Supervised learning tasks: A Comparative Study on Credit Card Default Prediction

Report for the CEGE0004: Assignment

Group Chaos

Jiaxuan Han
22032087
Uesjh6@ucl.ac.uk

Wanchen Liu
22096586
Uceswl5@ucl.ac.uk

Ruikun Wu
22062571
Ucesrw3@ucl.ac.uk

Jiajie Sun
22077268
Ucfaune@ucl.ac.uk

Zheyi Ju
22103036
Ucfazju@ucl.ac.uk

Alice Busby (Absent)
22203704
Ucesbus@ucl.ac.uk

Content

1. Introduction 3
    1.1. Background 3
    1.2. Research target 3
    1.3. Performance assessment 3
    1.4. Workflow 3
2. Learning Tasks 4
    2.1. Task Features 4
    2.2. Expected Output 4
    2.3. Mathematical formalization 4
        2.3.1. Decision Tree 4
        2.3.2. Instanced-based learning 5
        2.3.3. Neural Network 5
        2.3.4. Bayesian learning 6
        2.3.5. Model Ensemble 7
3. Material 8
    3.1. Data Exploration 8
    3.2. Attribute Analysis 8
        3.2.1. Attribute Names and Types 8
        3.2.2. Missing Values 10
        3.2.3. Noisiness and Noise Type 10
        3.2.4. Distribution Type 10
        3.2.5. Target Attribute Identification 10
        3.2.6. Other Data Visualization 10
        3.2.7. Common Transformations 11
        3.2.8. Test Set Sampling 11
4. Technology 11
    4.1. Jupyter Notebook 11
    4.2. Python 11
    4.3. Pandas 12
    4.4. NumPy 12
    4.5. Scikit-learn (sklearn) 12
    4.6. Pickle 12
    4.7. PyTorch 12

4.8. Matplotlib 12
4.9. Seaborn 12
5. Learning Algorithms 13
5.1. Decision Trees 13
5.1.1. Introduction of Decision Trees 13
5.1.2. Modelling 13
5.1.3. Hype Parameters and Validation 13
5.1.4. Comparisons of Models 16
5.2. Instance-Based Learning 17
5.2.1. Introduction of Instanced-based 17
5.2.2. Prepare dataset 17
5.2.3. Define accuracy measure 17
5.2.4. Try KNN implementation 17
5.2.5. Find Hyper-parameters 18
5.2.6. Comparison of Models 19
5.3. Neural Networks 20
5.3.1. Implementation 21
5.3.2. Hyperparameter Tuning 23
5.3.3. Validation of Performance 23
5.3.4. Further Improvement 23
5.4. Bayesian Learning 26
5.4.1. Introduction of Bayesian Learning 26
5.4.2. Implementation, Hype Parameter, and Validation 27
5.4.3. Comparison of Models 27
5.5. Model Ensemble 28
5.5.1. Task Description 29
5.5.2. Data Preprocessing 29
5.5.3. Ensemble Models 29
5.5.4. Hyperparameter and Validation 30
5.5.5. Performance Evaluation 31
6. Conclusion 33
7. Git Log 35
8. Reference 37

1. Introduction

1.1. Background

Credit card debt crisis and delinquency issues are the major concern for all financial institutions, as these problems could lead to significant liquidity crises that threaten the survival of financial institutions and the normal operation of financial markets (Juan, 2011). However, to increase the competitive advantages and capture market shares, many banks in Taiwan over-issued credit cards to unqualified applicants, which attracts the attention of the regulator (Yeh and Lien, 2009). As a result, reviewing and assessing cardholders' information to determine if there are potential defaults is a priority for banks in Taiwan.

1.2. Research target

Therefore, by analyzing and studying the 'default of credit card client dataset' from UCI machine learning repository, this project aims to provide several prediction models for financial institutions to assist in the assessment and prediction of customer defaults. To be specific, as decision tree model and neural networks model are valuable models for predicting customer default (Gui, 2019), these two algorithms will be used in this project. In addition, to further compare the performance of different models, instance-based learning model, bayesian learning model, and model ensemble algorithm will also be used. By assessing and comparing these models, this article will provide the best model for predicting customer defaults.

1.3. Performance assessment

In this project, the performance of different models will be evaluated based on the accuracy of predicting defaults. Specifically, the whole dataset will be divided to training set and test set. After fitting the models, the test set will be used for prediction and the accuracy of different models will be compared. Finally, according to the prediction accuracy of the best model, the degree of completion of the whole project is also assessed. The detailed information will be discussed in the following part.

1.4. Workflow

To successfully complete the entire project, the workflow is divided into the following sections. First of all, after acquiring the data, this dataset will be explored, analyzed, and processed to ensure the data quality. Next, the processed dataset will be divided into a training set (80%) and test set (20%) for model fitting and prediction. Subsequently, different models will be fitted. It is noted that the dataset may be processed again at this stage based on the needs of different models. Also, when fitting the models, different hype parameters of models will be tested and selected, and the validation will be implemented to ensure the best performance of different models. Finally, the prediction will be performed using different models and their accuracy will be assessed to discuss project completion and provide the best model for default prediction.

2. Learning Tasks

2.1. Task Features

The dataset comprises 23 features, which are attributes of credit card clients. These features include age, sex, education, marital status, and repayment status, among others. In the data preprocessing stage in some models, we modify and apply one-hot encoding to specific features (e.g., SEX and EDUCATION).

2.2. Expected Output

The model's objective is to predict whether a client will default in the subsequent month. The output is a binary classification result: 0 indicates no default, while 1 signifies default.0

2.3. Mathematical formalization

The credit default prediction problem can be formalized as follows:

In the dataset, each instance of i is represented as a tuple (x_i, y_i). Here, y_i is a binary label that denotes the credit default status, and x_i is a feature vector.

The feature vector for instance i is represented by x_i = (x_{i1}, x_{i2}, ..., x_{i23}), where x_{ij} is the value of feature j for instance i. These characteristics may include data on a person's credit history, loan amounts, and other pertinent financial details.

The binary label for instance i y_i, represents whether a person will default on their debt (1) or not (0).

The task is to learn a function f: X → Y, where X is the space of all possible feature vectors and Y is the set of binary labels {0, 1}.

2.3.1. Decision Tree

Decision tree is a commonly used machine learning model which could be used for both classification and regression tasks. In this project, classification will be used to predict two attributes, default and not default.

This algorithm works by splitting data according to their features. It starts with non-leaf node s which represent and attribute to test. Then some edges connected with node means the value that the data need to conform to. This step could repeat until having all the features included. After that some leaf nodes represent the predicted class of the instance. The final model will look like a tree.

2.3.2. Instanced-based learning

Instance-based algorithms store the entire training test which is used directly to make predictions on new data.

There are several learning tasks can be performed by instanced-based algorithms such as classification, regression, outliers detection. In classification, the algorithm is trained on a set of labeled examples and then used to classify new data. In regression, the algorithm is trained on input-output pairs and create its own regression model then used it on new data. In outliers detection, the algorithm is mainly used to detect the anomalous data. KNN algorithm can be used in all three tasks by taking advantage of the k nearest neighbors.

2.3.3. Neural Network

Mathematically, the MLP model can be represented as a series of transformations(Bishop, 2007):

$$h_1 = g_1 \times (W_1 \times x_i + b_1) \qquad \text{eqn.1}$$

$$h_2 = g_2 \times (W_2 \times h_1 + b_2) \qquad \text{eqn.2}$$

…...

$$h_L = g_L \times (W_L \times h_{L-1} + b_L) \qquad \text{eqn.3}$$

$$y_i = activate(h_L) \qquad \text{eqn.4}$$

Where,

L is the number of layers in the MLP, including input, hidden, and output layers.

W_i and b_i represents the weight matrix and bias vector for layer i, respectively.

g_i is the activation function applied element-wise to the output of layer i.

h_i is the output of layer i.

$activate(\quad)$ is the output activation function that converts the final layer's output into probabilities summing to 1.

Finding the ideal weight matrices W_i and bias vectors b_i for all layers i is the objective in order to minimize a loss function over the entire dataset, for example the cross-entropy loss:

$$L(W, b) = -\sum\{i = 1\}^m \left[y_i \times \log\big(f(x_i)\big) + (1 - y_i) \times \log\big(1 - f(x_i)\big)\right] \qquad \text{eqn.5}$$

Where,

m is the number of instances in the dataset, and $f(x_i)$ is the output of the MLP model for instance i.

In order to minimise the loss function, backpropagation and optimisation techniques like Adam or SGD are employed to iteratively update the weights and biases. By assessing the model's accuracy on a test set that wasn't used during training, the performance of the model is confirmed:

$$Accuracy = \frac{Number\ of\ correctly\ classified\ instances}{Total\ number\ of\ instances\ in\ the\ test\ set} \qquad \text{eqn.6}$$

2.3.4. Bayesian learning

Bayesian learning is a statistical method based on probability (Mitchell, 1997). Basically, this algorithm is based on the theorem as shown below.

$$P(h|D) = P(D|h)P(h)/P(D) \qquad \text{eqn.7}$$

Where,

$P(h)$ = prior probability of hypothesis h.

$P(D)$ = prior probability of training data D.

$P(h|D)$ = probability of h given D.

$P(D|h)$ = probability of D given h.

Specifically, in this project, the Bayesian learning algorithm is operated based on the number of occurrences of different tokens and the probability of different outcomes. As a result, to record different tokens and successfully implement the Bayesian learning for this project, all attributes are processed as discrete text variables. Next, for each row, the combination of these columns is converted to create a matrix of token counts. As a result, these token count matrixes are used as training sets by the Bayesian model to learn the number of different outcomes, the occurrences of different tokens corresponding to different outcomes, and the total number of occurrences of different tokens in the training set. After fitting the model, this information is stored by the model. In the future, when the model receives a new token count matrix from the test set, based on the tokens in this matrix, the model will calculate the scores of different outcomes to assess the probability. After the comparison, the outcome with the highest score will be output as the prediction result. For example, it is assumed that after fitting the model, the model records the information as below.

For outcome 1, the number of occurrences of token are:

Token A: 5

Token B: 3

Token C: 11

Token D: 9

And the number of the outcome 1 in training set is 5

For outcome 0, the number of occurrences of token are:

Token A: 1

Token C: 1

Token D: 7

And the number of the outcome 0 in training set is 2

For all tokens:

Token A: 6

Token B: 3

Token C: 12

Token D: 16

When the model receives a new test set for prediction with matrix {Token A, Token C, Token D}, the score of the outcome 1 is 5/6 * 11/12 * 9/16 * 5 = 2.148, while the score of the outcome 0 is 1/6 * 1/12 * 7/16 * 2 = 0.012, which is smaller than 2.148. As a result, the prediction result is 1. However, this is a simple mathematical demonstration of the model. In reality, the situation will be more complex than above and will consider the impact of different hype parameters on the calculation.

2.3.5. Model Ensemble

Suppose dataset D contains n characteristics $\{x_1, x_2, ..., x_n\}$ and a binary target variable y (indicating whether the customer will default).

Basic model:

Decision Tree: $f_{DT}(x)$

Example-based learning: $f_{KNN}(x)$

Bayesian Learning: $f_{GNB}(x)$

Neural network: $f_{MLP}(x)$

Integrated learning methods:

Bagging:

$$F_{bag}(x) = \frac{1}{T} \sum f_{DT\_T}\,(x) \qquad \text{eqn.8}$$

where T is the number of decision trees, $f_{DT\_T}\,(x)$ represents the prediction of the t-th decision tree.

Boosting:

$$F_{ada}(x) = sign(\sum \alpha_t \times f_{DT\_T}\,(x)\,) \qquad \text{eqn.9}$$

where α_T is the weight of the t-th decision tree.

Voting：

$$F_{vote}(x) = \ majority\_ vote(f_{DT}(x), f_{KNN}(x), f_{GNB}(x), f_{MLP}(x)) \qquad \text{eqn.10}$$

3. Material

The Credit Card Default dataset (*UCI Machine Learning Repository: default of credit card clients Data Set*, 2016) contains information on credit card clients in Taiwan, spanning from April 2005 to September 2005. It comprises 30,000 records and 24 attributes, including demographics, credit history, payment history, and bill statements. The target attribute indicates whether a client will default on their credit card payment.

3.1. Data Exploration

The dataset consists of 30,000 records with 24 attributes, including 14 categorical and 10 numerical attributes. The target variable, "default payment next month," is a binary variable that indicates whether the client will default on their payment.

3.2. Attribute Analysis

3.2.1. Attribute Names and Types

Table 1. Attributes

X1	X2	X3	X4	X5
LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE

Amount of the given credit	Gender
	(1= male;
	2 = female)
		Education level
		(1 = graduate school;
		2 = university;
		3 = high school;
		4 = others)
			Marital status
			(1 = married;
			2 = single;
			3 = others).
				Year

Table 2. Attributes

History of past payment (1 = pay duly; 1 = payment delay for one month ... 9 = payment delay for nine months and above)		Amount of bill statement		Amount of previous payment	
x6	the repayment status in September	X12	amount of bill statement in September	X18	amount paid in September
X7	the repayment status in August	X13	amount of bill statement in August	X19	amount paid in August
X8	the repayment status in July	X14	amount of bill statement in July	X20	amount paid in July
X9	the repayment status in June	X15	amount of bill statement in June	X21	amount paid in June
X10	the repayment status in May	X16	amount of bill statement in May	X22	amount paid in May
X11	the repayment status in Apirl	X17	amount of bill statement in Apirl	X23	amount paid in Apirl

3.2.2. Missing Values

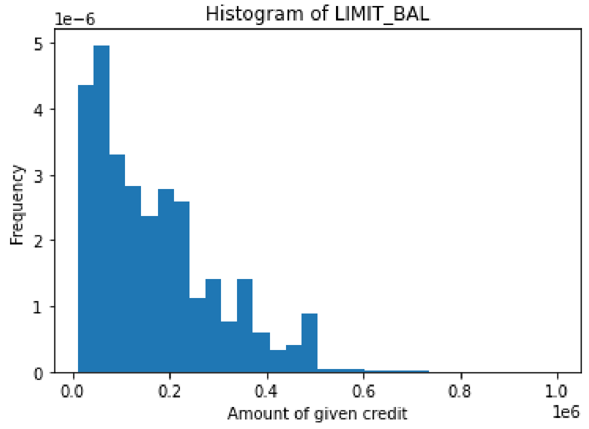
The dataset does not contain any missing values.

3.2.3. Noisiness and Noise Type

Although there are no obvious outliers, some numerical attributes (e.g., BILL_AMT1 to BILL_AMT6, PAY_AMT1 to PAY_AMT6) exhibit high variability, suggesting the presence of stochastic noise.

3.2.4. Distribution Type

Numerical attributes display various distributions. For instance, both LIMIT_BAL and AGE have a skewed distribution, as shown in **Figure 1 and 2**.



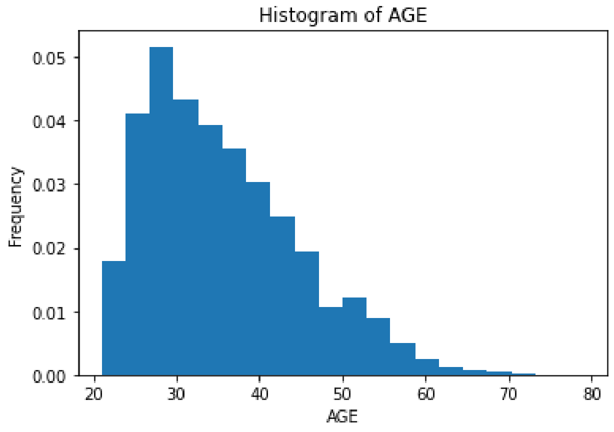


Figure 1. Distribution of LIMIT_BAL

Figure 2. Distribution of AGE

3.2.5. Target Attribute Identification

The target attribute, "default payment next month," is a binary variable that indicates whether a client will default on their credit card payment.

3.2.6. Other Data Visualization

Visualization can offer insights to understand different attributes. Histograms and scatter plots are useful for visualizing numerical data, whereas bar charts and pie charts are suitable for visualizing categorical data. Therefore, **Figure 3. and 4.** below are created to assist in understanding some attributes in the dataset.

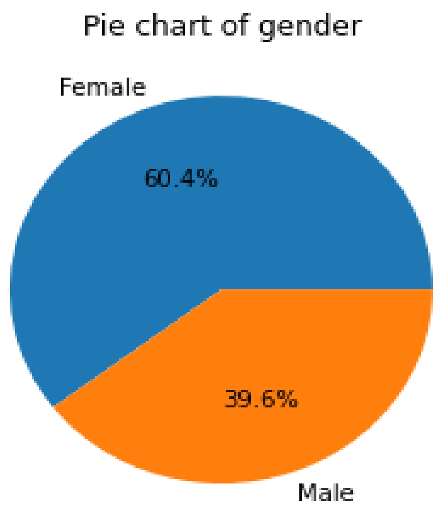


Figure 3. Distribution of Gender Attribute

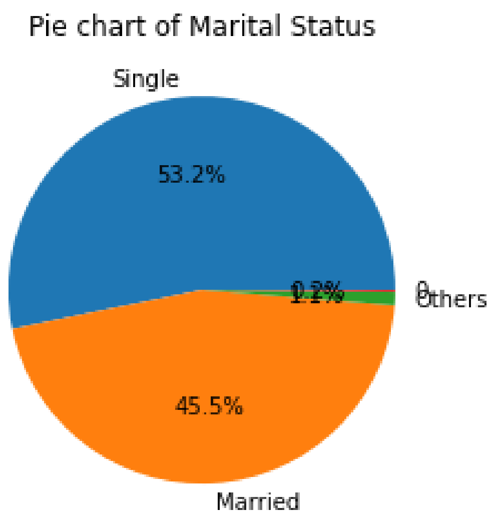


Figure 4. Distribution of Marital Status

3.2.7. Common Transformations

Potential transformations for this dataset include normalization of numerical data to reduce outlier influence, encoding of categorical data into numerical values, converting attributes into token matrix, and feature scaling to ensure all attributes have a similar scale.

3.2.8. Test Set Sampling

The test set can be randomly sampled from the dataset. To guarantee the test set is representative of the population, it should maintain the same proportion of target variable values as the original dataset. Alternatively, stratified sampling can ensure the test set has a similar distribution of values as the original dataset.

4. Technology

4.1. Jupyter Notebook

An open-source website tool for writing and exchanging narrative prose, equations, and live code. We used Jupyter Notebook to write, test, and document our machine learning code.

4.2. Python

A versatile, high-level programming language used for various purposes, including data analysis, machine learning, and web development. We chose Python for its simplicity, readability, and extensive libraries for machine learning.

4.3. Pandas

a well-known Python data analysis and manipulation library. We used Pandas to load, clean, and preprocess the data set.

4.4. NumPy

A Python library for working with arrays and performing numerical operations. We used NumPy for its efficient array manipulation capabilities, which helped in handling the data set.

4.5. Scikit-learn (sklearn)

A Python machine learning framework that provides simple and efficient tools for data analysis and mining. We used Scikit-learn for implementing Decision Trees, Instance-based Learning, Bayesian Learning, and Ensemble Methods. It also provided useful tools for preprocessing, model evaluation, and hyperparameter tuning.

4.6. Pickle

A Python module for serializing and de-serializing Python objects. We used Pickle to save and load the trained machine learning models.

4.7. PyTorch

A Torch-based, open-source machine learning toolkit for Python used for deep learning tasks like building neural networks. We used PyTorch to implement the Neural Network model.

4.8. Matplotlib

A Python module for data visualization. Specifically, it is a plotting library for creating high quality static, dynamic, and interactive charts.

4.9. Seaborn

A Python module for data visualization based on Matplotlib that provides handy functions and interfaces to make it easier for users to create high-quality, aesthetically pleasing statistical graphics.

5. Learning Algorithms

5.1. Decision Trees

5.1.1. Introduction of Decision Trees

Decision trees are a family of algorithms which could handle both regression and classification problems. Each node in a decision tree represents an attribute, and edge behind it shows its value or feature of this attribute. In this project, two widly-used kinds of decision tree algorithms, the ID3 and CART, are used to compare and select the best one with the highest accuracy. The optimal algorithm could use to forecast whether users will default on their credit card payments by inputting limit balance, age, sex, education, marriage and history of payment and paybacks as nodes. For data preprocessing, there is not some specific step in Decision Tree.

5.1.2. Modelling

The ID3 and CART are one of the hyperparameters which are used to train the data. At the begi nning of this algorithm, the max_depth of 6 is selected to train both algorithms. The accuracies a re 0.8257 and 0.8252 respectively. ID3's accuracy is a little bit higher than CART's. The code for this part is shown below.

```
ID3_model = DecisionTreeClassifier(max_depth=6,criterion='entropy')
ID3_model.fit(x_train,y_train)
ID3_y_pred = ID3_model.predict(x_test)
ID3_score =accuracy_score(ID3_y_pred, y_test)
ID3_score
CART_model = DecisionTreeClassifier(max_depth=6,criterion='gini')
CART_model.fit(x_train,y_train)
CART_y_pred = CART_model.predict(x_test)
CART_score =accuracy_score(CART_y_pred, y_test)
CART_score
```

5.1.3. Hype Parameters and Validation

If this project is trying to improve the model performance, both max_depth and criterion are important hyperparameters to adjust. The best way is using the **GridSearchCV** method. GridSearchCV is one of the most valuable and efficient methods to determine optimal hyperparameters. In reality, this method is widely used to fit the best model and perform cross-validation (Saidi et al., 2022). For this algorithm, two hyperparameters are assessed by GridsearchCV and 10 times cross validation are done. Specifically, the max_depth with range from 1 to 10 and two types of criterions are considered. By calculating all kinds of combinations, a criterion and a max_depth will be selected according to the value accuracy after 10-fold cross validation. The best hyperparameters are criterion of gini and max_depth of 3 for this dataset, and the accuracy of this best model is 0.8263, with slight improvement of accuracy. The performances of different models are compared, as shown in **Table 3. The code for this part is shown below.**

Table 3. Comparison of DecisionTree models

Model	Accuracy	Comment
DecisionTreeClassifier(max_depth=6,criterion='entropy')	0.8255	Not Best
DecisionTreeClassifier(max_depth=6,criterion='gini')	0.8252	Not Best
DecisionTreeClassifier(max_depth=3,criterion='gini')	0.8263	Best
DecisionTreeClassifier(max_depth=5,criterion='gini')	Eliminated by GridsearchCV	Not Best
DecisionTreeClassifier(max_depth=8,criterion='gini')	Eliminated by GridsearchCV	Not Best
DecisionTreeClassifier(max_depth=8,criterion='entropy')	Eliminated by GridsearchCV	Not Best

```
parameters = {"criterion":("gini","entropy") ,"max_depth":[*range(1,10)]}
clf = DecisionTreeClassifier(random_state=0)
GS = GridSearchCV(clf,parameters,cv=10)
GS = GS.fit(x_train,y_train)
print(GS.best_params_)
best_model = DecisionTreeClassifier(max_depth=3,criterion='gini')
best_model.fit(x_train,y_train)
y_pred = best_model.predict(x_test)
score =accuracy_score(y_pred, y_test)
score
```

Additionally, to display the validation process clearer, **Figure 5 and 6** which contain the accuracies with various max_depth in x-axis are plotted below. The accuracies in y-axis are calculated by 10-fold cross validation to make them more reliable.

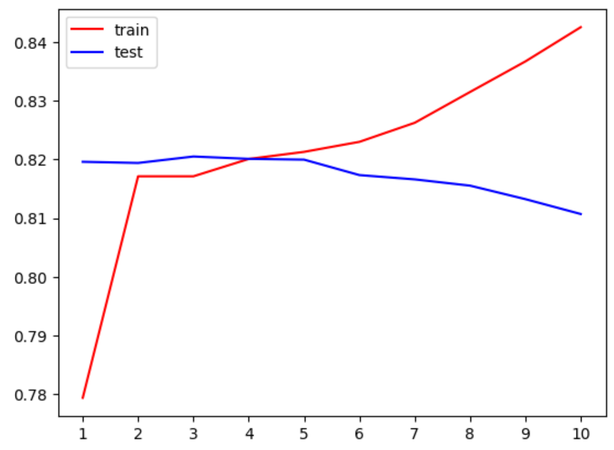


Figure 5. Accuracy of ID3

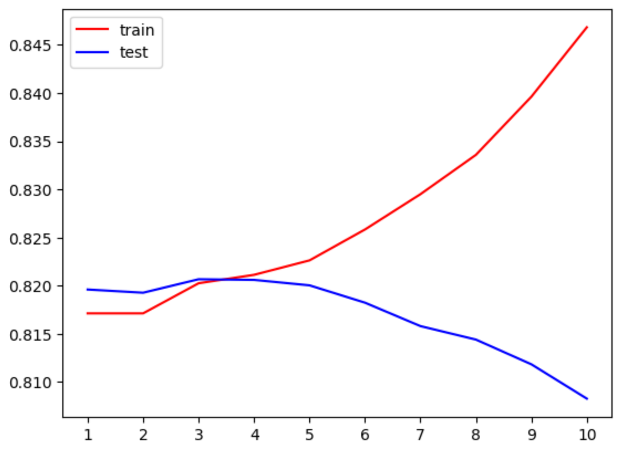


Figure 6. Accuracy of CART

The highest point on blue test line suggests the best value of max_depth. Two blue lines fluctuate around the value of accuracy 0.82 when max_depth is between 1 and 5. Then they both drop when max_depth increases. The red lines are the accuracies of train data. They keep increasing when max_depth getting bigger. This suggests that the model becomes overfitting to the train data when the max_depth is too large. **The code for this part is shown below**.

```
scoreTest = []
scoreTrain = []
for i in range(10):
    clf = DecisionTreeClassifier(random_state=0, max_depth=i+1, criterion="entropy")
    clf = clf.fit(x_train,y_train)
    onceTrain = clf.score(x_train,y_train)
    onceTest = cross_val_score(clf,x,y,cv=10).mean()
    scoreTest.append(onceTest)
    scoreTrain.append(onceTrain)

plt.figure()
plt.plot(range(1,11),scoreTrain,color="red",label="train")
plt.plot(range(1,11),scoreTest,color="blue",label="test")
plt.xticks(range(1,11))
plt.xlabel('Max_Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
scoreTest = []
scoreTrain = []
for i in range(10):
    clf = DecisionTreeClassifier(random_state=0, max_depth=i+1, criterion="gini")
    clf = clf.fit(x_train,y_train)
    onceTrain = clf.score(x_train,y_train)
    onceTest = cross_val_score(clf,x,y,cv=10).mean()
```

```python
    scoreTest.append(onceTest)
    scoreTrain.append(onceTrain)
plt.figure()
plt.plot(range(1,11),scoreTrain,color="red",label="train")
plt.plot(range(1,11),scoreTest,color="blue",label="test")
plt.xticks(range(1,11))
plt.xlabel('Max_Depth')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

5.1.4. Comparisons of Models

Overall, the CART algorithm with max_depth 3 is the optimal model. The accuracy of this model is 0.8263, which only has little improvement comparing with the result before. The detailed comparison of different models is shown in **Figure 7**.

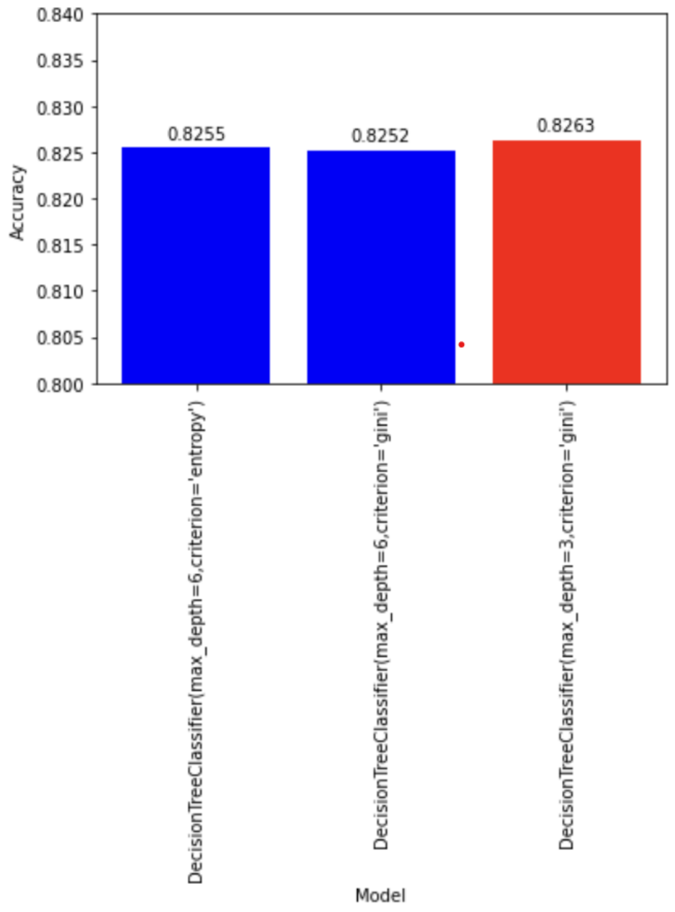


Figure 7. Comparison of decision tree models

5.2. Instance-Based Learning

5.2.1. Introduction of Instanced-based

Instance-based learning is a type of machine learning approach in which the model is trained by memorizing the examples from the training set rather than building a general model that can be applied to new data.('Instance-based learning', 2020) The most common example of Instance-based learning is k-nearest neighbor (KNN) algorithm, which assigns a class label to the new instance based on the class labels of its k-nearest neighbors in the training data.

5.2.2. Prepare dataset

Before modelling, the dataset has been accessed and prepared. During preparing data, y_train and y_test are converted to one-dimensional array.

5.2.3. Define accuracy measure

Next, the accuracy measure that calculates the accuracy of a predicted output compared to the actual output is defined. This function takes in two parameters, ys and ys_hat, which are lists containing the true labels and predicted labels, respectively.

```
def accuracy(ys, ys_hat):
    res = 0
    for y, y_hat in zip(ys, ys_hat):
        if y == y_hat:
            res += 1
    res /= len(ys)
    return res
```

5.2.4. Try KNN implementation

After defining accuracy measure, the KNeighborsClassifier of scikit-learn is implemented. Specifically, the packages needed is imported and an instance of the KNeighborsClassifier class from sklearn is created. Subsequently, KNN starts to work by finding the k-nearest neighbors of a given data point in the feature space, and assigning it to the class that is most common among its neighbors.

The n_neighbors parameter specifies the number of nearest neighbors to consider when making a prediction. Put the random number of neighbors that n=3 inside KNeighborsClassifier class using 'euclidean' metric. After creating the classifier object, the code fits it to the training data using the fit method, which trains the model by memorizing the training dataset. The x_train parameter contains the feature vectors for the training data, while y_train contains the corresponding target labels. Then output shows the test accuracy is 0.7425. Then try another distance metric ‘cosine’ with n = 4. The test accuracy is 0.7785 with n=4.

```python
knn_clf = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_clf.fit(x_train, y_train)
y_test_pred = knn_clf.predict(x_test)
print('Test accuracy of kNN', accuracy_score(y_test, y_test_pred))
knn_clf = KNeighborsClassifier(n_neighbors=4, metric='cosine')
knn_clf.fit(x_train, y_train)
y_test_pred = knn_clf.predict(x_test)
print('Test accuracy of kNN', accuracy_score(y_test, y_test_pred))
```

5.2.5. Find Hyper-parameters

To find these hyper-parameter values, the GridsearchCV of scikit-learn is used. Then the best hyper-parameter values could be found by the cross-validation. Firstly, the hyper-parameter of n_neighbors is tested in the range (1,11) and the best hyper-parameter values is 10, which is a marginal value. Therefore, to find the best hyper-parameter of the model, the next range of (8,25) is tested. Fortunetely, this time the GridsearchCV get 19 as the best hyper-parameters. As 19 is in the middle of the range, it is believed that the best hyper-parameter is found. Next, we put the best hype-parameters on the test set and get the highest accuracy of 0.7945.

The comparison table in shown below.

Table 4. Comparison of KNN models

Model	Accuracy	Comment
KNN(n_neighbors=3, metric='euclidean'	0.744	Not Best
KNN(n_neighbors=4, metric='cosine'	0.7907	Not Best
KNN(n_neighbors=10, metric='cosine'	0.7942	Not Best
KNN(n_neighbors=19, metric='cosine'	0.7945	Best
KNN(n_neighbors=20, metric='cosine'	Eliminated by GridsearchCV	Not Best
KNN(n_neighbors=20, metric='euclidean'	Eliminated by GridsearchCV	Not Best

The code is shown below.

```python
from sklearn.model_selection import GridSearchCV
param_grid = [{
    'weights': ["uniform", "distance"],
    'n_neighbors': range(1, 11),
    'metric':['euclidean', 'manhattan', 'cosine']}]
knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=2)
grid_search.fit(x_train, y_train)
grid_search.best_estimator_

knn_clf = KNeighborsClassifier(metric='cosine', n_neighbors=10, weights='uniform')
```

```
knn_clf.fit(x_train, y_train)
y_train_pred = knn_clf.predict(x_train)
y_test_pred = knn_clf.predict(x_test)
print('Train accuracy of kNN', accuracy(y_train, y_train_pred))
print('Test accuracy of kNN', accuracy(y_test, y_test_pred))
param_grid = [{
    'weights': ["uniform", "distance"],
    'n_neighbors': range(8, 25),
    'metric':['euclidean', 'manhattan', 'cosine']}]
knn_clf = KNeighborsClassifier()
grid_search = GridSearchCV(knn_clf, param_grid, cv=5, verbose=2)
grid_search.fit(x_train, y_train)
grid_search.best_estimator_
knn_clf = KNeighborsClassifier(metric='cosine', n_neighbors=19, weights='uniform')
knn_clf.fit(x_train, y_train)
y_train_pred = knn_clf.predict(x_train)
y_test_pred = knn_clf.predict(x_test)
print('Train accuracy of kNN', accuracy(y_train, y_train_pred))
print('Test accuracy of kNN', accuracy(y_test, y_test_pred))
```

5.2.6. Comparison of Models

In conclusion, we find the best model, KNN(n_neighbors=19, metric='cosine'), with the best accuracy by using the GridsearchCV, which helps us to find the best hype parameter and perform cross validation. The comparison of different model is shown below.

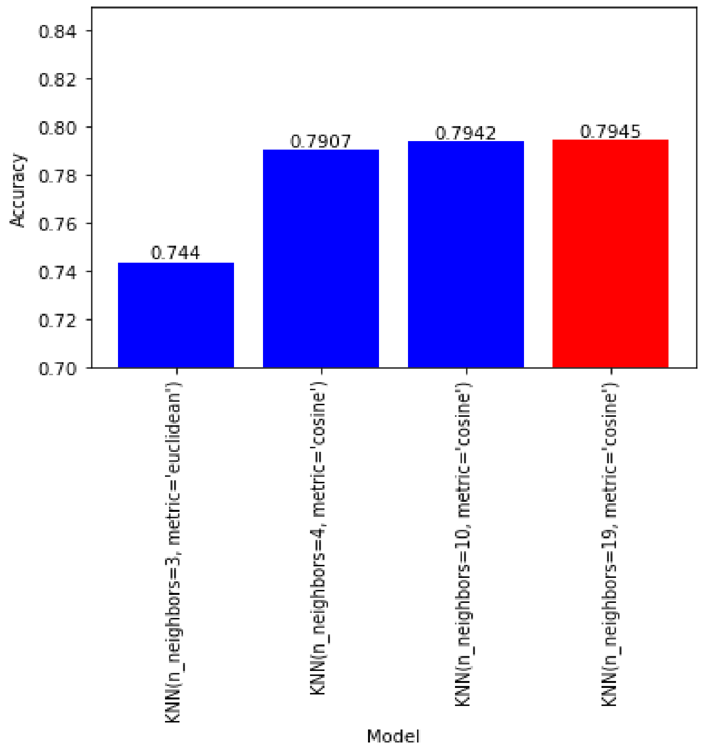


Figure 8. Comparison of KNN Models

5.3. Neural Networks

Neural networks are a family of algorithms influenced by the design and operation of the human brain. They are made to identify patterns and decide based on complicated data inputs. Due to its excellent performance in a variety of tasks, including game playing, speech and picture recognition, natural language processing, and others, neural networks have gained more and more popularity in recent years. **Figure 9.** categorizes some common neural networks into their respective branches. (Schmidhuber, 2015)

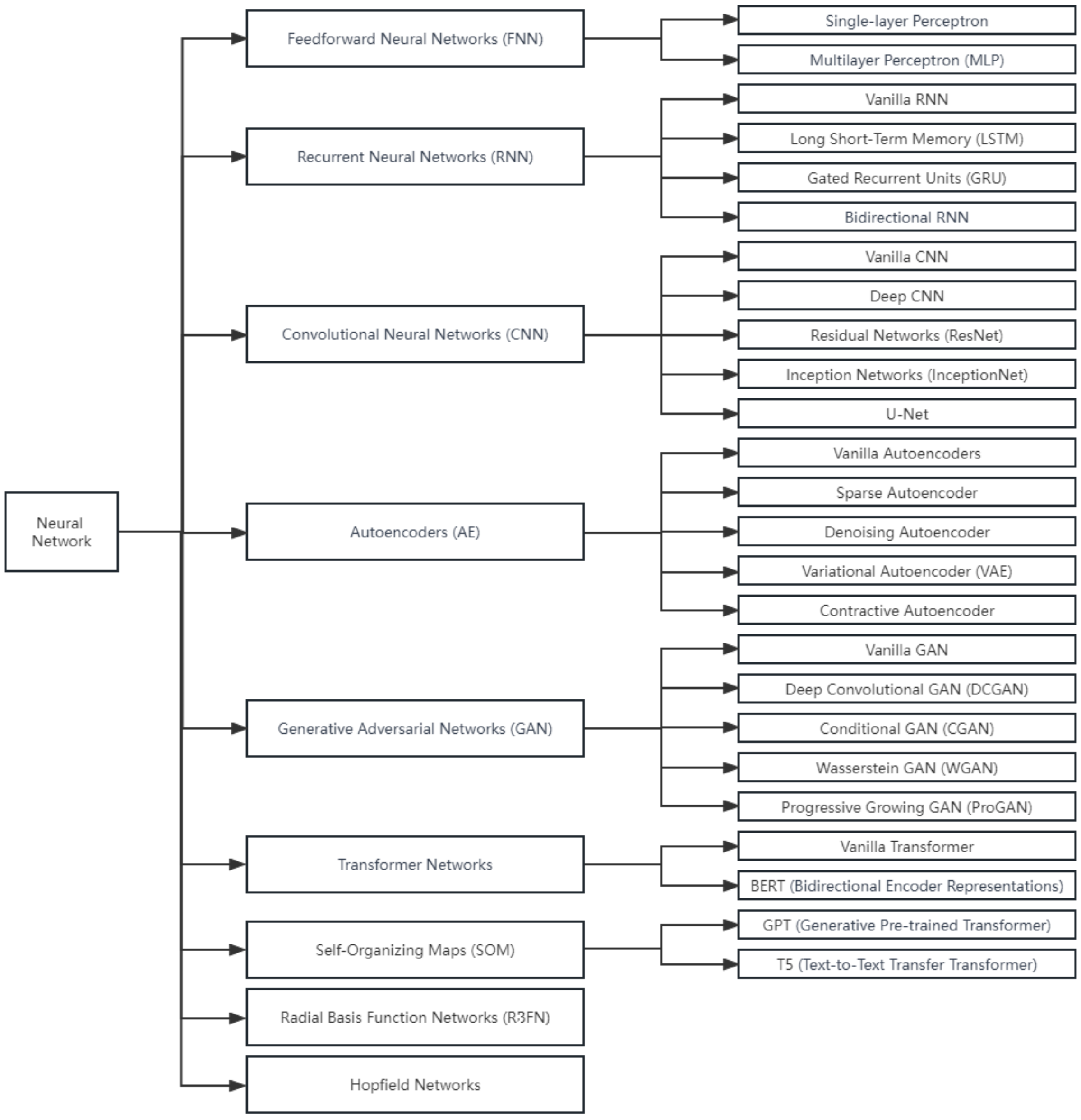

Neural Network
Feedforward Neural Networks (FNN)
Single-layer Perceptron
Multilayer Perceptron (MLP)
Recurrent Neural Networks (RNN)
Vanilla RNN
Long Short-Term Memory (LSTM)
Gated Recurrent Units (GRU)
Bidirectional RNN
Convolutional Neural Networks (CNN)
Vanilla CNN
Deep CNN
Residual Networks (ResNet)
Inception Networks (InceptionNet)
U-Net
Autoencoders (AE)
Vanilla Autoencoders
Sparse Autoencoder
Denoising Autoencoder
Variational Autoencoder (VAE)
Contractive Autoencoder
Generative Adversarial Networks (GAN)
Vanilla GAN
Deep Convolutional GAN (DCGAN)
Conditional GAN (CGAN)
Wasserstein GAN (WGAN)
Progressive Growing GAN (ProGAN)
Transformer Networks
Vanilla Transformer
BERT (Bidirectional Encoder Representations)
Self-Organizing Maps (SOM)
GPT (Generative Pre-trained Transformer)
T5 (Text-to-Text Transfer Transformer)
Radial Basis Function Networks (RBFN)
Hopfield Networks

Figure 9. Overview of Neural Network

The building blocks of neural networks are interconnected nodes or synthetic neurons arranged in layers. Input, hidden, and output layers are the three categories into which the layers can be divided. Raw data is received by the input layer, which processes it in the hidden layers before presenting the final conclusion or prediction. The weighted connections between each neuron and its neighbouring neurons, known as synapses, are altered during learning to enhance the functioning of the network.

A popular model for binary classification is the Multilayer Perceptron (MLP) neural network because it is versatile and good in learning intricate patterns in data and effectively dividing the data points into two groups. (LeCun, Bengio and Hinton, 2015)

The task is to predict whether a customer will default on their payments in the following month based on their historical data, which is sort of binary classification. It can be accomplished by a multi-layer perceptron (MLP) neural network. The following is the operating process.

5.3.1. Implementation

Firstly, the input dataset is extracted from pre-processed CSV files and split into training and testing sets. The PyTorch framework is then used to create an MLP neural network in a preliminary manner. An input layer, two hidden layers, and an output layer make up the model. The hidden layers are activated using the ReLU activation function. The code defines the network architecture, SGD as the optimizer, and CrossEntropyLoss as the loss function. The input characteristics and binary target labels are handled by a special PyTorch Dataset class (BinaryDataset), which transforms them into PyTorch tensors.

```python
class binarydataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X.values.astype(np.float32), dtype=torch.float32)
        self.y = torch.tensor(y.values.astype(np.float32), dtype=torch.long)
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
class MLP(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(MLP, self).__init__()
        self.layers = nn.ModuleList()
        next_hidden_size = input_size
        for hidden_size in hidden_sizes:
            self.layers.append(nn.Linear(next_hidden_size, hidden_size))
            next_hidden_size = hidden_size
        self.output = nn.Linear(next_hidden_size, output_size)
    def forward(self, x):
        out = x
        for layer in self.layers:
            out = layer(out)
            out = nn.functional.relu(out) # activation function
        out = self.output(out)
        return out
def train_model(params):
    model = MLP(23, params['hidden_sizes'], 2)
    model.to(device)
    optimizer = torch.optim.SGD(model.parameters(),
                                lr=params['lr'],
                                momentum=0.99,
                                weight_decay=params['weight_decay'])
    criterium = nn.CrossEntropyLoss()
    batch_size = 20
    train_loader = DataLoader(traindata, batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
    test_loader = DataLoader(testdata, batch_size=100, shuffle=False, num_workers=0, pin_memory=True)
    epochs = 5
    model.train()
    running_loss = 0
    training_loss = []
    testing_loss = []
    print(f'Testing model for parameters {params}!')
    for epoch in range(epochs):
        for i, batch in enumerate(train_loader):
            xs, ys = batch
            xs, ys = xs.to(device), ys.to(device)
            optimizer.zero_grad()
            pred_ys = model(xs)
            loss = criterium(pred_ys, ys)
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), max_norm=1)
            optimizer.step()
            running_loss += loss.item()
            if (i + 1) % 200 == 0:
                running_loss /= 200
                training_loss.append(running_loss)
                model.eval()
                test_loss = compute_loss_test(model, test_loader, criterium)
                testing_loss.append(test_loss)
                model.train()
                print('Epoch [%d/%d], Step [%d/%d], Train Loss: %.4f, Test Loss: %.4f' % (

                    epoch + 1,
                    epochs,
                    i + 1,
                    len(xs_train) // batch_size,
                    running_loss,
                    test_loss))
                running_loss = 0
    return model, train_loader, test_loader, training_loss, testing_loss
```

5.3.2. Hyperparameter Tuning

The number of hidden layers, learning rate, and weight decay are among the hyperparameters taken into account here for tuning. These hyperparameter combinations are used to conduct a grid search. The model is trained for each combination, and its performance is assessed based on the training accuracy. The highest training accuracy is used to choose the ideal collection of hyperparameters.

```
param_grid = {'hidden_sizes': [(15, 15, 2), (20, 20, 2), (25, 25, 2)],
                        'lr': [0.001, 0.0001, 0.00001],
best_accuracy = 0
for params in param_list:
    model, train_loader, test_loader, training_loss, testing_loss = train_model(params)
        model.eval()
    train_accuracy = accuracy(train_loader)
        if train_accuracy > best_accuracy:
        best_accuracy = trai
n_accuracy
        best_params = params
        print(f'Best parameter is updated to {best_params}')
```

5.3.3. Validation of Performance

The model is retrained and assessed on the test set after the optimal set of hyperparameters has been found. For the purpose of seeing how the model converges during training, the training and testing losses are shown. The training and test accuracies are reported in order to verify the model's performance. Due to the characteristics of the dataset and the fact that the activation function, optimizer, and loss function may not be the best options for this particular task, the accuracy improvement may not be significant.

```
params = best_params
model, train_loader, test_loader, training_loss, testing_loss = train_model(params)
model.eval()
train_accuracy = accuracy(train_loader)
print('Train accuracy of the MLP: {:.3f}'.format(train_accuracy))
test_accuracy = accuracy(test_loader)
print('Test accuracy of the MLP: {:.3f}'.format(test_accuracy))
```

5.3.4. Further Improvement

After implementing the multi-layer perceptron neural network using the PyTorch framework, an alternative approach is taken by using the scikit-learn package to train the model. The scikit-learn library offers a mature MLPClassifier model that allows for more extensive hyperparameter tuning, including the choice of activation functions and optimizers. GridSearchCV is used to search for the best combination of these hyperparameters. The model is trained and evaluated based on accuracy, using 5-fold cross-validation. The performance of this model is compared to the previous PyTorch implementation, and a significant improvement in accuracy is observed.

```python
scaler = StandardScaler()
xs_train = scaler.fit_transform(xs_train)
xs_test = scaler.transform(xs_test)
mlp_classifier = MLPClassifier(random_state=42)
param_grid = {
    'hidden_layer_sizes': [(15,15), (20, 20), (25, 25)], # Different hidden layer configurations
    'activation': ['relu', 'tanh'], # Activation functions
    'solver': ['adam', 'sgd'], # Optimizers
    'learning_rate_init': [0.00001, 0.0001, 0.001], # Initial learning rates
}
grid_search = GridSearchCV(mlp_classifier, param_grid, scoring='accuracy', cv=5, n_jobs=-1)
grid_search.fit(xs_train, ys_train)
print("Best hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
ys_pred = grid_search.best_estimator_.predict(xs_test)
accuracy = accuracy_score(ys_test, ys_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Based on a series of training, hyper parameter tuning and testing (as shown in **Table 5. & 6.**), the best model is sklearn MLP(hidden_layer=(20, 20), lr=0.0001, activation=tanh, solver=adam), which achieves a accuracy of 82.65%. The **Figure 10.** Illustrate the process of improvements.

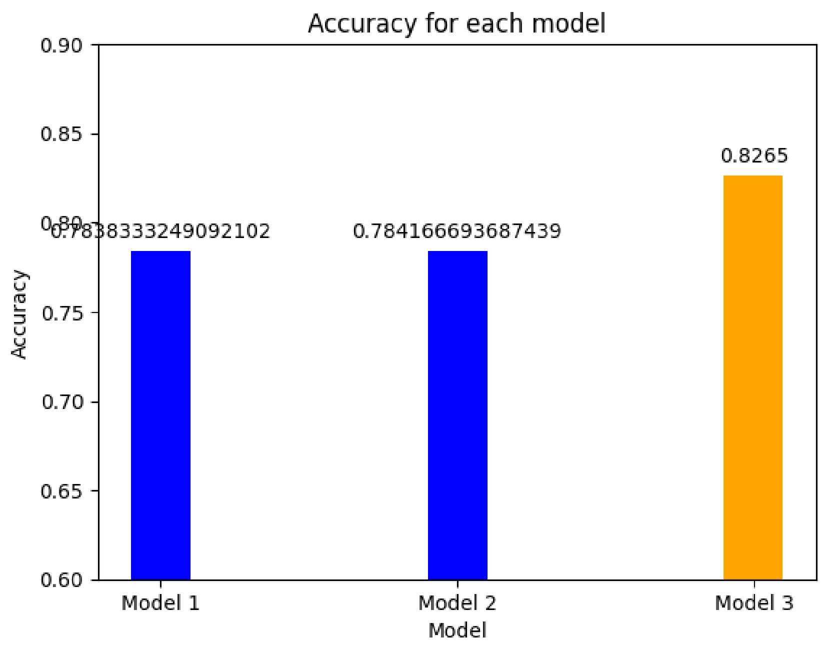


Figure 10. Accuracy of each model by the order of improvement

Table 5. Comparison of pytorch custom MLP models

Model	Accuracy	Comment
pytorch custom MLP(hidden_layer=(15, 15), lr=0.00001, weight_decay=0.001)	0.7838	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.0001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.0001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.0001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.00001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.00001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(15, 15), lr=0.00001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.0001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.0001, weight_decay=0.0001)	0.7842	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.0001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.00001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.00001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(20, 20), lr=0.00001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.0001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.0001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.0001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.00001, weight_decay=0.01)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.00001, weight_decay=0.0001)	Eliminated by ParameterGrid	Not Best
pytorch custom MLP(hidden_layer=(25, 25), lr=0.00001, weight_decay=0.00001)	Eliminated by ParameterGrid	Not Best

Table 6. Comparison of sklearn MLP models

sklearn MLP(hidden_layer=(15, 15), lr=0.00001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.00001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.00001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.00001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.0001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.0001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.0001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.0001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(15, 15), lr=0.001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.00001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.00001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.00001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.00001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.0001, activation=tanh, solver=adam)	0.8265	Best
sklearn MLP(hidden_layer=(20, 20), lr=0.0001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.0001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.0001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(20, 20), lr=0.001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.00001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.00001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.00001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.00001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.0001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.0001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.0001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.0001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.001, activation=tanh, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.001, activation=tanh, solver=sgd)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.001, activation=relu, solver=adam)	Eliminated by GridsearchCV	Not Best
sklearn MLP(hidden_layer=(25, 25), lr=0.001, activation=relu, solver=sgd)	Eliminated by GridsearchCV	Not Best

5.4. Bayesian Learning

5.4.1. Introduction of Bayesian Learning

Bayesian learning is a statistical method based on probability theory, and its theoretical basis can be traced back to the 18th century by the English mathematician Thomas Bayes, in which a preliminary form of Bayes' theorem was proposed (Stigler, 1982). This algorithm is based on the assumption that ‘the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about them’ (Aldo, 2023). In other words, by learning and analyzing the observed data, this algorithm could reason about other data and make the best decisions based on the probability of occurrence of different variables in the observed data.

5.4.2. Implementation, Hype Parameter, and Validation

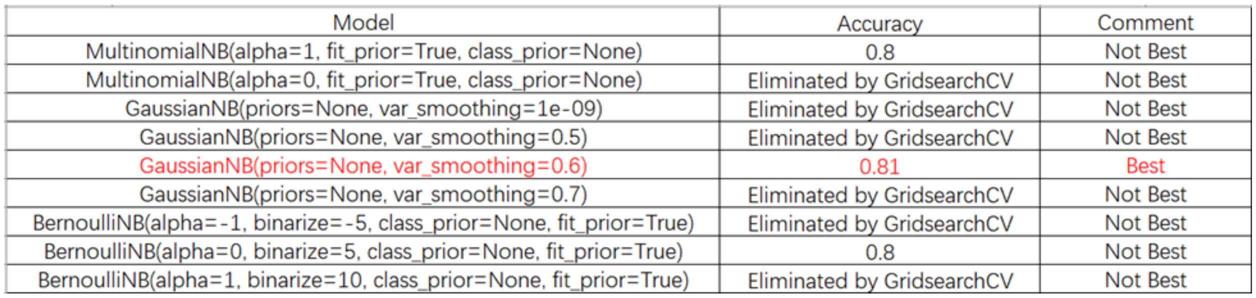
In reality, Bayesian learning is widely used and shows good performance, for example in text classification problems. In this project, this algorithm is also used to learn the characteristics of the customers and to predict defaults based on the knowledge and probability. To be specific, for this algorithm, all attributes of the dataset are preprocessed and classified before fitting the model. First of all, the data is classified for those columns that are continuous numbers to ensure that they become discrete text variables. Next, the information from all columns is combined so that each line of data is like a text. Subsequently, each line of data is converted into a matrix of token counts using the bag-of-words approach, CountVecorizer class, from Scikit-learn package. Finally, these matrixes are used to fit model and predict the result. It is noted that by calling the Scikit-learn package, all three Bayesian classifiers GaussianNB, MultinomialNB, and BernoulliNB are tested and compared to obtain the best model for this dataset. In addition, for each classifier, **GridsearchCV** from Scikit-learn package is used to find the best hyper parameters and perform cross-validation, ensuring that the best model is fitted. For example, for GaussianNB classifier, grid search is used to find the most suitable 'var_smoothing' value from 0.000000001 to 0.9. The result shows that 0.7 is the best hype parameters for 'var_smoothing' for this dataset. Some codes for this part are shown below, and all codes can be found on the GitHub.

```
param_grid = [{'var_smoothing': [1e-9, 0.5, 0.6, 0.7, 0.8, 0.9]}]
gaussion_nb = GaussianNB()
grid_search = GridSearchCV(gaussion_nb, param_grid, cv=5, verbose=2)
grid_search.fit(np.asarray(xs_train_prep.todense()), ys_train)
gaussion_nb = GaussianNB(priors=None, var_smoothing=0.7)
gaussion_nb.fit(np.asarray(xs_train_prep.todense()), ys_train)
ys_train_pred = multinomial_nb.predict(np.asarray(xs_train_prep.todense()))
accuracy_train = accuracy_score(ys_train, ys_train_pred)
ys_test_pred = gaussion_nb.predict(np.asarray(xs_test_prep.todense()))
accuracy = accuracy_score(ys_test, ys_test_pred)
```

5.4.3. Comparison of Models

After a lot of testing, hype parameter selection and cross-validation, GaussianNB(priors=None, var_smoothing=0.7) is considered as the best Bayesian learning model for this dataset. The comparison of different models is shown in **Table 7 and Figure 11**. Using this model for prediction, the accuracy for test set is 81%.

Table 7. Comparison of Bayesian learning models



Model	Accuracy	Comment
MultinomialNB(alpha=1, fit_prior=True, class_prior=None)	0.8	Not Best
MultinomialNB(alpha=0, fit_prior=True, class_prior=None)	Eliminated by GridsearchCV	Not Best
GaussianNB(priors=None, var_smoothing=1e-09)	Eliminated by GridsearchCV	Not Best
GaussianNB(priors=None, var_smoothing=0.5)	Eliminated by GridsearchCV	Not Best
GaussianNB(priors=None, var_smoothing=0.6)	0.81	Best
GaussianNB(priors=None, var_smoothing=0.7)	Eliminated by GridsearchCV	Not Best
BernoulliNB(alpha=-1, binarize=-5, class_prior=None, fit_prior=True)	Eliminated by GridsearchCV	Not Best
BernoulliNB(alpha=0, binarize=5, class_prior=None, fit_prior=True)	0.8	Not Best
BernoulliNB(alpha=1, binarize=10, class_prior=None, fit_prior=True)	Eliminated by GridsearchCV	Not Best

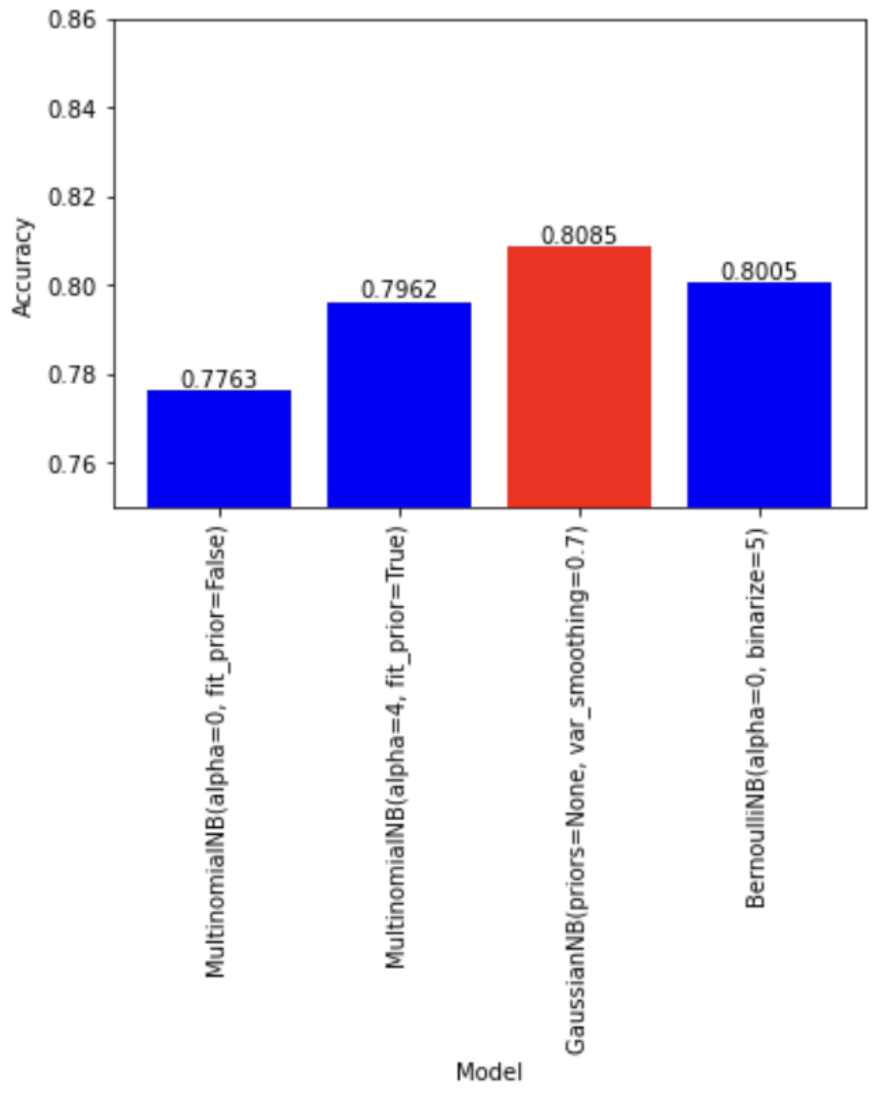


Figure 11. Comparison of Bayesian learning models

5.5. Model Ensemble

Ensemble methods are a popular approach in machine learning that aims to improve a model's performance by combining the predictions of multiple base models. This can lead to a more accurate and robust model, as it helps to overcome the weaknesses of individual models. This part explores three ensemble methods, Bagging, AdaBoost, and Voting Classifier, to solve the credit card default prediction task using previously trained models.

5.5.1. Task Description

The task is predicting credit card defaults based on various features, such as credit limit, repayment status, and demographic information. The "Default of Credit Card Clients" dataset from the UCI repository is used.

5.5.2. Data Preprocessing

Before applying ensemble methods, we perform the following preprocessing steps:

Replace invalid values in the EDUCATION column with a new category '4'.

Replace negative and zero values in the PAY_X columns with '-1' to indicate no delay in payment.

Drop the 'ID' column as it is not useful for prediction.

Create dummy variables for the 'SEX' and 'EDUCATION' columns to represent categorical data in a suitable format.

5.5.3. Ensemble Models

First, four pervious models are Implemed by pickle as follow:

```python
with open("DecisionTree.sav", "rb") as f:
    decision_tree = pickle.load(f)
with open("Instance_based.sav", "rb") as f:
    instance_based_learning  = pickle.load(f)
with open("gaussion_nb.sav", "rb") as f:
    bayesian_learning  = pickle.load(f)
with open("MLP_sklearn.sav", "rb") as f:
    neural_network  = pickle.load(f)
```

Three ensemble methods are implemented using the Decision Tree, K-Nearest Neighbors, Gaussian Naïve Bayes, and Multi-Layer Perceptron classifiers that were trained previously:

Bagging Classifier: Bagging ensemble method with the Decision Tree classifier as the base estimator is used. This method trains multiple instances of the base estimator on random subsets of the dataset and combines their predictions by majority voting. We fit the classifier on the training data and report its accuracy on the test data. The Bagging accuracy before parameter adjustment is 0.784.

```python
bagging_classifier = BaggingClassifier(
    estimator=decision_tree,
    max_samples=2,
    max_features=1.0
)
bagging_classifier.fit(X_train, y_train)
bagging_y_pred = bagging_classifier.predict(X_test)
```

```
bagging_accuracy = accuracy_score(y_test, bagging_y_pred)
print(f'Bagging accuracy: {bagging_accuracy:.4f}')
```

AdaBoost Classifier: AdaBoost ensemble method with the Decision Tree classifier the base estimator is used as. This method trains multiple instances of the base estimator sequentially, with each instance focusing on the samples that were misclassified by the previous one. The AdaBoost accuracy before parameter adjustment is 0.8058.

```
adaboost_classifier = AdaBoostClassifier(
    estimator=decision_tree,
    n_estimators=100
)
adaboost_classifier.fit(X_train, y_train)
adaboost_y_pred = adaboost_classifier.predict(X_test)
adaboost_accuracy = accuracy_score(y_test, adaboost_y_pred)
print(f'AdaBoost accuracy: {adaboost_accuracy:.4f}')
```

Voting Classifier: Voting Classifier ensemble method with the Decision Tree, K-Nearest Neighbors, Gaussian Naïve Bayes, and Multi-Layer Perceptron classifiers is used. This method combines the predictions of multiple classifiers by majority voting or weighted voting. The Voting accuracy before parameter adjustment is 0.7838.

```
voting_classifier = VotingClassifier(
    estimators=[
        ('decision_tree', decision_tree),
        ('instance_based_learning', instance_based_learning),
        ('bayesian_learning', bayesian_learning),
        ('neural_network', neural_network)
    ],
    voting='hard'
)
voting_classifier.fit(X_train, y_train)
y_pred = voting_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Stacking Classifier accuracy (before GridSearchCV): {accuracy:.4f}")
```

5.5.4. Hyperparameter and Validation

For each ensemble method, hyperparameter tuning is performed using GridSearchCV with 5-fold cross-validation. A range of hyperparameter values are searched for each method:

Bagging Classifier: 'n_estimators', 'max_samples', and 'max_features'

AdaBoost Classifier: 'n_estimators' and 'learning_rate'

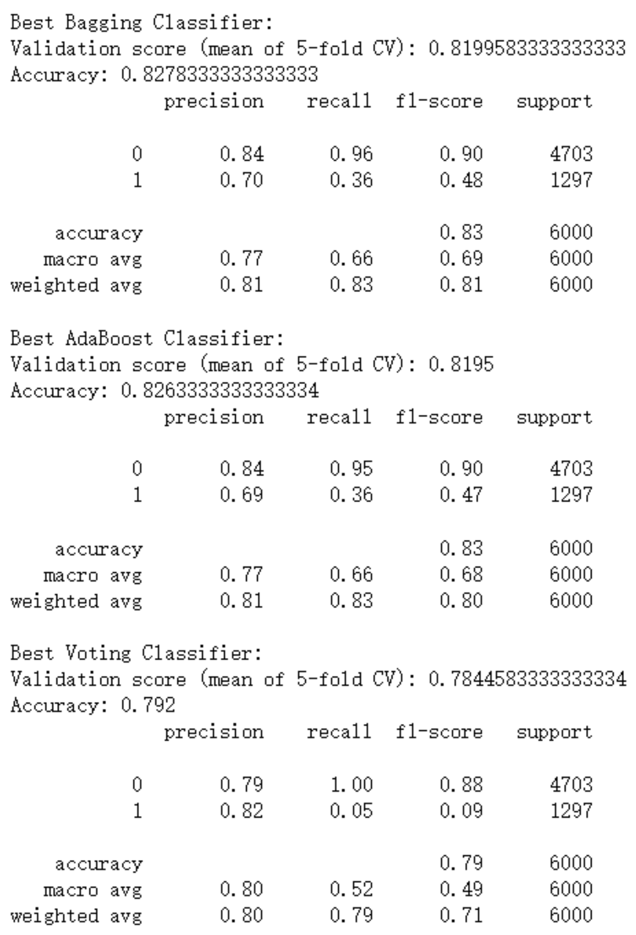
Voting Classifier: 'voting' (hard or soft)

After hyperparameter adjusting, Bagging's accuracy is up to 0.8283, AdaBoost's accuracy is up to 0.8263, and Voting's accuracy is up to 0.792. GridSearchCV is fitted through different combinations of hyperparameters, and cross-validation is performed internally. The best hyperparameters and their corresponding validation scores (mean of 5-fold CV) for each method are shown.

```python
bagging_params = {
    'n_estimators': [10, 50, 100, 200],
    'max_samples': [0.5, 1.0, 2],
    'max_features': [0.7, 0.9, 2]
}
bagging_grid = GridSearchCV(bagging_classifier, bagging_params, cv=5, n_jobs=-1)
bagging_grid.fit(X_train, y_train)
best_bagging = bagging_grid.best_estimator_
best_bagging_accuracy = accuracy_score(y_test, best_bagging.predict(X_test))
print(f'Best Bagging accuracy: {best_bagging_accuracy:.4f}')
# AdaBoost
adaboost_params = {
    'n_estimators': [2, 10, 20],
    'learning_rate': [0.001, 0.005, 0.01, 0.1, 0.5, 0.8, 1.0]
}
adaboost_grid = GridSearchCV(adaboost_classifier, adaboost_params, cv=5, n_jobs=-1)
adaboost_grid.fit(X_train, y_train)
best_adaboost = adaboost_grid.best_estimator_
best_adaboost_accuracy = accuracy_score(y_test, best_adaboost.predict(X_test))
print(f'Best AdaBoost accuracy: {best_adaboost_accuracy:.4f}')
# Stacking
voting_params = {
    'voting': ['hard', 'soft']
}
voting_grid = GridSearchCV(voting_classifier, voting_params, cv=5, n_jobs=-1)
voting_grid.fit(X_train, y_train)
best_voting = voting_grid.best_estimator_
best_votingg_accuracy = accuracy_score(y_test, best_voting.predict(X_test))
print(f'Best Stacking accuracy: {best_votingg_accuracy:.4f}')
```

5.5.5. Performance Evaluation

The performance of each ensemble method can be seen by using accuracy and classification report, which includes precision, recall, and F1-score for each class. The following graph shows the comparation of the performance of the best models found through hyperparameter tuning with the initial models before tuning.



```
Best Bagging Classifier:
Validation score (mean of 5-fold CV): 0.8199583333333333
Accuracy: 0.8278333333333333
              precision    recall  f1-score   support

           0       0.84      0.96      0.90      4703
           1       0.70      0.36      0.48      1297

    accuracy                           0.83      6000
   macro avg       0.77      0.66      0.69      6000
weighted avg       0.81      0.83      0.81      6000

Best AdaBoost Classifier:
Validation score (mean of 5-fold CV): 0.8195
Accuracy: 0.8263333333333334
              precision    recall  f1-score   support

           0       0.84      0.95      0.90      4703
           1       0.69      0.36      0.47      1297

    accuracy                           0.83      6000
   macro avg       0.77      0.66      0.68      6000
weighted avg       0.81      0.83      0.80      6000

Best Voting Classifier:
Validation score (mean of 5-fold CV): 0.7844583333333334
Accuracy: 0.792
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      4703
           1       0.82      0.05      0.09      1297

    accuracy                           0.79      6000
   macro avg       0.80      0.52      0.49      6000
weighted avg       0.80      0.79      0.71      6000
```

Figure 12. Accuracy and classification report of each ensemble method

Based on the above results, it can be observed that the performance of three integration methods (Bagging, AdaBoost, and Voting) on the test set.

```python
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
    print(classification_report(y_test, y_pred))
def print_validation_score(grid_search):
    print("Validation score (mean of 5-fold CV):", grid_search.best_score_)
print("Best Bagging Classifier:")
print_validation_score(bagging_grid)
evaluate_model(best_bagging, X_test, y_test)
print("Best AdaBoost Classifier:")
print_validation_score(adaboost_grid)
evaluate_model(best_adaboost, X_test, y_test)
print("Best Voting Classifier:")
print_validation_score(voting_grid)
evaluate_model(best_voting, X_test, y_test)
```

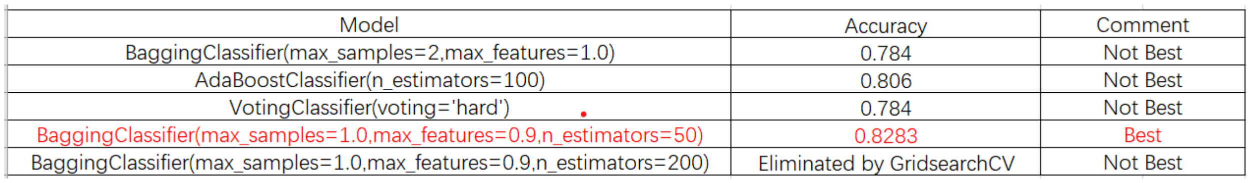
From an accuracy perspective, Bagging and AdaBoost performed closely, while voting performed slightly worse. However, accuracy does not fully reflect the performance of the model, especially when categories are unbalanced. Therefore, other evaluation indicators precision, recall, and F1-score should be considered.

Observing the results of precision, recall, and F1-score, Bagging and AdaBoost perform similarly in prediction category 0 (non-default), while AdaBoost has slightly higher precision than Bagging in prediction category 1 (default), but recall is lower. Overall, Bagging and AdaBoost have performed well in the F1 core.

At the same time, Voting's performance in predicting category 1 is poor, especially in recall, indicating its limited ability to identify defaulting customers. This may be due to the impact of the poor performance of some base learners on the Voting method.

After comprehensive consideration of various evaluation indicators, Bagging and AdaBoost performed well in this task, both better than the Voting method. Although there is no significant difference between Bagging and AdaBoost in terms of accuracy and F1-score, since Bagging's verification score (the average of 5-fold cross-validation) is slightly higher than AdaBoost, in conclusion, the Bagging method performs best in this task. The comparison of different Bagging models is shown in the following table.

Table 8. Comparison of models ensemble methods



Model	Accuracy	Comment
BaggingClassifier(max_samples=2,max_features=1.0)	0.784	Not Best
AdaBoostClassifier(n_estimators=100)	0.806	Not Best
VotingClassifier(voting='hard')	0.784	Not Best
BaggingClassifier(max_samples=1.0,max_features=0.9,n_estimators=50)	0.8283	Best
BaggingClassifier(max_samples=1.0,max_features=0.9,n_estimators=200)	Eliminated by GridsearchCV	Not Best

6. Conclusion

In conclusion, the task of predicting credit card default is completed well by using various machine learning models and ensemble methods. To be specific, a total of five kinds of models (Decision Tree, K-Nearest Neighbor, Bayesion Learning, Model Ensembles, and Multilayer Perceptron Neural Network) are chosen to train and test for clients' credit card default. In addition, during the modelling, GridSearchCV is used to assist in the selection of hype parameters and the cross validation for all five models, which efficiently improves the accuracy of prediction. By comparing with their accuracies, it could be concluded that the Model Ensembles are the best models for this dataset, with the accuracy of 82.83%. Meanwhile, both Decision Tree and Multilayer Perceptron Neural Network show good performance as well, demonstrated by the accuracy of 82.6% and 82.65% respectively. The remaining two models are slightly weaker than the others, with Bayesian Learning's accuracy at 81% and KNN's accuracy at 79.45%. The comparison of performance among different models is shown below.

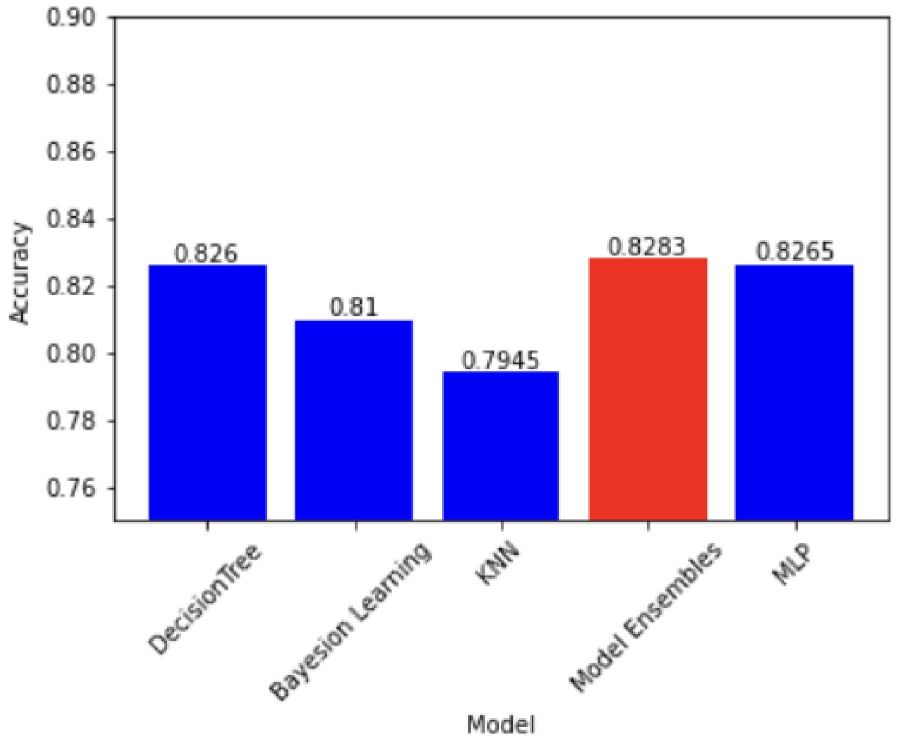


Figure 13. Accuracy comparison of all kinds of models

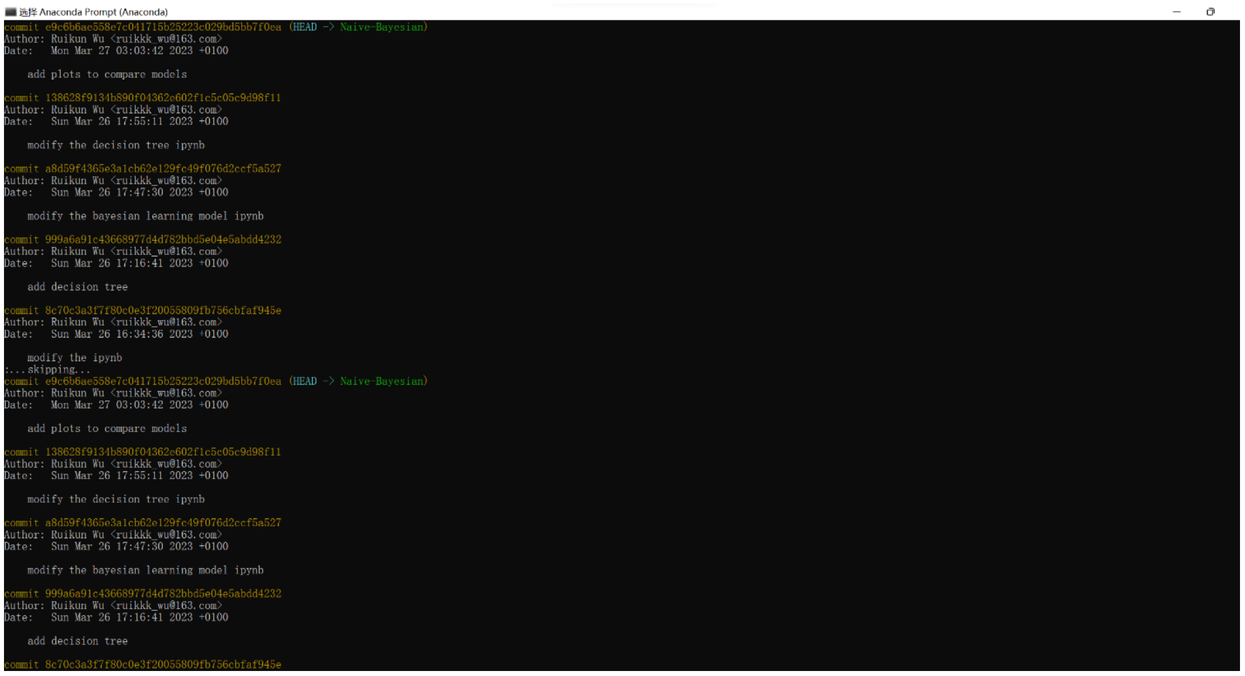
There are some reasonable explanations for the different prediction performance of different models. In terms of Model Ensembles, it combines the strengths of various models to implement predictions, so it is more likely to have better performance. As for Multilayer Perceptron Neural Network, this model generally has good ability to handle complex patterns and relationships. At the same time, the dataset credit card default has a large number of features. Thus, MLP show excellent prediction accuracy. When it comes to KNN model, as mentioned above, the number of features of this dataset is too much, and the distances between majority points are similar, which makes KNN hard to predict.

Admittedly, this project has some limitations. For example, the accuracy of all models is not very high when compared to excellent machine learning prediction models. The reason could be the low quality of the dataset especially the column 12 to 23. These attributes are not helpful for classification since they show the same pattern regardless of whether the classification is 0 or 1 (e.g. next month's payment amount is less than the previous month's bill amount), which significantly affects the accuracy of 1 classification, as there is more data in the dataset with a 0 classification. In the future, after having a greater knowledge base, these shortcomings could be optimized by using other datasets and implementing feature engineering for dataset.

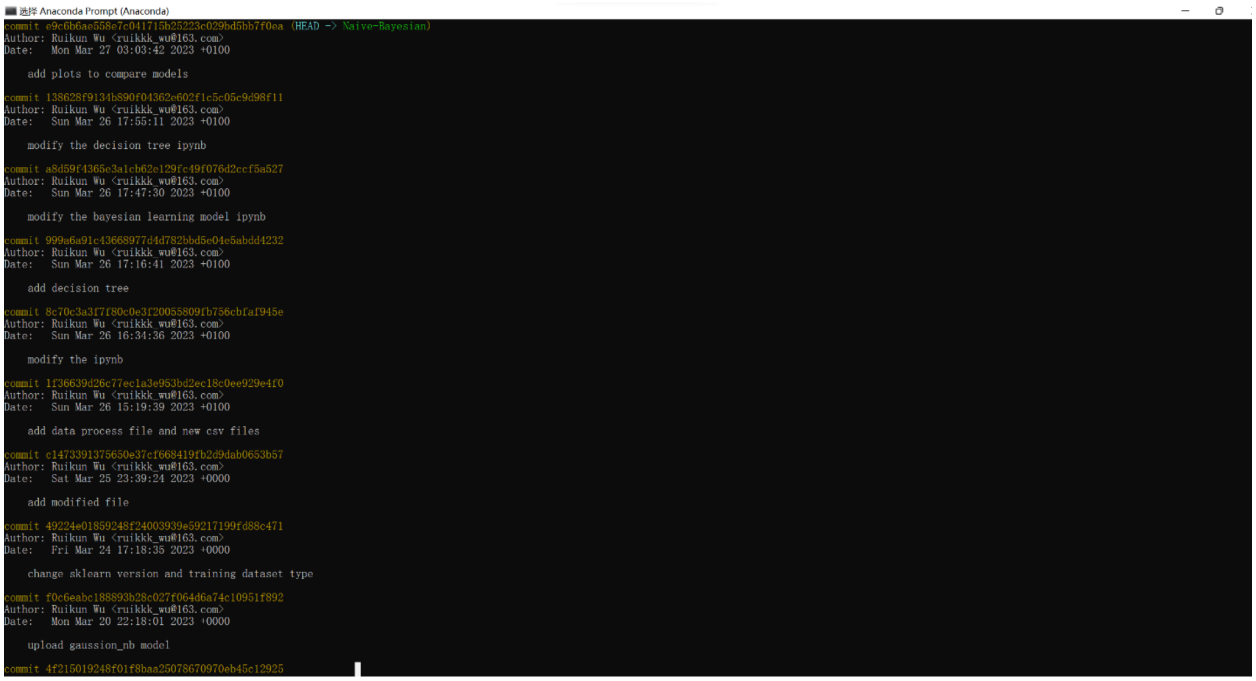
However, this project still provides valuable suggestions for future studies. Specifically, in terms of model optimisation and parameter selection, it is valuable to use GridsearchCV to find the best hype parameters and perform cross-validation. In terms of the model selection for credit card dataset, the Model Ensembles could be a good choice.

7. Git Log

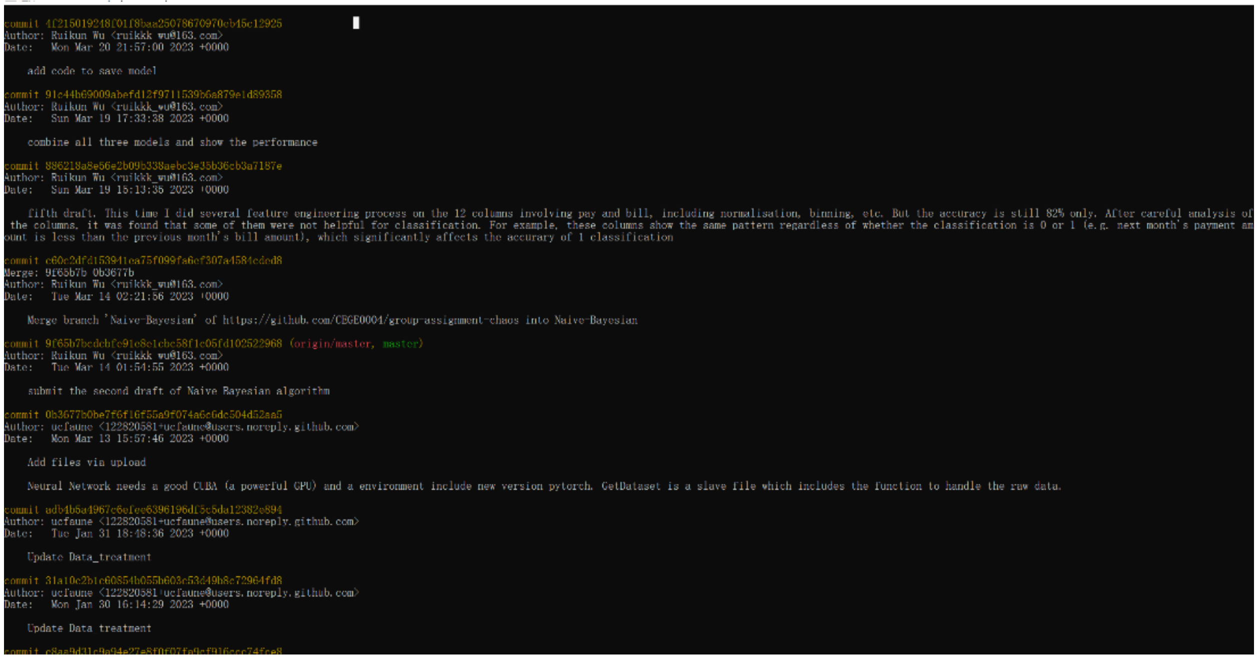
A total of **75** commits are submitted on the GitHub throughout the whole project. Some git log histories are shown below.



```
commit e9c6b6ae558e7c041715b25223c029bd5bb7f0ea (HEAD -> Naive-Bayesian)
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Mon Mar 27 03:03:42 2023 +0100

    add plots to compare models

commit 138628f9134b890f04362e602f1c5c05c9d98f11
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:55:11 2023 +0100

    modify the decision tree ipynb

commit a8d59f4365e3a1cb62e129fc49f076d2ccf5a527
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:47:30 2023 +0100

    modify the bayesian learning model ipynb

commit 999a6a91c43668977d4d782bbd5e04e5abdd4232
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:16:41 2023 +0100

    add decision tree

commit 8c70c3a3f7f80c0e3f20055809fb756cbfaf945e
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 16:34:36 2023 +0100

    modify the ipynb
:...skipping...
commit e9c6b6ae558e7c041715b25223c029bd5bb7f0ea (HEAD -> Naive-Bayesian)
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Mon Mar 27 03:03:42 2023 +0100

    add plots to compare models

commit 138628f9134b890f04362e602f1c5c05c9d98f11
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:55:11 2023 +0100

    modify the decision tree ipynb

commit a8d59f4365e3a1cb62e129fc49f076d2ccf5a527
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:47:30 2023 +0100

    modify the bayesian learning model ipynb

commit 999a6a91c43668977d4d782bbd5e04e5abdd4232
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:16:41 2023 +0100

    add decision tree

commit 8c70c3a3f7f80c0e3f20055809fb756cbfaf945e
```



```
commit e9c6b6ae558e7c041715b25223c029bd5bb7f0ea (HEAD -> Naive-Bayesian)
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Mon Mar 27 03:03:42 2023 +0100

    add plots to compare models

commit 138628f9134b890f04362e602f1c5c05c9d98f11
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:55:11 2023 +0100

    modify the decision tree ipynb

commit a8d59f4365e3a1cb62e129fc49f076d2ccf5a527
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:47:30 2023 +0100

    modify the bayesian learning model ipynb

commit 999a6a91c43668977d4d782bbd5e04e5abdd4232
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 17:16:41 2023 +0100

    add decision tree

commit 8c70c3a3f7f80c0e3f20055809fb756cbfaf945e
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 16:34:36 2023 +0100

    modify the ipynb

commit 1f36639d26c77ec1a3e953bd2ec18c0ee929e4f0
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 26 15:19:39 2023 +0100

    add data process file and new csv files

commit c1473391375650e37cf668419fb2d9dab0653b57
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sat Mar 25 23:39:24 2023 +0000

    add modified file

commit 49224e01859248f24003939e59217199fd88c471
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Fri Mar 24 17:18:35 2023 +0000

    change sklearn version and training dataset type

commit f0c6eabc188893b28c027f064d6a74c10951f892
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Mon Mar 20 22:18:01 2023 +0000

    upload gaussion_nb model

commit 4f215019248f01f8baa25078670970eb45c12925
```



```
commit 4f215019248f01f8baa25078670970eb45c12925
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Mon Mar 20 21:57:00 2023 +0000

    add code to save model

commit 91c44b69009abefd12f9711539b6a879e1d89358
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 19 17:33:38 2023 +0000

    combine all three models and show the performance

commit 886218a8e56e2b09b338aebc3e35b36cb3a7187e
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Sun Mar 19 15:13:35 2023 +0000

    fifth draft. This time I did several feature engineering process on the 12 columns involving pay and bill, including normalisation, binning, etc. But the accuracy is still 82% only. After careful analysis of
 the columns, it was found that some of them were not helpful for classification. For example, these columns show the same pattern regardless of whether the classification is 0 or 1 (e.g. next month's payment am
ount is less than the previous month's bill amount), which significantly affects the accurary of 1 classification

commit c60c2dfd153941ea75f099fa6ef307a4584cded8
Merge: 9f65b7b 0b3677b
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Tue Mar 14 02:21:56 2023 +0000

    Merge branch 'Naive-Bayesian' of https://github.com/CEGE0004/group-assignment-chaos into Naive-Bayesian

commit 9f65b7bcdcbfc91c8c1cbc58f1c05fd102522968 (origin/master, master)
Author: Ruikun Wu <ruikkk_wu@163.com>
Date:   Tue Mar 14 01:54:55 2023 +0000

    submit the second draft of Naive Bayesian algorithm

commit 0b3677b0be7f6f16f55a9f074a6c6dc504d52aa5
Author: ucfaune <122820581+ucfaune@users.noreply.github.com>
Date:   Mon Mar 13 15:57:46 2023 +0000

    Add files via upload

    Neural Network needs a good CUDA (a powerful GPU) and a environment include new version pytorch. GetDataset is a slave file which includes the function to handle the raw data.

commit adb4b5a4967c6efee6396196df5c5da12382e894
Author: ucfaune <122820581+ucfaune@users.noreply.github.com>
Date:   Tue Jan 31 18:48:36 2023 +0000

    Update Data_treatment

commit 31a10c2b1c60854b055b603c53d49b8c72964fd8
Author: ucfaune <122820581+ucfaune@users.noreply.github.com>
Date:   Mon Jan 30 16:14:29 2023 +0000

    Update Data treatment

commit c8aa9d31c9a94e27e8f0f07fa9cf916ccc74fce8
```

8. Reference

Aldo, L. (2023) *Course: CEGE0004: Machine Learning for Data Science (22/23)*. Available at: https://moodle.ucl.ac.uk/course/view.php?id=33287 (Accessed: 27 March 2023).

Bishop (2007) ‘Pattern Recognition and Machine Learning’, *Journal of Electronic Imaging*, 16(4), p. 049901. Available at: https://doi.org/10.1117/1.2819119.

Gui, L. (2019) *Application of Machine Learning Algorithms in Predicting Credit Card Default Payment*. M.S. Available at: https://www.proquest.com/docview/2248659802/abstract/A46F94332B984787PQ/1 (Accessed: 27 March 2023).

‘Instance-based learning’ (2020) *GeeksforGeeks*, 29 August. Available at: https://www.geeksforgeeks.org/instance-based-learning/ (Accessed: 27 March 2023).

Juan, M.J.S.-S. (2011) ‘A sociological inquiry into credit card delinquency in the Philippines’, 11, pp. 21–60.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) ‘Deep learning’, *Nature*, 521(7553), pp. 436–444. Available at: https://doi.org/10.1038/nature14539.

Schmidhuber, J. (2015) ‘Deep Learning in Neural Networks: An Overview’, *Neural Networks*, 61, pp. 85–117. Available at: https://doi.org/10.1016/j.neunet.2014.09.003.

Stigler, S.M. (1982) ‘Thomas Bayes’s Bayesian Inference’, *Journal of the Royal Statistical Society. Series A (General)*, 145(2), pp. 250–258. Available at: https://doi.org/10.2307/2981538.

UCI Machine Learning Repository: default of credit card clients Data Set (2016). Available at: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients (Accessed: 27 March 2023).

Yeh, I.-C. and Lien, C. (2009) ‘The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients’, *Expert Systems with Applications*, 36(2, Part 1), pp. 2473–2480. Available at: https://doi.org/10.1016/j.eswa.2007.12.020.