Classification of Bank Account Risk Credit Default Prediction task

Group Name: Morpheus

Supervisor

Dr. Aldo Lipani

Haijin Li
Zonghe Ma
Dodo Wang
Jishuo Zhang
Yiru Xu

January 9th, 2023

Report for CEGE0004:Machine Learning for Data Science

Faculty of Engineering Sciences
University College London

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Credit card default is a significant problem in the financial sector, with severe consequences for both lenders and borrowers. This project aims to investigate the default of credit card clients in Taiwan, China based on the dataset obtained from the UCI Machine Learning Repository(Uci.edu 2016). The main objective of this report is to develop and compare various machine learning methods to predict credit card default, with the aim of improving the accuracy of default prediction and reducing the financial risk for both lenders and borrowers.

The credit card default problem has a substantial impact on the financial sector, including banks, credit card issuers, and borrowers. In the United States, credit card debt reached a record high of 930 billion in 2020, with over 9% of credit card balances becoming delinquent during the pandemic(Newyorkfed.org 2019). Defaulting on a credit card payment can lead to substantial financial losses for lenders, with credit card issuers in the US having to write off over $83 billion in bad debt in 2020 alone (S&P Global Market Intelligence)(Tesher and York n.d.). For borrowers, it can result in a significant reduction in credit scores and creditworthiness. Therefore, there is a need to develop effective default prediction models to minimize financial risk for both lenders and borrowers.

In this project, the primary learning task in this report is binary classification. We aim to solve the classification problem of predicting whether a credit card client would default or not. Specifically, we used 5 kinds of machine learning algorithms to train models to predict the default status of a credit card client based on various attributes, such as their payment history, amount of credit, and demographic information.

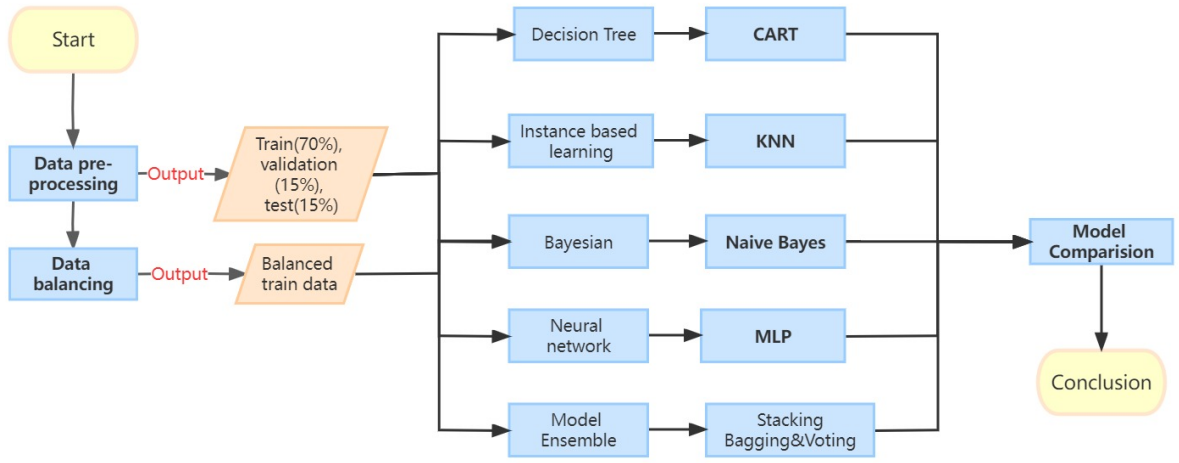


Figure 1.1: Project flowchart.

To measure the performance of the different models, we calculated standard classification

evaluation metrics such as accuracy, precision, recall, F1 score, confusion matrix etc(Berzal and Matín 2002). We also used cross-validation techniques to ensure that the models are robust and generalize well to unseen data. In addition, we aim to find the best combination of hyperparameters for each model to maximize its performance by using GridSearchCV(Nordhausen 2014). The hyperparameters we tuned include parameters such as the learning rate, the number of hidden layers and neurons in the neural network, the maximum depth of the decision tree, and the number of neighbours in the K-nearest neighbours algorithm etc(learn.org n.d.).

If we were to do this task by hand, we would look for patterns and trends in the data that could indicate which clients are likely to default on their credit card payments. We would start by examining the payment history of each client and identify those who have missed payments or paid late in the past. We would also look at the demographic information of the clients, such as their age, income, and education level, as these factors may also influence their creditworthiness. Finally, we would use our intuition and experience to make predictions about which clients are likely to default based on the patterns and trends we have identified in the data. However, given the complexity of the problem and a large amount of data, it is more effective to use machine learning algorithms to develop accurate and scalable models for default prediction.

Our team consisted of 5 members, each of whom contributed significantly to the final completion of the work. The team discussed and selected the research topic proposed by Yiru and are very grateful to Haijin and Zonghe for working together on the pre-processing and enhancement of the data. In order of Jishuo, Yiru, Zonghe, Haijin and Dodo (assisted Haijin), completed the training of the model for the five tasks. Zonghe and Jishuo worked together on the preamble, Haijin worked independently on the data cleaning report, and Yiru and Dodo worked on the final model analysis and conclusion.

Chapter 2

Learning Tasks

2.1 Dataset, Input and Expected output

The input dataset includes the 23 features and target, which contains the amount of credit, gender, education, marital status, age, history of past payments, bill statement (Uci.edu 2016), previous payment amount and the default condition. In detail, there are three types of input features, numerical features (amount of credit, age, history of past payment, bill statement and previous payment amount), categorical features (gender and marital status) and ordinal features (education level). The expected outputs of the machine learning algorithms are the correct class label based on the given input features.

2.2 Data Processing and Split

Then, in the pre-processing part, the raw dataset was split into the train (70%), validation (15%) and test (15%) parts. Then, the balanced train data was also outputted in this paper. The detailed principle and operations are described in the Material part. These three datasets, train, validation and test, need to be split into features and targets, for example:

```
    X_train = training_data.iloc[:, 1:-1]
    y_train = training_data.iloc[:, -1]
```

2.3 Machine Learning Model Training

There are 5 kinds of machine learning algorithms to train models to predict the default status of a credit card client. Decision Trees (Classification And Regression Trees (CART)), Instance-based Learning (KNN), Bayesian (Naive Bayes), Neural Network (MLP) and Model Ensemble (Stackink Bagging&Voting) (Figure 1.1). The training dataset split in Section 2.2 would be used to train the default machine learning models. Then, the hyperparameter tuning part is necessary, and the enhanced dataset was also used for the optimal model training.

2.4 Model Evaluation Methodology

Using the x_test dataset to test the optimal model and then storing the predicted target variables as y_pred. Evaluating the performance of the model by comparing the predicted target variables y_pred and true target variables y_test. Firstly, we need to calculate the confusion matrix, which is a table that summarizes the number of true positives, true negatives, false positives and false negatives for the predictions (Table 2.1). Then we can compute various metrics such as accuracy (ACC), precision, recall, F1-score and Matthews Correlation Coefficient (MCC). Conception and calculation details are shown in Table 2.2 (Thach, Rojanavasu, and Pinngern 2008).

Table 2.1: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Accuracy Accuracy is the proportion of correct predictions (true positives and negatives) out of the total predictions made. It is an intuitive metric to evaluate the overall performance of the model. it may not be suitable for imbalanced datasets where the data is imbalanced.

Precision-score Precision is the proportion of true positive predictions out of all the positive predictions made by the model. It measures how many of the instances predicted as positive are actually positive. Precision is important when you want to minimize false positives (e.g., in spam detection, where you don't want non-spam emails to be classified as spam). High precision indicates that the model is good at identifying defaulters without misclassifying too many non-defaulters as defaulters.

Recall-score Recall, also known as sensitivity or true positive rate, is the proportion of true positive predictions (actual defaulters) out of all the actual defaulters in the dataset. It measures how many of the positive instances are captured by the positive predictions. A high recall means that the model is good at identifying defaulters, even if it misclassifies some non-defaulters as defaulters. Recall is important when you want to minimize false negatives (e.g., in some cases where researchers don't want to miss any positive cases).

F1-Score The F1-score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall and provides a single metric that considers both false positives and false negatives. The F1 score is particularly useful when dealing with imbalanced datasets, where one class is significantly more prevalent than the other.

Matthews Correlation Coefficient MCC is essentially the correlation coefficient between the observed and predicted binary categories; it returns a value between -1 and +1. For the default credit card problem, a high MCC value (close to +1) indicates good classification performance, while a low value (close to -1) indicates poor performance. an MCC of 0 means that the classifier does not perform better than random chance.

Table 2.2: List of metrics

Metric	**Conception**	**Formula**
ACC	the proportion of correct predictions over the total number of predictions	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$
precision	the proportion of true positives over the total number of positive predictions	$precision = \frac{TP}{TP+FP}$
recall	the proportion of true positives over the total number of actual positive samples	$recall = \frac{TP}{TP+FN}$
F1-score	the harmonic mean of precision and recall	$\mathbb{F}_1 = 2 * \frac{precesion*recall}{precision+recall}$
MCC	the correlation between the predicted and true binary classifications, taking into account true and false positives and negatives	$\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Chapter 3

Material

3.1 Data Descriptions

The dataset consists of $30,000$ credit card clients in Taiwan, and it includes demographic information, credit history, and payment details from April 2005 to September 2005 (Uci.edu 2016). Each client has 23 attributes, including age, gender, education level, marital status, credit limit, payment history, and the amount of the billing statement. The dataset has a binary classification label indicating whether a client defaulted on their credit card payment in the next month.

3.2 Data characteristics

Continuous features description (14 variables with float datatype):

- $Limit\_BAL$: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- Age: the actual age in years.
- PAY_AMT1 to PAY_AMT6: Amount of previous payment (NT dollar).
  - PAY_AMT1 = amount paid in September, 2005;
  - PAY_AMT2 = amount paid in August, 2005;
  - PAY_AMT3 = amount paid in July, 2005;
  - PAY_AMT4 = amount paid in June, 2005;
  - PAY_AMT5 = amount paid in May, 2005;
  - PAY_AMT6 = amount paid in April, 2005;
- $BILL\_AMT1$ to $BILL\_AMT6$: Amount of bill statement (NT dollar);
  - $BILL\_AMT1$ = amount of bill statement in September, 2005;
  - $BILL\_AMT2$ = amount of bill statement in August, 2005;
  - $BILL\_AMT3$ = amount of bill statement in July, 2005;
  - $BILL\_AMT4$ = amount of bill statement in June, 2005;
  - $BILL\_AMT5$ = amount of bill statement in May, 2005;
  - $BILL\_AMT6$ = amount of bill statement in April, 2005.

Categorical features description (9 variables with integer datatype):

- Sex: two classes, with values (1 = male; 2 = female).

- *Education*: four classes, with values (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- *Marriage*: the customers marital status (1 = married; 2 = single; 3 = others).
- PAY_0 to PAY_6 with out PAY_1: History of past payment tracked the past monthly payment records (from April to September, 2005) as follows:
  - PAY_0 = the repayment status in September, 2005;
  - PAY_2 = the repayment status in August, 2005;
  - PAY_3 = the repayment status in July, 2005;
  - PAY_4 = the repayment status in June, 2005;
  - PAY_5 = the repayment status in May, 2005;
  - PAY_6 = the repayment status in April, 2005;

  The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.

Target variable description: default payment ($Yes = 1$, $No = 0$) with integer datatype.

3.3 Data statistical and graphical descriptions

The statistical description of the continuous variables can be seen in Table 3.1, where mean, standard deviation, maximum, minimum and 25%, 50%, and 75% quantiles. Referring to the STD columns, the data feature has exceptionally high variance, which can make our learning model unstable. From the maximum and minimum measurements, we see that some data columns exhibit a significant spread, which may cause the gradients of the model's parameters to be too large or too small, which can result in slow or unstable learning.

Table 3.1: Continuous feature descriptive statistics

	Mean	STD	MIN	25%	50%	75%	MAX
LIMIT_BAL	167489.24	129747.03	10000	50000	140000	240000	1000000
AGE	35.49	9.22	21	28	34	41	79
PAY_AMT1	5663.77	16563.52	0	1000	2100	5006	873552
PAY_AMT2	5921.34	23041.23	0	833	2009	5000	1684259
PAY_AMT3	5225.86	17607.23	0	390	1800	4505	896040
PAY_AMT4	4826.24	15666.40	0	296	1500	4013.5	621000
PAY_AMT5	4799.55	15278.54	0	253.5	1500	4032	426529
PAY_AMT6	5215.68	17777.74	0	118	1500	4000	528666
BILL_AMT1	51224.91	73636.58	-165580	3558.5	22382	67092	964511
BILL_AMT2	49180.61	71174.46	-69777	2984.5	21203	64008.5	983931
BILL_AMT3	47014.70	69350.03	-157264	2667.5	20089	60165.5	1664089
BILL_AMT4	43264.39	64333.44	-170000	2328	19052	54509	891586
BILL_AMT5	40312.74	60797.72	-81334	1763.5	18105	50196	927171
BILL_AMT6	38873.06	59554.68	-339603	1256	17074	49200.5	961664

PCA data Visualization and Principal

Moreover, Figure 3.1 shows the results plot by transferring the 23 predict variables to 3 principle components by calculating the eigenvalues and eigenvectors to see the theory behind this; let's suppose our variable predictor matrix is $X \sim (m, \Sigma)$ and the space we wish to map to hyperspace

with a new predictor matrix $X' \sim (m', \Sigma')$, the goal is to find a minimum linear transformation vector than can transform X to X'. The shortest distance can be considered to minimise the expected sum of squared perpendicular distance. By Pythagoras' theory we have:

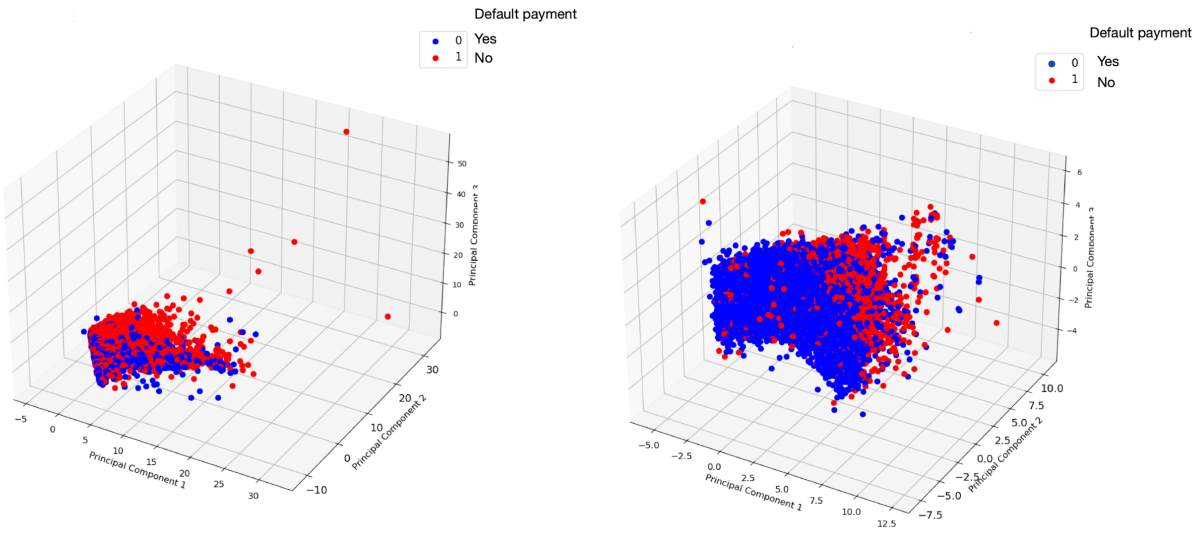


Figure 3.1: Principal Component Analysis

$$\mathbb{E}[d(X, X')^2] = \mathbb{E}[d(m, X)^2] - \mathbb{E}[d(m, X')^2], \tag{3.1}$$

where $\mathbb{E}[d(m, X)^2]$ is the distance between the mean and the original point and $\mathbb{E}[d(m, X')^2]$ is the distance between the mean and the projection point. Let assume γ be a unit vector pointing in the direction of the vector $X' - m$, and let $t \in \mathbb{R}$ such that the projected point can be calculated by:

$$X' = m + t\gamma \in \mathbb{R}^q, \tag{3.2}$$

where q is equal to 3 in our case. The euclidean distance between mean m and X' equal $|t|$.

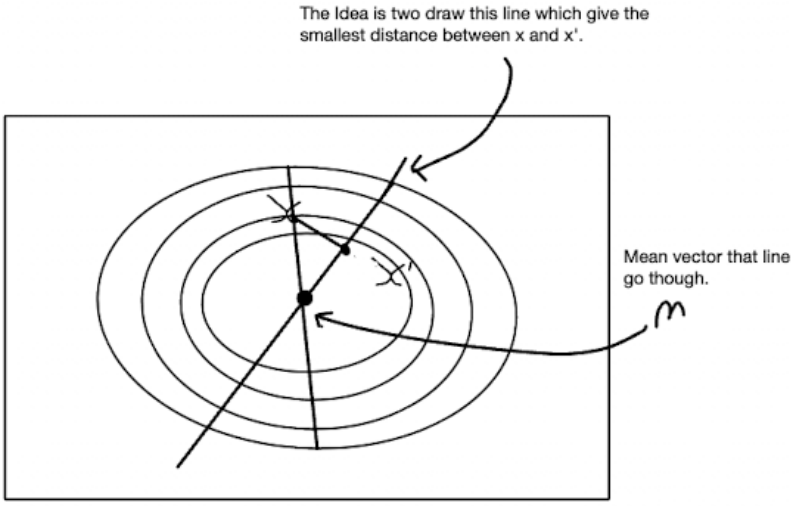


Figure 3.2: Principal Component Analysis theory visualization

By considering trigonometry theory and dot product t can be calculated as $\gamma^T(X - m) = t$. Effectively, we are trying to find vector γ that maximises $\mathbb{E}[d(m, X')^2]$ the squared root distance

between the main point and the projection (see Equation 3.1) :

$$\begin{aligned}
\mathbb{E}[d(m, X')^2]1 &= \mathbb{E}[d(m, X')d(m, X')], \\
&= \mathbb{E}[\gamma^T(X-m)(X-m)^t\gamma], \\
&= \mathbb{V}ar[\gamma^T(X-m)] + \mathbb{E}[\gamma^T(X-m))], \\
&= \mathbb{V}ar[\gamma^T X] \quad by \quad (\mathbb{E}[X-m]=0), \\
&= \gamma^T \mathbb{V}ar(X)\gamma \quad = \quad \gamma^T\Sigma\gamma. \qquad (3.3)
\end{aligned}$$

Hence we need to maximise $\gamma^T\Sigma\gamma$ under the constrain $\gamma^T\gamma = 1$, to solve this problem we can use the Lagrange multiplier method, let's define:

$$P(\gamma) = \gamma^T\Sigma\gamma - \lambda(\gamma^T\gamma - 1), \qquad (3.4)$$

where $(\gamma^T\gamma - 1) = 0$, by taking the derivative with respect to γ we get:

$$\begin{aligned}
\frac{\partial P}{\partial \gamma} &= 2\Sigma\gamma - 2\lambda\gamma, \\
\Sigma\lambda = \lambda\gamma \quad or \quad \gamma^T\Sigma\gamma &= \lambda. \qquad (3.5)
\end{aligned}$$

Therefore, to transform $\mathbb{R}^n$ to a lower space $\mathbb{R}^p$ the PCA algorithm finds the top p eigenvalue of the variance matrix Σ and the eigenvector is the γ that is needed for transformation in Equation 3.2. Note that Equation 3.5 can also be rewritten as $\Sigma = \zeta\kappa\zeta^T$ where ζ is a vector consist a list of eigenvalues and κ is an eigenvector matrix, this is also known as Eigen-decomposition.

Back to Figure 3.1, the left side shows the data reduction before the outlier detection, and the right side shows the PCA visualisation after the outlier is removed(Isolation Forests). The blue point represents the target variable with a default payment equal to Yes, and the red is No. The plot shows that there are some red points in the original dataset, which are far from the majority of data, suggesting that outlier detection and removal are required.

Data correlation Heat-Map

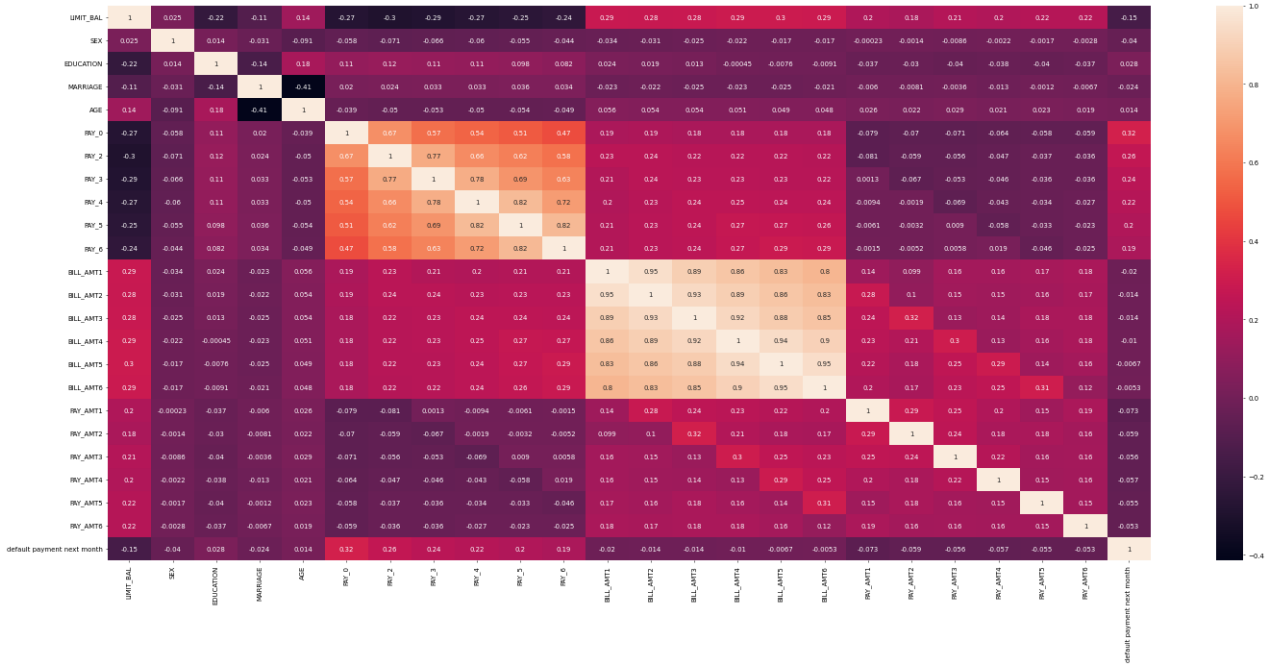


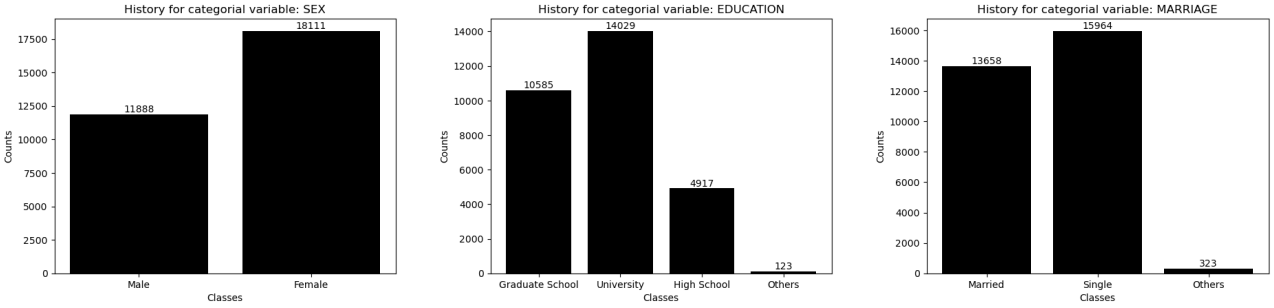
Figure 3.3: Data correlation matrix heat-map visualization

Figure 3.3 shows the correlation between data variables; there are incredibly high correlations between variables PAY_0 to PAY_6 and $BILL\_AMT1$ to $BILL\_AMT6$ this makes sense as those variables are collected through a time series. This can cause a problem as highly correlated variables can complicate interpreting the model's coefficients. They may be inflated or deflated due to multicollinearity and are more likely to observe model overfitting. One way to deal with this is regularization techniques like Lasso or Ridge regression. These methods can help reduce the impact of highly correlated variables by adding a penalty term to the algorithm's objective function.

Moreover, there is interesting that the time series variable PAY_AMT1 to PAY_AMT6 shows no autocorrelations, suggesting no correlation between the history of past payments tracked.

Categorical data Histogram plot for predictor variables

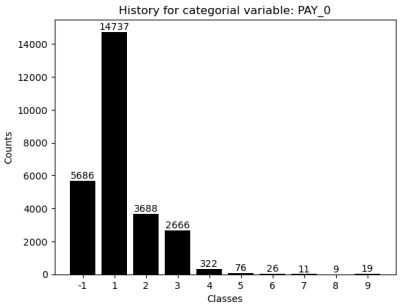
From Figure 3.4, we see that the data consists of more female clients rather than male, more university education history than others, and more single class than married. For time series variables PAY_0 to PAY_6, most clients are likely to repay the purchase that delays in one month.



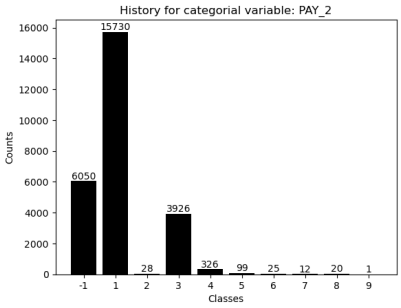
(a) Histogram Sex

(b) Histogram $Education$

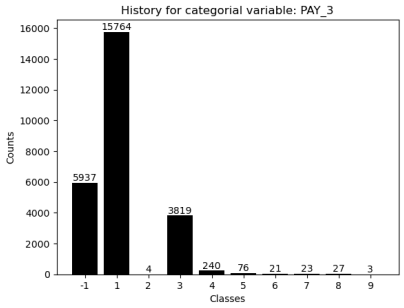
(c) Histogram $Marriage$



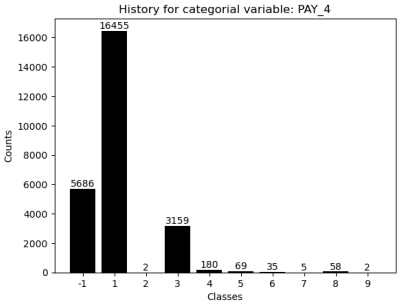
(a) Histogram PAY_0



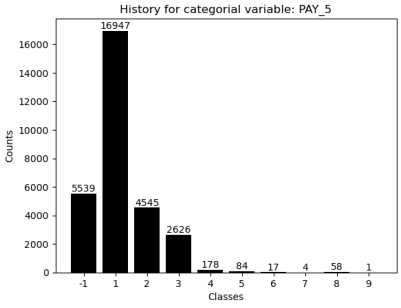
(b) Histogram PAY_2



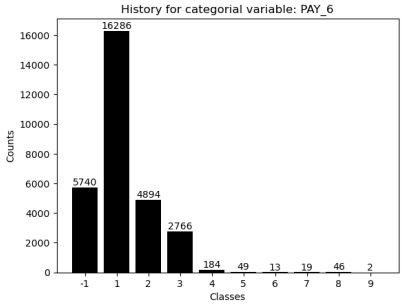
(c) Histogram PAY_3



(a) Histogram PAY_4



(b) Histogram PAY_5

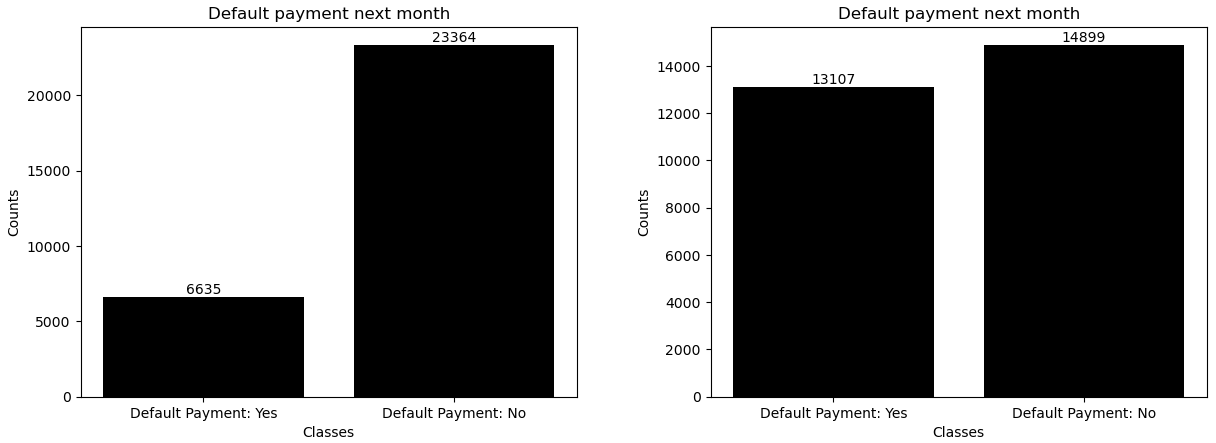


(c) Histogram PAY_6

Figure 3.4: The figure shows the categorical histogram plot for predictor variables.

Categorical data Histogram plot for target variable

Figure 3.5 shows the histogram plot for the target variable, where Figure 3.5(a) shows the histogram plot for the original dataset; the plot clearly shows the problem of target variable data imbalance, where the default payment of Yes is 3 times less than the default payment of No so the model may tend to be biased towards the majority class, which can lead to poor performance on the minority class. Figure 3.5(b) shows the target data distribution after data enhancement by running SMOTE algorithm, more on Section 3.5.



(a) Original target variable hist

(b) Target variable hist after data enhancement

Figure 3.5: The figure shows the histogram plot for the target variable.

3.4 Exploratory Data Analysis (EDA)

Section 3.3 outlines the limitations of our dataset. In summary, we have identified two main issues: the presence of outliers in our predictor variables, as evidenced by the high variance values in Table 3.1, and an imbalanced data distribution in our target variable. In this section, we will discuss the approaches we took to address these problems.

Firstly, the Isolation forest (Liu, Ting, and Zhou 2008) algorithm is used to deal with outlier detection and removal. The algorithm is based on the idea that anomalies are easier to isolate and remove from the rest of the dataset. Isolation Forest (Liu, Ting, and Zhou 2008) randomly selects a feature and a split point between the maximum and minimum values of that feature. This process is repeated recursively until all data points are isolated, or a predetermined number of trees is reached. Outliers are identified as the data points that require fewer splits to be isolated.

The method is called in **sklearn.ensemble** package, where the model parameters are set to $n\_estimators = 80$, $contamination = auto$; the algorithm will build 80 decision trees during the training process and automatically determine the value of contamination based on the size of the dataset. The algorithm takes the input of all continuous variables and has successfully removed 2473 outliers which is around 8% data removed from original datasets, and the results plot can be seen in Figure 3.6. Table 3.2 shows the continuous predictor variable descriptive statistics after removing the outlier from the Isolation Forest algorithm. This can be compared with the statistics of the original dataset 3.1 by comparing the results. We can see that the standard deviation has significantly reduced, and the data is more centralized (refer to PCA plot Figure 3.1).

Finally, the SMOTE (Fernández et al. 2018) algorithm is used to solve unbalanced data problems; SMOTE works by creating synthetic data points in the minority class by interpolating between existing minority class data points.

The algorithm works as follows:

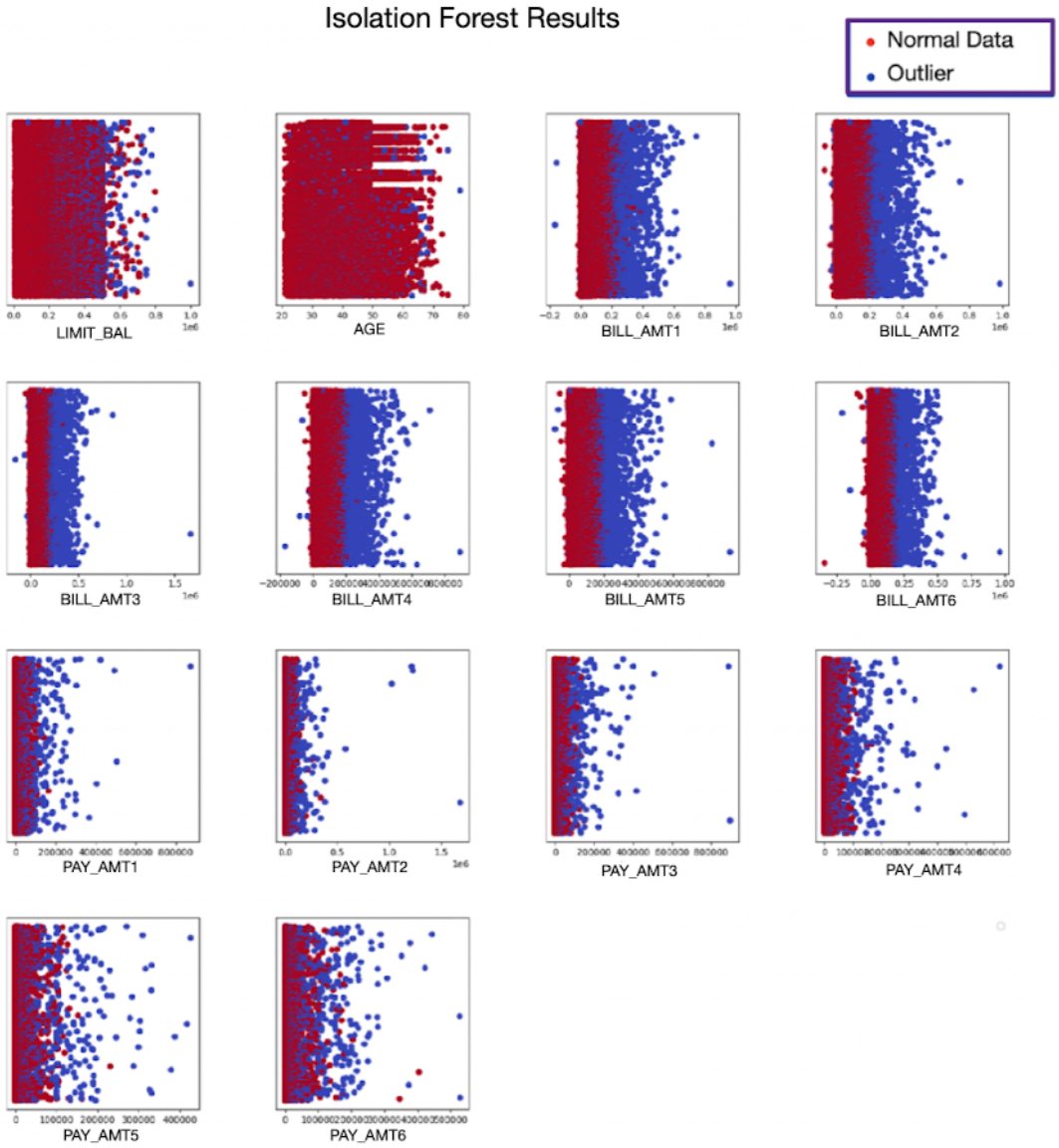


Figure 3.6: Isolation forest results for remove outliers.

Table 3.2: Continue predictor features descriptive statistics after outlier remove

	Mean	STD	MIN	25%	50%	75%	MAX
LIMIT_BAL	152457.30	119425.99	10000	50000	120000	210000	800000
AGE	35.26	9.20	21	28	34	41	75
PAY_AMT1	3900.90	7448.89	0	758	2000	4345.5	164820
PAY_AMT2	4004.56	9062.00	0	632	2000	4074	344467
PAY_AMT3	3466.00	7343.70	0	326	1579	3600	153400
PAY_AMT4	3203.72	7330.59	0	163.5	1249	3200	168459
PAY_AMT5	3233.42	7671.13	0	112	1300	3240.5	231133
PAY_AMT6	3372.37	9327.61	0	0	1200	3190.5	403500
BILL_AMT1	38669.45	47831.43	-15308	2946	19328	54917	457627
BILL_AMT2	36587.10	45198.53	-69777	2475	18720	51400	353481
BILL_AMT3	34566.69	42999.61	-46127	2154	18203	49710	323836
BILL_AMT4	31515.39	39861.52	-50616	1770	17091	46042	268084
BILL_AMT5	29286.08	38004.45	-53007	1320	15582	41524	268084
BILL_AMT6	28247.99	37611.77	-339603	937.5	14106	39997.5	271388

- For each minority class data point, find its k (used 5) nearest neighbours in the feature space.
- Randomly select one of the k nearest neighbours and create a new data point by interpolating between the selected neighbour and the original data point.
- Repeat steps 1 and 2 until the desired number of synthetic data points have been created.

After outlier detection by random forest algorithm, we have generated 6472 default payment: Yes class data; the histogram comparison can be seen in Figure 3.5. Although the data looks more balanced in the histogram, there are some drawbacks to using SMOTE. SMOTE can sometimes create synthetic data points that are too similar to the original data, resulting in overfitting and poor model generalisation performance. Moreover, the SMOTE algorithm uses the KNN model as a basis and another learning algorithm to generate new data; the data might be inaccurate. In general, it does increase the model concentration for minority class data.

3.5 Summarize EDA process and data training split

The EDA process and training, validation and test data split for model training are done as follows:

- Read the original data in python, which is named default_of_credit_card_clients.csv;
- Use the Isolation Forest algorithm to detect and remove data outliers and generate a new file named df_IFoutlier_c.csv;
- Use **sklearn.model_selection.train_test_split** to split the df_IFoutlier_c.csv data into train, validation and test, with 70%, 15% and 15%, saved as three separated .csv files name train.csv, validation.csv, test.csv.
- Use SMOTE algorithm to generate new 6472 datasets with the target variable Default Payment: Yes on the train.csv dataset (not on validation or test), and it is saved as balanced_train_data.csv.

Those generated datasets will be used for our five algorithm families; the validation.csv dataset is used to compare the model inside each family and for hyper-parameter tuning. The best model in each algorithm family is selected, and after the hyper-parameter tun, we compare them with training on the test.csv dataset.

Chapter 4

Technology

This section lists the hardware and software system we used to present this coursework and any open-source platform used.

Decision Tree

- OS: Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz DELL G15, Intel(R) UHD Graphics, NVIDIA GeForce RTX 3060 Laptop
- Environments: Python 3.8, numpy 1.23.5, pandas 1.5.2, scikit-learn 1.2.1, seaborn: 0.12.2, matplotlib: 3.6.2

Instance-Based Learning

- OS: Windows 11 with 16GB RAM
- Environments: Python 3.8
- Packages: scikit-learn 1.2.2, pandas 1.5.3, numpy 1.24.2, scikit-learn 1.2.2, matplotlib 3.6.3

Bayesian Learning

- OS: MacOS with Apple M1 Pro
- Environments: Python 3.8, scikit-learn 1.2.2, pandas 1.5.3

Nueral Network

- OS: Intel(R) Core(TM) i5-8259U@230 GHz Macbook pro 2020
- Platform: Python 3.10.8, pytorch 1.13.1, scikit-learn 1.2.1

Model Ensemble

- OS: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz 2.80 GHz
- Platform: Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
- GPU: NVIDIA GeForce GTX 1060 388.73

Collaboration platforms

- GitHub for project management.
- Jupyter Notebook was used to implement data analysis, program description and run results presentation
- Overleaf for report collaboration.

AI Tools

ChatGPT (OpenAl 2023) was used to better understand the topic requirements of the report and to embellish parts of the writing.

Chapter 5

Learning Algorithms

5.1 Decision Trees

The whole dataset was split into three parts in the pre-processed data: training, validation and testing. These three .csv documents were saved in the data file. Then, several tasks are done in this algorithm:

1. Loading the training, validation and test data in the data file and splitting these datasets into features and target parts. These three .csv documents have already been split in the pre-processing part;
2. Building a default decision tree model with the training data and printing the accuracy of the default decision tree model;
3. Using GridSearchCV for the hyper-parameter tuning part;
4. Training a new decision tree model with the best parameters from the earlier part;
5. Evaluating the new decision tree model via the accuracy of the validation and test data;
6. Plotting the decision tree via matplotlib package;
7. Using accuracy, precision score, recall-score, F1-score and confusion matrix of the model.

This decision tree model is a non-parametric supervised learning method used for classification. In this project, there are 23 features in the default credit card clients dataset, which can be operated in this decision tree model for the classification. These features include the amount of the credit, gender, education, marital status, age, history of past payments, bill statement and previous payment amount. In detail, there are three types of input features, numerical features (credit, age, history of past payment, bill statement and previous payment amount) and categorical features (gender, education and marital status).

The expected output of the decision tree classification model is a class label. In this project, the expected output is to classify whether customers will default payments in Taiwan (Yes= 1, No = 0).

Table 5.1: The accuracy of the default decision tree model and the decision tree model with best parameters.

	Default Decision Tree Model	**Best Parameters Decision Tree Model**	
		Validation	Test
Accuracy(%)	72.2209	81.3756	82.3002

The pre-processing step was described in the material part, and the raw data was split into three .csv documents (training, validation and test). This algorithm first loads the dataset (training, validation and test) and splits the three datasets into features and targets. Secondly, the decision tree model was built by training_data, and the accuracy of this decision tree model is 72.2209% with validation_data (Table.5.1). In addition, the hyperparameters tuning part, including "criterion", "max_depth", "min_samples_split", and "min_samples_leaf", and the $GridSearchCV()$ is used for this part. Then, the new decision tree model was built with the suitable parameters, and the validation accuracy was 81.3756%, and the test accuracy was 82.3002% (Table.5.1). Then, the $plot\_tree()$ function is used to plot the decision tree model after hyperparameters tuning (Fig.5.1) and the figure plot with the result (default and no default). In this project, the improvement in the hyperparameters tuning part was mentioned, and the accuracy can be improved significantly from 72.2209% to 81.3756%. As for the hyperparameters tuning part, two options for "criterion", ten options for "max_depth", three options for "min_samples_split", and three options for "min_samples_leaf". So, the total iterations in this part are 180, and the accuracy is used for choosing the suitable parameters.

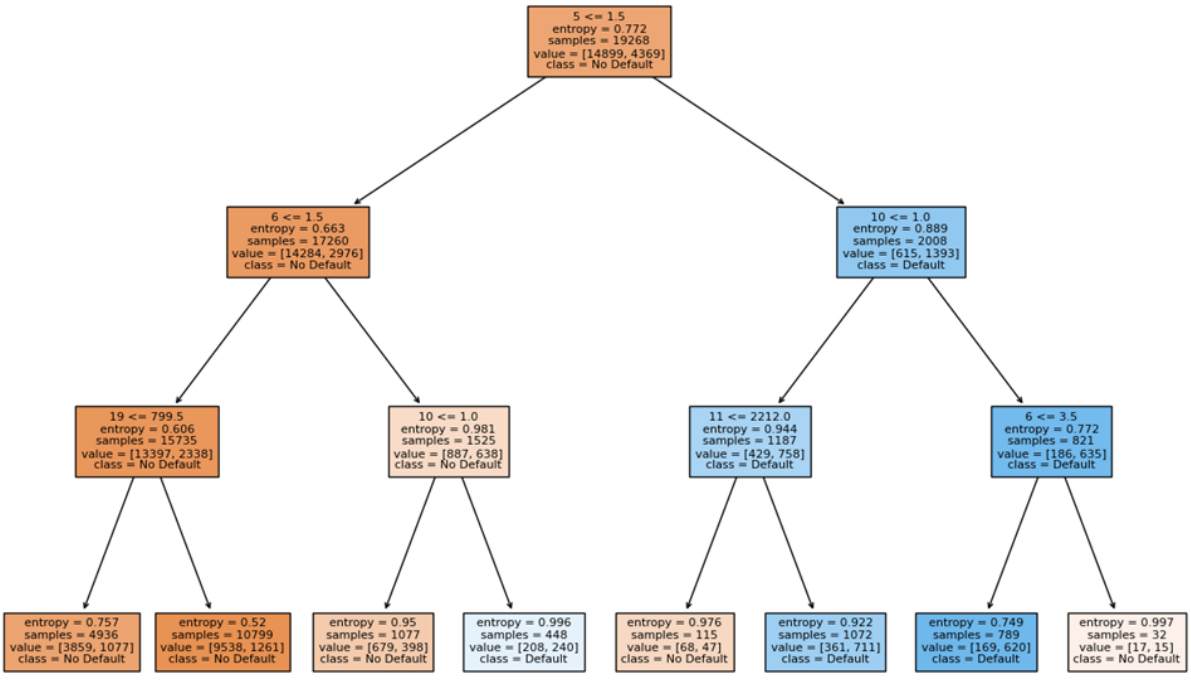


Figure 5.1: The decision tree model after hyperparameters tuning part.

There are some advantages of using the decision tree model to classify the default of credit card clients. Firstly, the decision tree model is easy to understand and visualise. Secondly, this model can deal with the numerical and categorical features and output the class label. Thirdly, this decision tree model can be validated and accounted for by statistical tests (Pedregosa et al. 2011). However, there are also some disadvantages, like the overfitting problem, and the decision tree model grows until it fits the data perfectly. The other problem is that the imbalanced data cannot be operated perfectly in the decision tree model, and this project tries to solve this problem in the pre-processing part (Pedregosa et al. 2011).

5.2 Instance-Based Learning

5.2.1 Introduce KNN

In contrast to general machine learning algorithms, which have an explicit description of the target function when training examples are provided, instance-based learning algorithms develop the target function only when a new instance must be classified. Each time a new query instance is encountered, its relationship to the previously stored examples is examined to assign a target function value for the new instance.

The advantages of instance-based learning algorithms are:

1. Training is very fast.
2. Learn complex target functions.
3. Don't lose information.

Shortcomings of instance-based learning algorithms are:

1. Slow at query time, and the computing cost can be expensive.
2. Many algorithms typically consider all attributes of instances, which require lots of storage.
3. Easily fooled by irrelevant attributes.

Instance-based learning algorithms include K-nearest neighbour (KNN), locally weighted regression (LWL), case-based reasoning (CBR), etc. As the most basic instance-based algorithm and simple to implement, KNN was chosen as the method to solve the tasks, which will be described in the next section.

KNN assumes that all instances correspond to points in the n-dimensional space. The nearest K neighbours of an instance are defined in terms of distance (Euclidean, Manhattan, Cosine), and thus the instance can be classified (discrete-valued target) or estimated (continuous-valued target) by the most common class among the K nearest neighbours. The major advantage of KNN is that it is not required to establish a predictive model before classification or regression. However, it may be biased when the training examples are imbalanced, with many outliers and different data scales. If any of the circumstances occurs, corresponding preprocessing steps such as balancing the data, removing outliers, and standardisation should be taken.

The distanced-weighted KNN algorithm is the refinement to the KNN algorithm as it weights the contribution of each k neighbour according to their distance to the query instance, i.e., giving greater weight to closer neighbours. Distanced-weighted KNN is especially useful when the data distribution is not uniform. Whereas, it also has some drawbacks, such as being sensitive to outliers and having a higher computing cost than the standard KNN algorithm.

5.2.2 Tasks

From the material section, it can be seen that the original dataset is imbalanced and has different data scales, which may influence the accuracy of classification. Therefore, it is necessary to first standardise and balance the data in the preprocessing steps. Both balanced and imbalanced (original) data would be put in the algorithm, then the performances of the two models can be compared with indicators such as accuracy, precision, and recall etc. so that we can choose the optimal model for KNN.

5.2.3 Preprocessing

Step 1: Separate the target and reshape the dataset. Step 2: Standardise the dataset.

5.2.4 Algorithm Implementation

5.2.4.1 Tuning KNN's hyperparameters and validation

In general, hyperparameters in KNN include K-value, weights, and distance. To pick out the proper K-value, we first plot a graph with the x-axis being the K-value and the y-axis being the corresponding error rate to give a rough estimation of the potential K-value. Then we create a hyperparameter search grid that includes weights ("uniform": standard KNN, "distance": distance-weighted KNN), the scale of K-value ("n_neighbors"), and distance ("euclidean", "manhattan", "cosine"). Considering the size of the hyperparameter search gird is pretty large, we used randomized search instead of grid search to pick out the optimal hyperparameters through 10-fold cross-validation. Then we put these hyperparameters in the algorithm and compared the performances of the model that used different datasets. Confusion matrix and classification report were used to evaluate the model performance.

5.2.4.2 KNN - original dataset

The error rate can be calculated using the training and testing datasets in a For loop, with the K-value being a large scale of 1-30.

```
# choose K-value with the lowest error rate.
error_rate = []
for i in range(1, 30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_x, train_y)
    pred_i = knn.predict(test_x)
    error_rate.append(np.mean(pred_i != test_y))
```

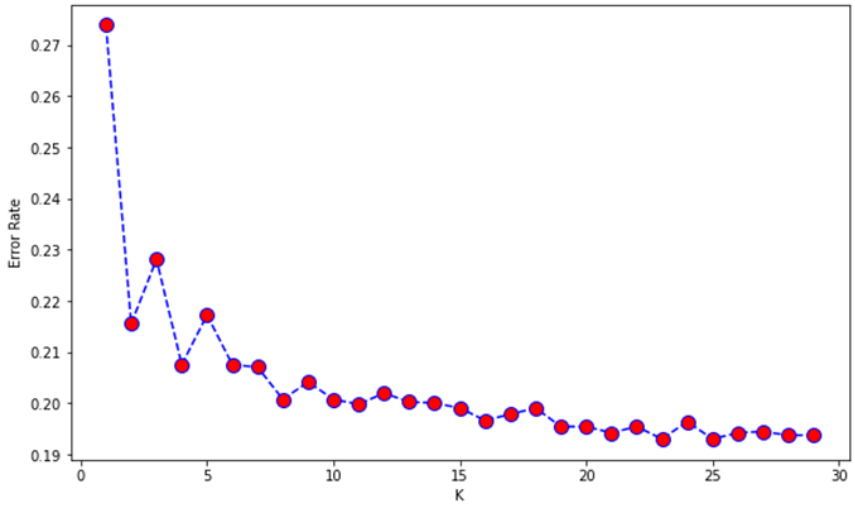


Figure 5.2: The error rate and K Value for original dataset

As is shown in Fig.5.2, the error rate fluctuates steadily after the K-value of 15. randomized search would be used next to choose the actual proper K-value and other hyperparameters.

```
# randomized search
param_grid = [{
    'weights': ["uniform", "distance"],
    'n_neighbors': range(1, 20), # according to the plot the scale can be narrowed down  to 1
                                 -20
    'metric':['euclidean', 'manhattan', 'cosine']}]
knn_clf = KNeighborsClassifier()
randomized_search = RandomizedSearchCV(knn_clf, param_grid, cv=10, verbose=2)
randomized_search.fit(train_x, train_y)
```

The search gave the best-estimated parameters of weights being uniform, K-values being 15, and distance being euclidean. Then hyperparameters were put into the algorithm and evaluation indicators could be generated.

```
Train accuracy of kNN 0.8183516711646253
Test accuracy of kNN_0.8009203196899976
```

```
Confusion Matrix:
 [[3023  191]
 [ 631  284]]
Classificaiton Report:

              precision    recall  f1-score   support
         0.0       0.83      0.94      0.88      3214
         1.0       0.60      0.31      0.41       915

    accuracy                           0.80      4129
   macro avg       0.71      0.63      0.64      4129
weighted avg       0.78      0.80      0.78      4129
```

The result shows that using the original data, KNN predicts class 0 more accurately than class 1. However, considering the imbalance of the dataset, it is possible that the high accuracy of 80% is misled by the dominant class 0. Therefore, balanced data should be used in the algorithm for comparison and drawing a proper conclusion.

5.2.4.3 KNN – balanced dataset

Using the balanced dataset mentioned in the material section and following the same procedures above, the graph of error rate vs. K value is generated as below. As is shown in Fig.5.3, the

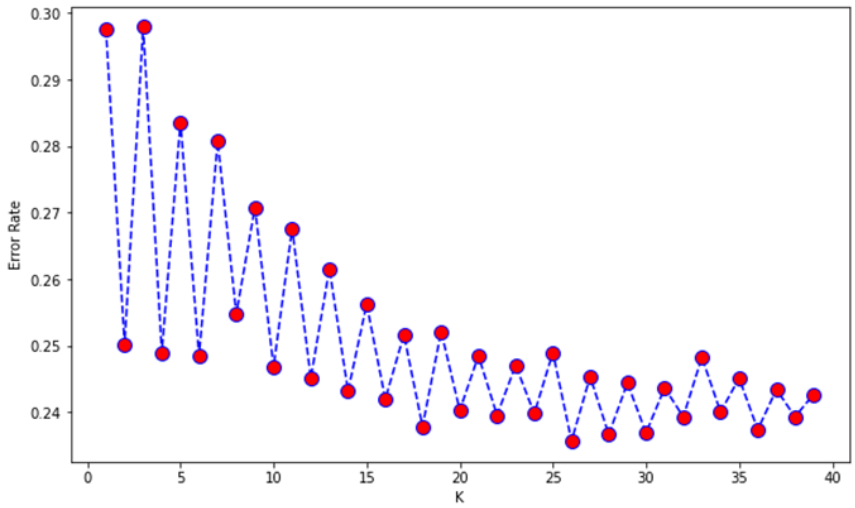


Figure 5.3: The error rate and K Value for balanced dataset

error rate also fluctuates steadily after the K-value of 15. Randomized search was used next to choose the actual proper K-value and other hyperparameters. The search gave the best-estimated parameters of weights being distance, K-values being 14, and distance being manhattan. Then hyperparameters were put into the algorithm and performance indicators could be generated.

```
Train accuracy of kNN 0.9996072270227808
Test accuracy of kNN 0.7519980624848631
```

```
Confusion Matrix:
 [[2630  584]
 [ 440  475]]
Classificaiton Report:

              precision    recall  f1-score   support
         0.0       0.86      0.82      0.84      3214
         1.0       0.45      0.52      0.48       915

    accuracy                           0.75      4129
   macro avg       0.65      0.67      0.66      4129
weighted avg       0.77      0.75      0.76      4129
```

The result shows that using the balanced data, KNN also predicts class 0 more accurately than class 1, which is similar to the result of the original data. According to the confusion matrix, true positives decrease by 13% while true negatives increase by 40%. In the meantime, the precision values show a 0.06% increase in class 0 and a 15% decrease in class 1. This suggests that using the balanced data, the algorithm is becoming more aggressive in classifying class 1. The decrease in precision means that the algorithm is now more likely to make false positive predictions for class 1, while the increase in recall means that the algorithm is now identifying more actual instances of class 1. In terms of accuracy, the algorithm using balanced data has lower accuracy than the previous one, indicating that the algorithm is making more incorrect predictions overall.

5.2.5 Section Sum Up

According to the comparison, it can be seen that after balancing the data, KNN can see more instances of the minority class(class 1) and learn to classify them more accurately, which greatly decreases the risk of default. However, this is achieved at the cost of more false positives, as the algorithm includes more instances in class 1 which may not actually belong to this class.

In general, considering the context of operations and benefits, though banks expect fewer default risks, they couldn't afford the cost of losing more potential customers, damaging clients' trust, and ultimately, cutting off the margin. Therefore, it is more reasonable to choose the version that has higher accuracy and precision for both of the classes as the optimal model for the KNN algorithm, which is the one using the original dataset. Additionally, further work such as changing the algorithm to balance the data, and comparing it with other algorithms of instance-based learning can still be done to improve the model's performance.

5.3 Bayesian Learning

5.3.1 Introduce Bayesian Learning

Bayesian learning is a type of machine learning that leverages probability theory and Bayesian inference to make predictions and decisions based on data (Barber, 2012). At the heart of Bayesian learning lies the concept of updating a prior probability distribution with new data to obtain a posterior probability distribution that represents the updated knowledge about the problem being modeled (Bishop and Nasrabadi 2006).

Bayesian learning is rooted in Bayes' theorem as shown in eq.5.1, which states that the posterior probability of a hypothesis is proportional to the product of the prior probability and the likelihood of the data given the hypothesis. In Bayesian learning, the prior probability distribution reflects the initial beliefs or knowledge about the problem being modeled, and the likelihood function represents the probability of the observed data given the hypothesis(Gelman et al. 2013).

$$\mathbf{P}\left(A|B\right) = \frac{\mathbf{P}\left(B|A\right) * \mathbf{P}\left(A\right)}{\mathbf{P}\left(B\right)} \tag{5.1}$$

where $\mathbf{P}(A|B)$ is the probability of A given B; $\mathbf{P}(B|A)$ is the probability of B given A; $\mathbf{P}(A)$ is the prior probability of A; and $\mathbf{P}(B)$ is the prior probability of B.

Bayesian learning is guided by four key principles: prior knowledge, likelihood function, Bayesian inference, and decision-making. Prior knowledge refers to the initial belief or prior probability distribution about the problem being modeled, while the likelihood function represents the probability of the observed data given the hypothesis being tested. Bayesian inference is the process of updating the prior probability distribution with new data to obtain the posterior probability distribution. Finally, decision-making involves using the posterior probability distribution to make predictions or decisions.

Bayesian learning has several advantages over other machine learning methods, such as the ability to incorporate prior knowledge and handle uncertainty(Ghahramani 2015). It has numerous applications in various fields, including computer vision, natural language processing, and healthcare(Abdullah, Hassan, and Mustafa 2022).

One popular type of Bayesian learning algorithm is Naive Bayes, which is particularly suitable for classification tasks. Despite the assumption of independence among features, Naive Bayes models can perform well in practice, especially for high-dimensional data(Zhang 2005).

5.3.2 Model implementation

Our study focused on various Naive Bayes models, including Gaussian Naive Bayes, Multinomial Naive Bayes, Bernoulli Naive Bayes, Categorical Naive Bayes, and Complement Naive Bayes. We also included the Gaussian Mixture Model as a probabilistic model for comparison(Scikit-learn.org n.d.). These Naive Bayes models were constructed based on different assumptions about feature conditions, as summarized in Table.5.2.

Before fitting the models, the dataset needs to be pre-processed in order to meet the assumptions of the model on the feature conditions. In constructing the six models, the pre-processed data were fitted primarily through Naive Bayes models in scikit-learn. In addition, we compare the model performance using pre-processed and enhanced data, and based on this, the hyperparameters are tuned by **GridSearch** to boost the model performance further.

Table 5.2: Comparison of models for Bayesian learning part

Model Name	Assumption on Feature Distribution	Pros	Cons
Gaussian Naive Bayes	Features follow Gaussian distribution	Simple and fast algorithm; performs well on continuous features	Sensitive to outliers
Multinomial Naive Bayes	Features are discrete variables	Performs well on text classification and other similar tasks	Cannot handle continuous features; not robust to long-tail feature distributions
Bernoulli Naive Bayes	Features are binary variables	Performs well on text classification and other similar tasks	Cannot handle multivariate discrete features; not robust to long-tail feature distributions
Categorical Naive Bayes	Features are multivariate discrete variables	Performs well on classification tasks; more robust than Multinomial Naive Bayes	Cannot handle continuous features
Complement Naive Bayes	Features are discrete variables	Performs well on imbalanced datasets	Requires more computation resources compared to Multinomial Naive Bayes
Gaussian Mixture Model	Features follow multivariate Gaussian distribution	More accurate modeling of continuous features	More parameters to estimate; longer training time; sensitive to initialization

The training process consists of:

1. Divide the dataset pre-processed for the model into a training set, a validation set and a test set according to Data processing in Material.
2. Fit the above six models using the training and validation sets;
3. Print the results of each model in terms of diagnostic information, confusion matrix, etc;
4. Replace the enhanced dataset to conduct training, diagnostics on the models;
5. Tune the hyperparameters of the models by the GridSearch method and printing the diagnostic information;
6. Validate the model performance with the test set;
7. Visualise and compare the model performance and parameters.

5.3.3 Model Performance Optimisation

The model behaves as follows. Based on the confusion matrices from the Fig.5.4, we can see that the Gaussian Naive Bayes model has the highest accuracy among all models, with an accuracy of 0.757. However, when we look at the confusion matrix, we can see that the model has a high false negative rate, meaning that it incorrectly predicts a lot of defaults as non-defaults. On the other hand, the Multinomial Naive Bayes model has an accuracy of 0.778, but it predicts all instances as non-defaults. This means that the model is not suitable for this problem and the accuracy is misleading.

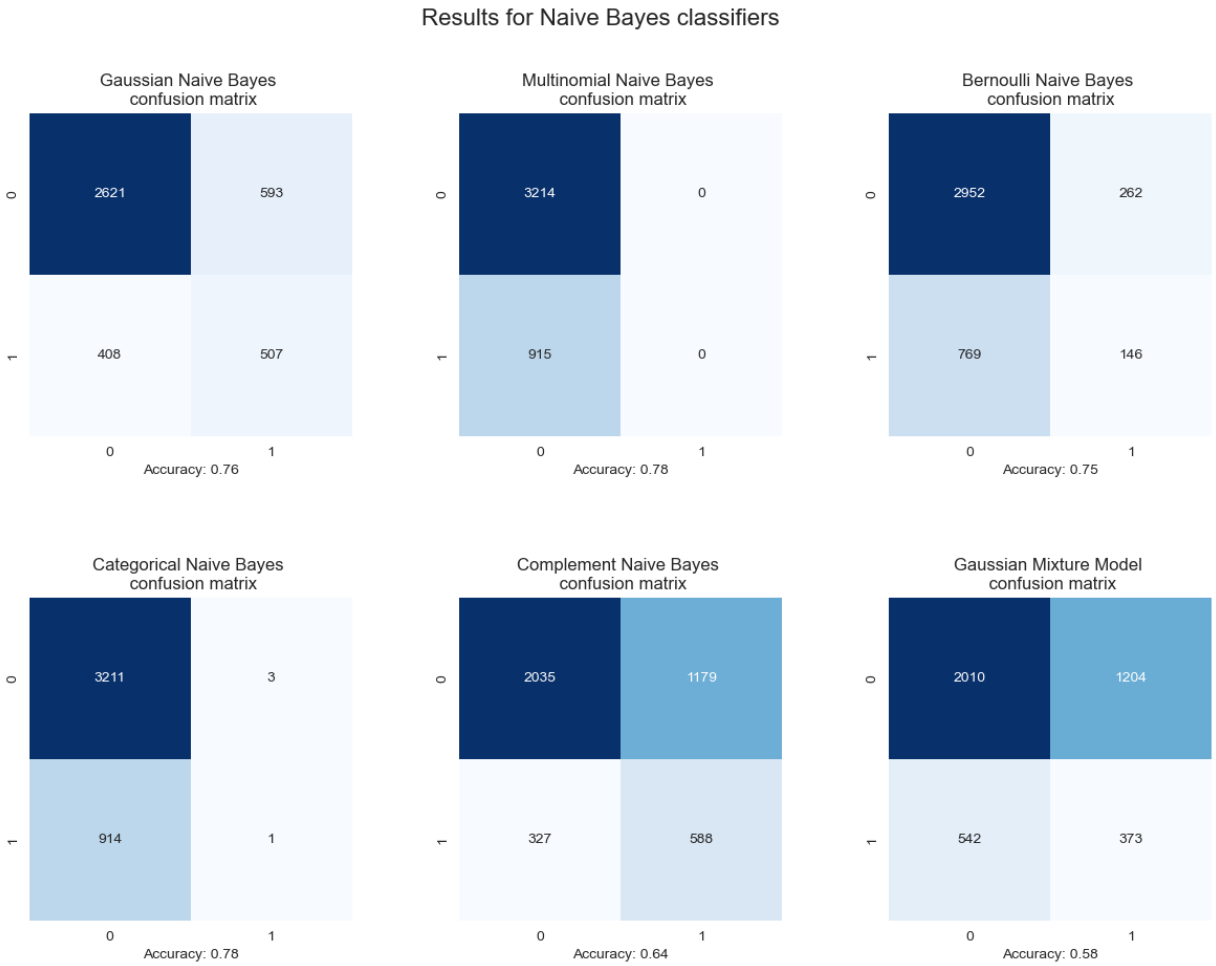


Figure 5.4: Confusion matrix for each model

Table 5.3: Comparison of model performance using pre-processed and enhanced datasets

Model Name	**Accuracy** (Pre-processed data | Enhanced data)	**Weighted avg F1 score** (Pre-processed data | Enhanced data)	**Weighted avg Precision** (Pre-processed data | Enhanced data)	**Optimisation scale of accuracy**
Gaussian Naive Bayes	0.57 | 0.76	0.60 | 0.77	0.75 | 0.78	33.33%
Multinomial Naive Bayes	0.76 | **0.78**	0.76 | 0.68	0.75 | 0.61	2.63%
Bernoulli Naive Bayes	0.49 | 0.75	0.53 | 0.71	0.70 | 0.70	**53.06%**
Categorical Naive Bayes	0.58 | **0.78**	0.61 | 0.68	0.67 | 0.66	34%
Complement Naive Bayes	0.64 | 0.64	0.67 | 0.67	0.74 | 0.74	**0.00%**

According to the results in Table.5.3, it can be seen that by enhancing the data set, different models respond differently. Multinomial Naive Bayes and Categorical Naive Bayes achieve the highest accuracy and weighted average F1 score on the enhanced data. Complement Naive Bayes has the highest weighted average precision on both pre-processed and enhanced data. Gaussian Mixture Model has the lowest performance metrics among all the models, while with the highest degree of accuracy enhancement. At this stage, we believe that Categorical Naive Bayes models explain the highest degree of cases of credit card clients default at 78%.

Table 5.4: GridSearch results for model parameter tuning

Model Name	**Tuning parameters**	**Best parameter**	**Accuracy**	**Optimisation scale**
Gaussian Naive Bayes	var_smoothing	0.0660	**0.8**	5.26%
Multinomial Naive Bayes	alpha	1	0.78	0%
Bernoulli Naive Bayes	alpha	1	0.75	0%
Categorical Naive Bayes	alpha	1	0.78	0%
Complement Naive Bayes	alpha	1.8936	0.63	-1.56%
Gaussian Mixture Model	n_components, covariance_type	10, 'full'	0	-100%

For Gaussian Naive Bayes, the tuning parameter was var_smoothing and the best parameter was 0.066. The model achieved an accuracy of 0.8, which represents a 5.26% increase in accuracy from the default setting. For Multinomial, Bernoulli, and Categorical Naive Bayes, the tuning parameter was alpha, and the best parameter for all three models was 1. These models achieved accuracies of 0.78, 0.75, and 0.78, respectively. For Complement Naive Bayes, the best parameter was 1.8935 and the model achieved an accuracy of 0.63. This represents a decrease in accuracy of 1.56% from the default setting. The results suggest that the Gaussian Naive Bayes model is the most suitable for this dataset, with an accuracy of 0.8 and the ability to identify credit card clients who are likely to default on their payments. The poor performance of the Complement Naive Bayes model and the Gaussian Mixture Model highlight the importance of selecting an appropriate model for a given dataset and an event.

5.3.4 Results analysis

We can see that the Gaussian Naive Bayes model has the highest accuracy, but it may not be the best model for this problem due to its high false negative rate. The Multinomial Naive Bayes model, on the other hand, is not suitable for this problem. The Gaussian Mixture Model has the lowest accuracy and is also not suitable for this problem. The Bernoulli Naive Bayes, Categorical Naive Bayes, and Complement Naive Bayes models have similar performance, with high false negative rates. Therefore, we may need to improve the performance of these models or even consider other models to better predict credit card defaults.

On balance, the analysis suggests that the **Categorical Naive Bayes model** is the best for predicting credit card defaults, achieving a reliable degree of cases explained. However, the performance of all models could be improved, and other models may also need to be considered.

By conducting an in-depth study and analysis, we can conclude that Naive Bayes algorithm has some advantages and disadvantages when it comes to solving the credit card default dataset as follows.

Advantages:

1. Naive Bayes algorithm is computationally efficient and can handle large datasets with high dimensionality.
2. Naive Bayes assumes that features are independent, which may not be true in reality, but it often still performs well in practice.
3. Naive Bayes can handle both continuous and categorical features, making it suitable for datasets with mixed data types.
4. Naive Bayes provides probabilities of the predicted classes, which can be useful in credit risk assessment.

Disadvantages:

1. Naive Bayes assumes that features are independent, which may not be true in reality and can lead to inaccurate predictions.
2. Naive Bayes can be sensitive to irrelevant features, which may decrease its performance.
3. Naive Bayes is a simple algorithm and may not capture complex relationships between features.
4. Naive Bayes assumes that the training set is representative of the test set, which may not always be the case in credit risk assessment.

5.3.5 Section Sum Up

In conclusion, Naive Bayes can be a useful algorithm in solving the credit card default dataset due to its efficiency, ability to handle mixed data types, and ability to provide probabilities of predicted classes. However, its assumption of feature independence and sensitivity to irrelevant features may limit its performance, especially in complex credit risk assessment scenarios.

5.4 Neural Networks

5.4.1 Introduce Neural Networks

The main difference between Neural Networks(Noriega 2005) and other algorithms is that the Neural Network consists of multiple interconnected layers of neurons, where each neuron receives input from other neurons in the previous layer, applies a weighted sum to these inputs, and passes the result through an activation function to produce an output. The weights and biases associated with each neuron are learned during training so the network can accurately predict new data.
The output of an artificial neuron can be formulated as:

$$y_l = \alpha(x_1 w_1 + x_2 w_2 + x_3 w_3 + ... + x_n w_n + b), \tag{5.2}$$

where w_i are the learning width, b is biased term and α is an activation function. The index l is the l number of the hidden layer, and the size of the hidden layer is a hyper-parameter. For multiple layers, we will then pass the results of y_l as the input to the next layer artificial neurons.

Moreover, another huge difference in Neural Networks is that they can capture complex, non-linear relationships between the input and output data by using non-linear activation functions in the neurons, which introduce non-linearities into the network.

Some popular non-linear activation functions (Sharma, Sharma, and Athaiya 2017) used in neural networks include the Sigmoid function, the hyperbolic tangent function, and the rectified linear unit (ReLU) function. These functions introduce non-linearities into the network, enabling it to capture complex data relationships and make predictions on new data.

Let's introduce the commonly used one used in the experiment (Sigmoid). It is a smooth, S-shaped curve that maps any real-valued number in $\mathbb{R}$ to a value between 0 and 1. The Sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{5.3}$$

When using the Sigmoid activation function in the output layer of a neural network for binary classification, the threshold is commonly set to 0.5 where if the outputs greater than or equal to 0.5 as indicating a positive class and else if outputs less than 0.5 as indicating a negative class. This threshold can also be modify for better fitted Neural Network models.

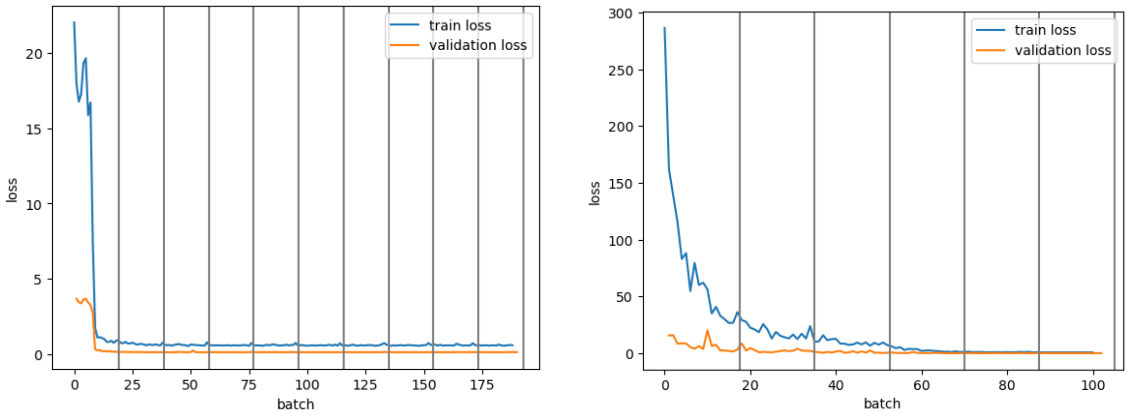
5.4.2 Neural Networks Model Choice and Hyper-parameter tune

Two main packages are used for Neural Networks, namely pytorch and scikit-learn, where pytorch is used to train ANN, and for scikit-learn MLPClassifier is used.

For pytorch package the process goes as follows:

- Write code for customer data-loader with the __len__() method to return the number of samples in the .csv dataset, and the __getitem__() method to take an index as input and return the corresponding sample from .csv dataset.
- Load pre-processed data (see Section 3.5) by calling the customer data-loader.
- Extend the nn.Moudle to write a one-layer ANN model, with hidden dimension 15 and output dimension 2, which represent the two class probability, we choose the maximum one to be our classification results.
- Hyper-parameter setting: batch size: 5, learning rate: 0.001, optimizer: *Adam*, ℓ_2: 0.1, epochs: 10, from Section 3 we see that there are some features with high correlation. Therefore, a ℓ_2 penalty is added to avoid high variance and overfitting problems.

- Train the model on train.csv and evaluate on validation.csv; the measurement of the first ANN model can be seen in Table 5.5 with the column name "ANN (Pytorch)", although the validation accuracy is not too bad, the issue here is that the model almost predicts all results to class 0, this would be the problem of the unbalanced target variable.
- So the model is then trained on a balanced training dataset (balanced_train_data.csv); the model performance can also be seen in Table 5.5 with column name "ANN vs SMOTE(Pytorch)", we see that although the model does obtain some concentration on classify the class 1, it is still not a good classify model, and the accuracy is slightly dropped.
- Another experiment was carried out, where the Network architecture was slightly changed. Instead of having two outputs from ANN, we changed to 1 and applied a sigmoid activation function to return a probability, adding another layer. This way, we could manipulate the thresholds of predicting 1 classes. It turns out the model performed better by setting thresholds as 0.5. The results can be seen in Table 5.5, with column "Logistic ANN(Pytorch)", we now have slightly better classifying the class label 1.
- The model learning curve for the models "Logistic ANN(Pytorch)", and "ANN vs. SMOTE(Pytorch)" can be seen in Figure 5.5, where the Figure 5.5 (a) shows the ANN with sigmoid activation, the model convergent very fast, with less learning carry on, on the hand the learning curve ANN with data enhancement shows a reasonable learning curve. More experiments are required, and the MLPClassifier in scikit-learn is then used to investigate further.



(a) ANN with sigmoid activation (b) ANN with data enhancement(SMOTE)

Figure 5.5: ANN learning curve with two output against one output with logistic activation.

Table 5.5: Neural Netowrk models performance results on validation set.

	ANN (Pytorch)	ANN vs SMOTE (Pytorch)	Logistic ANN (Pytorch)	MLP (scikit-l)	MLP hyper-tun (scikit-l)	MLP hyper-tun SMOTE
ValAccuracy	0.778	0.771	0.769	0.778	**0.817**	0.585
Precision	0.999	0.971	0.930	1.00	**0.881**	0.596
Recall	0.7781	0.771	0.769	0.778	**0.817**	0.585
F1	0.875	0.8573	0.836	0.875	**0.84**	0.552
TP	3212	3166	3108	3214	**3029**	1827
TN	0	15	65	0	**345**	**558**
FP	915	900	850	915	**602**	359
FN	1	47	105	0	**154**	1356

For scikit-learn MLP classifier the process goes as follows:

- We first load the dataset by using the pandas package and normalize data to see if this helps and set the model parameter the same as the ANN as mentioned above model, fit the model on the normalized train.csv set, and evaluate it on the validation set, the model performance can be seen in Table 5.5, with column name "MLP (scikit-l)", we see a similar problem where the model can only predict the 0 class and fail to classify the 1's. Therefore, changing the model architecture and data enhancement does not significantly help to solve the unbalanced target variable problem, and this could be the case that our SMOTE returns too much similar new data.

- The final thing we could do is to do hyper-parameter tunning by running a Random Search CV; the examine hyper-parameters are set as:

  - learning_rate_init: $[0.001, 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40]$;
  - solver : $['lbfgs', 'sgd', 'adam']$;
  - hidden_layer_sizes : $[(12, 2), (8, 2), (7, 2), (15, 2)]$;
  - batch_size: $[2, 3, 4, 5, 6, 7, 8, 9, 15, 18]$;
  - alpha: $[0.00001, 0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.3]$.

  With total 80 fits carried out with cross-validation set to 4, the best model MLP parameters are solver: $'lbfgs'$, learning_rate_init: 0.35, hidden_layer_sizes: $(8, 2)$, batch_size: 15, alpha: 0.01. The model performance can be seen in Table 5.5, with column name "MLP hyper-tun (scikit-l)", we can now see the model is now optimal.

- We have also trained our best MLP network with a balanced training dataset, and the results can be seen in Table 5.5, with column name "MLP hyper-tun SMOTE". However, the correct classification of the class 1 significantly increases from 345 to 558, but the model drops 30% accuracy.

- Finally, the "MLP hyper-tun (scikit-l)" is chosen as the best classifier for the Neural Network family class and evaluates the model on test-set for a final comparison.

5.5 Model Ensemble

5.5.1 Introduce Model Ensemble

Model Ensemble (Zhang 2021) accomplishes a learning task by constructing and combining multiple learners. Individual learners are generated from training data by an existing learning algorithm. Ensemble learning usually starts by generating a set of individual learners and then combining them through some strategy. In homogeneous integration, the individual learners are called base learners, while in heterogeneous integration, the individual learners consist of different learning algorithms and are called component learners.

The most representative families of Ensemble Learning are bagging, boosting, and stacking. In this case, the Morpheus team has applied **Bagging** and **Stacking** in the training process. Fig.5.6 describes our main approach to this task.

The primary goal of this approach is to create a powerful ensemble with strong generalization ability by leveraging the best-trained Decision Trees, Instance-based Learning, Neural Networks, and Bayesian Learning classifiers from the earlier sections. For each classifier, we employ bagging as the base learners, ensuring a heterogeneous primary integration. Building a robust ensemble requires individual learners to be both accurate and diverse, which is why we selected the best-performing learners for bagging and subsequent voting. We utilized both **soft and hard voting** mechanisms to determine the ultimate learner, providing a comprehensive and precise decision-making process.

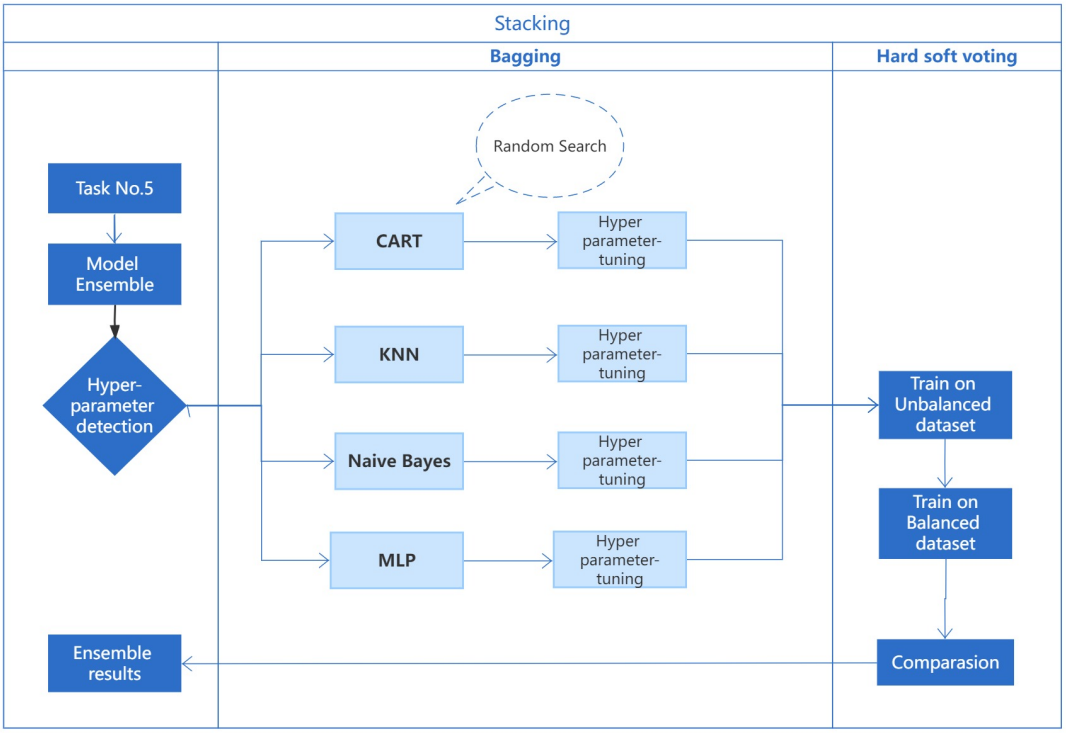


Figure 5.6: The stacking method applied to bagging and hard-soft voting

5.5.2 Bagging

Through sampling with replacement, bagging, also known as bootstrap aggregating, makes copies of the training dataset and trains a single learner on each one. The team used bagging on four different learners for the credit default problem, there are several advantages and disadvantages

to consider. Table. are some advantages and disadvantages of using bagging with four different learners for this problem:

Table 5.6: Comparison of advantages and disadvantages of bagging

	Advantages:	Disadvantages:
Diversity	By training multiple learners on slightly different subsets of the data, the ensemble can capture a variety of perspectives, which can improve the model's overall performance.	Choosing appropriate base learners is crucial for the success of the ensemble. If the base learners are too similar or weak, the ensemble may not provide significant improvements.
Computational cost:	Parallelism: Bagging can be easily parallelized, as each learner is trained independently, which can speed up the training process.	Training multiple learners and tuning their hyperparameters can be computationally expensive, especially if the dataset is large or the algorithms are complex.
Reduction of overfitting:	Bagging helps reduce overfitting since each individual learner is trained on a different subset of the data. This results in a more robust model that is less sensitive to noise.	
Accuracy and complexity	The combination of multiple learners can lead to better overall performance, as the ensemble can compensate for the weaknesses of individual learners.	Combining multiple learners can result in a more complex model, which can be harder to interpret and explain.
Hyperparameter tuning:	By using different learners, it's possible to explore a wider range of hyperparameters, which can further improve the model's performance.	While tuning hyperparameters can improve the model's performance, it can also be time-consuming and computationally expensive, especially when dealing with multiple learners.

In summary, using bagging on four different learners for the credit default problem can lead to improved model performance and generalization. However, it also comes with some challenges, such as increased complexity and computational cost. Hyper-parameter tuning, while potentially beneficial, may further increase these challenges.

5.5.3 Stacking using voting

After training the base learners using bagging, the team use a voting mechanism to combine predictions. Voting has been supposed to be either hard or soft:

Hard voting:
The final prediction is the class with the most votes from the individual base learners. In hard voting, the final prediction is the class with the most votes from the individual base learners. Suppose you have N base learners, and each learner provides a predicted class label for a given input. The final prediction is determined by selecting the class with the highest frequency among the N predicted labels.

Mathematically, let $y_i^{(j)}$ denote the predicted class label for the *i-th* input by the *j-th* base learner, where $i = 1, 2, \ldots, N$ and $j = 1, 2, 3, 4$. The ensemble prediction y_i can be obtained using a majority vote:

$$y_i = mode\left(y_i^{(1)}, y_i^{(2)}, y_i^{(3)}, y_i^{(4)}\right), \tag{5.4}$$

where mode$(\cdot)$ is a function that returns the most common element in a set.

Soft voting:
The final prediction is obtained by averaging the predicted class probabilities of the individual base learners and selecting the class with the highest average probability. In soft voting, the final prediction is obtained by averaging the predicted class probabilities of the individual base learners and selecting the class with the highest average probability. This method takes into account not only the predicted class labels but also the confidence of each base learner in its prediction.

Mathematically, let $p_i^{(j)(k)}$ denote the predicted probability of the *i-th* input belonging to class k by the *j-th* base learner, where $i = 1, 2, \ldots, N$, $j = 1, 2, 3, 4$, and $k = 1, 2, \ldots, K$, K is the

number of classes. The ensemble prediction y_i can be obtained by averaging the predicted class probabilities and selecting the class with the highest average probability:

$$y_i = argmax_k \left(\sum_{j=1}^{4} \frac{\left(p_i^{(j)(k)} \ for \ j \right)}{4} \right) \tag{5.5}$$

where argmax$(\cdot)$ is a function that returns the index (or class) that maximizes the expression inside the parentheses.

Hard voting focuses on the most common predicted class label, while soft voting considers the confidence of each base learner in its prediction by averaging the predicted class probabilities. Soft voting can provide more accurate predictions in some cases, as it takes into account the base learners' confidence, but it requires the base learners to output class probabilities, which is not always available or feasible for all algorithms. In this project, the team tried both of them and compared the outputs as follows.

5.5.4 Model training process and results

We use the best-trained Decision Trees, K-Nearest Neighbour, MLP, and Gaussian naive Bayesian classifiers to build model ensembles where the hyper-parameter is defined as the same as the models mentioned earlier tuning section.

Table 5.7: Compare the soft and hard volting stacking with bagging-stacking

	Stacking Soft Voting	Stacking Hard Voting	Bagging-Stacking Soft Voting	Bagging-Stacking Hard Voting
Train Accuracy	0.869	0.831	**0.836**	0.832
Val Accuracy	0.810	0.809	**0.813**	0.810

The overall model training process goes as follows:

- First, load best-trained Decision Trees, K-Nearest Neighbour, MLP, and Gaussian naive Bayesian classifies and then stack them together using a voting mechanism to train a stacking ensemble model. Since there are few hyper-parameters in the model input, the model is fitted twice with hard and soft voting methods. The results can be found in Table 5.7 with column "Stacking Soft Voting", and "Stacking Hard Voting", where stacking soft voting is slightly better than soft voting with validation accuracy around 0.810.

- For Bagging and Boosting, we can only pass one estimator to base_estimator, and the main idea behind it is to train one model on random samples of the training data to reduce its variance. Therefore, we can train 4 Bagging and Boost for each classifier and pass it into the stacking classifier to get a final result.

- Bagging ensemble method combines multiple independently trained models to reduce variance and improve stability. In Bagging, multiple models are trained on different subsets of the training data using bootstrapping (sampling with replacement). The final prediction is made by aggregating the predictions of all the models. On the other hand, the Boosting ensemble method combines multiple weak models to create a robust model, where the models are trained sequentially, and each subsequent model tries to correct the mistakes made by the previous models. Therefore, In our case, the Bagging ensemble method is more suitable for classifier trains; we then implemented the bagging method for each classifier and passed all the bagging models to stacking to get a single model ensemble stacking classifier.

- For each Bagging classifier, we used random search cross-validation to do hyper-parameter tuning; the parameter spaces are defined as:

- n_estimators: $[[3, 5, 8, 10, 12, 14]]$;
- max_samples: $[0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0]$;
- max_features: $[0.1, 0.3, 0.5, 0.7, 0.9, 1.0]$.

with total 150 fits and cross-validation is set to 10. The results of bagging classifiers before and after hyper-parameter tuning can be seen in Table 5.8, where after hyper-parameter tuning, all four classifiers show a dramatic validation accuracy increase.

- Finally, we passed those four bagging classifiers to the stacking to generate one model ensemble with a voting mechanism. We tested it with soft, hard, and balanced training sets generated from the SMOTE algorithm. The results can be seen in Table 5.9, where bagging-stacking soft voting is our best ensemble model with a validation accuracy of 0.820. It is noticeable that for fitting the backing-stacking model on a balanced dataset, we obtained approximately 86.1% for classifying the correct default payment:Yes class, but the accuracy is only around 40.4%.

Table 5.8: Compare the bagging hyper-parameter tuning results

	Bagging-KNeighbors	Bagging-GaussianNB	Bagging-Decision Trees	Bagging-MLP
Before tuning:				
Train Accuracy	0.928	0.767	0.815	0.821
Val Accuracy	0.798	0.765	0.817	0.812
After tuning:				
Train Accuracy	0.931	0.797	0.816	0.820
Val Accuracy	0.803	0.798	0.818	0.814

Table 5.9: Compare the bagging-stacking results

	Bagging-Stacking Soft Voting	Bagging-Stacking Hard Voting	Bagging-Stacking SMOTE Soft
ValAccuracy	0.820	0.817	0.4046
Precision	0.884	0.899	0.719
Recall	0.820	0.817	0.406
F1	0.842	0.846	0.402
TP	3037	3069	856
TN	351	307	815
FP	596	640	132
FN	146	114	2327

Chapter 6

Conclusion

6.1 Results and Discussion

The project aimed at the case of customers' default payments in Taiwan. The Morpheus team has applied 4 representative algorithms from different families and compared the predictive accuracy. In the last part a stacking method from Ensemble Learning family has been picked to combine all the above learners.

Table 6.1: Results metrics for best models

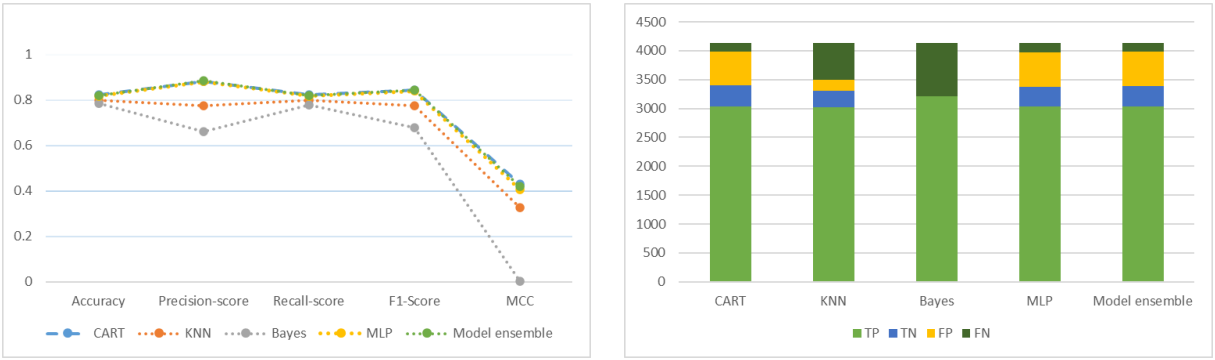
	CART	**KNN**	**Bayes**	**MLP**	**Ensemble**
Accuracy	0.823	0.8009	0.7841	0.8169	0.8203
Precision-score	0.8834	0.7765	0.66	0.8807	0.8839
Recall-score	0.823	0.8009	0.78	0.8169	0.8203
F1-Score	0.8439	0.7758	0.68	0.8393	0.8424
MCC	0.4308	0.3267	0.0021	0.4075	0.4196
TP	3036	3023	3211	3029	3037
TN	363	284	1	345	351
FP	584	191	3	602	596
FN	147	631	914	154	146
sum	4130	4129	4129	4130	4130

6.1.1 Confusion Matrix Evaluation:

Bayes has the highest True Positive rate, suggesting that it is effective at identifying positive cases, but it also has a very low False Positive rate, which may indicate some bias toward the positive class. CART and KNN both have higher rates of False Positive and False Negative, suggesting that they need further optimization to improve their performance. MLP has a higher False Positive rate, meaning it is incorrectly classifying some negative cases as positive. Ensemble has a higher False Positive rate, but a relatively lower False Negative rate, indicating that it may be more conservative in classifying cases as positive.

6.1.2 Metrics Evaluation:

CART has the highest accuracy at 0.823, followed by Model Ensemble at 0.8203, MLP at 0.8169. CART correctly classifies default and non-default instances more often than the other models in this credit default prediction task. A high accuracy is desirable in credit default prediction, as it can help financial institutions make better decisions when providing credit.



(a) Classification report comparison

(b) Confusion matrix components comparision

Figure 6.1: The metrics evaluation for five tasks.

Model Ensemble has the highest precision score at 0.8839, closely followed by CART at 0.8834, MLP at 0.8807. This indicating that when the ensembled model predicts a credit default, it is correct 88.39% of the time. The high precision minimizes the risk of misclassifying non-default instances as default instances, which could lead to a loss of potential customers.

CART has the highest recall score at 0.823, followed by Model Ensemble at 0.8203, MLP at 0.8169. CART has a high recall score, which reduces the chances of missing default instances and helps financial institutions avoid providing credit to customers who are likely to default. CART has the highest F1-score at 0.8439, followed by Model Ensemble at 0.8424, MLP at 0.8393. The high F1-score of CART indicates it has a good balance between precision and recall, optimizing the trade-off between false positives and false negatives.

CART has the highest MCC at 0.4308, followed by Model Ensemble at 0.4196, MLP at 0.4075, KNN at 0.3209, and Naive Bayes at 0.0021. The highest coefficient which measures the quality of binary classification and takes into account true and false positives and negatives. This indicates CART has a better model performance, especially in imbalanced datasets like this project.

Based on these metrics and the context of credit default prediction, CART is the best-performing model. It not only achieves the highest accuracy but also maintains a good balance between precision and recall, which is crucial for making informed credit decisions and managing risk effectively.

6.2 Limitations and Future Work

The patterns that were seen in the credit default dataset might be explained by things like intricate interactions between characteristics, correlations between unrelated variables, the existence of outliers or noise, and unequal class distribution. These elements must be taken into account when choosing the best machine learning algorithms since they may affect how well the models function.

Due to their independence, susceptibility to irrelevant characteristics, distance-based nature, and model simplicity requirements, KNN and Bayesian models performed relatively worse than other models on the credit default dataset. Scaling of features and unbalanced datasets are two further problems that may affect performance. Investigating various modelling approaches or ensemble methods may be able to assist credit default prediction perform more effectively overall. Compared to KNN and Naive Bayes, CART, MLP, and model ensemble have shown improved performance in this challenge. Their excellent performance in this particular challenge is a result of their proficiency with complicated relationships, feature representation, and generalisation approaches. All three models profit from their unique advantages: MLP's potential to learn complicated patterns through several layers, CART's hierarchical structure and interpretability, and Model Ensemble's ability to draw on the knowledge of numerous models.

Further analysis and feature engineering could help address these issues and improve model performance. Consider the uncertainty, the real probability is unknown,the accuracy of the predicted default likelihood will yield more valuable results than the classification of clients as either credible or not credible, which yields a binary outcome. When viewed through the lens of risk management, the significance of this study is to reduce the risk of bad records caused by banks or credit institutions providing loans to risky accounts by improving the accuracy of predicting the possibility of default.

Bibliography

Abdullah, Abdullah A, Masoud M Hassan, and Yaseen T Mustafa (2022). "A review on bayesian deep learning in healthcare: Applications and challenges". In: *IEEE Access*.

Berzal, F. and N. Matín (2002). "Data mining". In: *ACM SIGMOD Record* 31.2, p. 66. DOI: https://doi.org/10.1145/565117.565130.

Bishop, Christopher M and Nasser M Nasrabadi (2006). *Pattern recognition and machine learning*. Vol. 4. 4. Springer.

Fernández, Alberto et al. (2018). "SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary". In: *Journal of artificial intelligence research* 61, pp. 863–905.

Gelman, Andrew et al. (2013). *Bayesian data analysis*. CRC press.

Ghahramani, Zoubin (2015). "Probabilistic machine learning and artificial intelligence". In: *Nature* 521.7553, pp. 452–459.

learn.org, scikit (n.d.). *3. Model selection and evaluation - scikit-learn 0.24.2 documentation*. URL: https://scikit-learn.org/stable/model_selection.html.

Liu, Fei Tony, Kai Ming Ting, and Zhi-Hua Zhou (2008). "Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE, pp. 413–422.

Newyorkfed.org (2019). *The Center for Microeconomic Data - FEDERAL RESERVE BANK of NEW YORK*. URL: https://www.newyorkfed.org/microeconomics/hhdc.html.

Nordhausen, K. (2014). "An Introduction to Statistical Learning-with Applications in R by Gareth James, Daniela Witten, Trevor Hastie & Robert Tibshirani". In: *International Statistical Review* 82.1, pp. 156–157. DOI: https://doi.org/10.1111/insr.12051_19.

Noriega, Leonardo (2005). "Multilayer perceptron tutorial". In: *School of Computing. Staffordshire University* 4, p. 5.

OpenAl (2023). "OpenAl, ChatGPT". In: URL: https://chat.openai.com/APA.

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

Scikit-learn.org (n.d.). *1.9. Naive Bayes — scikit-learn 0.21.3 documentation*. https://scikit-learn.org/stable/modules/naive_bayes.html.

Sharma, Sagar, Simone Sharma, and Anidhya Athaiya (2017). "Activation functions in neural networks". In: *Towards Data Sci* 6.12, pp. 310–316.

Tesher, D. and N. York (n.d.). *Regional Credit Conditions Chair*. Accessed 26 Mar. 2023. URL: https://www.spglobal.com/_assets/documents/ratings/research/101566870.pdf.

Thach, Nguyen Huy, Porntep Rojanavasu, and Ouen Pinngern (2008). "Cost-Xensitive XCS Classifier System Addressing Imbalance Problems". In: *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. Vol. 2, pp. 132–136. URL: https://ieeexplore.ieee.org/document/4666094.

Uci.edu (2016). *UCI Machine Learning Repository: default of credit card clients Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients.

Zhang, Harry (2005). "Exploring conditions for the optimality of naive Bayes". In: *International Journal of Pattern Recognition and Artificial Intelligence* 19.02, pp. 183–198.

Zhang F., Li J. Wang Y. Guo L. Wu D. Wu H. Zhao H. (2021). "Ensemble Learning Based on Policy Optimization Neural Networks for Capability Assessment." In: *Available at SSRN 2634092*. URL: https://doi.org/10.3390/s21175802.