesis is not only applicable for binary classification in loan eligibility, and can also be utilized for other binary classification problems. By investigating the method section, one might be able to capture valuable information that might help picking the most fitting algorithm for their specific situation.

1.1 Motivation for research

Our thesis focuses on comparing various Machine Learning algorithms and models for loan eligibility prediction. This can be applicable to other binary classification problems as well. The reasoning for choosing this topic is as follows:

- This thesis will provide valuable information for individuals/companies who want to develop a binary classification model for loan eligibility.
- Limited research and empirical studies for comparative studies in this specific area.
- Gain valuable insight into each algorithm and the inner workings of each model.

1.2 Research question

Our thesis dives into each relevant algorithm and discusses each model, how it operates and behaves, and the various results. We will elaborately compare and discuss the results before we discuss various aspects of our thesis. We will also make a comparison to other research.

Our research question is as follows:

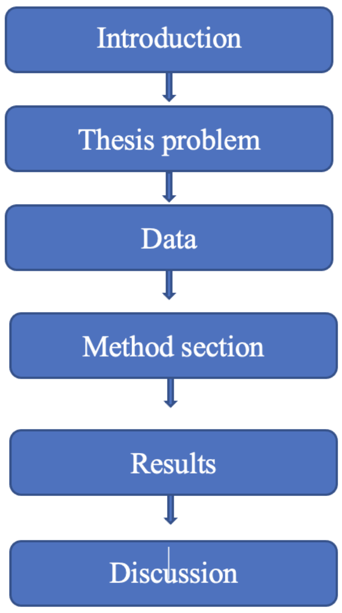
” For a chosen set of machine learning algorithms, which algorithm demonstrates the best performance in loan eligibility classification prediction with regards to several model evaluation metrics?”

In order to answer this question, we will train, tune and test five Machine Learning algorithms and compare the performance of those algorithms to each other. In addition, we will compare those models to other related research within the same topic as well.

1.3 Thesis structure

Figure 1 represents the structure of our thesis, where the first part includes an introduction to the study and motivation for research. The second part introduces the thesis problem and research question. The third part represents the data used for this thesis, the pre-processing steps, feature engineering, data splitting, and unbalanced classes. For the fourth section, we will go through each algorithm of our choice and discuss the model, tuning, and the individual results. For the results section, we will compare the results and models from the latter section. Finally, we will discuss various aspects of this thesis, such as strengths and weaknesses, the impact this has on the banking industry, and some further research of applying a loan eligibility prediction model.

Figure 1 Structure for the thesis



2 Research methodology

The research methodology for our thesis is split into four main parts. First, we explain why we select these specific machine learning techniques to predict loan approvals. The next part describes various performance measures used to evaluate the algorithms, such as accuracy, recall, and precision, making it possible to rank the different algorithms. The third main part describes the different features in the data set and the data preprocessing used to transform the raw data into a valuable and efficient format. Where we first in the data preprocessing explain the necessary data cleaning processes and feature engineering. Further, we use the variance inflation factor to test the data set for multicollinearity. Another technique used is feature scaling which standardizes the independent features. In addition, we utilize a synthetic minority upsampling technique to deal with imbalances in our dataset. Lastly, we explain the train validation test split, which prevents the model from overfitting and accurately evaluating the model. The last part describes how the different machine learning algorithms operate. Moreover, we discuss the different hyperparameters and their representative optimal values. In the end, we evaluate the model's results.

2.1 Selection of algorithms

Table 1 lists and shortly describes the predictive algorithm commonly employed in the literature on machine learning methods for loan approval (Teply, P., & Polena, M. 2020). In addition, we wanted to include something different than supervised learning to see how it affected the results. Hence, we choose to add a Stochastic gradient descent algorithm, which is also used in the literature of loan approval (Nabende et al., 2019). Each classification technique will be detailed in chapter 3, with a focus on their unique hyperparameters.

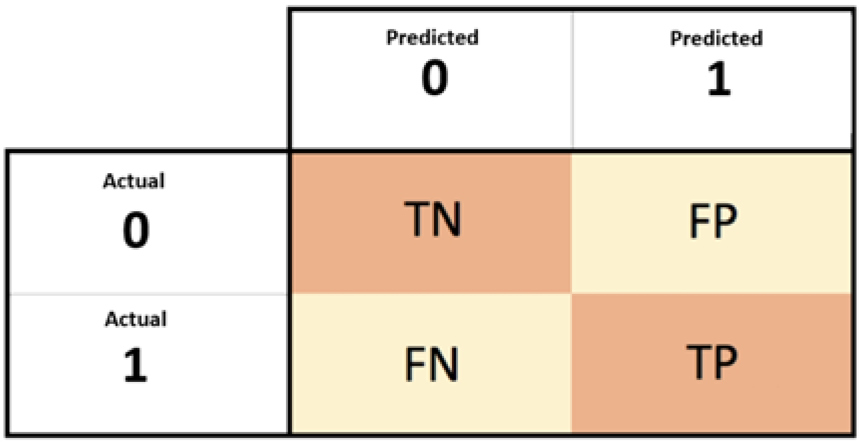
Table 1 Selection of algorithms

Machine learning algorithms	**Description**
Decision trees	The data is partitioned via a series of branching processes.
Support Vector Machines	Fitting hyperplane in an N-dimensional space that distinctly classifies the data points
K-Nearest Neighbors	Stores all available cases and classifies new cases based on a similarity measure.
Logistic regression	Is used to model the probability of a certain class or event
Stochastic gradient descent	Optimization method to find a local minimum of a function

2.2 Performance measures

One typical way to evaluate the performance of a model in binary classification is to use a confusion matrix. The instance of accepted loan approval is defined as positive and rejected loan approvals as negative. The potential outcomes are then true positive (TP) if the model correctly predicts approved loans and false positive (FP) if rejected loan approvals are predicted as approved loans. True negative (TN) if rejected loan approvals are correctly predicted, and false-negative (FN) if accepted loan approvals have been predicted as rejected loan approvals, as illustrated in figure 2.

Figure 2 Confusion matrix



	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Figure 2. «Confusion Matrix », (n.d.).

Several metrics can be derived from a confusion matrix. The most frequent metric is accuracy, which is the fraction of predictions the model predicted correctly. Accuracy is calculated by (2.2.1) (Google, n.d.).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2.1}$$

In terms of business sense, the goal is to balance the cost of losing money on non-performing consumers and the opportunity cost of losing a potentially profitable customer (Huilgol, 2021). As a result, it's crucial to investigate how alternative strategies affect recall and precision since recall measures how many accepted loan approvals the model captures. In contrast, precision refers to the possible opportunity cost. Recall and precision are defined in function 2.2.2 and 2.2.3.

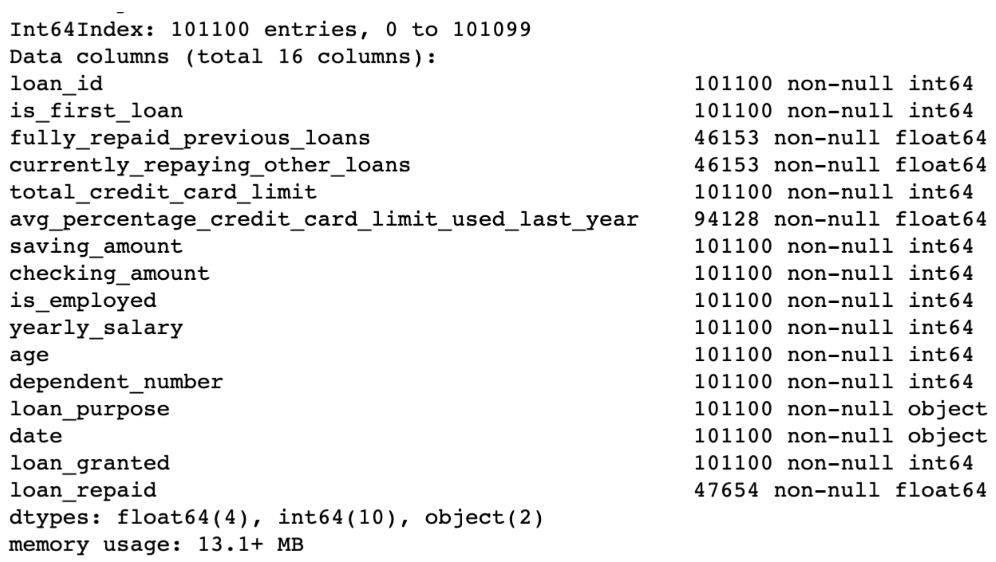
$$Recall = \frac{TP}{TP + FN} \tag{2.2.2}$$

$$Precision = \frac{TP}{TP + FP} \qquad (2.2.3)$$

2.3 Data collection

Our data is composed of two datasets, “borrower_table.csv” and “loan_table.csv”. It is collected from Kaggle (*Bank Loan Data*, 2020) and contains 101.100 rows of various loan applications collected by the given bank. There are two datasets which are named “borrower_table.csv” and “loan_table.csv”. “borrower_table.csv” contains various features (12) based on the borrower/applicant, and the “loan_table.csv” dataset contains five features including the purpose of the loan, whether the loan was granted or not, and whether the loan is repaid or not. We can merge these two datasets together in order to create one complete dataset. In addition, these datasets are fairly new and unused, and date back to June 2020. Below in figure 3 is metadata for the combined bank loan dataset.

Figure 3 Feature description



```
Int64Index: 101100 entries, 0 to 101099
Data columns (total 16 columns):
loan_id                                      101100 non-null int64
is_first_loan                                101100 non-null int64
fully_repaid_previous_loans                  46153 non-null float64
currently_repaying_other_loans               46153 non-null float64
total_credit_card_limit                      101100 non-null int64
avg_percentage_credit_card_limit_used_last_year    94128 non-null float64
saving_amount                                101100 non-null int64
checking_amount                              101100 non-null int64
is_employed                                  101100 non-null int64
yearly_salary                                101100 non-null int64
age                                          101100 non-null int64
dependent_number                             101100 non-null int64
loan_purpose                                 101100 non-null object
date                                         101100 non-null object
loan_granted                                 101100 non-null int64
loan_repaid                                  47654 non-null float64
dtypes: float64(4), int64(10), object(2)
memory usage: 13.1+ MB
```

The original datasets had the following features:

- loan_id: an identification number representing a loan application
- is_first_loan: whether it’s the appliers first loan or not (0/1)
- fully_repaid_previous_loans: whether the applicant has repaid its previous loans or not.
- currently_repaing_previous_loans: whether the applicant is currently repaying other/previous loans.
- total_credit_card_limit: the applicant's current credit card limit.
- avg_percentage_credit_card_limit_used_last_year: the average percentage utilized on the credit card limit by the applicant the previous year.
- saving_amount: the appliers savings amount
- checking_amount: the appliers checking amount
- is_employed: whether the applier is employed or not
- yearly_salary: the appliers yearly salary
- age: the appliers age
- dependent number: Undisclosed feature
- loan_purpose: what is the purpose of the given loan
- date: submission date for application
- loan_granted: whether the given loan is granted or not
- loan_repaid: whether the given loan is repaid or not.

2.4 Pre-Processing:

In the upcoming section, we will discuss our preprocessing steps in order to prepare our data for training. We will discuss our data cleaning, which includes removal of NaN values, feature exclusions, multicollinearity and variable inflation factor, feature engineering, feature scaling, how we tackle the imbalanced classes, and finally the data splitting process.

2.4.1 Data cleaning:

2.4.1.1 NaN values in “loan_repaid”:

The dataset needed to be cleaned and prepared before implementing it into our various ML algorithms. The first order of operation was to investigate our target variable **“loan_repaid”**. It is quickly noticed that our target variable contained a great amount of NaN values, with a count of 53446 instances. This occurs because there are several instances of customers not being granted a loan from the bank, which results in that there is no loan to repay for that instance. Since our models are binary classification, either 1 or 0, the NaN instances needed to be corrected. However, since our original dataset was decently sized, we decided to remove those rows while maintaining the integrity of the dataset.

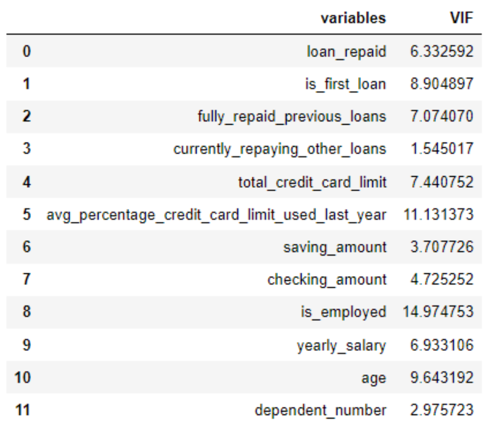
2.4.1.2 Feature exclusions:

- The column “*date*” is removed because of its low variance. After investigating this aspect, we couldn't find any indications that time of year affects whether a loan is granted or not. Therefore, we can confidently conclude that this feature doesn’t affect the results in any way.
- The feature “*is_employed*” is excluded because a customer can not have any yearly salary if they’re not employed. Therefore, if yearly salary equals 0, then their employment status would reflect the same result.
- *"avg_percentage_credit_card_limit_used_last_year”* and *"fully_repaid_previous_loans"* is excluded due to multicollinearity. See the upcoming chapter for in depth information.

2.4.2 Multicollinearity

Multicollinearity occurs in situations where there are high intercorrelations between two or more independent variables. In our case, there is a strong intercorrelation between *"avg_percentage_credit_card_limit_used_last_year"* and a handful of other features in our dataset. While *"fully_repaid_previous_loans" and "is_first_loan"* are directly intercorrelated. This can be confirmed from the variable inflation factor before and after removal of those features (Figure 4 and Figure 5).

Figure 4 VIF score original dataset



	variables	VIF
0	loan_repaid	6.332592
1	is_first_loan	8.904897
2	fully_repaid_previous_loans	7.074070
3	currently_repaying_other_loans	1.545017
4	total_credit_card_limit	7.440752
5	avg_percentage_credit_card_limit_used_last_year	11.131373
6	saving_amount	3.707726
7	checking_amount	4.725252
8	is_employed	14.974753
9	yearly_salary	6.933106
10	age	9.643192
11	dependent_number	2.975723

Between these two, we decided to remove *"fully_repaid_previous_loans"* because *"is_first_loan"* is a better explanatory variable explained by a higher feature importance, as shown in table 2. The feature importance scores below are gathered from our decision tree model.

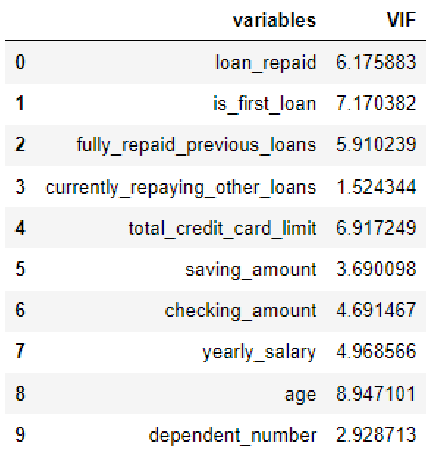
Table 2 Feature Importance

Feature name:	Feature importance score:
"fully_repaid_previous_loan"	0.00300
"is_first_loan"	0.01135

One might argue that multicollinearity is more applicable to algorithms in a multiple regression model. However, we claim that one should always be concerned about this aspect, regardless of the model being linear or not. Let's say we have a random forest algorithm with a set of linearly correlated features. The random selection within the tree might select collinear features that often result in a weak selection. This can add up the further we work down our nodes, therefore affecting the results negatively (Allison, 2019). We want to avoid features being highly correlated with each other. We must preserve the value of interdependence between our features, which will result in not meddling with the standard error of the regression coefficient (*Multicollinearity*, n.d.).

Both features mentioned above have a high variance inflation factor (VIF), which measures the amount of multicollinearity. In general, a variance inflation factor above 10 is considered high correlation and could be a red flag (Bock, 2020), which our dataset has as shown in figure 4. However, as we can see from figure 5, when we remove the mentioned features from our dataset, the VIF scores balances out:

Figure 5 VIF score after removing features dataset



	variables	VIF
0	loan_repaid	6.175883
1	is_first_loan	7.170382
2	fully_repaid_previous_loans	5.910239
3	currently_repaying_other_loans	1.524344
4	total_credit_card_limit	6.917249
5	saving_amount	3.690098
6	checking_amount	4.691467
7	yearly_salary	4.968566
8	age	8.947101
9	dependent_number	2.928713

2.4.3 Feature engineering:

The feature “Loan_purpose” is non-numeric, and states the purpose of the given loan with text. Since this feature is non-numeric, it can create confusion within our machine learning models and algorithms. The various selections within the loan purpose are: investment, home, business, emergency funds, and other. In order to bypass this string, we have created numerical values that will replace the string conditions. In other words, we replace the text with numerical values. After this process was completed, we currently stand with (Table 3):

Table 3 Feature engineering

Loan_purpose	**Numerical values**
Other	1
Emergency_funds	2
Business	3
Home	4
Investment	5

2.4.4 Feature scaling

In our given dataset, there are numerous features that vary in magnitude and scope. We wanted to bring all the features within similar standing, so that one specific feature does not influence our model too much compared to other features. For instance, if one feature contains large numbers, combined with a high standard deviation, it might affect the model to a larger degree compared to other features that might be smaller numbers with minor deviations. This preprocessing step can be the difference between a strong machine learning model, and a fragile and inaccurate model (Gupta, 2020). Two techniques dominate this market, and those are normalization and standardization (Roy, 2020). We have utilized a standardization technique due to the fact that it transforms our data to have a zero mean and a variance of 1 and gives our dataset a standard normal distribution. In addition, it makes our data unitless. This scaling technique transforms our features to a range between *[-1,1]* (Gupta, 2020).

Standard scores are calculated by:

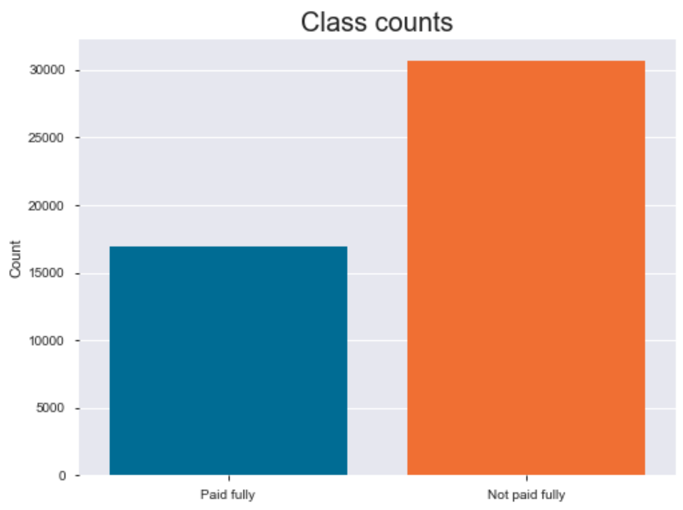
$$z = \frac{x - \mu}{\sigma} \tag{2.4.4.1}$$

Generally speaking, machine learning algorithms only capture numbers, and don't know what the numbers represent. If one of the given features contains large numbers, while other features hold lower numbers, then it might make an underlying assumption that larger numbers are of higher importance. We want each feature to be at a level playing field (Roy, 2020).

2.4.5 Imbalanced class

When examining our original dataset, we see a case of imbalanced data. As mentioned in the data description, there are 16948 cases where the given loan is paid fully, which results in 36% of the original dataset. However, we have a larger counterpart of 30706 instances where the loan is not paid fully, which results in 64% of the dataset, as shown in figure 6. This results in an imbalance within our class with a ratio of approximately 1:3. Therefore, we can see a moderately unbalanced dataset. Considering the number of cases where the loan is not paid in full, we decided to implement a synthetic minority oversampling technique, also known as SMOTE, to increase the number of cases where the loan is repaid to match the number of loans that are not repaid (Bownlee, 2020).

Figure 6 Count of target variable



When investigating the data, more specifically the independent variable (y), there were too few examples in the minority class for the model to efficiently learn the decision boundary. SMOTE operates by picking examples located close to the feature space rather than in the data space. It draws a line between those examples and creates a new sample at a point along that given line (Brownlee, 2020).

More specifically, SMOTE starts by selecting a minority class instance at random and then finds its k-nearest neighbors. It picks one of those neighbors and creates a synthetic instance by connecting those two examples to form a line segment in the feature space. The synthetic instances are generated as a convex combination for the two chosen instances (Brownlee, 2020b).

2.4.6 Data splitting

As a standard when dealing with prediction algorithms, we have to create a training dataset, possibly a validation dataset, and a test dataset. For valuation purposes, we decided to make a validation dataset. Instead of utilizing the common 80% training and 20% test data split, we performed a split of 60%

training and a 40% test data split. Furthermore, we split the test data once more into a validation set and a test set. In summary, we now have a 60% training dataset, a 20% validation dataset, and a 20% holdout dataset. This was performed in python by utilizing the sklearn package, more specifically, the train_test_split function (*Sklearn.Model_selection.Train_test_split — Scikit-Learn 0.24.2 Documentation*, n.d.). This function takes random partitions for the two subsets, and creates the two sets based on our test size of choice, and the random state (*Splitting Datasets With the Sklearn Train_test_split Function*, 2019). The previous step is repeated in order to create the validation set. We utilize this type of split (60/20/20) because it enables us to check whether the model overfits or underfits and it allows us to accurately evaluate the model. In addition, the validation set can be utilized to test various hyperparameter values.

3 Method section

In this section, we will present and dive deep into our algorithms of choice. We will discuss in general how the algorithms operate, the process of optimizing the hyperparameters and their outcome, and the various results of the models. In addition, we will talk about some features and techniques that are consistent throughout every model. The given Python scripts for each model can be found on GitHub (Ruud & Boen, 2021).

3.1 Supervised learning

As mentioned earlier, we will go through five algorithms and compare them. Four of these algorithms have something in common, which is that they are supervised learning algorithms. Supervised learning is defined by its utilization of a labeled dataset that trains algorithms in order to classify data or predict outcomes precisely. The prediction variable in our thesis is “loan_repaid”. The dataset is already classified, and the algorithm is "learning" from our data.

When input data is inserted into our model, the algorithm regulates its weights in order to fit the model to the data (IBM, 2021).

3.2 Common notations

Throughout this thesis, we mention our X and y with regards to the Machine Learning algorithms and models. Most models are described as learning a target function (f) that best maps the given input variables (X) to our output/predictor variable (y). The ultimate goal is to make accurate predictions in the future (y) with new examples of input variables (X) (Brownlee, 2019).

$$y = f(x) \tag{3.2.1}$$

3.3 Technique for Optimizing Hyperparameters

A hyperparameter is a parameter whose value is used to control the learning process (Wikipedia, 2021). Tuning hyperparameters is considered a difficult task to perform in machine learning, where the goal is to find the optimal hyperparameters. A machine learning model contains a heap of various parameters that ultimately decide the accuracy of the model. Therefore, it can be extremely important to locate the optimal values for the model hyperparameters. The hyperparameters vary from algorithm to algorithm, as various models require diverse types of tuning. We want to search the hyperparameter space in order to find a set of values that will optimize our model architecture (Mujtaba, 2021).

One possibility is to try all the different values when it comes to hyperparameters in order to find the ideal combination. However, this is a tedious process, and therefore, we have utilized SK-learn's automated procedure called GridSearchCV (*Sklearn.Model_selection.GridSearchCV*

— *Scikit-Learn 0.24.2 Documentation*, n.d.). This function loops through our predefined hyperparameters and fits our estimators on our training set.

GridSeachCV explores every combination from the values passed in the dictionary. Thereafter, it evaluates our model for any given combination using the cross-validation method. We utilized a 10-fold cross-validation strategy. After GridSearchCV is finished, we receive our model scores for every combination of hyperparameter values, and we can choose the combination that gives the best model performance. This was performed on the training dataset.

3.4 Support Vector machine

A support vector machine is a supervised machine learning model that utilizes classification algorithms in order to classify the data. The idea behind the support vector machine is relatively simple. It creates a hyperplane within the data that classifies the data into approved or rejected bank loans.

The way a support vector machine model works is that it finds a separating line or a so-called hyperplane that divides the data into two classes. Once that line is decided, it’s essential to find the best or optimal line to increase the model's accuracy and avoid false positives and false negatives in our confusion matrix (Gandhi, 2018). However, there are infinite potential lines that will help separate our two classes, so how does the support vector machine find the most effective hyperplane? Within machine learning in general, we want to find a generalized separator to isolate our two groups of classes (Gandhi, 2018).

Once we have this general line that separates the data, we will need to find the two points closest to that given line. These points are called support vectors, and we calculate the distance between these two support vectors and our original line (Gandhi, 2018). The objective for this model is to maximize this margin that separates the support vectors, i.e find the optimal hyperplane. The support vector machine model attempts to create a decision boundary where the distance between the support vectors is as wide as possible. There are two goals that we were looking for when building this model: setting a larger margin and lowering our misclassification rate to the best of our ability, as shown in figure 7 (ale, 2020).

Figure 7 Optimal Hyperplane

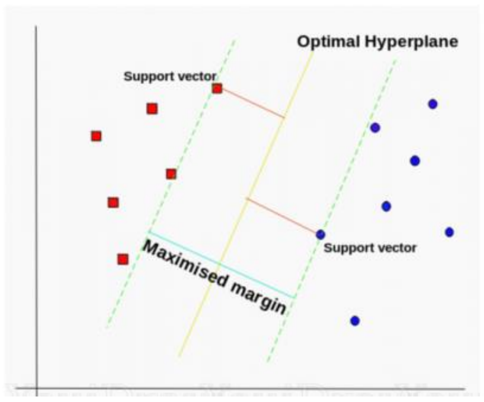


Figure 7 “Optimal hyperplane using SVM algorithm” 2020, by Bhosale.

3.4.1 Hyperparameters tuning

The parameters of interest when it comes to tuning the Support Vector Machine model are C and gamma, as shown in table 4. As mentioned earlier in this section, we want to get as large a margin as possible while also lowering the misclassification rate. However, these two aspects might sound contradictory. If we increase our margin, it leads to a higher misclassification rate and vice versa. It’s here the parameter C comes into play. Proper choice of C and gamma is crucial for the performance of our model (Yıldırım, 2020).

Table 4 SVM Hyperparameters

Hyperparameter	Default value	Search space	Optimized values
C	1.0	0.1, 1, 10, 100, 1000	1.0
Gamma	“scale”	1, 0.1, 0.01, 0.001, 0.0001	1.0

Parameter C

The larger our C value is, the smaller our margin becomes. We want to find the optimal margin to receive the best results for our testing set. On the opposite side, the smaller our C value is, the larger is the margin. In theory, the C parameter tells the Support Vector Machine how willing we are to misclassify each training example. It is difficult to manually find the optimal C value since you must run the model numerous times to find the lowest misclassification rate (Yıldırım, 2020). In addition, when the C parameter is large, the model tries to fit all the classifiers correctly, which can lead to a curved or wiggly decision boundary. If this boundary is directly imported into our testing data, it will most likely not yield good results because the points will be in different positions, and the customized decision boundary is not applicable to that data. This would most likely result in overfitting, and we want to create a generalized model that can fit in several situations and datasets. This is where GridSearchCV will help us enormously.

Gamma

The gamma parameter controls the distance of influence of a single training point. If we have a high gamma value, the points need to be tight to one another in order to be considered in the same class. With a low gamma value, the points don’t need to be close to each other in order to be regarded within the same class (Yıldırım, 2020). We needed to avoid a high gamma value since that most likely will result in overfitting.

Kernel

There are three major kernel functions that are available for our support vector machine model. Kernel functions are techniques used to take our data as input and convert it into the necessary form of processing data. It is a set of mathematical functions used in support vector machines, and it creates an opening to manipulate the data. We decided to use the Radial Basis Function (RBF) because it is the most generalized form of kernelization and has a strong similarity to the gaussian distribution (Sreenivasa, 2020). Furthermore, it is suitable for our model because gamma is a parameter that needs to be highly considered to avoid overfitting in our data.

3.4.2 Results

In order to measure the performance of the Support Vector Machine algorithm, we initially ran the model without any hyperparameter tuning on the validation set. The default hyperparameter values for the SVM algorithm is C=1.0 and gamma=" scale". When gamma is passed with "scale" it utilizes 1/ (n_features * X.var()) as its value (*Sklearn.Svm.SVC — Scikit-Learn 0.24.2 Documentation*, n.d.). With these default values, we received an accuracy score of 0.9256, which is considered a fairly good score. In addition, recall and precision have been calculated as shown in table 5.

Table 5 Performance score on validation dataset without tuning

Performance measure:	**Score:**
Accuracy:	0.9256
Precision:	0.9558
Recall:	0.8936

After tuning the SVM hyperparameters, we received our optimal values of C = 1.0 and Gamma = 1.0. When these values were inserted into the given model, we received an accuracy score of 0.9321, which is a small increase from the first result. For our additional performance measure, we received the following after utilizing the optimal hyperparameters. See table 6.

Table 6 Performance score on validation dataset with tuning

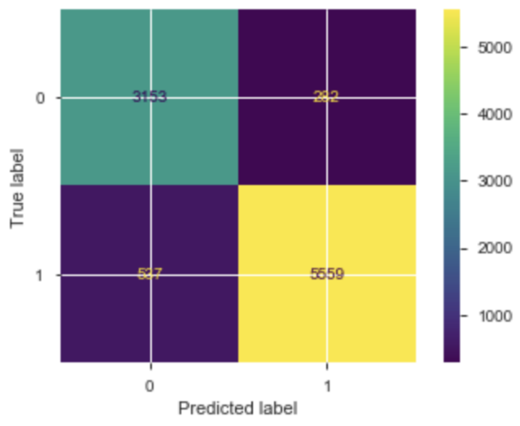
Performance measure:	**Score:**
Accuracy:	0.9321
Precision:	0.9543
Recall:	0.9086

When running the optimized algorithm on our unseen test data, we receive similar scores, with no indication of over/underfitting. For accuracy, the score is 0.914 on the test dataset, which is fairly consistent with the scores from our previous paragraphs. Below are the performance measures and a confusion matrix for our optimized algorithm on the test dataset. See table 7 and figure 8.

Table 7 Performance score on test dataset with tuning

Performance measure:	**Score:**
Accuracy:	0.9140
Precision:	0.9517
Recall:	0.9119

Figure 8 Confusion matrix on test dataset



3.5 Logistic regression

Logistic regression is a supervised machine learning classification algorithm used to predict a given target variable, and the model is dichotomous. It is a very common and simple machine learning algorithm that predicts P(Y=1) as a function of X. At its core, logistic regression is driven by the logistic function, also called the sigmoid function. It is an S-shaped curve that transforms a real number into a value between 0 and 1 (Swaminathan, 2019).

Figure 9 Illustration of logistic regression

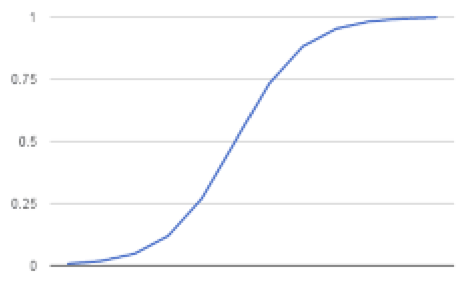


Figure 9 “logistic function” 2020c, by Brownlee.

The logistic function is given by:

$$f(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{3.5.1}$$

In the given function above, the $f(x)$ signifies the probability of our predictor variable. β_o represents the linear regression intercept, whileβ_1 is the multiplication of the regression coefficient by various values of the predictor variable (Brownlee, 2020c).

If the output of the given function is more than 0.5, it will be classified as "eligible", and if the function output is less than 0.5, it will be classified as "non-eligible". The function works almost like a percentage calculation. If our output is 0.80, then there is an 80 percent chance that the loanee is eligible for the given loan. However, we are not predicting probabilities. The probability prediction is transformed into binary values (0 or 1) (Brownlee, 2020c).

$$0 \;\; if \;\; p(non - eligible) < 0.5 \tag{3.5.2}$$

$$1 \;\; if \;\; p(eligible) > 0.5 \tag{3.5.3}$$

3.5.1 Hyperparameters tuning

The hyperparameters of interest in a logistic regression algorithm are C and penalty, as illustrated in table 8. As our previous models, we are often interested in the best combination of our hyperparameters, as they often work in tandem.

Table 8 Logistical regression Hyperparameters

Hyperparameter	**Default value**	**Search space**	**Optimized values**
Penalty	l2	l1, l2	l2
C	1.0	np.logspace(0,4,10)	7.74

Penalty

This parameter is used to specify the norm used in the penalization. l2 regularization adds an l2 penalty which is equal to the square of the magnitude of coefficients, while all coefficients are shrunk by the same factor (*L1 Penalty and Sparsity in Logistic Regression — Scikit-Learn 0.24.2 Documentation*, n.d.).

C

The parameter C is the inverse of regularization strength in logistic regression. It’s a control variable that preserves the strength of regularization by being inversely positioned to the lambda regulator (Charleshsliao, 2017). In general, regularization is adding a penalty in order to increase the scale of parameter values to reduce overfitting. Thus, with a high value of C, the model inserts extra weight to the training data and a lower weight to the complexity penalty. When running our C parameter in GridSearchCV, logspace() was utilized in order to receive numbers that were evenly spaced on a log scale. (*Sklearn.Linear_model.LogisticRegression — Scikit-Learn 0.24.2 Documentation*, n.d.).

Our objective when optimizing our hyperparameters is minimizing the error between our predicted variable compared to what the dependent variable actually is.

3.5.2 Results

When investigating the results for the logistic regression model, we first ran our validation data without any optimization. The default values for the logistic regression model are C =1.0 and penalty =l2, which can be found in table 8 above. Without hyperparameter tuning, our model yielded an accuracy of 0.9058 on the validation set, which is considered a respectable and reliable score.

Table 9 Performance scores on validation dataset without tuning

Performance measure:	**Score:**
Accuracy:	0.9058
Precision:	0.9202
Recall:	0.8894

After running GridSearchCV on the logistic regression algorithm, we received optimal values of C = 7.742, and penalty = l2, as seen in table 8. After we inserted the optimal values into the logistic regression algorithm on the validation dataset, we received an accuracy of 0.9057 as seen in table 10. This is actually a minimal decrease from the default values, but it's insignificant because it's extremely small.

Table 10 Performance score on validation dataset with tuning

Performance measure:	**Score:**
Accuracy:	0.9057
Precision:	0.9200
Recall:	0.8894

When running the optimized model on our test data, we received a similar score of 0.9026, while precision received a score of 0.9492 and recall with a score of 0.8956. See table 11.

Table 11 Performance score on test dataset

Performance measure:	**Score:**
Accuracy:	0.9026
Precision:	0.9492
Recall:	0.8956

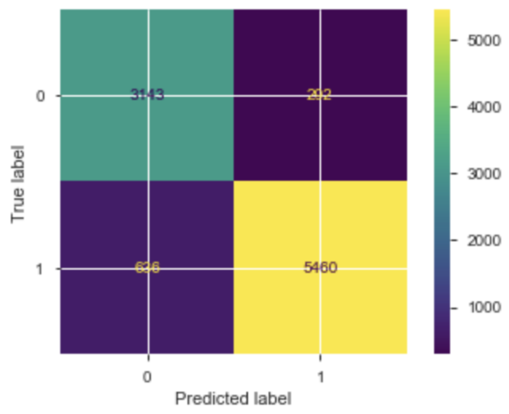
There were a couple of hyperparameter values that yielded the exact same results when tuning the training data as shown in the table below (Table 12). Logistic regression is a very robust algorithm. It can map many types of data in various forms. Therefore, tuning a logistic regression model isn't necessarily crucial for model performance and accuracy. The performance of the models stays fairly level, regardless of the hyperparameter values.

Table 12 Result from the GridSearchCV

C	**Penalty**	**Params**	**Mean test score**	**Std test score**
7.74264	l2	“C: 7.74264”, “Penalty: l2”	0.90253	0.00479
21.5443	l2	“C: 21.5443”, “Penalty: l2”	0.90253	0.00479
59.9484	l2	“C: 59.9484”, “Penalty: l2”	0.90253	0.00479

The confusion matrix below represents the validity of our model. We can compare these results as if we ran the model in a real-life situation since this is retrieved from unseen test data. As shown in figure 10, this is a confusion matrix that has a good score. However, with a bit higher accuracy, we could steer clear of some false negatives and false positives in order to avoid granting a loan to someone who isn't able to repay or denying a loan to a customer who should, in theory, be able to repay.

Figure 10 Confusion matrix from test dataset



3.6 K-Nearest Neighbors

The k-nearest neighbors (KNN) are a non-parametric algorithm that can be used to solve classification and regression problems. Non-parametric algorithms do not make any strong assumptions about the form of the mapping function, which makes the algorithm free to learn any functional form from the training data (Brownlee, 2020d). This is helpful in practice since most real-world data do not follow any mathematical theoretical assumptions.

Figure 11 Illustration on how the KNN algorithm work

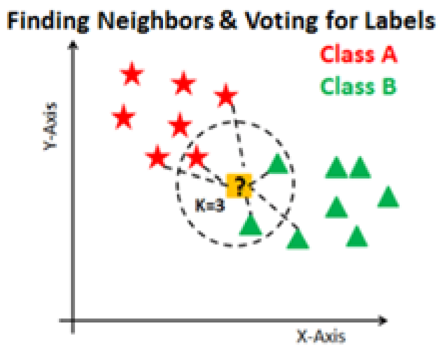


Figure 11. « KNN Classification using Scikit-learn», 2018, by Navlani, A.

The *k* in the KNN algorithm, is the number of nearest neighbors, which are the main deciding factor in the algorithm. We will provide an example to understand the algorithm better. Suppose k=3 and P1 is the label that needs to be predicted, as shown in figure 11. First, the algorithm finds the nearest *k* neighbors to P1. This is generally done by calculating the Euclidean distance, which calculates the distance from P1 to all classified data points (Navlani, 2018).

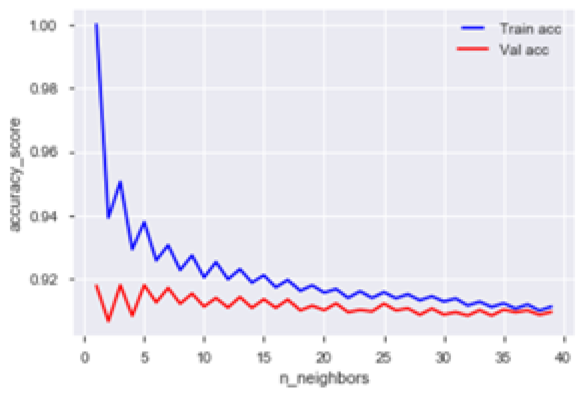
$$(x,\ y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3.6.1}$$

Next, the algorithm classifies P1 by majority votes of its k neighbors. Each neighbor votes for their class, and the class with the most votes predicts P1 (Navlani, 2018). In our examples, P1 will be classified as Class B since class B has the most votes.

3.6.1 Hyperparameters tuning

The KNN algorithm has many different hyperparameters, such as the number of neighbors, different weights on data points, and different metrics that can be used for calculating the distance to each data point. We will review the number of neighbors since this hyperparameter is the main deciding factor. First, we did a GridSearchCV where we tested the accuracy of 1 to 40 neighbors. The GridSearchCV computed the optimal number of neighbors to be 1 with an accuracy of 1,00, which overfits the data. Then, to prevent the model from overfitting, we used the validation dataset to visualize the accuracy score of the training and validation dataset for different neighbors, as we see in figure 12.

Figure 12 Overfitting using small number of neighbors



From figure 12, we see that the training dataset overfits when the numbers of neighbors are small, which means that the model learns the detail and noise in the training data to the extent that it negatively impacts the model on a new dataset (Brownlee, 2019b). When the number of neighbors is high, the model underfits, which means that the model cannot generalize the datasets enough (Brownlee, 2019b). This leads to bad performance on the data. We choose the smallest number of neighbors in our model to be seven since the relative accuracy between training and validation does not significantly lower if the number of neighbors increases. Next, we did a grid search from 7 to 40 neighbors, where the best parameter was seven neighbors, as we see in table 13.

Table 13 KNN Hyperparameters

Hyperparameter	**Default**	**Search Space**	**Optimal values**
n_neighbors	5	7 to 40	7

3.6.2 Results

To measure the performance of the KNN algorithm, we first tested the KNN algorithm on the validation dataset with no hyperparameter tuning. As displayed in table 13, the default number of neighbors is five, which gives an accuracy score of 0.9177. In addition, we calculate the precision and recall score. The precision score was 0.9678, and the recall score was 0.8653, as shown in table 14.

Table 14 Performance score on validation dataset without tuning

Performance measure:	**Score:**
Accuracy:	0.9177
Precision:	0.9678
Recall:	0.8653

After running the GridSearchCV on the KNN model, we received the optimal number of neighbors to be 7, as shown in table 13. As a result, the KNN model had an accuracy of 0.9149 on the validation dataset, a precision score 0.9656, and a recall score of 0.8616. Thus, the model had a marginal lower accuracy, recall, and precision score, which can result from the model becoming more generalized.

Table 15 Performance score on validation dataset with tuning

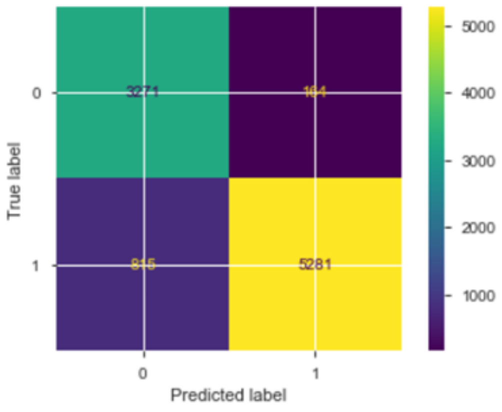
Performance measure:	**Score:**
Accuracy:	0.9149
Precision:	0.9656
Recall:	0.8616

The optimized model had an accuracy of 0.8972, a precision score of 0.969, and a recall score of 0.8671 on the test data set, as displayed in table 16. Figure 13 displays the confusion matrix on the optimized model on the test dataset.

Table 16 Performance score on test dataset

Performance measure:	**Score:**
Accuracy:	0.8972
Precision:	0.9690
Recall:	0.8671

Figure 13 Confusion matrix test dataset



3.7 Decision Tree Classifier

The Decision Tree Classifier algorithm creates a flowchart structured as a tree where the nodes represent the features, the branches show the decision rules, and the leaves display the outcome (Navlani, 2018). The topmost node of the tree is the root node, which usually is the node that maximizes the reduction in cross-entropy. Afterward, finding the best split combination, the data are partitioned into two new nodes, and the splitting process repeats on each of the two new nodes. This method repeats on each subset in a recursive manner called recursive partitioning (Navlani, 2018). The recursive partitioning completes when the data within the subsets are sufficiently homogenous or another stopping criterion has been met. Figure 14 illustrated a decision tree with only one split. The leaf (child) nodes are "purer" than the root node(parent) since the share of each type is more dissimilar in the leaf node than in the root node, also called more homogenous (Provost & Fawcett, 2015, p. 53)

Figure 14 Illustrated a decision tree with one split

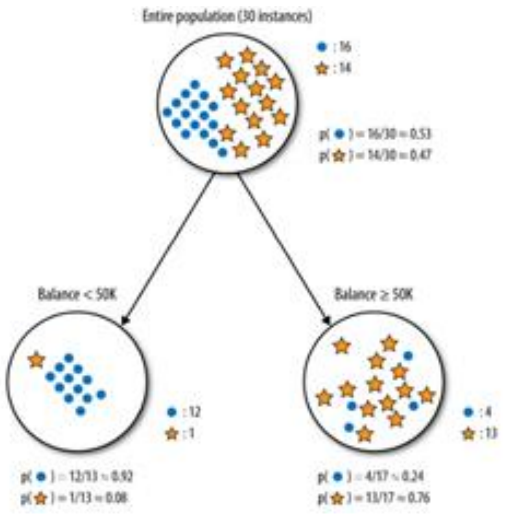


Figure 14. From «Splitting the “write-off” sample into two segments, based on splitting the Balance attribute (account balance) at 50K» by Provost. F., & Fawcett. T, 2015, Data Science for Business, s.54, United States: O'Reilly Media.

Several splitting rules can be used to partition the dataset. One typical method is information gain (IG) based on a purity measure called entropy. Entropy is defined as (Provost & Fawcett, 2015, p. 52):

$$entropy = -p_1\, log\, log\, (p_1) - p_2 log(p_2) - \cdots \tag{3.7.1}$$

Each p_i is the fraction of examples in a given class. Entropy calculates the disorder of the set, ranging from minimum disorder (the set has members all with the same, single property) to one at maximal disorder (the properties are equally mixed) (Provost & Fawcett, 2015, p. 53). Disorder resembles how impure the set is. Information gain measures the change in entropy due to any amount of new information being added and are calculated by (Provost & Fawcett, 2015, p. 52):

$$\begin{gathered} IG(parent, children) = entropy(parent) - \\ [p(c_1) \times entropy(c_1) + \\ p(c_2) \times entropy(c_2) + \cdots] \end{gathered} \tag{3.7.2}$$

3.7.1 Pruning

The extensive partitioning will frequently lead to a big decision tree that will produce good predictions on the training set but weak predictions on the test set because of the decision tree's complexity. As a result, the algorithm overfits the training data, which could be improved with smaller tree size. Pruning is a technique that involves growing a tree to its full size before "cutting" off branches, ultimately reducing the tree's size by removing sections of the tree that are non-critical and redundant to classify instances (Hoare, 2020). This process leads to a reduction in variance at the cost of some extra bias.

3.7.2 Hyperparameters tuning

Table 17 shows the different hyperparameters we tune in our decision tree algorithm. The search space represents the different values tested, and optimal values show the result from the grid search. The criterion parameter chooses which function to measure the quality of a split. Ccp_alpha is the parameter used for pruning, as explained in the section above. Furthermore, max_depth, min_samples_split, and min_samples_leaf is all stopping criteria. Max_depth specifies the maximum depth of the tree, min_samples_split is the minimal number of samples required to split a parent node, and min_samples_leaf is the minimal samples needed to be at a child (leaf) node (Mithrakumar, 2019).

Table 17 Decision tree Hyperparameters

Hyperparameter	**Default**	**Search Space**	**Optimal values**
Criterion	Gini	Gini or Entropy	Gini
ccp_alpha	0.0	0 to 0.1	0.00007
max_depth	None	1 to 10	9
min_samples_split	2	1 to 10	2
min_samples_leaf	1	1 to 5	1

3.7.3 Results

We tested the algorithm with no hyperparameter tuning to measure the performance of the decision tree algorithm on the validation data set. The default parameters are shown in table 17. With these hyperparameters, the model gives an accuracy score of 0.8972, a precision score of 0.8986, and a recall score of 0.8971.

Table 18 Performance score on validation dataset without tuning

Performance measure:	**Score:**
Accuracy:	0.8972
Precision:	0.8986
Recall:	0.8971

Table 18 displays the optimized hyperparameter from the GridSearchCV. As a result, the model had an accuracy of 0.914, a precision score of 0.9472, and a recall score of 0.8781. Thus, tuning of the hyperparameters increases the accuracy and precision score of the model, which boosts the model's performance.

Table 19 Performance score on validation dataset with tuning

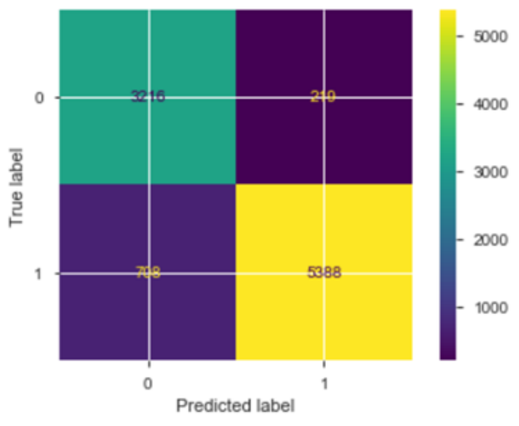
Performance measure:	**Score:**
Accuracy:	0.9140
Precision:	0.9472
Recall:	0.8781

The optimized model had an accuracy of 0.9006, a precision score of 0.9668, and a recall score of 0.8746 on the test data set. Thus, the model has nearly as good performance scores as the validation dataset, which indicates that the model has a good generalization. Below is the confusion matrix on the optimized model on the test dataset.

Table 20 Performance score on test dataset

Performance measure:	**Score:**
Accuracy:	0.9006
Precision:	0.9668
Recall:	0.8746

Figure 15 Confusion matrix test dataset



3.8 Stochastic gradient descent

Stochastic gradient descent is an optimization algorithm used to find the parameters values that minimize loss function. The algorithm has many different loss functions, such as sum of squared residuals, mean squared error, hinge loss, and cross-entropy function:

$$w_{t+1} = w_t - n_t g_t(w_t)$$
$$where\ g_t(w_t) = \lambda w_t + \iota'(w_t^T x_t, y_t) \qquad (3.8.1.1)$$

The algorithm starts with a randomly chosen point on the vector w_t (Krzysztof & Paweł, 2015). Next, it advances to the next point in the direction of the fastest decrease of the loss function $n_t g_t(w_t)$, where ι' is the gradient of the loss function and n_t is the learning rate that determines how large the moving step will be (Krzysztof & Paweł, 2015). The moving step is usually big when the point is far away from the local minimum and small when it comes close to the local minimum, as shown in figure 16 (Stojiljković, 2021). The algorithm stops when the moving step will be close to 0 or the maximum number of steps is reached.

Figure 16 Illustrates the movement of the gradient through the iterations

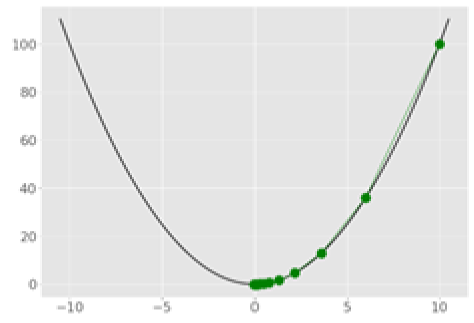


Figure *16*. «Stochastic Gradient Descent », 2006, by Stojiljković, M

3.8.1 Hyperparameters tuning

In the stochastic gradient descent, we mainly analyze the loss, penalty, and alpha hyperparameter since these are the main deciding parameters. The loss parameter determines which loss function the algorithm will have. The penalty is the regularization parameter added to the loss function that reduces the model parameters towards the zero vector (Stephanie, G. 2020). Alpha is a constant that multiplies with the penalty parameter. The higher the value of alpha, the stronger the regularization. Alpha is also used to compute the learning rate (sklearn, n.d.), as we see in equation 3.8.1.2.

$$n_t = \frac{1}{\alpha(t + t_0)} \tag{3.8.1.2}$$

GridSearchCV computed that the best loss function is hinge loss with a penalty of l1, limiting the size of the coefficients and the alpha to be 0.01, as we see in table 21. When the hinge loss is used as the loss function, it gives a linear support vector machine.

Table 21 SGD Hyperparameters

Hyperparameter	Default	Search Space	Optimal values
Loss	Hinge	Hinge, Modified_huber, Log, Squared_hinge, Perceptron	Hinge
Alpha	0.0001	0.0001, 0.001, 0.01, 0.1, 1, 10, 100	0.01
Penalty	None	None, l1, l2, Elasticnet	l1

3.8.2 Results

We tested the algorithm with no hyperparameter tuning to measure the performance of the Stochastic gradient descent algorithm on the validation data set. With the default parameter values shown in table 21, the model gives an accuracy score of 0.9038, a precision score of 0.9246, and a recall score of 0.8808.

Table 22 Performance score on validation dataset without tuning

Performance measure:	Score:
Accuracy:	0.9038
Precision:	0.9246
Recall:	0.8808

After running the GridSearchCV on the SGD model, we received the optimal values for the hyperparameters, as shown in table 21. As a result, the SGD model had an accuracy of 0.9063 on the validation dataset, a precision score 0.9398, and a recall score of 0.8695. Thus, the model had a marginal higher accuracy and precision score. In addition, the model had a slight decrease in recall score.

Table 23 Performance score on validation dataset with tuning

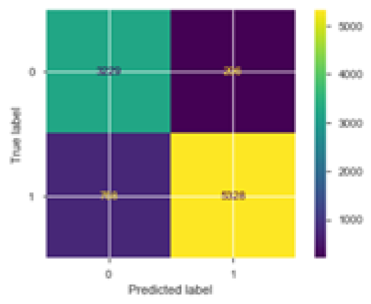
Performance measure:	**Score:**
Accuracy:	0.9063
Precision:	0.9398
Recall:	0.8695

The optimized model had an accuracy of 0.8978, a precision score of 0.9627, and a recall score of 0.8741 on the test data set, as displayed in table 24. Thus, the model has almost as good performance scores as the validation dataset, which indicates that the model has a good generalization. Below is the confusion matrix on the optimized model on the test dataset.

Table 24 Performance score on test dataset

Performance measure:	**Score:**
Accuracy:	0.8978
Precision:	0.9627
Recall:	0.8740

Figure 17 Confusion matrix on test dataset



4 Comparing Results and Additional Findings

In this section, a comparison of the performance and results of our various machine learning algorithms will take place. In table 25, we can see the given scores that were received from the baseline models on the validation set, scores on the validation set after tuning the model, and finally, the optimized model utilized on the unseen test data.

Table 25 Accuracy from the algorithms

Algorithm:	**Accuracy: before parameter tuning**	**Accuracy: after parameter tuning**	**Accuracy: on testset**
SVM	0.925	0.932	0.914
Logistic Regression	0.906	0.906	0.903
KNN	0.917	0.914	0.897
Decision Tree	0.897	0.914	0.899
Stochastic gradient descent	0.903	0.906	0.898

The Support Vector Machine model performed very well on all aspects of our data, such as accuracy, precision and recall. It resulted in an accuracy of 0.914 on the unseen test data, with no signs of over-fitting. The benefit of our Support Vector Machine is that it effectively utilized its advantages. There is a clear margin of separation between our classes. In combination with the “rbf” kernel, it performed well on unseen data. On the other hand, a Support Vector Machine does not perform extremely well on very large datasets. In our case, the dataset is not particularly large which suits the Support Vector Machine algorithm. In addition, there isn’t a lot of noise in the dataset, with few overlapping features that help the support vector machine algorithm perform on top of its potential (K, 2020). Finally, a general disadvantage for support vector machines is that it doesn’t predict probabilities, since the algorithm places our data points below or above the classifying hyperplane (K, 2020), which is mentioned in the more detailed section about Support Vector Machines (Chapter 3.4).

Our second best model is the logistic regression model. We received good overall scores for both the validation data and the unseen test data. However, the hyperparameter tuning didn’t excel the model in any noteworthy direction, with only a small increase in accuracy. Compared to our best model, the support vector machine, the logistic regression algorithm does, in fact, calculate predicted probability. In short, it learns the linear relationships that occur in the dataset while introducing non-linearity in the form of the sigmoid function (GeeksforGeeks, 2020a). In general, logistic regression is easy to implement, interpret, and can train efficiently and effectively on all kinds of data. In theory, a logistic regression algorithm does not require any hyperparameter tuning, as we can see from our results. Something we need to keep in mind when considering implementing a logistic regression model in Machine Learning is that the algorithm assumes linearity between our prediction variable and or independent variables (GeeksforGeeks, 2020a). In addition, even though a logistic regression model rarely overfits, it has a tendency to overfit in extremely high dimensional space.

The remaining three algorithms yielded decent results, however, based on the results we would choose either logistic regression or a support vector machine for this type of classification problem. Decision tree, even though it requires some fine-tuning in order to perform relatively well with an accuracy on the unseen test data of 0.902. KNN received an accuracy on the test data of 0.897, see our k-nearest-neighbors section (Chapter 3.6) for more information on results and reasons. In addition, we wanted to include something different other than supervised learning in order to see how it affected the results. Stochastic gradient descent is an optimization algorithm and received an accuracy on the test data of 0.897, which is considered a decent score. For more information, see the method section about SGD (Chapter 3.8).

5 Discussion and Conclusion

In the following chapter, we will discuss related work and research, answer our thesis question, the impact on the banking industry and further research.

5.1 Related Work

Hand and Henley analyzed the different credit scoring methodologies used in the financial industry. The techniques they list as industry standards are logistical regression, decision tree, linear regression, and discriminant analysis (Hand & Henley, 1997). Along with k-nearest neighbors and neural networks. Turiel and Aste provide an extensive and in-depth look at loan acceptance and default prediction with machine learning (Turiel & Aste, 2020). They used logistic regression, support vector machine algorithms, and deep neural networks. The study concluded that machine learning could improve current credit risk models, reducing the default risk of issued loans by as much as 70%. The best default prediction algorithm was deep neural networks, and for loan approval was logistic regression best.

Furthermore, Tepy and Polena investigated which classification algorithms perform the best in peer-to-peer lending (Tepy & Polena 2020). They tested ten different classification algorithms and ranked them on six different performance measures. The best overall performer was logistic regression. Other authors also found this, such as Peng, who analyzed the credit risk and fraud risk in six different countries using eight classification methods (Peng, et al, 2011).

Table 26 illustrates research relevant to the theses with their representative performance accuracy for the different machine learning algorithms. As the table shows, Support Vector Machine and Logistic regression frequently perform relatively well, as our research also achieves.

The relative differences in predictive power between the algorithms may be overestimated, according to Hand (Hand, 2006). This could be due to the "reject inference" problem. One algorithm that performs well on a dataset does not naturally perform better than other classifiers on through-the-door population. Additionally, Hand claims that the classifier's goal should be to maximize profit. Therefore, if a confusion matrix is utilized as an evaluation measure, the results may differ from those obtained using the ROC AUC metric.

Table 26 Accuracy comparison of previous research

	Baesens (2003)	**Peng et al (2011)**	**Bao et al 2019**	**Nabende & Senfuma, 2019**	**Vieira et al (2019)**	**Fan 2020**	**Tepy et al (2020)**
SVM	0.797	0.83	0.914		0.924	0.774	0.788
Decision trees	0.77	0.817	0.842		0.888		0.765
K-NN	0.782	0.802	0.904	0.932			0.765
Logistic regression	0.793	0.853	0.855	0.903	0.921	0.701	0.791
SGD				0.921			

5.2 Summary

Our research question is as follows:

” For a chosen set of machine learning algorithms, which algorithm demonstrates the best performance in loan eligibility classification prediction with regards to several model evaluation metrics?”

As mentioned in chapter 4, Support Vector Machine and Logistic Regression yielded the best results in terms of accuracy, precision, and recall for our data. Support Vector Machine has a better potential to be accurate but also has its limitations. On the other hand, Logistic Regression is very user-friendly, easy to implement and interpret, and does not require any specific tuning.

We would argue that our results are quite similar to other related work within the same topic. According to Turiel and Aste (Turiel & Aste, 2020), and Tepy and Polena (Tepy & Polena 2020) their best loan approval prediction model was Logistic Regression based on its excellent characteristics as a binary classifier. When investigating table 26, where we illustrate the accuracy scores of the mentioned related work, we can see a strong and clear consistency that Support Vector Machine and Logistic Regression performs best, compared to the other algorithms.

Our accuracies, precisions and recalls generally had very good measures throughout this study. When compared to the related studies, we can see that our model performance generally outperforms the other scores. This doesn't necessarily tell us that our models are better, with more accurate tuning and testing. It might boil down to the differences within each dataset, or that we utilize different techniques when it comes to pre-processing, tuning, and testing. Every performance measure is relative to the data available, baseline model, and tuning techniques. In conclusion, we would argue that the logistic regression model is overall best in this specific situation because of its flexibility and user-friendliness. A Support Vector Machine can outperform the logistic regression algorithm, but it requires more work as well.

5.2 Impact on the banking industry

The implementation of machine learning algorithms regarding loan eligibility can have a significant effect on the banking industry. The algorithms can reduce the staff of loan officers for banks, which lower their cost. Many countries such as the US, Singapore, and China have already started using AI to determine creditworthiness and streamline the loan process (Faggella, 2020), which increases the competition in the banking industry. Furthermore, the algorithms can help banking institutions reduce their risk. With more personalized loans, the business and clients can get loan approvals based on their whole financial and personal situation, not just the standard loan-to-value ratio and/or debt-to-income ratio, which will reduce loan defaults. For example, Amazon saves a huge amount of data on what products are sold, customer's reviews on those products, the economic status of the businesses which make those products, and the likely future demand for these products. This information uses Amazon machine learning models to predict which companies they should offer loans to (Faggella, 2020).

5.3 Further research

The research presented in this thesis could serve as a good starting place for further investigation into loan eligibility. A deeper analysis of the variables used in the models as well as creating additional variables can improve the prediction of the models. Furthermore, other models could be added, such as neural network algorithms, which have proven good performance in default prediction. However, the data accessible in this thesis has some limitations regarding the number of years covered by the data. In addition, customers' behavior may influence the results of this research since customers in a specific location could have different spending habits than customers in other locations. Another approach that can be applied is if there's data available for more extended periods and broader geography of clients, it would be interesting to implement macroeconomic variables, leading to new insights into the factors that influence loan eligibility and which machine learning methods are best for this problem.

Furthermore, it would be interesting to research which performance metrics are the most helpful for this type of issue. A cost-sensitive classification model is another option for implementation, Where the model gives different costs to misclassifying loan approvals. For example, the model penalizes incorrectly classifying a loan acceptance higher than incorrectly classifying a loan rejection. It would also be interesting to make the cost of misclassifying loans from clients with higher credit ratings more expensive than loans from lower ratings because customers with higher credit ratings usually take bigger loans and are riskier for the bank. This implementation could lower the overall risk and possibly increase the models' performance. Moreover, it would be interesting to analyze which features are the most important when it comes to predicting loan approvals.

6 Bibliography

Ruud, M. & Boen, H. (2021, June 28). *Loan-Eligiblity-Prediction*. GitHub. https://github.com/MikkeljRuud/Loan-Eligiblity-Prediction

Brownlee, J. (2019, August 12). *How Machine Learning Algorithms Work (they learn a mapping of input to output)*. Machine Learning Mastery.https://machinelearningmastery.com/how-machine-learning-algorithms-work/

IBM. (2021, May 7). *Supervised Learning*. www.Ibm.com. https://www.ibm.com/cloud/learn/supervised-learning

Krzysztof, S., & Paweł, D. (2015, September 20). Stochastic Gradient Descent with Barzilai–Borwein update step for SVM. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0020025515002467

Stojiljković, M. (2021, January 27). Stochastic Gradient Descent Algorithm With Python and NumPy. Realpython. https://realpython.com/gradient-descent-algorithm-python/

Stojiljković, M. (2021b, January 27). Stochastic Gradient Descent Algorithm With Python and NumPy [Photo]. Realpython. https://realpython.com/gradient-descent-algorithm-python/

Stephanie, G. (2020, July 7). Regularization: Simple Definition, L1 & L2 Penalties. Statistics How To. https://www.statisticshowto.com/regularization/

sklearn. (n.d.). sklearn.linear_model.SGDClassifier — scikit-learn 0.24.2 documentation. Scikit-Learn. Retrieved June 26, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

Navlani, A. (2018, December 28). Decision Tree Classification in Python. Datacamp. https://www.datacamp.com/community/tutorials/decision-tree-classification-python

Navlani, A. (2018, August 2). KNN Classification using Scikit-learn. Datacamp. https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=229765585183&utm_targetid=aud-299261629574:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=9047324&gclid=CjwKCAjwtJ2FBhAuEiwAIKu19tq7P3si5w30q7pGz2NEJS4FQdtW8_9B5HmcvZMZWZIgQuLkToeMgBoCDcsQAvD_BwE

Navlani, A. (2018). KNN Classification using Scikit-learn [Photo]. https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn?utm_source=adwords_ppc&utm_campaignid=898687156&utm_adgroupid=48947256715&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=229765585183&utm_targetid=aud-299261629574:dsa-

429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=9047324&gclid=CjwKCAjwtJ2FBhAuEiwAIKu19tq7P3si5w30q7pGz2NEJS4FQdtW8_9B5HmcvZMZWZIgQuLkToeMgBoCDcsQAvD_BwE

Brownlee, J. (2020d, August 15). Parametric and Nonparametric Machine Learning Algorithms. Machine Learning Mastery. https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/

Brownlee, J. (2019b, August 12b). Overfitting and Underfitting With Machine Learning Algorithms. Machine Learning Mastery. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

Provost, F., & Fawcett, T. (2015). Data Science for Business. United States: O'Reilly Media.

Hoare, J. (2020, November 19). Machine Learning: Pruning Decision Trees. Displayr. https://www.displayr.com/machine-learning-pruning-decision-trees/

Mithrakumar, M. (2019, November 12). How to tune a Decision Tree? - Towards Data Science. Medium. https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

SAS. (n.d.). Predictive Analytics: What it is and why it matters. Retrieved June 26, 2021, from https://www.sas.com/en_us/insights/analytics/predictive-analytics.html

Hope, D. (2020, November 21). How Big Data Is Changing The Nature of Consumer Lending. SmartData Collective. https://www.smartdatacollective.com/how-big-data-changing-nature-of-consumer-lending/

Côrte-Real, N., Ruivo, P., & Oliveira, T. (2017, January 1). Assessing business value of Big Data Analytics in European firms. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0148296316304982

Google. (n.d.). Classification: Accuracy | Machine Learning Crash Course. Google Developers. Retrieved June 26, 2021, from https://developers.google.com/machine-learning/crash-course/classification/accuracy#:%7E:text=Accuracy%20is%20one%20metric%20for,predictions%20Total%20number%20of%20predictions

Teply, P., & Polena, M. (2020, January 1). Best classification algorithms in peer-to-peer lending. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S1062940818302262?casa_token=VhZtxmVItZYAAAAA:ffhaxzcLLt634nO2mNo1vNlPnTtphMqg8yQW2ZvTxGCd8C84ingkqsnaknD2i2Px5VjYrkYgHQ

Nabende, P., & Senfuma, S. (2019). A study of machine learning models for predicting loan status from Ugandan loan applications. Int'l Conf. Artificial Intelligence. https://csce.ucmss.com/cr/books/2019/LFS/CSREA2019/ICA7034.pdf

Huilgol, P. (2021, March 9). *Precision vs. Recall – An Intuitive Guide for Every Machine Learning Person*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

Confusion Matrix. (n.d.). [Image]. Confusion Matrix. https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781838555078/6/ch06lvl1sec34/confusion-matrix

Faggella, D. (2020, April 3). Artificial intelligence applications for lending and loan management. Emerj. https://emerj.com/ai-sector-overviews/artificial-intelligence-applications-lending-loan-management/

Hand, D. J., & Henley, W. E. (1997, September 1). Royal Statistical Society Publications. Royal Statistical Society. https://rss.onlinelibrary.wiley.com/doi/epdf/10.1111/j.1467-985X.1997.00078.x

Baesens, B. (2003). Developing Intelligent Systems for Credit Scoring Using Machine Learning Techniques. [Ph.D. thesis Faculteit Economische en Toegepaste Economische Wetebnschappen, Katholieke Universiteit, Leuven]. http://www.dataminingapps.com/wp-content/uploads/2015/04/Phd-Bart-Baesens.pdf

Bao, W., Lianju, N., & Yue, K. (2019, August 15). *Integration of unsupervised and supervised machine learning algorithms for credit risk assessment*. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S0957417419301472

Fan, S. (2020, November 4). *Improved ML-Based Technique for Credit Card Scoring in Internet Financial Risk Control*. Hindawi. https://www.hindawi.com/journals/complexity/2020/8706285/

Peng, Y., Wang, G., Kou, G., & Shibc, Y. (2011, March 1). *An empirical study of classification algorithm evaluation for financial risk prediction*. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S1568494610003054?casa_token=fwSxDfev9ukAAAAA:CnVqJxzzpuiuEhZcKMlRbxeK06uFpu80EJ6iAu0sJzlIxjqrEuhGoRNldP92jRSdw_UJ7ZXYJA

Turiel, J. D., & Aste, T. (2020, May 18). *Peer-to-peer loan acceptance and default prediction with artificial intelligence*. Royalsocietypublishing. https://royalsocietypublishing.org/doi/pdf/10.1098/rsos.191649

Vieira, J. R. C., Barboza, F., Sobreiro, V. A., & Kimura, H. (2019, October 1). *Machine learning models for credit analysis improvements: Predicting low-income families' default*. ScienceDirect. https://www.sciencedirect.com/science/article/abs/pii/S156849461930420X

Hand, D. J. (2006, February 1). *Classifier Technology and the Illusion of Progress*. Projecteuclid. https://projecteuclid.org/journals/statistical-science/volume-21/issue-1/Classifier-Technology-and-the-Illusion-of-Progress/10.1214/088342306000000060.full

Brownlee, J. (2020, August 15). *8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset*. Machine Learning Mastery. https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

Brownlee, J. (2020b, January 14). *A Gentle Introduction to Imbalanced Classification*. Machine Learning Mastery. https://machinelearningmastery.com/what-is-imbalanced-classification/

Roy, B. (2020, April 7). *All about Feature Scaling - Towards Data Science*. Medium. https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

Multicollinearity. (n.d.). Investopedia. Retrieved May 13, 2021, from https://www.investopedia.com/terms/m/multicollinearity.asp

Gupta, M. (2020, September 2). *What, When and Why Feature Scaling for Machine Learning*. Medium. Retrieved March 15, from https://medium.com/technofunnel/what-when-why-feature-scaling-for-machine-learning-standard-minmax-scaler-49e64c510422

Bock, T. (2020, December 9). *What are Variance Inflation Factors (VIFs)? | Displayr.com*. Displayr. Retrieved April 17, from https://www.displayr.com/variance-inflation-factors-vifs/

Bank Loan Data. (2020, January 23). Kaggle. https://www.kaggle.com/matthew2001/bank-loan-data

Mujtaba, H. (2021, June 23). *Hyperparameter Tuning with GridSearchCV*. GreatLearning Blog: Free Resources What Matters to Shape Your Career! Retrieved May 12, from https://www.mygreatlearning.com/blog/gridsearchcv/

sklearn.model_selection.GridSearchCV — scikit-learn 0.24.2 documentation. (n.d.). Scikit-Learn.Org. Retrieved June 28, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Yıldırım, S. (2020, June 1). *Hyperparameter Tuning for Support Vector Machines — C and Gamma Parameters*. Medium. https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167

Bhosale, G. (2020, July 18). *Support Vector Machines (SVM)*. All Geek Life. http://www.allgeeklife.com/support-vector-machines-svm/

Bhosale, G. (2020, July 18). [Image]. *Support Vector Machines (SVM)*. All Geek Life. http://www.allgeeklife.com/support-vector-machines-svm/

sklearn.svm.SVC — scikit-learn 0.24.2 documentation. (n.d.). Scikit-Learn. Retrieved June 16, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Swaminathan, S. (2019, January 18). *Logistic Regression — Detailed Overview - Towards Data Science*. Medium. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Brownlee, J. (2020c, August 15). [Image]. *Logistic Regression for Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/logistic-regression-for-machine-learning/

Brownlee, J. (2020c, August 15). *Logistic Regression for Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/logistic-regression-for-machine-learning/

L1 Penalty and Sparsity in Logistic Regression — scikit-learn 0.24.2 documentation. (n.d.). Scikit-Learn. Retrieved May 18, 2021, from https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html

sklearn.linear_model.LogisticRegression — scikit-learn 0.24.2 documentation. (n.d.). Scikit-Learn. Retrieved June 28, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

K, D. (2020, December 26). *Top 4 advantages and disadvantages of Support Vector Machine or SVM*. Medium. https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107

GeeksforGeeks. (2020a, September 2). *Advantages and Disadvantages of Logistic Regression*. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logistic-regression/

Allison, P. (2019, November 27). *When Can You Safely Ignore Multicollinearity? | Statistical Horizons*. Statistical Horizons | Statistics Training That Makes Sense. https://statisticalhorizons.com/multicollinearity

sklearn.model_selection.train_test_split — scikit-learn 0.24.2 documentation. (n.d.). Scikit-Learn. Retrieved June 28, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Sreenivasa, S. (2020, October 12). *Radial Basis Function (RBF) Kernel: The Go-To Kernel*. Medium. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a

Splitting Datasets With the Sklearn train_test_split Function. (2019, November 25). BitDegree. https://www.bitdegree.org/learn/train-test-split

Wikipedia contributors. (2021, April 23). *Hyperparameter (machine learning)*. Wikipedia. https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

Charleshsliao, V. A. P. B. (2017, May 20). *Logistic Regression in Python to Tune Parameter C*. Charles' Hodgepodge. https://charleshsliao.wordpress.com/2017/05/20/logistic-regression-in-python-to-tune-parameter-c/