Classification Project

Python Fundamentals for Machine Learning

Group 9

Andreas Rumin Kjær - 202208867

Emil Daasbjerg - 202009199

Filip Mzyk - 202402549

Magnus Sejer Lind - 202005246

Aarhus University

07/11/2025

Contents

1 Introduction **3**

2 Data **4**
2.1 Dataset description 4
2.2 Data pre-processing 5
2.2.1 Data cleaning 6
2.2.2 Data manipulation 7
2.2.3 Data visualization 8
2.2.4 Data correlation 10

3 Machine Learning **12**
3.1 Theory 12
3.1.1 Support Vector Classifier 13
3.1.2 Random Forest 13
3.2 Experimental setup 14
3.2.1 The default models 14
3.2.2 Hyperparameter tuning 15
3.2.3 Removing the credit score 15
3.3 Evaluation and Performance metrics 16
3.4 Feature importance 18

4 Results **19**
4.1 Main setup 19
4.2 Removal of the credit score 21

5 Discussion **22**
5.1 Key Findings 22
5.2 Further research 22
5.3 Limitations 23

6 Conclusion **24**

7 Appendix **26**

1 Introduction

The financial crisis of 2008 left a lasting impact on the banking sector worldwide. Over the years, many analyses have been performed in order to determine what caused the collapse of the stock market and, later on, the economies at large, such as in [1]. The various approaches can be summed up, in layman's terms, in very few words, "excessive risk taking", be it risk of particular financial instruments, like Credit Default Swaps, or the excessive risk of mortages given by banks, due to the relaxed lending standards present prior to the crisis [2].

In the years after the crisis, society finds itself in an interesting situation, where banks are more careful with their loan approval process, while people want to finance their endevaours in the evolving economy. At the same time, despite a more throught approval process, the credit line (or loans in general) continues to be the fundamental way a bank funds its existence, and creates profit [3].

Especially in the USA, a Credit Score of an individual, a numeric value representing the "creditworthiness" is used to either accept or reject a person's loan request. In the days of Big Data and easily accessible Machine Learning methods, it would be logical to implement other characteristics of an individual in the approval process. This is not unheard of, especially when considering China's Social Credit System, in which various actions and characteristics of an individual or a business are taken into account when requesting loans, searching for work, or purchasing an apartment [4].

This report tries to identify whether, on a given dataset, the loan approval outcome (Accepted or Rejected), can be predicted based on further characteristics, and not only from the credit score. As such, our problem can be characterized as a classification one in the machine learning domain. In order to predict the Accepted or Rejected state, two machine learning algorithms are employed: a linear Support Vector Classifier (SVC) and a Random Forest ensemble method. Both are tested in their default states as well as in hyperparameter-tuned versions.

The dataset used for the analysis in this report has been acquired from Kaggle. For each individual's loan approval outcome (Accepted or Rejected) the dataset includes information such as the credit score, the loan's term, the number of dependents in the household, the individual's education, and their assets. This creates a promising experimentation ground, in which we try to identify other viable predictors, possibly initially hidden, which may be useful during the loan approval process.
This problem can be viewed from both the bank's point of view, as well as the individual who requests a loan. The bank could use our analysis as a guide to, at least in the first steps of the loan approval process, partially automate the admission procedure. The individual could use the model by inputting his or hers data in order to verify a likely outcome of the loan status (accepted or rejected).

Figure 1.1 is a description of the 5 steps of using ML, which will be followed in this project. The report begins with a section inserting and describing the dataset, followed by a section on data pre-processing, where cleaning, manipulation, and visualization are performed. The report continues with a chapter on machine learning where the implemented models are presented, together with some theoretical explanation. The next section presents the execution of the algorithms, together with a "what-if" scenario where the credit score is removed from the predictors, evaluation metrics, and the general performance evaluation.

The report ends with an aggregated presentation of the results, a discussion of said results, and a conclusion where limitations and future research are discussed.

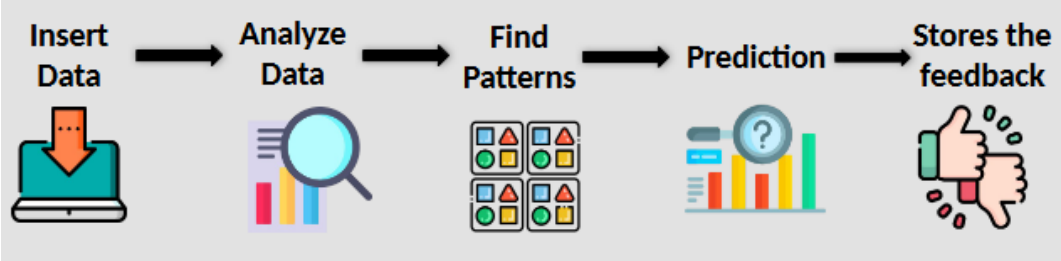


Figure 1.1: The 5 steps of ML

2 Data

2.1 Dataset description

This project is based on the dataset "Loan Approval" by Vishav Gupta, available at https://www.kaggle.com/datasets/vishavgupta01/loan-approval. The dataset is synthetic and contains 13 columns with 4269 rows. The columns contain information about the loan application status and the applicant. A complete list of the columns is provided in Table 2.1. There are very minimal descriptions provided about the data columns, therefore, some assumptions are made about the meaning of the data. For example, we assume that the currency used in the dataset is the Indian Rupee, or that the *loan_term* is in years.

Data columns	**Description**
loan_id	Identification number of the loan application
education	Binary education level of applicant
self_employed	Is the applicant self-employed?
income_annum	Annual income of applicant
loan_amount	The amount of loan requested by the applicant
cibil_score	Credit score of the applicant
loan_term	Term of requested loan
no_of_dependents	Number of applicant's dependents (eg. children)
bank_asset_value	Value of applicant's bank assets
residential_assets_value	Value of applicant's residential assets
commercial_assets_value	Value of applicant's commercial assets
luxury_assets_value	Value of applicant's luxury assets
loan_status	Status of the loan application (Approved/Rejected)

Table 2.1: Description of data columns

To gain further understanding of the data, before starting the pre-processing, the data types and some statistical parameters are investigated using the `.info()` and `.describe()` functions. The function output may be seen in Figure 2.1 and 2.2 respectively. It may be

observed that some of the columns are given as objects, this requires some data manipulation, and some encoded variable columns are implemented (_enc). This is described later on in Section 2.2.2 . Furthermore the data is visualized in Section 2.2.3.

```
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   loan_id                   4269 non-null   int64
 1    no_of_dependents         4269 non-null   int64
 2    education                4269 non-null   object
 3    self_employed            4269 non-null   object
 4    income_annum             4269 non-null   int64
 5    loan_amount              4269 non-null   int64
 6    loan_term                4269 non-null   int64
 7    cibil_score              4269 non-null   int64
 8    residential_assets_value 4269 non-null   int64
 9    commercial_assets_value  4269 non-null   int64
 10   luxury_assets_value      4269 non-null   int64
 11   bank_asset_value         4269 non-null   int64
 12   loan_status              4269 non-null   object
dtypes: int64(10), object(3)
```

Figure 2.1: Data information (loans.info())

```
                          count         mean         std       min  \
 no_of_dependents          4269.0         2.50        1.70       0.0
 income_annum              4269.0   5059123.92  2806839.83  200000.0
 loan_amount               4269.0  15133450.46  9043362.98  300000.0
 loan_term                 4269.0        10.90        5.71       2.0
 cibil_score               4269.0       599.94      172.43     300.0
 residential_assets_value  4269.0   7472616.54  6503636.59 -100000.0
 commercial_assets_value   4269.0   4973155.31  4388966.09       0.0
 luxury_assets_value       4269.0  15126305.93  9103753.67  300000.0
 bank_asset_value          4269.0   4976692.43  3250185.31       0.0
loan_status_enc            4269.0         0.62        0.48       0.0
education_enc              4269.0         0.50        0.50       0.0
self_employed_enc          4269.0         0.50        0.50       0.0

                               25%         50%         75%         max
 no_of_dependents              1.0         3.0         4.0         5.0
 income_annum            2700000.0   5100000.0   7500000.0   9900000.0
 loan_amount             7700000.0  14500000.0  21500000.0  39500000.0
 loan_term                     6.0        10.0        16.0        20.0
 cibil_score                453.0       600.0       748.0       900.0
 residential_assets_value 2200000.0   5600000.0  11300000.0  29100000.0
 commercial_assets_value  1300000.0   3700000.0   7600000.0  19400000.0
 luxury_assets_value      7500000.0  14600000.0  21700000.0  39200000.0
 bank_asset_value         2300000.0   4600000.0   7100000.0  14700000.0
loan_status_enc                0.0         1.0         1.0         1.0
education_enc                  0.0         1.0         1.0         1.0
self_employed_enc              0.0         1.0         1.0         1.0
```

Figure 2.2: Data description (print(df.describe().round(2).T))

2.2 Data pre-processing

Before applying structured machine learning, it is essential to preprocess the data to ensure its quality and consistency for analysis. In structured ML, the presence of missing values and inconsistent data formats can significantly reduce the model's performance.

The following section will cover the data preprocessing steps applied to clean and prepare the structured dataset in a way that maximizes the effectiveness and accuracy of the machine learning models.

2.2.1 Data cleaning

It is important to ensure that the dataset has been cleaned because missing values will reduce the quality of the machine learning algorithm. Additionally, most ML algorithms cannot work with missing values [5].

The acquired data set is synthetic, so it is not expected to have any NaN values. But just to be sure, the first step is to write a function that checks the number of missing values in our dataset. This is done with the following code, which searches for NaN values in the dataset and sums how many times it finds a NaN value in each respective column.

```
print('Null values present:\n', loan_app.isna().sum())
```

As expected, Figure 2.3 shows that when the code was run, no NaN values were found, which can be seen in the table below. Additionally, a code was written to check for duplicates in all rows of the 'loan_id' column of the dataset - no duplicates were found. The entire code can be found in the Google Colab file A.1.

```
Null values present:
 loan_id                    0
 no_of_dependents           0
 education                  0
 self_employed              0
 income_annum               0
 loan_amount                0
 loan_term                  0
 cibil_score                0
 residential_assets_value   0
 commercial_assets_value    0
 luxury_assets_value        0
 bank_asset_value           0
 loan_status                0
```

Figure 2.3: none values present in dataset

Because of this, no further data cleaning is needed. However, if NaN values were found, there would be two approaches to be considered:

1. Delete rows or columns with NaN values

   ```
   df.dropna()    # Delete all rows which contain NaN values
   df.dropna(axis=1)   # Delete all collums which contain NaN values
   ```

2. Replace the NaN values with appropriate substitutes (e.g., 0, 1, the mean, etc.)

   ```
   df.fillna(x)   # Replace NaN's with x
   s.fillna(s.mean())   # Replace NaN's with mean value of that column
   ```

2.2.2 Data manipulation

The first data manipulation that was performed, was to remove the spacing before the column names. Additionally a for loop was made to remove all spacing for all rows of column "education" and "loan_status". The code is shown below:

```
df.columns = df.columns.str.strip()   # removes spacing from all columns.
for col in ["education", "loan_status"]:
    df[col] = df[col].str.strip()     # removes spacing of all rows for specific columns.
```

Most machine learning algorithms perform better with numerical data, so the columns with string data was encoded into numerical values [5]. The code which was used is shown in Figure 2.4.

loan_status:
The loan_status column was label encoded into a new column loan_status_enc, where 'Rejected' will be converted to 0 and 'Approved' to 1.

education:
The education column was similarly encoded into education_enc, assigning 0 to 'Not Graduate' and 1 to 'Graduate'.

self_employed:
The self_employed column was label encoded into self_employed_enc, where 'No' corresponds to 0 and 'Yes' to 1.

loan_id:
Lastly the loan_id column was set as the index of the DataFrame, since it only serves as a unique identifier and does not provide any value for determining loan approval outcomes.

```
# Make the loan ID column the Index
df = loans.set_index('loan_id', inplace = False)
```

```
# Label encoding
# dtype set as int64 for consistency
df['loan_status_enc'] = df['loan_status'].map({
    'Rejected' : int(0),
    'Approved' : int(1)
})

df['education_enc'] = df['education'].map({
    'Not Graduate' : int(0),
    'Graduate' : int(1)
})

df['self_employed_enc'] = df['self_employed'].map({
    ' No' : int(0),
    ' Yes' : int(1)
})

print(df.info())
```

Figure 2.4: Data manipulation

2.2.3 Data visualization

Before applying any machine learning algorithm, it is important to get a good understanding of the dataset yourself. Both for troubleshooting when training and testing the ML model, but also for picking out the most effective model. A strong tool for gaining understanding of the dataset is the pair plot via the *seaborn* module in python, which gives a wide range of scatter plots. We are not going to cover all the scatterplots from the pairplot, but we will look at two that are the most representative of our pairplot (All of the scatterplots can be seen in the Appendix).

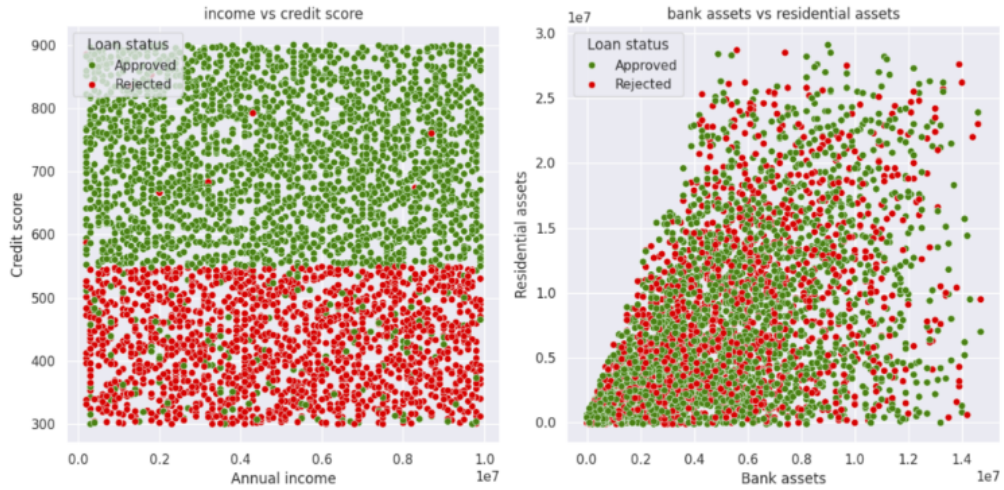


Figure 2.5: Credit score against annual income and Residential assets vs Bank assets

For the scatterplot with credit score against annual income, we see a linear separation at around 550 credit score. However, annual income does not show any effect on the approval. We also see some outliers, especially in terms of the loans being approved, even though they are under the 550 credit score threshold. This, of course, makes sense from the standpoint of the bank. This is one of the first indicators that the credit score is going to have an important effect on the approval, the loan status *y label*.
For the scatterplot with residential assets against bank assets, we see a correlation between the residential assets and the bank asset, however, we see no clear pattern in terms of bank loan approvals from these factors. We have a suspicion that some pattern could be hidden in the dataset for the ML models to discover, especially since it is synthetic.

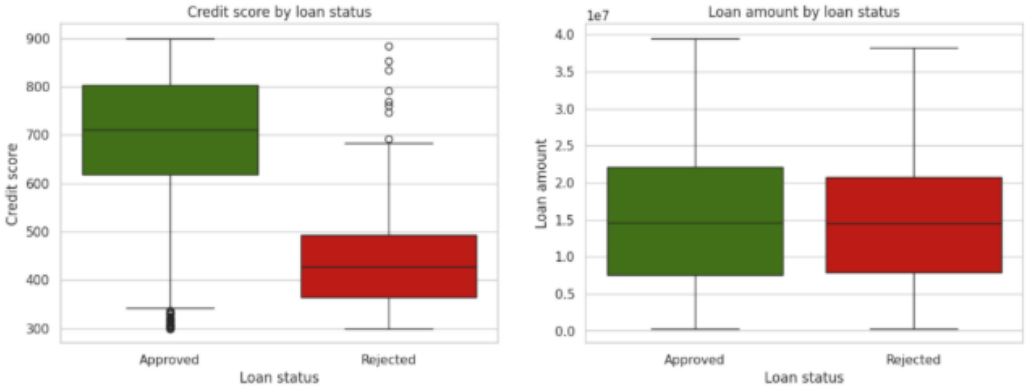


Figure 2.6: Boxplots of credit score with loan status

The box plot of the credit score against the loan status confirms what we saw from the scatter plots, but outliers are seen more clearly. The median for credit score is just above 700 and the median for rejected is 450.
Again for the box plot of loan amount against the loan status we confirm what we saw from

the scatterplot. We see no patterns, with the medians being nearly identical.

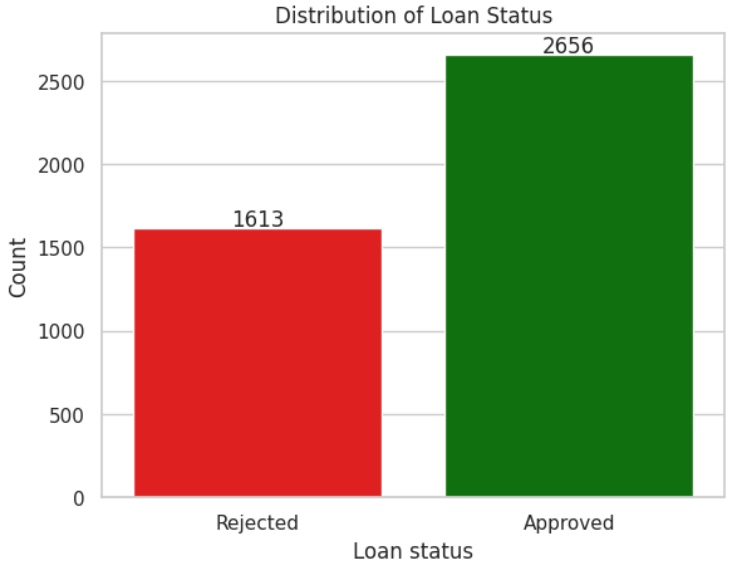


Figure 2.7: Loan status count

A count of the approved vs rejected was performed. We see a clear bias, with 62.2% approved and 37.8% rejected. We will take this into consideration for choosing the ML-model

2.2.4 Data correlation

To develop a further understanding of the relationships between the columns in the dataset, a correlation matrix is calculated using the `corr()` function. The correlation matrix of all of the correlation coefficients ρ of the data columns can be calculated by applying the following formula:

$$\rho_{X,Y} = \frac{n(\sum XY) - (\sum X)(\sum Y)}{\sqrt{[n\sum X^2 - (\sum X)^2][n\sum Y^2 - (\sum Y)^2]}} \tag{2.1}$$

where n is the number of observations and X and Y are the data column variables, it may be expressed as the ratio between the covariance and the standard deviation σ of the variables as

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{2.2}$$

The range of the correlation coefficient is thereby [-1,1] where a correlation coefficient of 1 indicates a perfect positive linear relationship, meaning that as one variable increases, the other variable increases proportionally. A correlation coefficient of -1 indicates a perfect negative linear relationship, meaning that as one variable increases, the other decreases proportionally. A correlation coefficient of 0 indicates no linear relationship between the two variables. Values between -1 and 1 indicate the degree of linear relationship, with values closer to 1 or -1 signifying a stronger linear relationship and values closer to 0 indicating a weaker linear relationship. [6]

The full correlation matrix of all the data columns may be seen in Figure 2.8

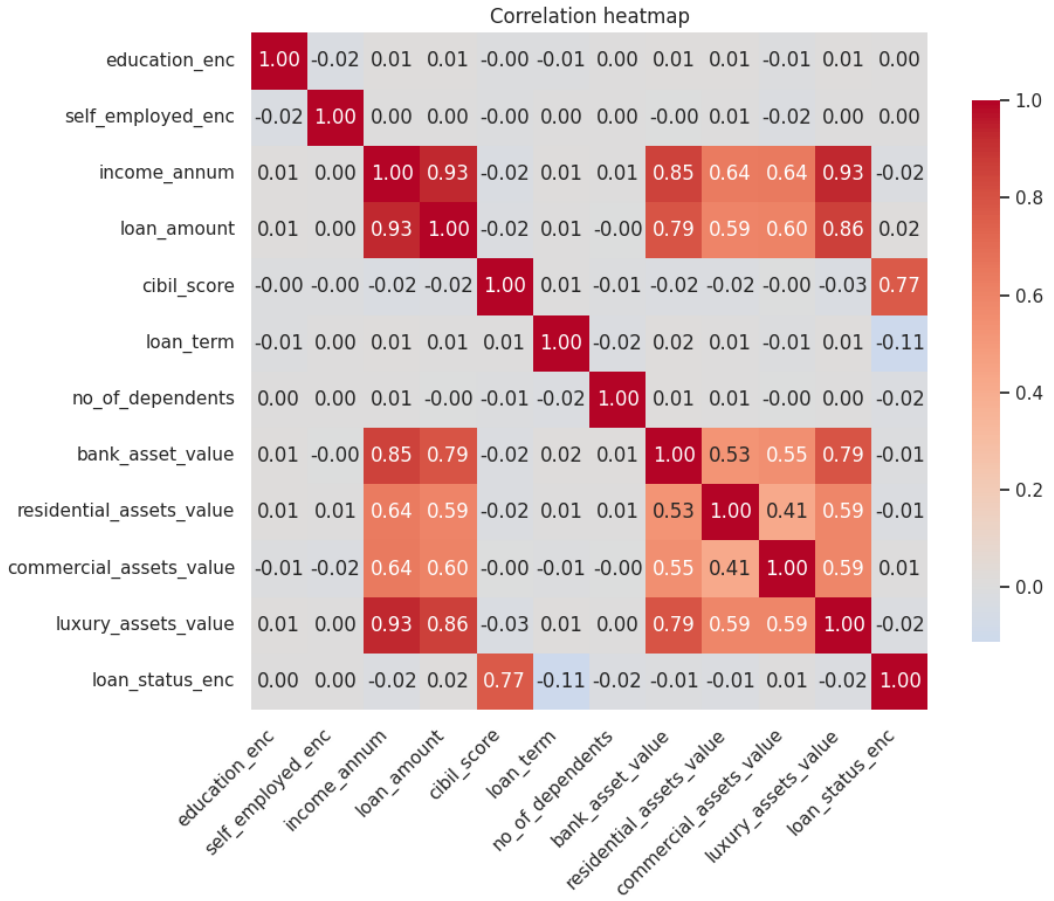


Figure 2.8: Correlation matrix

It may be observed that there is a block of relatively highly correlated variables in the bottom right corner, all representing a particular category of assets, the setup of combining these into a total asset variable (`total\_assets`) is thereby implemented as

$$\begin{aligned} \text{total\_assets} = \text{residential\_assets\_value} + \text{commercial\_assets\_value} \\ + \text{luxury\_assets\_value} + \text{bank\_asset\_value} \end{aligned} \tag{2.3}$$

The heatmap of the correlation matrix using the total asset variable may be seen in Figure 2.9

Figure 2.9: Correlation matrix using total assets

From the correlation matrices, it may be observed that the only statistical significant relationship between the ylabel loan_status and any other variable is the credit score (cibil_score), which has a correlation coefficient of

$$\rho_{cibil\_score,\ loan\_status\_enc} = 0.77 \tag{2.4}$$

3 Machine Learning

3.1 Theory

Both methods used in this report come from the field of machine learning, which in itself is a subfield of artificial intelligence that focuses on developing systems where a computer can learn patterns from data. The goal of those systems is to acquire accurate predictions or uncover hidden patterns.

In our setup, the problem is a supervised learning one, which means the algorithms are fed labeled data (each outcome corresponds to a set of other variables). Furthermore, it is a classification problem, where we try to predict a binary outcome, Accepted (1) or Rejected (0).
In this section, a brief theoretical foundation of the two models used in the analysis is presented. The models are described using some mathematical notation, but a graphical representation, where possible, is also included.

3.1.1 Support Vector Classifier

Our first model is the Support Vector Classifier (SVC), which is a specific application of the Support Vector Machine constructed for classification tasks. Based on our data, we decided to choose a simple linear kernel for our model, expecting it to use the information provided by the credit score variable (*cibil_score*). It is easy to visually showcase how the SVC works, as seen in 3.1. In simple terms, the SVC model tries to separate the data-points into the two groups (Accepted or Rejected) by applying a straight line.

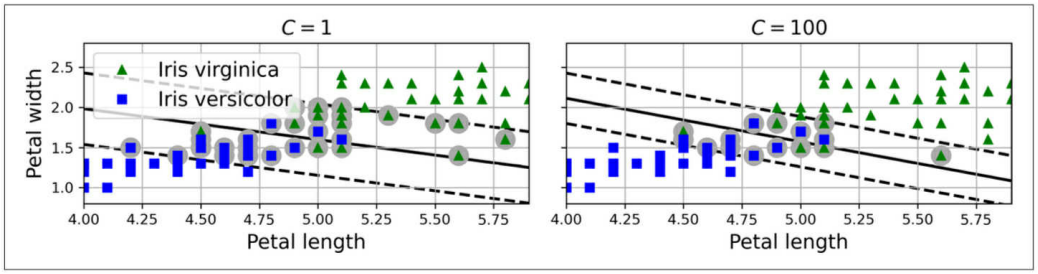


Figure 3.1: SVC and margins visualization adapted from [5]

The main decision to be made when applying the SVC, apart from the kernel, is the C hyperparameter. It dictates how wide the margins of the linear separation are. Usually, the larger the margin (low C), the more margin violations (data-points present inside the separation or even separated wrongly 3.1. The objective is to find a good balance between the width of the margin and the amount of margin violations [5]. The choice of the C parameter boils down, as it usually does in machine learning, to how much we overfit or underfit our model, leading to better or worse performance when generalizing (predicting).

As noted in [5], this type of model performs better when scaling of the features is applied. We achieve this by applying the *StandardScaler* of the *Sklearn* python package on our training and testing data that will be given to the SVC model (See Appendix A.2 for code snippet).

3.1.2 Random Forest

The Random Forest algorithm is an ensemble method, based on decision trees (for a simple decision tree example see Appendix A.1), introduced in 2001 [7]. It tries to improve not only on the simplistic decision tree algorithm, but also on the bagged (or bootstrap aggregated) trees algorithm. The problem with the decision tree algorithm is its tendency to overfit the data, making it inefficient in predicting during the testing phase. The bagged trees try to fix that by growing trees on bootstrap samples of the original data, and basing their prediction on the average of all the trees. However, the bootstrapped trees may present a high degree of correlation with each other. This is what Random Forest improves on. When splitting

each bootstrapped tree, the Random Forest also randomly chooses the variables on which the split will be performed.
Nevertheless, the following formula is the same whether talking about Bagged Trees or Random Forest. The formula presents the final prediction $\hat{f}(x)$, where $\hat{f}^{*b}(x)$ denotes the b-th bootstrap prediction, and B is the total number of bootstrap samples.

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x) \tag{3.1}$$

In our case, instead of considering all eleven variables at each split, the Random Forest will consider only a subset of those each time, as seen in 3.2. Ideally, this results in tree diversity, trading higher bias for lower variance [5]. We believe that the Random Forest will perform well in our setup. Firstly, because of the imbalanced dataset, which ensemble methods tend to handle well, and secondly, detecting possible non-linear relationships in the data.

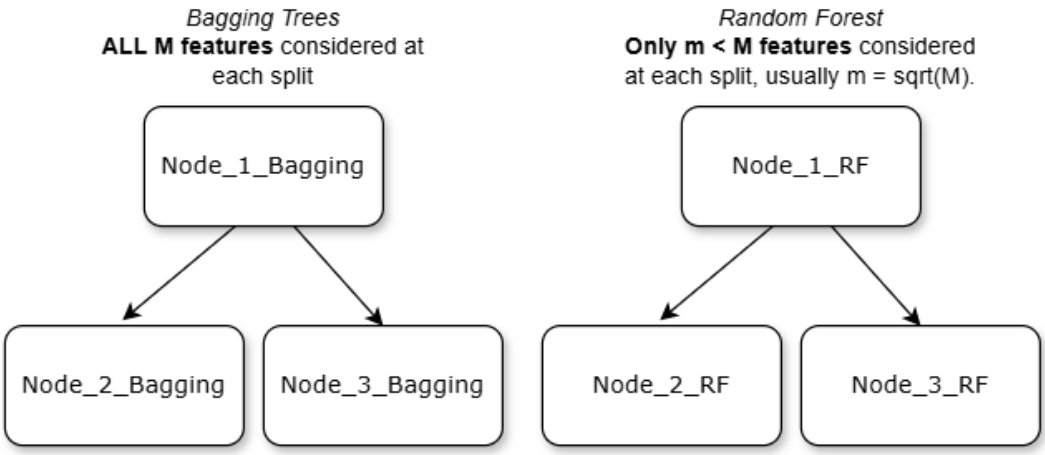


Figure 3.2: Bagging versus Random Forest.

The main drawback of a Random Forest is the difficulty of representing it in its entirety (akin to a Black-Box model), as opposed to a single tree. On the other hand, a big advantage is the minimal data pre-processing needed, and the possibility of graphing the feature importance, where each variable is presented in terms of how important it was on the tree's decision to split, something that is shown later in the report.

3.2 Experimental setup

In this section, we present our experimental set-ups. It is important to note that for each one, the data is split into 80% training and 20% testing chunks. Since our data is not temporal, the rows of data in each chunk are chosen randomly. Another general note is that the training and testing data is always scaled for the SVC model (see Appendix for the code snippet A.2), as is usually advised [5].

3.2.1 The default models

Here we briefly describe the settings of our two models in their default state. As seen from the table following table 3.1, for our SVC model, we opted for a linear kernel, believing that

a simple line could be enough to separate the two binary outcomes thanks to the clear cutoffs created by the credit score. As for the C hyperparameter, we set it to 1000, prioritizing correct classification, but creating a narrower margin, which could have an effect on the model's predictive ability.

	SVC	RandomForest
C	1000	*NonApplicable*
n_estimators	*NonApplicable*	100
max_depth	*NonApplicable*	*None*

Table 3.1: The default models' hyperparameters

Our default Random Forest is set to grow 100 trees, the *n_estimators* parameter in the table. *RandomForestClassifier* from the *sklearn* package sets the maximum depth to which the trees grow to *None* as the default value. This means that the trees grow until their leafs are pure.

3.2.2 Hyperparameter tuning

An attempt was made to optimize and improve the two machine learning models through hyperparameter tuning.
Hyperparameter tuning refers to the process of adjusting a model's hyperparameter or hyperparameters. In our case, this is done using the *GridSearchCV* of the *Sklearn* package (See Appendix A.3 for code snippet). Due to computational restrictions, we opted for tuning the hyperparameters using randomly chosen fixed values from a grid, and not a whole range. The process can be described as:

1. Creating a grid of hyperparameters.
2. Training the model on each combination in the grid.
3. Evaluating each model using cross-validation.
4. Selecting the best combination.

In our setup, the hyperparameter to be tuned in the SVC model is the C regularization parameter, which determines the margins that the SVC tries to maximize. A smaller C value creates larger margins and may lead to misclassifications in the training data, but a better predictive performance. The C values tested are $[0.1, 10, 10000]$.
In the Random Forest, we tune the *n_estimators*, determining the number of trees in the forest, and the *max_depth*, determining how far a tree grows. The grid was randomly generated, resulting in $[69, 234, 378]$ for *n_estimators* and $[17, 6, 14]$ for *max_depth*.
Our original training dataset is split into 5 folds for the purpose of evaluating the models. A usual choice during hyperparameter optimization.
The resulting best parameter for the SVC model is $C = 0.1$ and for the RF model *n_estimators* $= 378$ and *max_depth* $= 17$.
After this process is done, the best SVC and RF models are fitted once again on the entirety of the training dataset. The usual procedure of predicting on the testing dataset is conducted, with the results presented later in the report.

3.2.3 Removing the credit score

The high correlation of credit score (*cibil_score*) with the outcome of the loan approval process was already determined when exploring the data. As it will be further seen, as was

expected, the models heavily rely on this particular variable to predict the Approved or Rejected classification.
In our report, we decided to conduct a "what-if" experiment, in which we put our best models to a test. The test aims to answer questions such as:

- What would happen if we removed the credit score?
- How dependent are the models on this variable?
- Is there enough information in the other variables for the models to perform in an acceptable manner?

In order to perform the experiment we create a new train and test dataset, in which the *cibil_score* column is dropped. The dataset for the SVC model is again scaled. Both the best SVC model and the best RF model are fitted on the training data, and then used to predict by inputting the training data. We expect the SVC model to perform much worse in comparison to the Random Forest one, as it should be more challenging for a linear kernel to split the data now. For the sake of investigating the matter further, we also tried a polynomial kernel. Results are discussed in the appropriate section of the report.

3.3 Evaluation and Performance metrics

To figure out the quality of the trained ML algorithms, different evaluation methods can be used. This section gives a general overview of the evaluation methods, and in chapter 4 the results of these will be shown. The fastest way to gain insight into the accuracy of an ML algorithm is the "accuracy_score" function from *sklearn.metrics*. This function can be used to calculate the proportion of correctly classified predictions when comparing the ML algorithm's y_predictions from the training dataset with the actual results from the y_test dataset. The result is an accuracy level from 0–1, where 1 is 100% correct predictions.

A better way to assess the performance of the ML algorithm is the confusion matrix. Figure 3.3 is an illustration of the confusion matrix, which highlights the four outcomes of classification [5]. The confusion matrix consists of four quadrants, where the essence is to count the number of times a class is correctly classified and the number of times it is falsely classified as another class. The reason why this is better than the "accuracy" is that the confusion matrix provides information about which classes/columns have low precision by showcasing the number of true positives, false positives, true negatives, and false negatives.

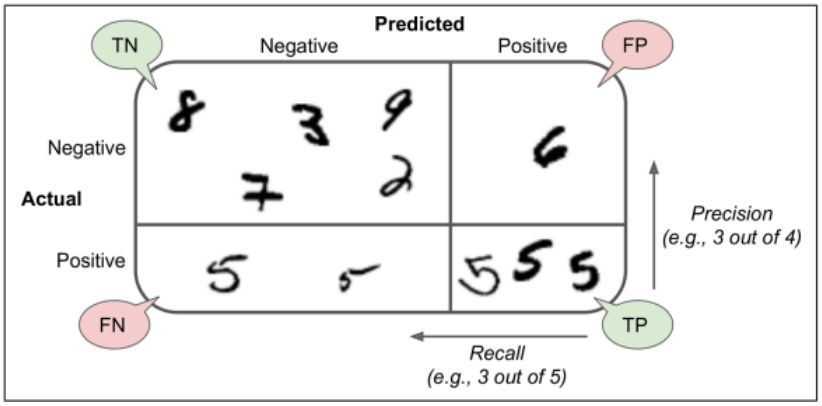


Figure 3.3: Illustration of Confusion matrix with true negatives in the top-left, false positives in the top-right, false negatives in the bottom-left, and true positives in the bottom-right corner. [5]

Additionally, the information from confusion matrix can be used to calculate the precision and recall. The precision is a calculation of how many of the positive predictions were actually correct out of all positive predictions the model made for a specific class:

$$\text{Precision} = \frac{TruePositive}{TruePositive + FalsePositive}$$

Recall measures how well the model can identify the true positive out of all positive cases, which is done by considering the false negatives:

$$\text{Recall} = \frac{TruePositive}{TruePositive + FalseNegative}$$

Lastly we have the f1-score, which is often used to compare the performance of multiple models. The f1-score is a harmonic mean of the precision and the recall, which ensures that f1-score will only be high if both the precision and recall values are high [5]. The equation for f1-score can be seen below:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN+FP}{2}}$$

It is desirable to achieve both high precision and high recall, but there is often a trade-off between them. Figure 3.4 demonstrates how precision and recall are influenced by the decision threshold. When the threshold is low, the model classifies more cases as positive, which increases recall because it captures more of the actual positive cases. However, this also leads to more false positives, which lowers precision. On the other hand, raising the threshold makes the model more conservative in predicting positives. This tends to improve precision, since fewer false positives are included, but it reduces recall, as more true positives are missed.

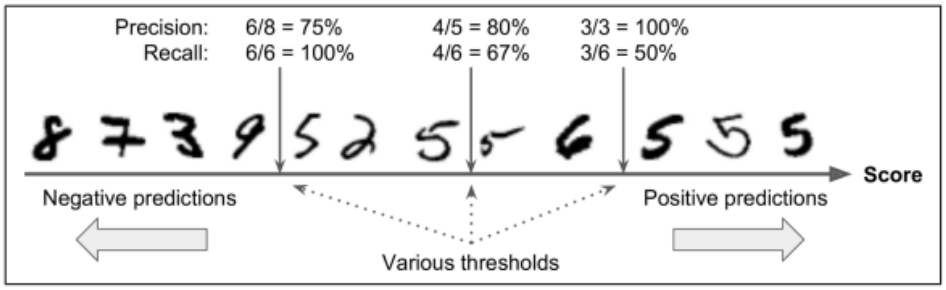


Figure 3.4: Precision and recall trade-off [5]

When looking at precision and recall from the point of view of the banks, precision is likely the most important factor, because it relates directly to the risk of whether the customer will pay back the loan. With high precision, the bank avoids approving customers who are false positives (which would lead to financial loss), whereas rejecting a good customer (a false negative) is financially less risky.

3.4 Feature importance

From the summation of the individual decision trees in the random forest classifier, the features of the dataset can be ranked in terms of their importance for the classification. The importance refers to how much a particular feature is used during a tree's splitting decision. The normalized importance scores of the features is given in Figure 3.5. It may be observed that the credit score is by far the most important feature for classification, which is also in agreement with the correlation matrix of the data.

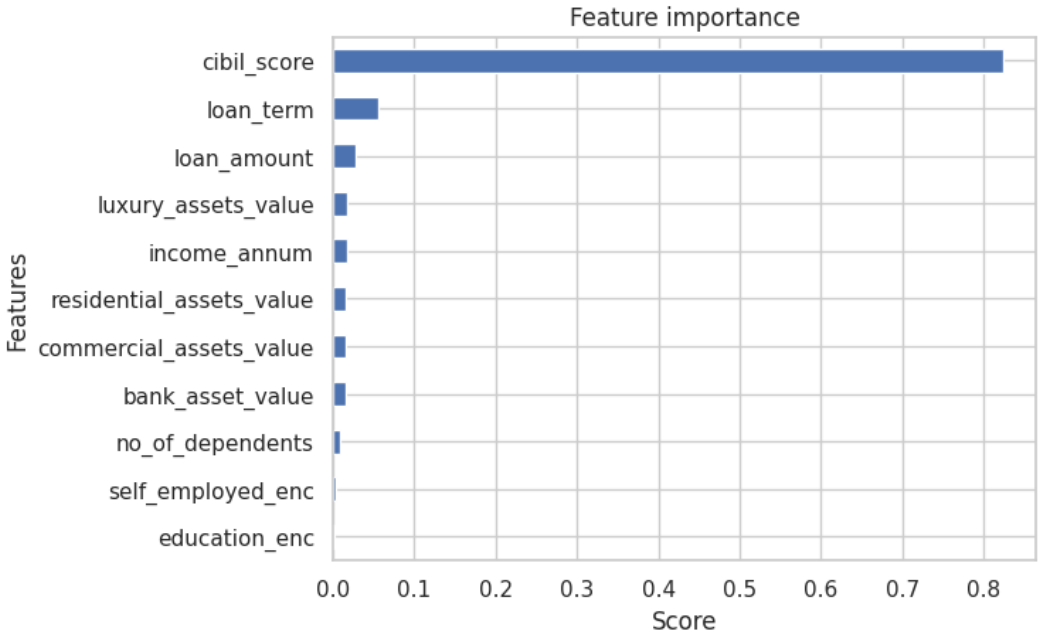


Figure 3.5: Feature importance

4 Results

4.1 Main setup

The setup of both the default and the tuned SVC and RandomForest models was discussed in chapter 3.2. In this section, the results of the different models will be shown, and in chapter 5 they will be discussed.

Figure 4.1 shows the code and accuracy of the default/tuned SVC vs the default/tuned RandomForest - the results are 0.909, 0.912, 0.980, and 0.982 respectively.

```
print(accuracy_score(SVC_def_prediction, y_test))
```

```
0.9086651053864169
```

(a) Accuracy score of the default SVC

```
print(accuracy_score(SVC_tun_prediction, y_test))
```

```
0.9121779859484778
```

(b) Accuracy score of the tuned SVC

```
print(accuracy_score(rf_def_prediction, y_test))
```

```
0.9800936768149883
```

(c) default RandomForest

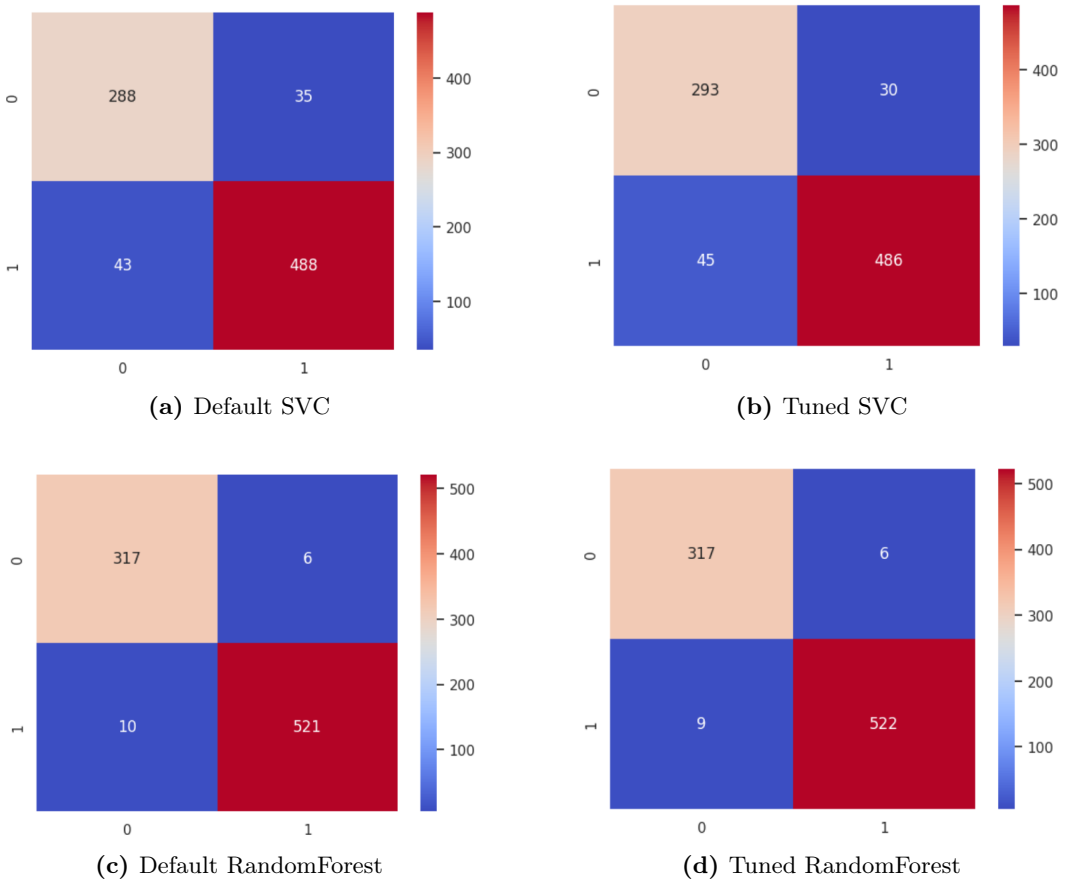
```
print(accuracy_score(rf_tun_prediction, y_test))
```

```
0.9824355971896955
```

(d) Tuned RandomForest

Figure 4.1: Accuracy scores of default/tuned SVC and RandomForest

Figure 4.2 shares an overview of confusion matrices for default and tuned SVC and Random-Forest. The -y-axis of figure 4.2 is the actual 0 and 1, while the X-axis shows the predicted 0 and 1. The changes in results when going from default to tuned are minor, but tuning shows a positive impact by increasing the number of True positves and true negatives. Additionally it can be seen that the RandomForest significantly reduce the number of False positive and False negatives.



(a) Default SVC **(b)** Tuned SVC

(c) Default RandomForest **(d)** Tuned RandomForest

Figure 4.2: Overview of confusion matrices of default/tuned SVC and RandomForest

Figure 4.3 shows the classification report, which include precision, recall and f1-score. Again It can be seen that the tuning makes the model slightly better overall when compared with the default model and the precision, recall and f1-score is way better for the RandomForest when compared with SVC. Additionally, it can be seen that the SVC models are biased towards approvals (1's), since the precision and recall is far better for approval than rejected - this correlates with the fact that 62% of the dataset results in loan approvals.

```
print(classification_report(SVC_def_prediction, y_test))
```

	precision	recall	f1-score	support
0	0.89	0.87	0.88	331
1	0.92	0.93	0.93	523
accuracy			0.91	854
macro avg	0.91	0.90	0.90	854
weighted avg	0.91	0.91	0.91	854

(a) Classification report for the default SVC

```
print(classification_report(SVC_tun_prediction, y_test))
```

	precision	recall	f1-score	support
0	0.91	0.87	0.89	338
1	0.92	0.94	0.93	516
accuracy			0.91	854
macro avg	0.91	0.90	0.91	854
weighted avg	0.91	0.91	0.91	854

(b) Classification report for the tuned SVC

```
print(classification_report(rf_def_prediction, y_test))
```

	precision	recall	f1-score	support
0	0.98	0.96	0.97	330
1	0.98	0.99	0.98	524
accuracy			0.98	854
macro avg	0.98	0.98	0.98	854
weighted avg	0.98	0.98	0.98	854

(c) Default RandomForest

```
print(classification_report(rf_tun_prediction, y_test))
```

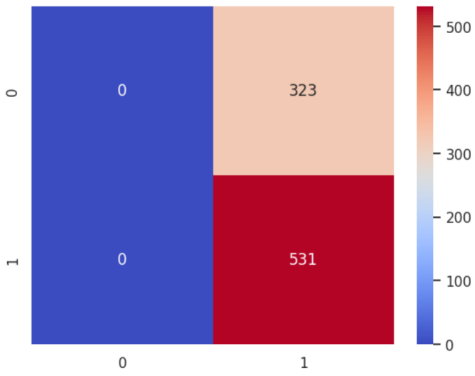
	precision	recall	f1-score	support
0	0.98	0.97	0.98	326
1	0.98	0.99	0.99	528
accuracy			0.98	854
macro avg	0.98	0.98	0.98	854
weighted avg	0.98	0.98	0.98	854

(d) Tuned RandomForest

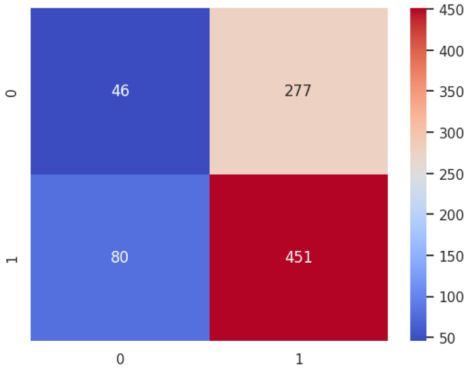
Figure 4.3: Comparison of classification reports for default/tuned SVC and RandomForest model

4.2 Removal of the credit score

For the setup with the credit score removed we apply the tuned versions of our SVC and a RandomForest models, as determined earlier. As expected, the predictions are much less accurate in comparison to the earlier case with the credit score included. The resulting accuracy from our "what-if" scenario is given to be 0.622 for the tuned SVC model and 0.582 for the tuned RandomForest model. Those results are also indicative of the bias of our dataset, the SVC model, in this case, categorizes as "Approved" all of the testing data.
The confusion matrix for our "what-if" scenario of the tuned SVC and tuned RandomForest models may be seen in Figure 4.4. In accordance with the accuracy metrics, the classification performance is significantly decreased.



(a) Tuned SVC without credit score



(b) Tuned random forest without credit score

Figure 4.4: Confusion matrices without credit score

To try to improve the classification predictions without the credit score, an SVC model with

a third-degree polynomial kernel was tried out. This resulted in an accuracy of 0.622.
It is thereby determined that the models are unable to accurately classify the loan status when the credit score is removed from the data.

5 Discussion

5.1 Key Findings

After conducting our experiments and having viewed the results, we can sum up and comment on our main findings.
As expected, our Random Forest model performs best, both in the default as well as in the hyperparameter-tuned version (when compared correspondingly to the default and tuned SVC model). This highlights the usefulness of applying an ensemble method, such as our Random Forest, to imbalanced datasets. We also note that the tuning process led to only a minuscule improvement in either of our models' performance.
From the exploration of our data and viewing the correlation matrix, we suspected that the credit score variable would be the most important predictive variable. This was indeed confirmed when inspecting the feature importance graph generated by our tuned Random Forest model.
This is why we also performed a "what-if" experimental scenario, removing the credit score variable from our training and testing datasets and applying our best, tuned models. Both models failed to perform well, with the tuned SVC just classifying every data point as "Accepted", underlining the nature of our imbalanced data. Interestingly, our tuned Random Forest performed worse than the tuned SVC when the credit score was removed. Having also those results in mind, we conclude that there were no hidden patterns in the data.

5.2 Further research

Our first recommendation for further research would be to tackle the bias of our dataset. Two possible solutions exist: either oversampling or undersampling. Both approaches present some advantages and disadvantages, which should be investigated by the interested reader. The distribution of the loan status with random undersampling implemented may be seen in Figure 5.1, the random undersampling may be observed to minimize the loan status bias in the data.

If computational resources were abundant, the hyperparameter tuning section of this paper would ideally be performed on a whole range, not on specific randomly selected values. This could potentially lead to finding better, more optimized hyperparameter settings, at the cost of the time associated with computing them.
Another recommendation would be setting up a dimensionality reduction experiment, relevant to our Random Forest model. Using the feature importance graph as our guide, we could discard the least important features (removing the corresponding columns from our dataset) and train the model again on the reduced dataset. It is hypothesized that there would not be any substantial differences in the performance of the Random Forest. As we saw in our what-if scenario, the credit score is the most important variable on which the predictions almost solely depend on.

The next logical step in expanding this study would be acquiring real-life data. In our case, the data is synthetic, which means there are doubts about how realistically our models could be applied (for example, we did not have to deal with NaN values in any dimension). If real-life data is possible to acquire, we also suggest utilizing webscraping for adding additional dimensions to the dataset, possibly leading to finding a new predictor variable.
Finally, as a last suggestion for further research, we would like to recommend trying other machine learning models. It would be interesting to view how a boosted tree-based model performs (for example, XGBoost) or a model with a much more complicated architecture, such as some type of neural network. Then, it should be investigated if the gain offsets the costs associated with computing more complicated models.

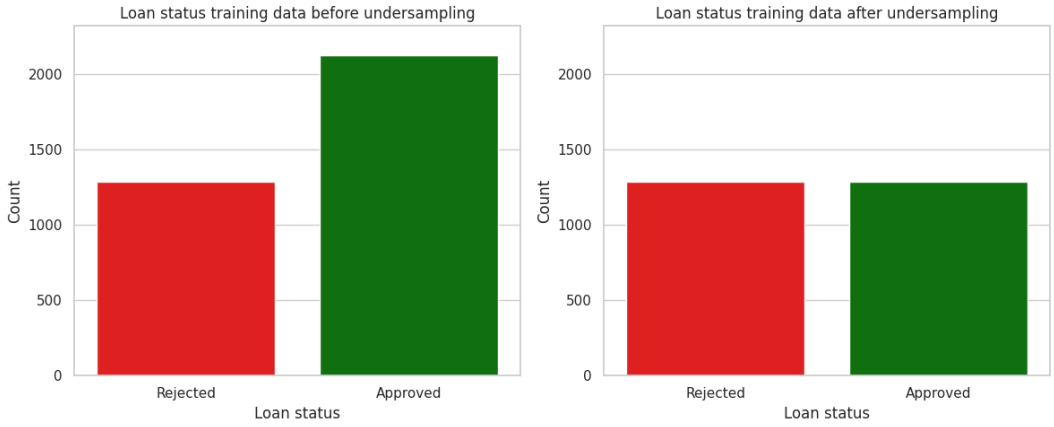


Figure 5.1: Undersampling ylabel distribution

5.3 Limitations

As in most empirical studies, our project possesses some limitations, with the main one being the data itself. Our dataset, accessed from *Kaggle* and adhering to the project's guidelines, is synthetic. This means that our findings may not reflect the real-world cases in the loan approval processes. Furthermore, concerning the data, its description is limited, and the author did not provide many details on the methods with which the dataset was synthesized. The author of the dataset also did not clarify some of the labels of the columns or details on the currency of the loans. This means that understanding the data involved some educated guesses. For example, the column named "no_of_dependents" was interpreted as the number of family members in the household of the person applying for the loan. As for the currency of the monetary values in the dataset, we concluded that it is the Indian Rupee. We based our interpretation, based on the magnitude of the entries and the dataset author's nationality, The synthetic dataset and the lack of clarification present a substantial limitation of this study.

Another limitation comes from the computational constraints that we faced during our hyperparameter tuning processes. Ideally, we would like to tune all of our hyperparameters (and not only, for example, the number of trees and maximum growth) for each model, as well as base our tuning not on particular values but on a whole range of them. This limitation could possibly result in not finding the best hyperparameter values for our models. Quite possibly, better settings could be found, resulting in more accurate and efficient models.

6 Conclusion

In this study, we attempted to predict the binary loan approval status (Accepted or Rejected) of an individual, utilizing a synthetic dataset accessed from *Kaggle*. To achieve this, we employed two machine learning algorithms: a Support Vector Classifier and a Random Forest. It was crucial to recognize this problem as a supervised learning one in the classification category. Firstly, we investigated our data, finding correlations mostly present when looking at the *cibil_score* variable, the credit score, something that we anticipated given the way banks provide loans. We also suspected that the credit score would be the variable utilized mostly by our algorithms.

We decided on an 80% training and 20% testing data split, on which we fitted and then tested our models. Both models were executed in a default state and in a hyperparameter-tuned version. We achieved a minor improvement in the models' predictive ability, with the Random Forest performing better in general.

We also conducted a what-if scenario where we removed our strongest predictor from the dataset, which was, as expected, the credit score. The models failed to perform in a satisfactory manner, highlighting the asymmetric pattern in our dataset.

Finally, we reflected on the results and provided guidance for further research on the matter of loan approval prediction.

Bibliography

[1] J. C. Coffee. “What Went Wrong? An Initial Inquiry Into the Causes of the 2008 Financial Crisis”. *Journal of Corporate Law Studies* 9 (2009), pp. 1–22. DOI: https://doi.org/10.1080/14735970.2009.11421533. URL: https://www.tandfonline.com/doi/abs/10.1080/14735970.2009.11421533.

[2] Yuliya Demyanyk and Otto Van Hemert. “Understanding the Subprime Mortgage Crisis”. *The Review of Financial Studies* 24.6 (2011), pp. 1848–1880. DOI: https://doi.org/10.1093/rfs/hhp033. URL: https://academic.oup.com/rfs/article-abstract/24/6/1848/1583661.

[3] Robert DeYoung, Tara Rice, et al. “How do banks make money? A variety of business strategies”. *Economic Perspectives-Federal Reserve Bank of Chicago* 28.4 (2004), p. 52.

[4] Fan Liang et al. “Constructing a data-driven society: China’s social credit system as a state surveillance infrastructure”. *Policy & Internet* 10.4 (2018), pp. 415–453.

[5] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O’Reilly Media, Inc.", 2022.

[6] Erwin Kreyszig, Herbert Kreyszig, and Edward J. Norminton. *Advanced Engineering Mathematics*. 10th ed. John Wiley Sons, Inc., 2023.

[7] Leo Breiman. “Random forests”. *Machine learning* 45.1 (2001), pp. 5–32.

7 Appendix

A.1

Link to Python notebook:
https://colab.research.google.com/drive/1F-wIN32TY2vONckWUtLye1BDI7GJqbAr?usp=sharing

A.2

Visualization of a single decision tree, example adapted from *An Introduction to Statistical Learning, Second Edition* by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani.

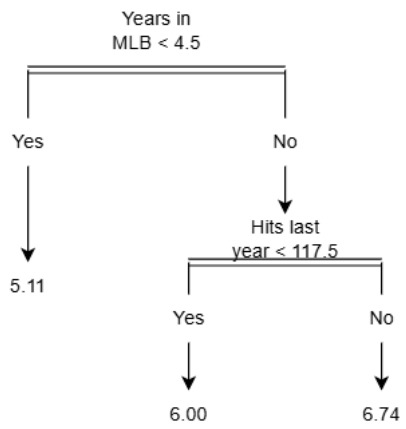


Figure A.1: A simple regression decision tree. Predicting log wage of a baseball player based on years in MLB and the number of hits in the previous year.

A.3

Code snippet of data scaling used for the SVC model. The particular snippet is from the "what if" section of our code. We use the *StandardScaler* from *Sklearn*.

```
[ ] # Drop the cibil_score from the X_train and X_test
    # _dr stands for "dropped"

    X_train_dr = X_train.drop(columns=['cibil_score'])
    X_test_dr  = X_test.drop(columns=['cibil_score'])

[ ] # Scaling for the SVC

    X_train_dr_sc = sc.fit_transform(X_train_dr)
    X_test_dr_sc  = sc.transform(X_test_dr)

[ ] # Fitting the best models without the credit score.

    # SVC tuned at C = 0.1 without cibil
    SVC_no_cibil = SVC(kernel='linear', C = 0.1, random_state = seed)
    SVC_no_cibil.fit(X_train_dr_sc, y_train)

[ ] # Testing (prediction) phase of SVC without cibil
    SVC_no_cibil_prediction = SVC_no_cibil.predict(X_test_dr_sc)

[ ] # Evaluation of the tuned SVC

    # Accuracy score
    print(accuracy_score(SVC_no_cibil_prediction, y_test))
```

Figure A.2: Data Scaling.

A.4

Code snippet of hyperparameter tuning used for the RF model. We use 5 folds to tune *n_estimators* and *max_depth*. We use the *GridSearchCV* from *Sklearn*.

```
rng   = np.random.RandomState(seed)

# Generating values to be tested.
n_estimators_vals = rng.randint(50, 500, 3).tolist()
max_depth_vals    = rng.randint(1,  20, 3).tolist()

# Putting the values in a list.
param_grid = {
    'n_estimators': n_estimators_vals,
    'max_depth':    max_depth_vals
}

# Specifying the 5 folds so they are reproductible.
cv = StratifiedKFold(n_splits = 5,
                     shuffle = True,
                     random_state = seed)

# Specyfying RandomForest
rf = RandomForestClassifier(random_state=seed)

# Our "model"
rf_grid = GridSearchCV(
    estimator   = rf,           # Our base rf.
    param_grid  = param_grid,   # Our values to test.
    cv          = cv,           # Splitting into 5 regions.
    n_jobs      = -1,           # Using all CPU cores. May not work for colab,
    verbose     = 3,            # So we know where we are in the process.
    refit       = True          # Refit so we keep the best one.
)

# Performing the Cross Validation for tuning.
rf_grid.fit(X_train, y_train)
```

Figure A.3: Hyperparameter tuning.

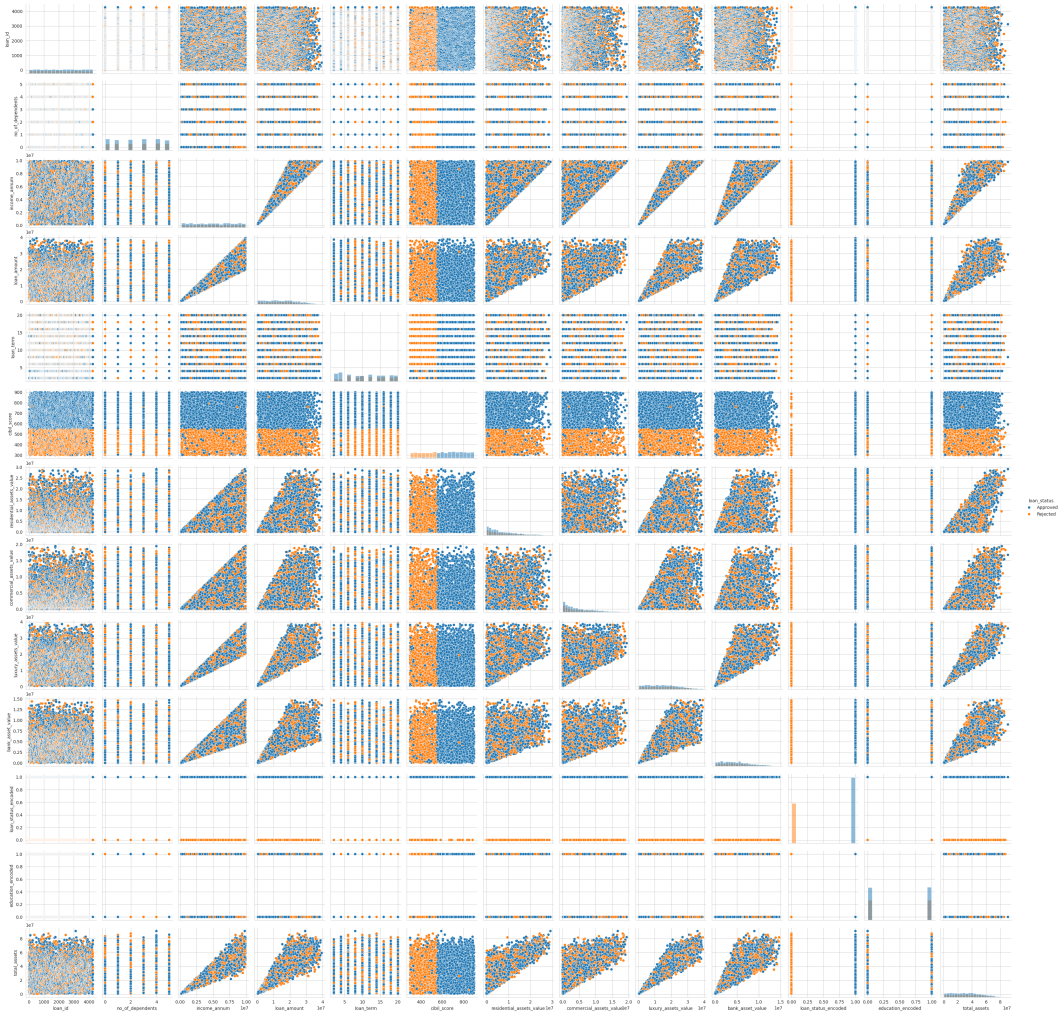


Figure A.4: pairplots