



Predictive Modeling for Loan Approval

A Data-Driven Approach

Table of Contents

Introduction	2
Background of Study	2
Objective	2
Exploratory Data Analysis (EDA).....	3
Data Preprocessing	13
Handling Missing Values:	13
Handling Duplicated Data:	13
Label Encoding:	14
Data Splitting.....	14
Synthetic Minority Over-sampling Technique (SMOTE)	15
Algorithm Selection	16
Model Implementation.....	20
Model Comparison	24
Results and Interpretation	25
Conclusion	33
References.....	34

Introduction

Background of Study

In today's financial landscape, the efficient processing of loan applications is crucial for both lenders and borrowers. With the advent of data mining techniques, lenders can leverage vast amounts of data to make informed decisions regarding loan approvals. This report aims to dive into the application of data mining methodologies to determine whether loan applications for 20 new applicants should be approved or rejected.

Objective

The primary purpose of this report is to utilize data mining techniques to assess the creditworthiness of the 20 new loan applicants. By analysing various factors such as credit score, income, debt-to-income ratio, and other relevant parameters, we aim to develop predictive models that can help in decision-making regarding loan approvals. This approach not only makes the loan application process more efficient but also reduces the risks of defaulting loans.

Moreover, the utilization of data mining techniques allows for a more objective evaluation of loan applications, minimizing the influence of subjective biases. Through this analysis, we aim to offer insights that help financial institutions make smart lending decisions while ensuring fairness for all applicants.

In the following sections, we'll discuss the methods used, how data was prepared, the features chosen, building the model, assessing its effectiveness, and deciding whether to approve or reject the new loan applications.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a fundamental step in the data analysis process, aimed at understanding the underlying structure, patterns, and relationships within a dataset. It involves summarizing the main characteristics of the data, identifying key features, detecting anomalies, and visualizing the data to gain insights that inform subsequent analysis and modelling decisions. This report provides an overview of EDA techniques and their significance in uncovering valuable information from datasets.

In this academic report, we undertake the exploration of the first five rows of the dataset titled "BankLoanApproval.csv". This process involves examining both categorical and numeric variables to understand the dataset comprehensively. Since we observed that the categorical variable "LoanID" doesn't give any insight to the dataset, we decided to remove it. Here's the categorical variables and numerical variables in the data.

Categorical Variables:

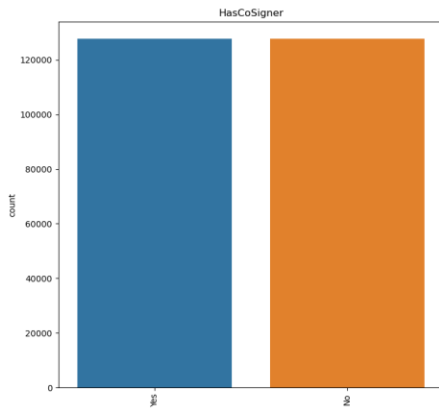
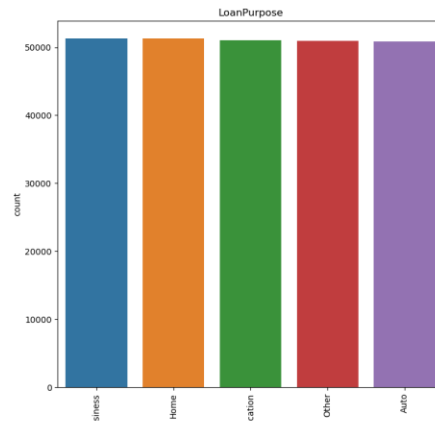
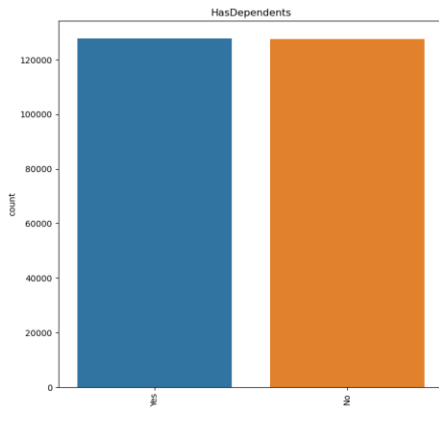
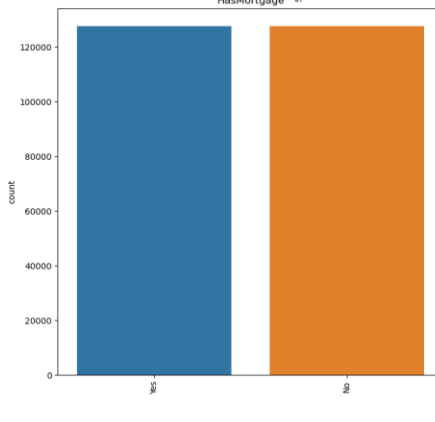
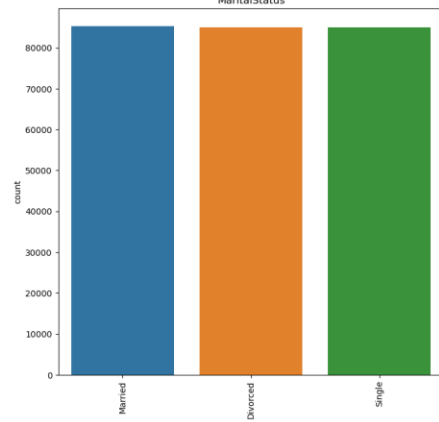
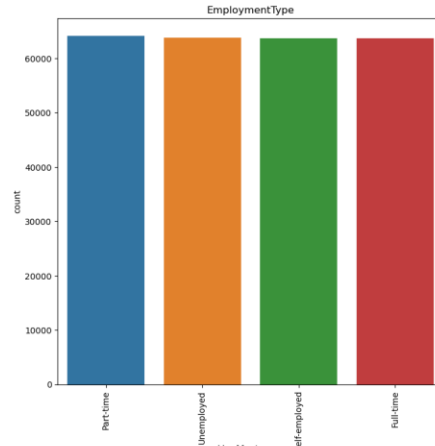
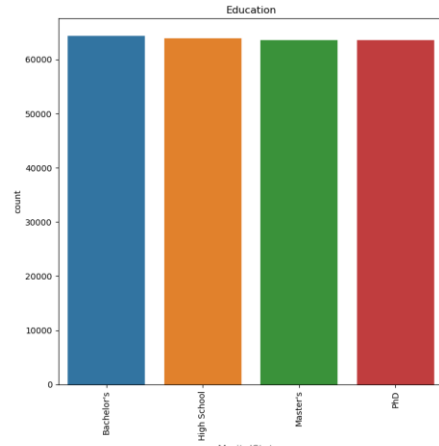
Variable name	Count of unique values	Unique values
Education	4	-Bachelor's -High School -Master's -PhD
EmploymentType	4	-Part-time -Unemployed -Self-employed -Full-time
MaritalStatus	3	-Married -Divorced -Single

HasMortgage	2	-Yes -No
HasDependents	2	-Yes -No
LoanPurpose	5	-Business -Home -Education -Other -Auto
HasCoSigner	2	-Yes -No

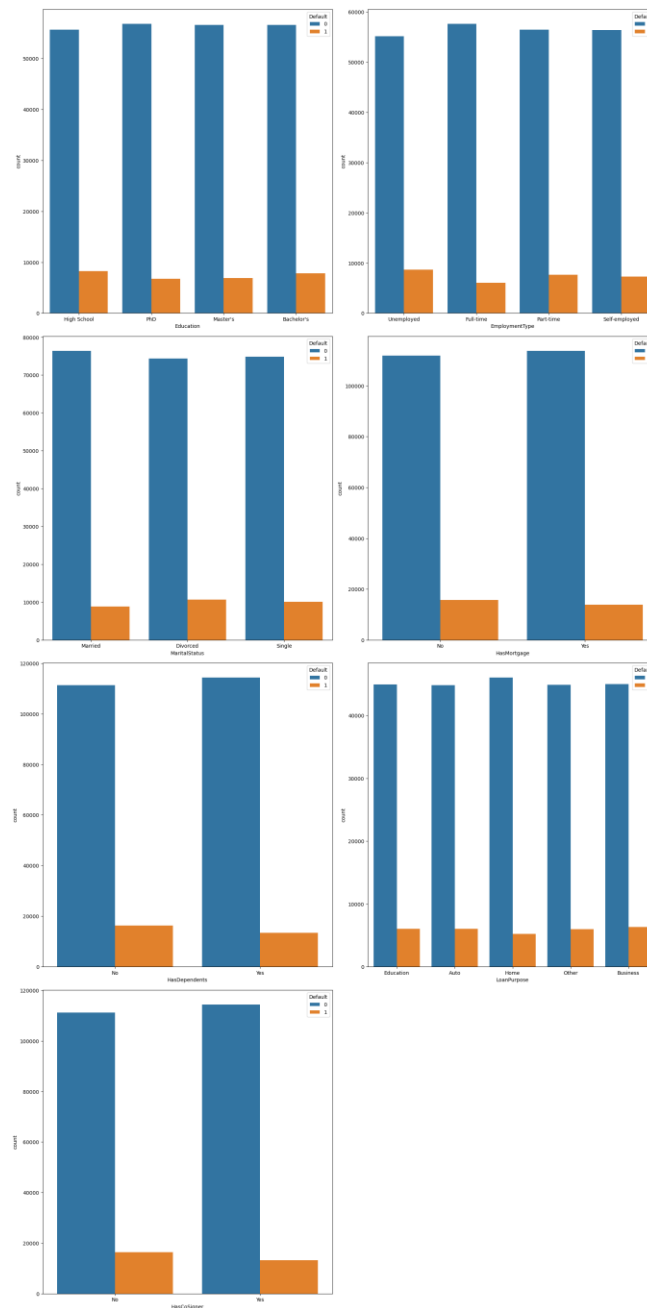
Numerical Variables:

Variable name	Min - Max
Age	18 - 69
Income	15,000 – 149,999
LoanAmount	5,000 – 249,999
CreditScore	300 – 849
MonthsEmployed	0 – 119
NumCreditLines	1 – 4
InterestRate	2 – 25
LoanTerm	12 - 60
DTIRatio	0.1 - 0.9

After that, we present the visualization of categorical data through bar plots. The dataset contains several categorical variables, including 'Education', 'EmploymentType', 'MaritalStatus', 'HasMortgage', 'HasDependents', 'LoanPurpose', 'HasCoSigner', and 'Default'. Each bar plot illustrates the distribution of observations within a categorical variable.



From the graphs presented above, we observed that no any values are dominating the variables. It means that there isn't a single category within each categorical variable that significantly outnumbers the others. In other words, the distribution of observations across the categories is relatively balanced. This can be beneficial in our analysis as it indicates diversity within the dataset and prevents bias towards a particular category. However, the bar plots above doesn't give us insight into the relationship between categorical variables and our targeted outcomes, "Default".



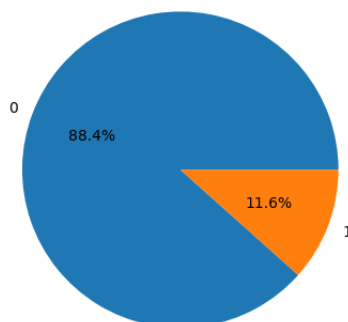
By plotting the categorical variables on count plots with the hue set to "Default" allows for a visual comparison of the distribution of each category within the variables based on the default status.

In this context:

- **Default = 0:** Represents instances where the loan application was approved or did not result in default.
- **Default = 1:** Represents instances where the loan application resulted in default or was not approved.

From the graph above, we observed that "Default = 0" significantly outnumbers "Default = 1" in every graph. It implies that most loan applications have been approved or did not lead to default, while a minority have resulted in default. From there, it highlights the imbalance in the dataset's target variable and may warrant further investigation into the factors contributing to loan defaults. Additionally, class imbalance can affect the performance of machine learning models and may require strategies such as resampling techniques or adjusting class weights to address.

We might not see the pattern of class imbalances from the graph above. Therefore, we created a pie chart for "Default" variable to provides a clearer picture of the class imbalance between "Default = 0" and "Default = 1".

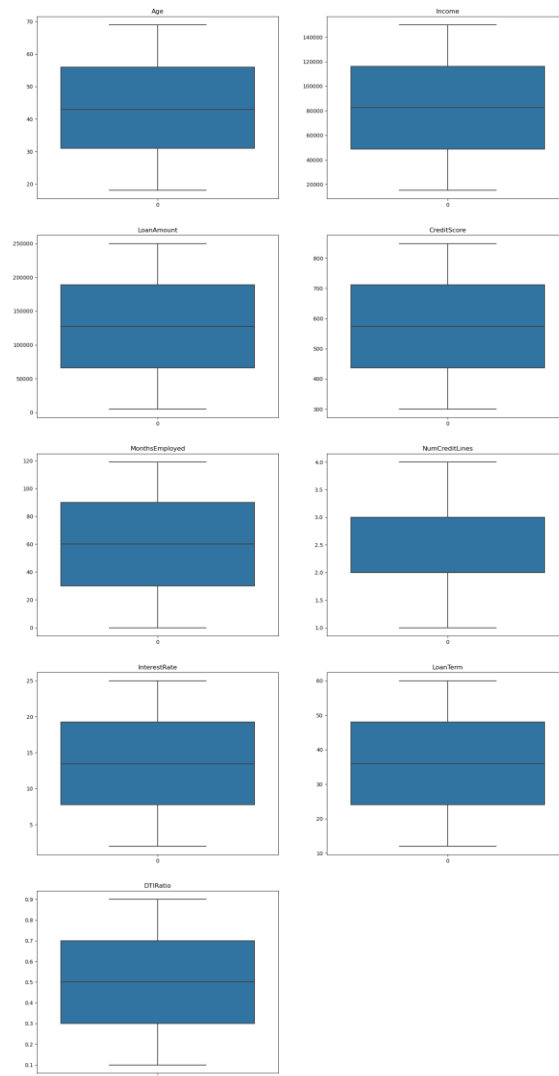


From the pie chart above, we clearly observed the distribution of the “Default” classes:

- "Default = 0" comprises 88.4% of the dataset.
- "Default = 1" comprises 11.6% of the dataset.

This visualization highlights the challenge of class imbalance in the dataset, with a vast majority of instances being non-default cases.

After analysing the categorical variables and dependent variables, we then dive into the analysis of numerical variables. First, we observe the pattern of numerical variables with boxplots.



From the boxplots above, we observed that all numerical variables have different value ranges but exhibit similar patterns in their boxplots. For example, they have similar patterns of medians, quartiles, and they are all absence of outliers. It suggests that these variables share similar distributions or trends despite their different scales or magnitudes.

- **Similar Central Tendencies Pattern:**

The consistent medians observed across all numeric variables suggest that they share similar central tendencies. This indicates that, on average, the numeric variables have comparable midpoint values or typical values, despite differences in their absolute ranges.

- **Consistent Quartiles:**

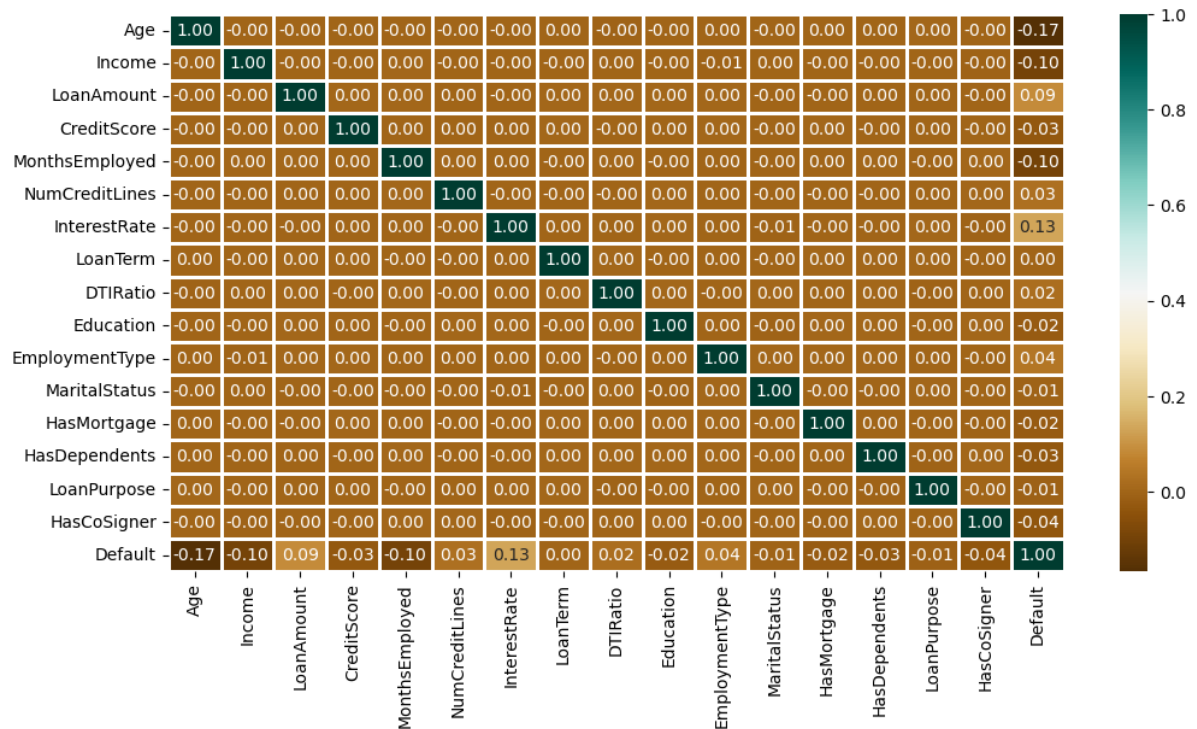
The consistency in the first quartile (Q1) and third quartile (Q3) values across the boxplots indicates that the spread or dispersion of values in the dataset is following a similar pattern. This suggests that the lower and upper ends of the distributions are consistent across the variables.

- **No Outliers:**

The absence of outliers in the boxplots suggests that there are no extreme or unusual values present in the dataset for any of the numeric variables. This indicates that the data points are relatively close to the central tendency and do not deviate significantly from the rest of the distribution.

The consistent central tendencies and quartiles across the numeric variables indicate that they contribute similarly to the overall distribution of the data. When training your model, we can leverage this information to ensure that each variable is appropriately weighted and considered in the modelling process. Ultimately, this could lead to more accurate predictions and better performance of our models.

Lastly, we plotted heatmap for the dataset. This could potentially help you understand the relationship between each variable. And most importantly, it gives us the insights of the relationship between each variable with the targeted variable “Default”.



From the heat map above, we came out with the results below:

Positive Correlation:

Feature	Correlation with Default	Interpretation
InterestRate	0.13	Higher interest rates may be associated with a slightly higher likelihood of default.
EmploymentType	0.04	Certain employment types may be associated with a slightly higher likelihood of default.

LoanAmount	0.09	Larger loan amounts may be associated with a slightly higher likelihood of default.
DTIRatio	0.02	Individuals with higher debt-to-income ratios may be slightly more likely to default.

Negative Correlation:

Feature	Correlation with Default	Interpretation
Age	-0.17	As age increases, likelihood of default decreases slightly.
Income	-0.1	Higher income individuals may be slightly less likely to default.
MonthsEmployed	-0.1	Individuals with longer employment tenure may be slightly less likely to default.
CreditScore	-0.03	Individuals with higher credit scores may be slightly less likely to default.
NumCreditLines	-0.03	Having more credit lines may be associated with a slightly higher likelihood of default.
HasDependents	-0.03	Individuals with dependents may be slightly less likely to default.
Education	-0.02	Individuals with higher education levels may be slightly less likely to default.

MaritalStatus	-0.01	Married individuals may be slightly less likely to default compared to unmarried individuals.
HasMortgage	-0.02	Individuals with mortgages may be slightly less likely to default.
LoanPurpose	-0.01	Certain loan purposes may be associated with a slightly lower likelihood of default.

These tables present the correlations between each feature and the default variable, along with interpretations of the correlations, separately for positive and negative correlations.

Data Preprocessing

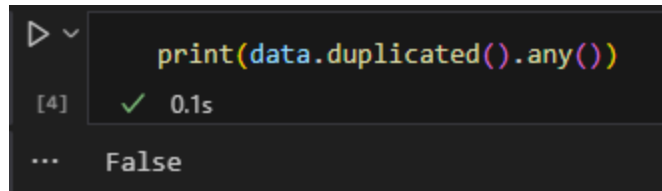
Data preprocessing is a critical step in the data analysis that involves transforming raw data into a clean, structured format suitable for analysis and modelling. It goes through a lot of techniques aimed at addressing issues such as missing values, inconsistencies, and outliers, to ensure the quality and reliability of the data. Effective data preprocessing is the foundation for accurate analysis, and effective modelling. Ultimately, this led to a more reliable and meaningful conclusions.

Handling Missing Values:

One essential aspect of data preprocessing is ensuring that the dataset does not contain any missing values. Missing values can arise due to various reasons, such as data entry errors, sensor failures, or simply the absence of information. While missing values may seem inconsequential, they can significantly impact the analysis and modelling process. From the dataset above, we see that no missing values existed. Hence, we do not need to handle missing values.

Handling Duplicated Data:

Another aspect of data preprocessing involves identifying and addressing duplicated data entries within the dataset. Duplicated data refers to the instances where same records or observations are present more than once in the dataset. While duplicated data might not seem like a big deal, it can causes biases, skew statistical analyses, and distort model performance. In this case , in the image below which is Figure 1. We can see that the there is no duplicated data as the it shows False.



```
▶ print(data.duplicated().any())  
[4] ✓ 0.1s  
... False
```

Figure 1

Luckily, we do not find any missing values in our dataset. Therefore, we do not need to address with this issue.

Label Encoding:

In our dataset, transforming categorical data into numerical representations is necessary for effective modelling and analysis. One common technique for accomplishing this task is using label encoding, which assigns a unique numerical label to each category within a categorical variable. By applying label encoding, we can effectively preprocess categorical variables and incorporate them into machine learning algorithms for predictive modelling and analysis.

Data Splitting

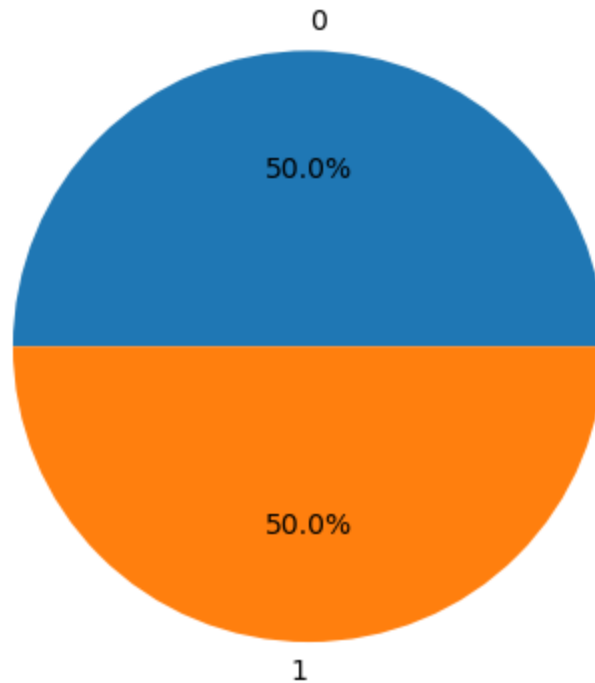
It is essential to evaluate the performance of our models on unseen data to ensure their generalization capabilities. To achieve this, the dataset is typically divided into training and testing sets, allowing us to train the model on one subset and evaluate its performance on another. The dataset splitting process involves partitioning the dataset into two disjoint subsets: a training set and a testing set. The training set is used to train the model, while the testing set is employed to evaluate its performance. In our case, we choose 80:20 ratio, where 80% of the data is allocated to the training set and 20% to the testing set.

Recall to the problem we encountered while we exploring data with categorical variables, we had an issue with imbalanced class. To address this issue, we decided to use the technique Synthetic Minority Over-sampling Technique (SMOTE).

Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE is a method used to tackle class imbalance in datasets. It does this by creating artificial examples of the minority class. Imagine you have a dataset where one class is much less common than the other. SMOTE helps by making up new instances of the minority class by looking at existing ones and creating similar ones nearby. This helps balance out the numbers, making the dataset more even. This process enhances the model's performance by giving it more balanced data to learn from, which in turn improves its ability to make accurate predictions on new data.

After applying SMOTE to address the class imbalance, we visualize the distribution of the "default" variable using a pie chart. The pie chart shows an equal distribution, with 50% of the instances labelled as "0" and the other 50% labelled as "1."



We observed that this balanced distribution indicates that SMOTE successfully addressed the class imbalance issue by generating synthetic samples of the minority class. As a result, both classes now have an equal representation in the dataset. This balanced distribution is beneficial for machine learning model training, as it ensures that the model is exposed to same number of observations from each class. Eventually, this led to an improved model performance.

Algorithm Selection

Selecting the appropriate machine learning algorithm is a critical step in building effective predictive models. Each algorithm has its strengths and weaknesses, making it essential to consider factors such as dataset characteristics, computational efficiency, and interpretability when choosing the right algorithm for a specific task. Below are the algorithms selected, associated with their strengths and weaknesses:

Algorithm	Strengths	Weaknesses
K-Nearest Neighbours Classifier	<ul style="list-style-type: none"> • Intuitive and easy to understand. • Non-parametric approach suitable for complex decision boundaries. • Does not assume any underlying data distribution. • Effective for datasets with well-defined clusters. 	<ul style="list-style-type: none"> • Computationally expensive during inference, especially with large datasets. • Sensitive to the choice of distance metric and the number of neighbours (k). • Performs poorly on high-dimensional datasets.
Random Forest Classifier	<ul style="list-style-type: none"> • Robust to overfitting and noise. • Handles both numerical and categorical data. • Performs well on high-dimensional datasets. • Provides feature importance for interpretability. 	<ul style="list-style-type: none"> • May require more computational resources and longer training times compared to simpler algorithms. • Less interpretable compared to decision trees. • May not perform well on extremely imbalanced datasets.
Logistic Regression	<ul style="list-style-type: none"> • Simple and interpretable. • Efficient for binary classification tasks. 	<ul style="list-style-type: none"> • Assumes linear relationship between features and target variable.

	<ul style="list-style-type: none"> • Provides probabilities as predictions. • Handles feature scaling well. 	<ul style="list-style-type: none"> • May underperform when features are highly correlated. • Limited to linear decision boundaries.
Decision Tree Classifier	<ul style="list-style-type: none"> • Simple and interpretable. • Handles both numerical and categorical data. • Robust to outliers and missing values. • Non-parametric approach suitable for nonlinear relationships. 	<ul style="list-style-type: none"> • Prone to overfitting, especially with deep trees. • Sensitive to small variations in the data. • Does not perform well on datasets with highly correlated features.
XGBoost Classifier	<ul style="list-style-type: none"> • Highly scalable and efficient. • Provides regularization to prevent overfitting. • Handles missing values and outliers well. • Supports parallel and distributed computing. • Widely used in machine learning competitions. 	<ul style="list-style-type: none"> • Requires tuning of hyperparameters. • May not perform well with small datasets. • Less interpretable compared to simpler algorithms.

Gaussian Naive Bayes	<ul style="list-style-type: none"> • Simple and computationally efficient. • Requires minimal tuning of hyperparameters. • Performs well on text classification tasks. • Handles high-dimensional datasets well. 	<ul style="list-style-type: none"> • Assumes feature independence, which may not hold true in some datasets. • Performs poorly on datasets with highly correlated features. • May not capture complex relationships in the data.
MLPClassifier	<ul style="list-style-type: none"> • Capable of learning complex patterns in data. • Handles both numerical and categorical data. • Suitable for large-scale datasets with high computational resources. 	<ul style="list-style-type: none"> • Requires careful tuning of hyperparameters. • Longer training times compared to simpler algorithms. • Sensitive to feature scaling and initialization of weights.

Model Implementation

Before we train a model, we need to initialize the classifier first and after that we will evaluate the classifier after we fit the training data into the classifier. Each classifier can be requiring no parameter or more than one or one parameter in order to start the modelling process. After training the model with sample dataset, we will then measure the accuracy, F1 score and confusion matrix of each models .After that we are required to make the prediction based on the testing set which has been segregated earlier in the data splitting process and we will measure the accuracy , F1 score and confusion matrix of the testing model . Next, we will need to measure the FPR which is False positive Rate in order to see which model we trained is accurate enough. The after that, we need to train the models with training dataset. Before the deployment, we still need to see the ROC curve for each model.

Firstly, We will start off from KNN classifier or known as K-nearest Neighbours algorithm. KNN classifier is relatively easy to implement in the python environment which only require us to install the scilearn.kit modules and after that we import KNN classifier from it. KNN algorithm can easily runs on any CPU as it only requires a CPU to run the algorithm. In term of ease of use and integration, we can say that KNN classifier is relatively easy to code as it requires us to standardize the training values before we can fit into the KNN classifier. The majority voting concept is used by kNN classifiers to determine the class of a given data point. The classes of the five closest points are looked at if k is set to 5. Predictions are made based on the dominating class. In a similar vein, kNN regression uses the average of the five closest locations(www.digitalocean.com, n.d.) .However, we can tune the KNN classifier K value to other odd values if needed to further improve the accuracy of the model by using Hyperparameter tuning method while most of the time , the value of K is set to 5 , we still can further improve the accuracy of the training or testing model by using other K values. Additionally, KNN classifier may struggle to implement when it deals with large dataset due to the curse of dimensionality.

Secondly, The Random Forest classifier is an algorithm or supervised learning technique that relies on group learning. This algorithm is highly versatile and intuitive. Using the provided data samples, it builds decision trees, extracts predictions from each tree, and uses voting to

determine which answer is optimal. Additionally, it is a fairly accurate measure of feature importance (kaggle.com, n.d.). The Random Forest classifier will first split the larger model into smaller models then the smaller model is based on multiple decision trees which are created with different random subsets of the data and features. Then, Random Forest classifier can also be found in scilearn.kit modules which can be imported from sklearn libraries, then we in the Random Forest Classifier , we need to put values in `n_estimators` which normally is 7 , `criterion` and `random_state`. `n_estimators` is the parameter that defines the number of decision trees and we can also tune this parameter to higher value in order to improve the performance of random forest. Next , `criterion= "entropy,"` which describes the function that evaluates each decision tree split's quality. Since the random forest classifier's unpredictability is controlled by `random_state`, repeatability of the findings is guaranteed.

Thirdly, Logistic regression uses a given data set of independent variables to estimate the likelihood of an event occurring, such as voting or not. Predictive analytics and categorization are two common applications for the logistic regression model. Because the result is a probability, the dependent variable has a range of 0 to 1. A logit transformation is performed to the odds in logistic regression, which are the probability of success divided by the probability of failure (IBM, 2022). Logistic regression can be found from scilearn.kit modules which is sklearn . We can simply import Logistic regression from sklearn and we also do not need specify anything in the parameter and if need to further improve the accuracy of the model , we can tune the `random_state` values and parameter tuning the Logistic regression model by tuning the number of iterations , choose the solver algorithm for logistic regression to use and stating the `class_weight` to 'balance' if any unbalanced data sampling is in the Logistic Regression model . However , in our case we do not need to parameter tune our Logistic model as the class imbalance issued is solved.

Fourth, Decision Tree classifier is a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model. (scikit-learn, 2009). Decision Tree classifier can also be imported from

the `sklearn.tree` module. The Decision Tree classifier only needs `random_state` parameter in order to initialize. However, if we wish to further increase the accuracy of the model. We can consider pruning the decision tree which decision tree is tends to overfitting by setting the `max_depth` and `min_samples_split` to prevent overfitting in the decision tree model.

Fifth, XGBoost is a machine learning technique that is a part of the gradient boosting framework, which is a subset of ensemble learning. It makes use of regularization techniques to improve model generalization using decision trees as foundation learners. XGBoost is a popular choice for computationally demanding tasks including regression, classification, and ranking due to its proficiency in feature importance analysis, management of missing information, and computational economy. (Analytics Vidhya, 2018) As for the XGBoost classifier we need to install `xgboost` modules into the Python environment. In `XGBClassifier`, we do not need to enter any parameters inside the `XGBClassifier`'s parameter. However this classifier may sounds similar like Random Forest classifier, While Random Forest Classifier creates its tree independently, `XGBClassifier` works by fixing mistakes from its previous tree, which sets it apart from Random Forest Classifier. (Analytics Vidhya, 2018) .There are ways to improve the accuracy in the XGBoost classifier by increasing the learning rate or lowering the learning rate and number of estimator by tuning tree-based parameters like `max_depth` or `min_child_weight`.

Sixth, A supervised machine learning technique called the Naïve Bayes classifier is utilized for classification tasks like text classification. To carry out classification jobs, they apply probability concepts. The goal of Naïve Bayes, like other generative learning algorithms, is to simulate the distribution of inputs inside a specific class or category. It doesn't figure out which features are most crucial for class differentiation, in contrast to discriminative classifiers like logistic regression. Naïve Bayes Classifier can be imported from `sklearn.naive_bayes` modules and it does not require any parameters by default. However if you planned to increase the accuracy, we can handle the continuous variables in our data or paralyzing the probability calculations on each attributes. (Tokuç, 2021)

Lastly, `MLPClassifier` which known as one of the neural network models but it is a supervised model. An suitable collection of outputs is mapped from input data sets to the multilayer perceptron (MLP), a feedforward artificial neural network model. Each layer in an

MLP is completely connected to the layer above it. An MLP has many layers. All of the neurons in the layers—aside from the input layer's nodes—have nonlinear activation functions. Nonlinear hidden layers may exist in one or more layers between the input and the output layer. (michael-fuchs-python.netlify.app, n.d.) We can improve the MLPClassifier's accuracy in the hyperparameter by adjusting the learning rate, number of Epochs via GridSearchCV and RandomSearch. (Singh, 2020) .However , we can also utilize other modules such as TensorFlow to train the neural network as TensorFlow modules allowed us to switch to GPU for training neural network model.

Model Comparison

Firstly, let's talk about each algorithm when dealing large datasets or small datasets, KNN classifier will not perform well on large datasets which will result in higher computational resource needs meanwhile XGBoost classifier does not work well with smaller datasets due XGBoost unable to split the trees properly with fewer samples. Other than that, other classifiers work well with the datasets.

Secondly, let's talk about which algorithm requires hyperparameter tuning to improve the accuracy. KNN classifier, Random Forest classifier, Decision Tree classifier, XGBoost classifier and MLPClassifier require extensive tuning due to their complexity and has larger number of hyperparameters.

Finally, let's talk about computational resource usage, among all the classifiers, KNNclassifier , Random Forest Classifier, MLP classifier and neural network model requires more computational resources than other classifiers due to their complexity. However, it really depends on the feature size that is fitted in the classifier. Meanwhile, Naïve Bayes classifier is the fastest among all classifiers due to it requires less computational resources than other classifiers but it does have a limitation which are performing poorly on datasets with highly correlated features and it may not capture complex relationships in the data.

Results and Interpretation

Firstly, let's talk about the neural network classifier which can utilize both CPU and GPU. Here is the side-by-side comparison for both CPU and GPU, refer to **Figure 2**.

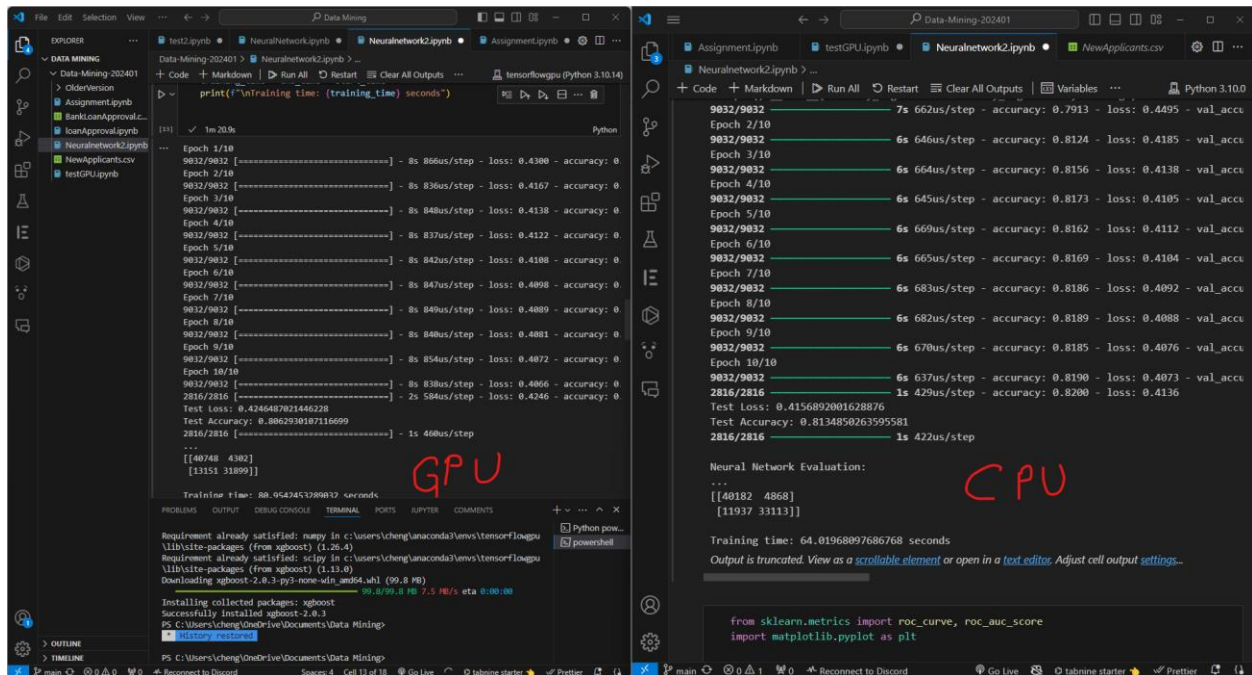


Figure 2

The result might be varies due to the hardware used on the neural network using tensorflow.keras libraries (As the hardware used in this TensorFlow neural network are CPU (I7-11800H) and GPU (RTX-3070M)) However, as you can see the CPU are faster than GPU and also the confusion matrix in CPU environment is higher than GPU environment. In this case, anaconda prompt is used to install the GPU environment for tensorflow (refer here : <https://www.youtube.com/watch?v=GFTlcKzhmoE&t=589s>). There might be several factors that caused the GPU to be slower than CPU , the first factor is the CPU clock speed might be higher than the GPU resulting in slower performance in GPU environment and also there might model size and complexity factor which the overhead of transferring the data to the GPU and managing parallel computations might be higher than the actual computation time , thus CPU is faster.

Model	Time Taken in second	Accuracy	F1 Score	Confusion Matrix
KNN Classifier	0.0030	68.49	0.67	[1343 519] [619 1131]
Random Forest Classifier	0.0648	98.48	0.98	[1836 26] [29 1721]
Logistic Regression	0.0309	66.36	0.65	[1280 582] [633 1117]
Decision Tree Classifier	0.0339	100.00	1.00	[1862 0] [0 1750]
XGBoost Classifier	0.1925	100.00	1.00	[1862 0] [0 1750]
Gaussian Naive Bayes	0.0060	73.92	0.75	[1255 607] [335 1415]
MLPClassifier	0.2833	52.46	0.15	[1689 173] [1544 206]

Tabel 1: Model Training Result with Sample Data

Sample data is used in this model training to have the idea of which model would be performing ideally in data prediction. Decision Tree Classifier and XGBoost Classifier perform in an ideally way which Accuracy is 100 and F1 score is 1.0, however XGBoost Classifier takes 0.1925 second to train and retrieve the data however Decision Tree Classifier takes 0.0339 second to train the model. We can expect that Decision Tree Classifier and XGBoost Classifier will be performing well in actual dataset.

Model	Time Taken in second	Accuracy	F1 Score	Confusion Matrix
KNN Classifier	0.0040	55.11	0.53	[26584 18466] [21981 23069]
Random Forest Classifier	0.0689	78.96	0.79	[36427 8623] [10340 34710]
Logistic Regression	0.0359	67.28	0.66	[31893 13157] [16321 28729]
Decision Tree Classifier	0.0387	71.47	0.72	[32095 12955] [12748 32302]
XGBoost Classifier	0.1965	85.83	0.85	[40304 4746] [8025 37025]
Gaussian Naive Bayes	0.0040	73.97	0.76	[30285 14765] [8687 36363]
MLPClassifier	0.2534	44.81	0.36	[26507 18543] [31184 13866]

Table 2: Model Testing Result with Actual Dataset

Based on the table, the result shows that the XGBoost Classifier has the highest accuracy at 85.83 and F1 score at 0.85 among other models however the time taken for it to train the data is 0.1965 second which is the second time taken to train the model. Decision Tree Classifier which performs well in previous data sample only has 71.47 accuracy and 0.72 F1-score which is the third idea model among the models.

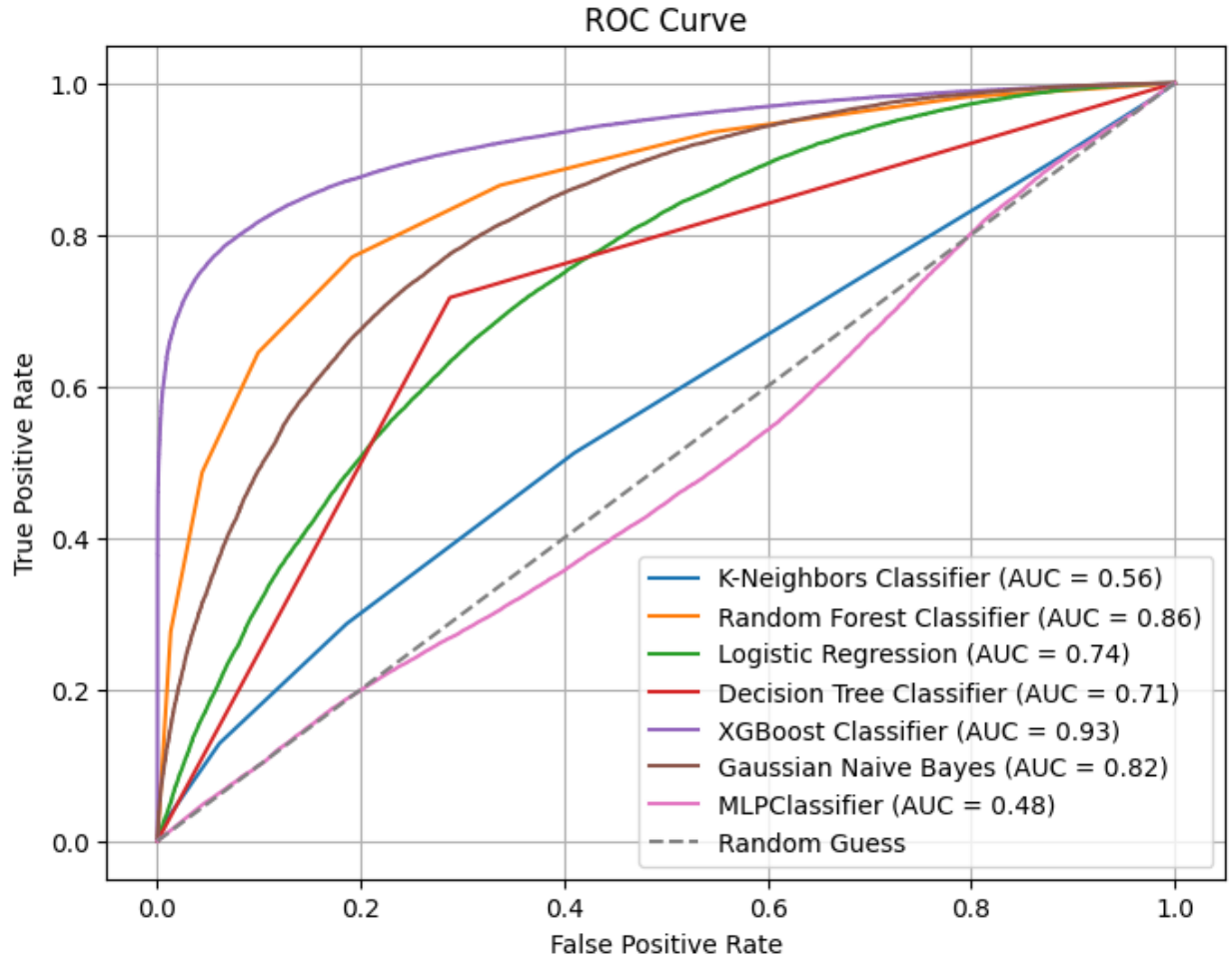


Figure 3: ROC Curve with Model without tuning

According to ROC Curve, it shows that XGBoost Classifier has the highest AUC which is 0.93 without tuning model follow by Random Forest with 0.86 AUC. We can expect that AUC of XGBoost Classifier will increase after tuning the model.

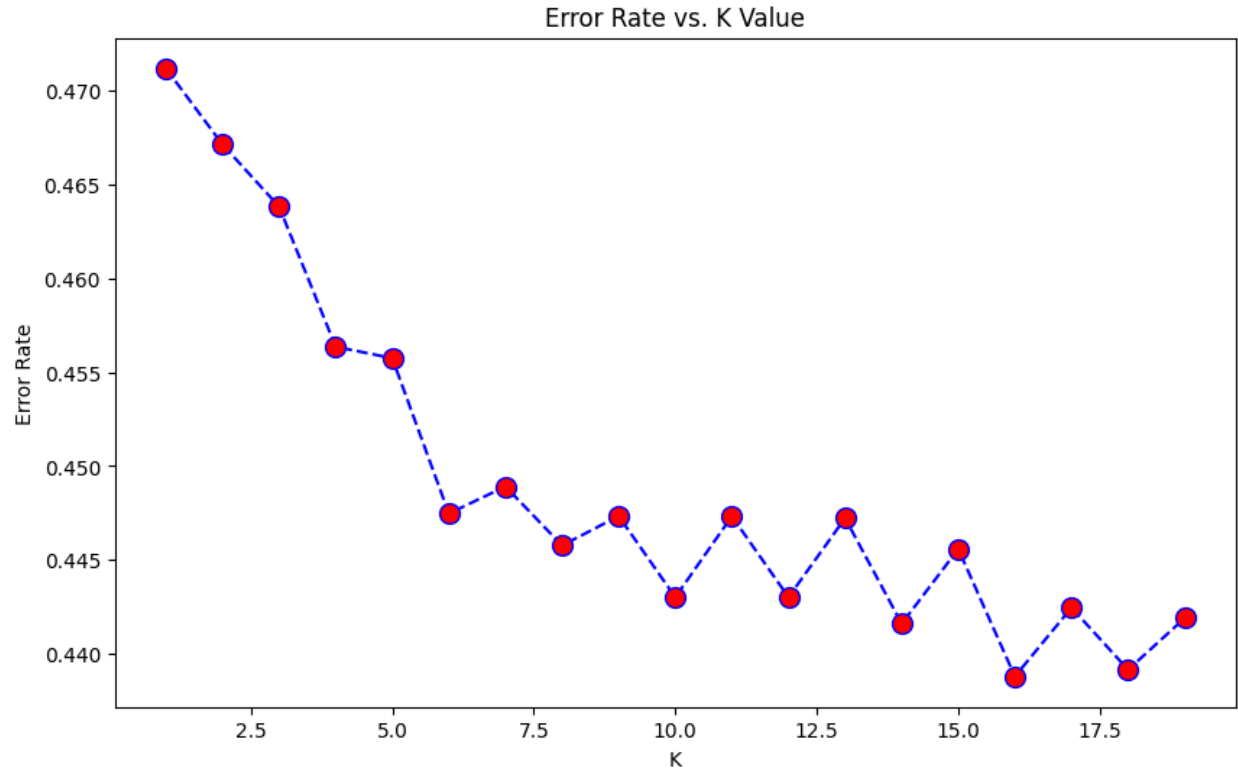


Figure 4: Error Rate vs K Value Graph

In order to get the idea number for k value, we use Error Rate vs K Value graph. We decide to use 7 as the k value to tune the model according to the graph. This method should increase the accuracy of the model and lower the error rate of the model.

Model	Time Taken in second	Accuracy	F1 Score	Confusion Matrix
KNN Classifier	0.0728	53.01	0.47	[29029 16021] [26321 18729]
Random Forest Classifier	12.8297	82.72	0.82	[39413 5637] [9932 35118]
Logistic Regression	2.5492	67.66	0.68	[30284 14766] [14376 30674]
Decision Tree Classifier	5.9720	78.55	0.78	[36418 8632] [10691 34359]
XGBoost Classifier	2.1772	89.50	0.89	[42530 2520] [6945 38105]
Gaussian Naive Bayes	0.1945	73.70	0.76	[29574 15476] [8221 36829]
MLPClassifier	911.0060	77.72	0.79	[33286 11764] [8311 36739]
Neural Network	123.8537	82.00	0.81	[39474 5576] [10707 34343]

Table 3: Model Testing Result with Actual Dataset after tuning model

After tuning model, most of the model has increment for the accuracy and F1 score except KNN Classifier which has a decrement. Accuracy of XGBoost Classifier increases from 85.83 to 89.50 and F1-score from 0.85 to 0.89.

At this point we add Neural Network model as we tried to look for a higher accuracy for the prediction but accuracy 82% is the highest that we could find which is still lower than XGBoost Classifier.

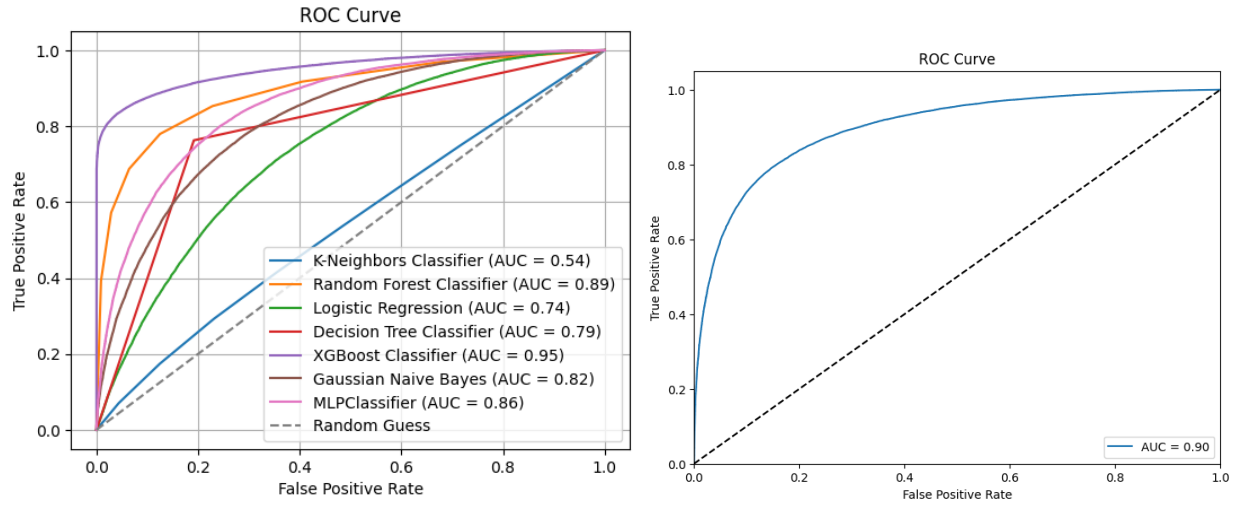


Figure 3: ROC Curve with Model after tuning

After tuning the model, every AUC of model has increment expect for KNN Classifier which decreases. AUC of XGBoost Classifier has increment as well from 0.93 to 0.95 which is the highest among the other models.

LoanID	Model							
	K-N	RF	LR	DT	XGB	GNB	MLP	NN
A01	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
A02	Approve	Reject	Reject	Reject	Reject	Reject	Reject	Reject
A03	Reject	Reject	Approve	Reject	Reject	Approve	Approve	Reject
A04	Reject	Reject	Reject	Reject	Reject	Approve	Approve	Reject
A05	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
A06	Reject	Reject	Approve	Reject	Reject	Approve	Approve	Reject
A07	Reject	Reject	Reject	Reject	Reject	Approve	Reject	Reject
A08	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
A09	Reject	Reject	Approve	Approve	Reject	Approve	Approve	Reject
A10	Approve	Reject	Reject	Reject	Reject	Reject	Reject	Reject
B01	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
B02	Approve	Reject	Reject	Reject	Reject	Reject	Reject	Approve
B03	Reject	Approve	Reject	Reject	Approve	Reject	Approve	Reject
B04	Reject	Reject	Approve	Reject	Reject	Approve	Reject	Reject
B05	Approve	Reject	Reject	Reject	Reject	Reject	Reject	Reject
B06	Reject	Approve	Approve	Reject	Reject	Approve	Approve	Approve
B07	Reject	Approve	Approve	Approve	Approve	Approve	Approve	Reject
B08	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
B09	Reject	Reject	Approve	Approve	Reject	Reject	Reject	Reject
B10	Reject	Reject	Reject	Reject	Reject	Reject	Reject	Reject
K-N -> KNN Classifier					XGB -> XGBoost Classifier			
RF -> Random Forest Classifier					GNB -> Gaussian Naive Bayes			
LR -> Logistic Regression					MLP -> MLPClassifier			
DT -> Decision Tree Classifier					NN -> Neural Network			

Table 4: The Result of Loan Approval

According to the result it shows that XGBoost Classifier only approve loan B03 and B07 and reject the rest of the loan.

Conclusion

After training model with 8 different model with different method which are KNN Classifier, Random Forest Classifier, Logistic Regression, Decision Tree Classifier, XGBoost Classifier, Gaussian Naive Bayes, MLPClassifier and Neural Network. We decide to use XGBoost Classifier to train a model as we retrieve the highest accuracy which is 89.5. We are come out with a result that only loan B03 and B07 will be approve and the 18 other loans will be rejected. We have 89.5% of confidence that the result will be accurate.

References

www.digitalocean.com. (n.d.). K-Nearest Neighbors (KNN) in Python | DigitalOcean. [online] Available at: <https://www.digitalocean.com/community/tutorials/k-nearest-neighbors-knn-in-python>.

kaggle.com. (n.d.). Random Forest Classifier Tutorial. [online] Available at: <https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial>.

IBM (2022). What Is Logistic Regression? [online] IBM. Available at: <https://www.ibm.com/topics/logistic-regression>.

scikit-learn (2009). 1.10. Decision Trees — scikit-learn 0.22 documentation. [online] Scikit-learn.org. Available at: <https://scikit-learn.org/stable/modules/tree.html>.

Analytics Vidhya. (2018). Understanding the Math behind the XGBoost Algorithm. [online] Available at: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>.

Tokuç, A.A. (2021). How to Improve Naive Bayes Classification Performance? | Baeldung on Computer Science. [online] www.baeldung.com. Available at: <https://www.baeldung.com/cs/naive-bayes-classification-performance>.

michael-fuchs-python.netlify.app. (n.d.). NN - Multi-layer Perceptron Classifier (MLPClassifier) - Michael Fuchs Python. [online] Available at: <https://michael-fuchs-python.netlify.app/2021/02/03/nn-multi-layer-perceptron-classifier-mlpclassifier/>.

Singh, A.P. (2020). Steps You Should Follow TO Successfully Train MLP. [online] Analytics Vidhya. Available at: <https://medium.com/analytics-vidhya/steps-you-should-follow-to-successfully-train-mlp-40a98c3b5bb3>