COMP7409A Project Report:
Fraud detection on The Bank Account Fraud (BAF) suite of datasets

Group 12
Liu Yutong, Zheng Wenjing, Li Chentao, Zhou Yuen and Xu Yunshi

Introduction:

In the digital age, credit card fraud has become a concern for individuals and financial institutions alike. The ability to detect fraudulent cases among millions of legitimate ones is a complex task. Our focus for this project is to compare several models' performance over a credit card fraud dataset named "*Bank Account Fraud Dataset Suite (NeurIPS 2022)*". The dataset is acquired from kaggle and it contains 1 base csv file with roughly 1 million records along with 5 variant csv files with different distributions.(See Appendix A for details of each attribute for these csv)

As for data processing, we made a python script for that purpose. There are several methods for: data selection, data sampling, one hot encoding and data printout. It is the foundation to make our project highly modular, therefore we can write a main python script to handle user inputs such as algorithm selection, data selection, data sampling and visualization. The algorithms are in different scripts named by algorithms' names and can train and predict once the user chooses to select an algorithm to perform on selected dataset. With this convention we can easily add or delete algorithms without impacting other parts of the project. In addition, there is one more script called "*plot.py*", that is for drawing graphs for F-score or False Positive line charts and name the graph with parameters chosen for that algorithm. Thanks to the design, the workload of the project can be distributed easily.

Problem Formulation:

After consideration of our computational power and time budget, we settled our goal to compare performances among algorithms over datasets. We are aware of the high imbalance between fraud and non-fraud cases. Thus we cannot use accuracy as our prediction if we sample them in an imbalanced manner. Therefore we use k-fold cross validation with two scores to evaluate our algorithms: False Positive rate and F-score.

Algorithms:

Random Forest:

Introduction:

A. Background information on the classification problem:

A classification problem is the task of assigning predefined categories or labels to input data based on its characteristics. In the experimental environment we chose, the code relies on the column named "fraud_bool" in the CSV file to study and solve classification problems related to fraud detection.

B. Brief description of the random forest model:

The random forest model is an ensemble learning method that combines multiple decision trees for prediction. It operates by creating an ensemble of decision trees, where each

tree is trained on a different subset of data, using a random subset of features. The final prediction is determined by aggregating the predictions of the individual trees, usually by voting or averaging.

C. Purpose and Scope of the Report:

The purpose of this section of the code and related reports is to train a random forest classifier for fraud detection on the dataset provided by Kaggle. It uses the Pandas library to process data and the Random Forest Classifier (RandomForestClassifier) in the sci-kit-learn library for model training and prediction. The code also uses some other libraries and functions, such as the drawing library (plot) and the database processing library (database). Aims to gain insight into the model's performance, analyze key metrics, and discuss results compared to other classification models.

Method:

A. Data collection and preprocessing:

The data comes from Kaggle and has been processed and classified by professionals, and different variant versions are listed (for example, variant I: has a higher group size difference than base, variant II has a higher prevalence difference than base) to facilitate the model to focus on the performance of different emphases.

Data preprocessing steps:

1. `display_data` function: This function is used to display data loaded from CSV files. Depending on the parameters provided, it can selectively sample non-fraud records and provide either a summary or details of the data.

2. `data_lanudry` function: This function is used to load the data file and sample from non-fraud records according to the provided parameters. It will try to load the CSV file with the specified name from the "./database/" directory and return a DataFrame object.

3. `data_selection` function: This function is used to select records from data. It classifies data into non-fraud records and fraud records and samples from non-fraud records and fraud records based on the provided parameters. The sampling process is random and returns a DataFrame object containing the sampled records.

4. `one_hot` function: This function is used to perform one-hot encoding of categorical variables. It receives a dataset, a training data identifier, and a one-hot encoder as input, and returns a DataFrame object containing the one-hot encoded data. Finally, use StandardScaler to standardize the data.

B. Training and evaluation of random forest models:

In the random_forest_train function, the input feature data and label data are first divided into a training set and a test set. The data preparation phase also includes one-hot encoding of categorical variables and standardization of numerical variables. Then use the random forest classifier (RandomForestClassifier) in the scikit-learn library to pass the feature data and label data of the training set into the model for training. During the training process, the random forest model learns based on the combination of features and the construction of decision trees. After the training is completed, use the trained random forest model to predict the feature data of the test set. The prediction results are compared with the true labels of the test set, and evaluation metrics such as error rate and false positive rate are calculated. The calculation of evaluation indicators uses methods such as confusion matrix, precision rate, and recall rate.

Results and Analysis:

A. Overview of the datasets used:

The Bank Account Fraud (BAF) dataset suite has been released at NeurIPS 2022, which contains a total of 6 different synthetic bank account fraud table datasets. Each dataset consists of 10,000 instances, 30 real-world features used in the fraud detection use case, a "month" column that provides time information about the dataset, and four protected attributes (age group, employment status, and income percentage) Partially composed.

B. Evaluation metrics used to evaluate model performance:

Fraud is a highly imbalanced task (approximately 1% of fraudulent labels), so accuracy is not a sufficient evaluation metric. The constant negative classifier achieves over 98% accuracy but cannot predict any fraud.

Therefore we only use the F1-score and FPR:

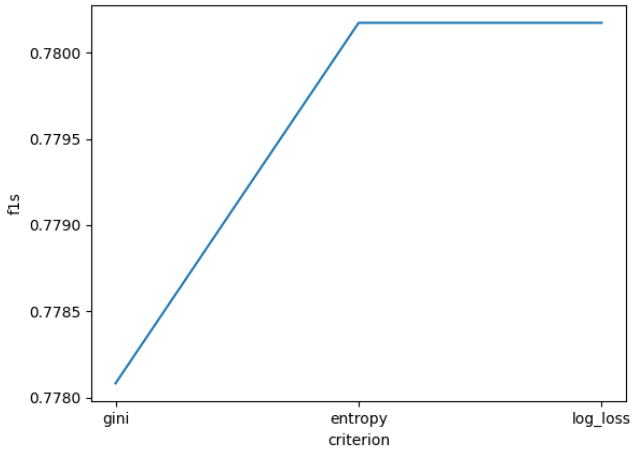
1. **False Positive Rate (FPR)**: The false positive rate is the ratio of false positive predictions to the total number of actual negative instances. It measures the proportion of negative instances that are incorrectly classified as positive. FPR is calculated using the formula: FPR = FP / (FP + TN), where FP represents false positives and TN represents true negatives. By using FPR, a model's false positive rate in normal transactions can be more accurately measured, allowing a better understanding of the model's misclassification costs. At the same time, it reduces the probability of incorrectly labeling normal transactions as fraudulent transactions and improves the accuracy and reliability of the model. F1-score is the harmonic average of precision and recall. Its value is between 0 and 1. The closer it is to 1, it means that the model has achieved a better balance between precision and recall and has better overall performance.

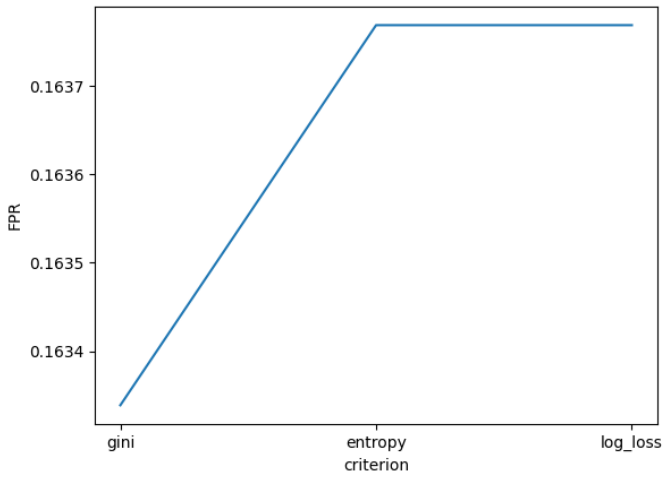
2. **F1-score**: It is the harmonic mean of precision and recall, providing a balanced measure of model performance. In fraud tasks, fraudulent transactions are usually in the minority category,

while normal transactions are in the majority category. F1-score is relatively robust in imbalanced data sets. By combining precision and recall, the F1 score is better able to handle the challenges posed by imbalanced data sets. It is more resistant to bias in results caused by data skew, providing a more accurate assessment of model performance. The value of FPR is between 0 and 1. The closer to 0, the lower the false positive rate of the model in negative samples and the better the performance.

C. Detailed analysis of random forest model performance:

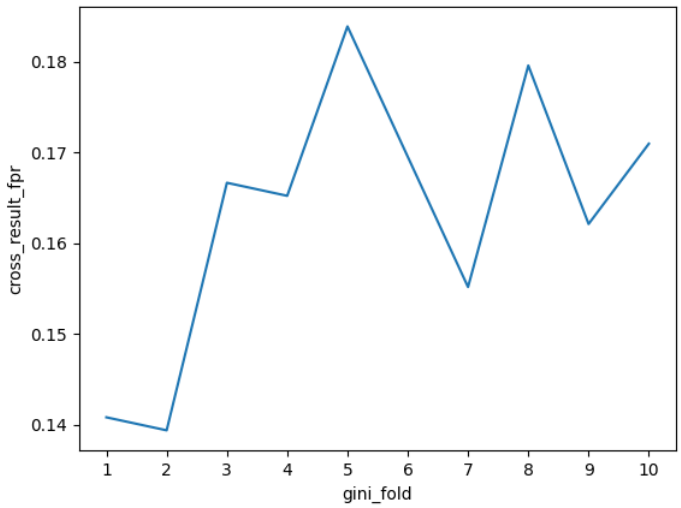
We load RandomForestClassifier from sklearn.ensemble package, manually adjust the parameters inside, and use grid_search to find the int-type parameters that are most suitable for the current data set, such as n_estimators, random_state, etc. Finally, we chose to conduct a comparative test on the criterion parameters. Select three criteria parameters: 'gini', 'entropy', and 'log_loss'. These strings represent different criteria used to measure the quality of splitting in a random forest model.

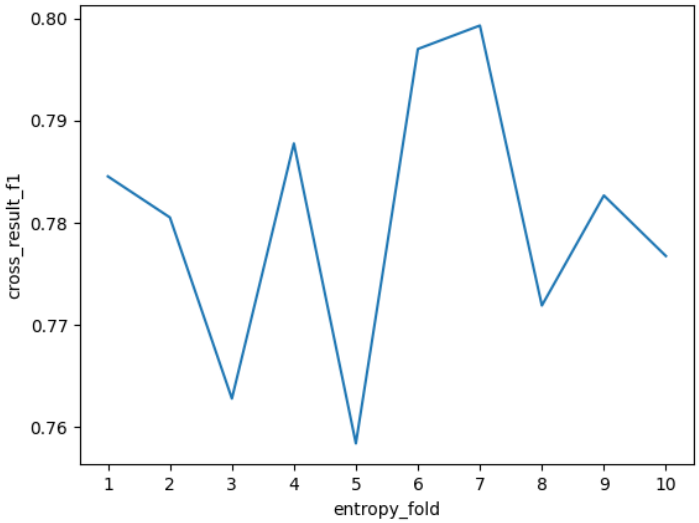




In the for loop, the code iterates through each element in the criteria list (i.e. each criterion) and passes it as a parameter to the random_forest_train function. This function will train a random forest model using the specified criteria and return the false positive rate (fpr) and error rate (lr_error) of the model.
These false positive and error rates are then added to the FPR and error lists. In this way, we can compare the performance of models under different criteria. After training a random forest model on a dataset and evaluating it on a test set, these metrics can be calculated to evaluate the performance of the model.

Next, let's put the pictures for each of the two criteria parameters:





Discussion:

A. Interpretation of the results:

Based on the results using the Random Forest model, we can perform the following explanations and analyses. First of all, (the first two figures) show that when the criterion is entropy, a higher f1-score can be obtained compared to other criterion parameters, but for FPR only gini achieves the best effect. Therefore, these two criterion parameters are used to complete the test of the model. The best F1-score came to 0.779475764616622, and the best FPR was 0.16362544457087572.

B. Comparison with other classification models:

Compared with all other models in this study: Logistic Regression, SVM, RNN, and LGBM, their models have almost equal f1-score, only slightly ahead of RNN, and ahead of all other models in terms of FPR.

C. Strengths and weaknesses of the Random Forest model:

The Random Forest model has the following advantages:

- Able to handle high-dimensional and large-scale data sets, with good scalability.
- When processing data containing a large number of features, important features can be automatically selected to reduce the workload of feature selection.
- Able to handle missing values and outliers, with relatively low data preprocessing requirements.
- By integrating multiple decision trees, the risk of overfitting can be reduced and the generalization ability of the model can be improved.

However, the Random Forest model also has some limitations:

- For dealing with highly imbalanced data sets, there may be a bias towards the dominant class.
- Model training and prediction may be slower due to the use of multiple decision trees.
- The interpretability of the model is relatively weak and it is difficult to provide detailed explanations of feature importance and decision-making processes.

D. Potential areas for improvement:

To further improve the performance of the Random Forest model, the following improvements can be considered:

- Data balancing processing: For highly imbalanced data sets, data resampling techniques (such as undersampling, and oversampling) or category weights can be used to balance the data to improve the model's ability to identify minority categories.
- Hyperparameter tuning: Optimize the performance and generalization ability of the model by adjusting the model's hyperparameters, such as the number of decision trees, tree depth, feature sampling ratio, etc.

- Model integration: Consider integrating the Random Forest model with other types of models, such as Gradient Boosting Trees, to further improve the overall prediction performance.

Logistic Regression:

Approach Introduction:

Although being called regression, it is used for classification tasks. The way to achieve it is fitting a logistic regression curve to the records and predicting the probability of the class being positive by using the independent attributes.

The process of Logistic Regression is: first, initialize the weights θ_0, θ_1, θ_2 …… in

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ...$$

Then, use the sigmoid function to calculate the predicted result, that is the probability of the class being positive:

Sigmoid function: $h = g(z) = 1/(1 + e^{-y})$

Next, fit the curve by maximizing the likelihood:

Likelihood of the model: $\prod_{i=1}^{n_{-ve}} (1 - p(x_i)) \prod_{i=n_{+ve}}^{n} p(x_i)$

Finally, using gradient descent to get the result and update the weights. Repeat these steps above until the cost function converges to the minimum. And the final weights and the model are then used to predict the records without class labels.

Select parameters:

The LR model is implemented in logistic.py. In order to find the best model, two parameters are being tuned.

Solver: which is used to choose the algorithms used in the optimization problem to minimize the loss function.

Penalty: is the regularization choice parameter, which is used to prevent overfitting by adding penalty to loss function to reduce the complexity of the model.

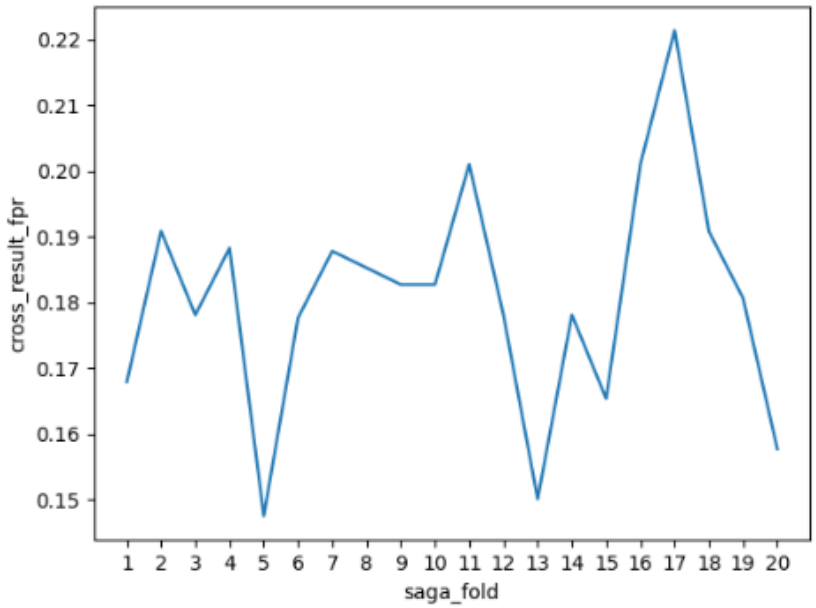
Note that some solvers can only be combined with certain penalties. Tuning combination shown:

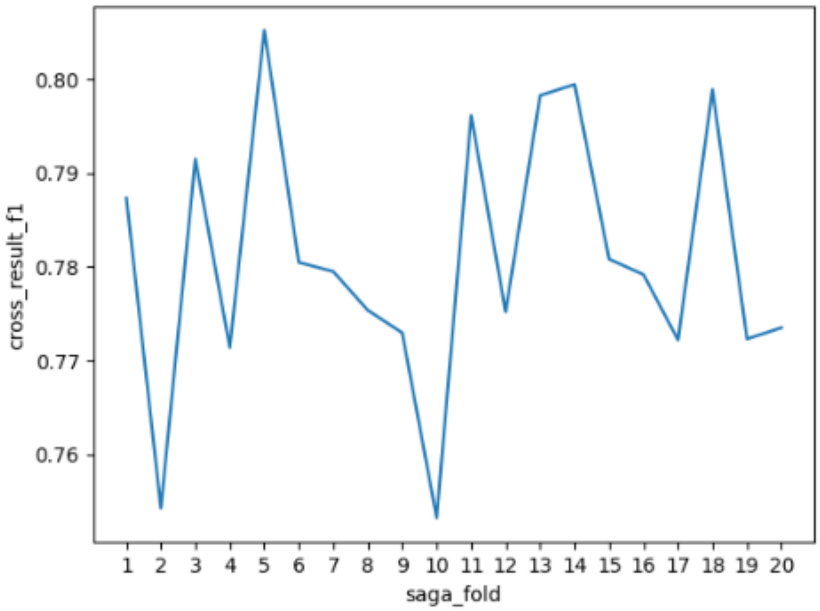
```
param_grid = [
    {'solver': ['lbfgs'], 'penalty': ['l2', None]},
    {'solver': ['liblinear'], 'penalty': ['l1', 'l2']},
    {'solver': ['newton-cg'], 'penalty': ['l2', None]},
    {'solver': ['newton-cholesky'], 'penalty': ['l2', None]},
    {'solver': ['sag'], 'penalty': ['l2', None]},
```

{'solver': ['saga'], 'penalty': ['elasticnet', 'l1', 'l2', None]}
]

The result of best parameters combination is solver = ‘saga’, penalty = ‘l1’

Model Result:





According to the above result, the average f1 score of LR is 0.7822943411688543 and the average FPR score is 0.17450909207736293.

Strengths and weaknesses of Logistic Regression model:

There are some advantages in using LR:

- Compared with complex models such as neural networks, its calculation efficiency is very high, and the optimized gradient descent algorithm can be used to further improve the efficiency.
- Insensitive to noise. Since maximum likelihood estimation is used in model creation, it is not easily affected by noise and outliers.
- This model is also easy to interpret, as the weights can be printed out. It will be easy for users to understand the performance of the model.

However, it also has many limitations:

- As the curve is built relying on the labeled class, if the labeled class quality cannot satisfy the standard, the model performance will be influenced.
- Logistic Regression is a kind of linear model which means that it could only find a linear decision boundary. So if the dataset has nonlinear relationships, it will be hard for the model to find a suitable result.
- When handling high-dimensional datasets, if logistic regression wants to prevent overfitting, it needs a larger training set therefore the cost of computing will increase a lot.

SVM:

Approach Introduction:

Support vector: The data to be classified can be divided into positive and negative classes by a hyperplane called decision boundary. The decision boundary that meets is moved up and down to the positive and negative classes. The outermost boundary of the class is the interval boundary. The distance between the two interval boundaries is $d=2/||\overline{w}||$, where $||\overline{w}||$ is the slope coefficient of the decision boundary function. The support vector is the samples on the two interval boundary. SVM is trying to find an optimal decision boundary so that the distance between closest two samples in the two categories, that is, the distance between the support vectors is the furthest. So the model need to minimize $||\overline{w}||^2$.

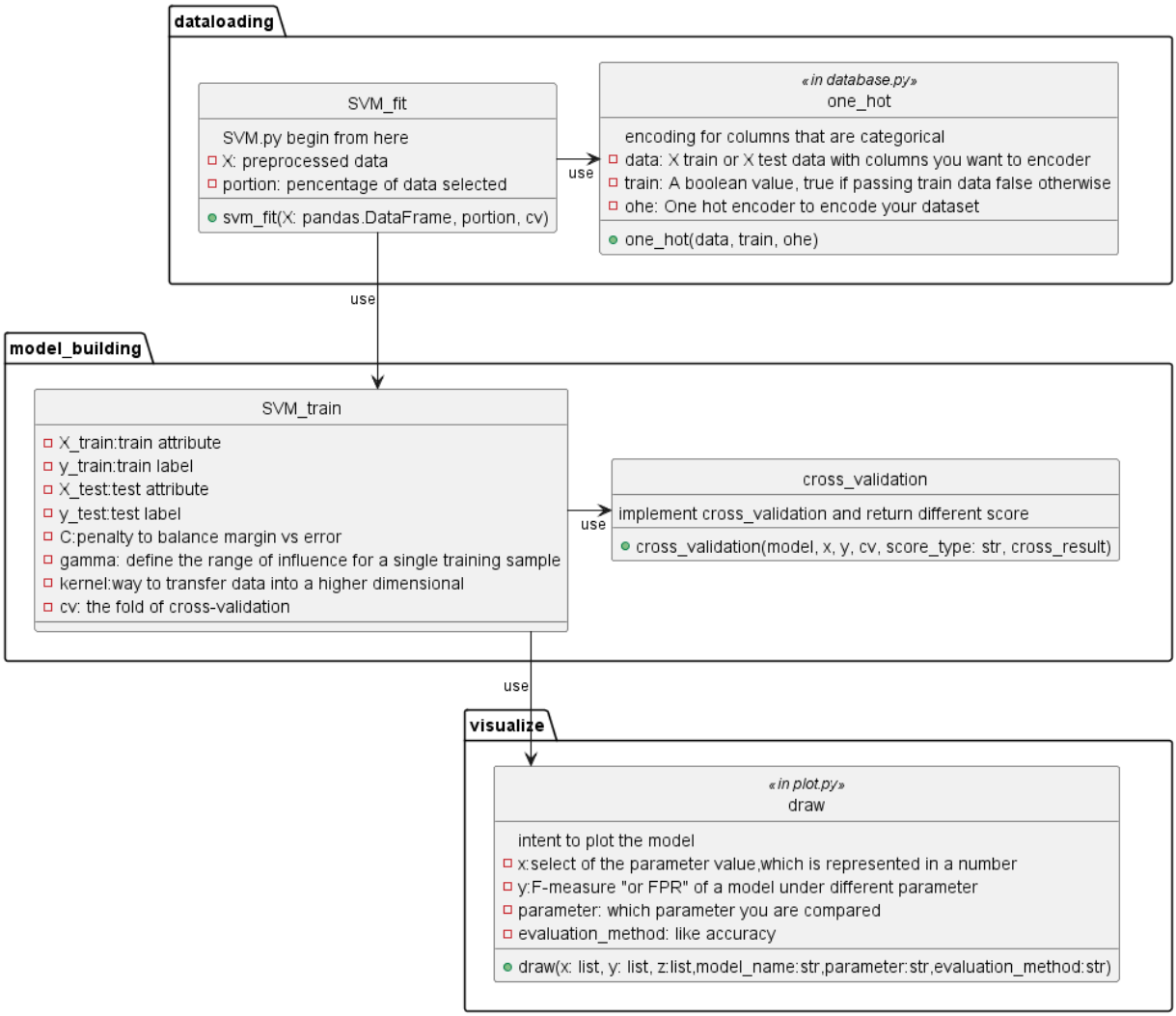
Kernel method: When the data to be classified is not linearly separable, the nonlinear data is often mapped into a higher dimensional space and then a linear interval boundary is found. Generally, computing the support vectors requires computing many 'dot products' of data points (because we need to measure similarity), computing each point in the high dimensional space will be very complicated, so we define a kernel Φ, if $\Phi(\overline{x})\cdot\Phi(\overline{y})$ can be computed by a function $K(\overline{x}\cdot\overline{y})$in the original low space to simplify the calculation.

Slack variables: Another way to separate the not linearly separable dataset is to give each point a slack variable. It allows objects to be misclassified but penalizes the model by the total amount of slack. So at last the target of the model is to minimize the formula:

$$||\overline{w}||^2 + C(\sum_{i=1}^{N} \xi_a)^k$$

Build model:

The SVM model is implemented in svm.py and the whole process is shown in the flowchart below:



First part is “dataloading” which loads the partitioned data and calls one_hot function in database.py to preprocess the categorical attribute and scale the data.

Next part is “model_building” in which SVM_train called by SVM_fit will train the SVM model and get the model performance by 20 cross validation.

Finally, use the “draw function” in plot.py to show the F1 and FPR cross validation results in pictures.

Tune parameters:

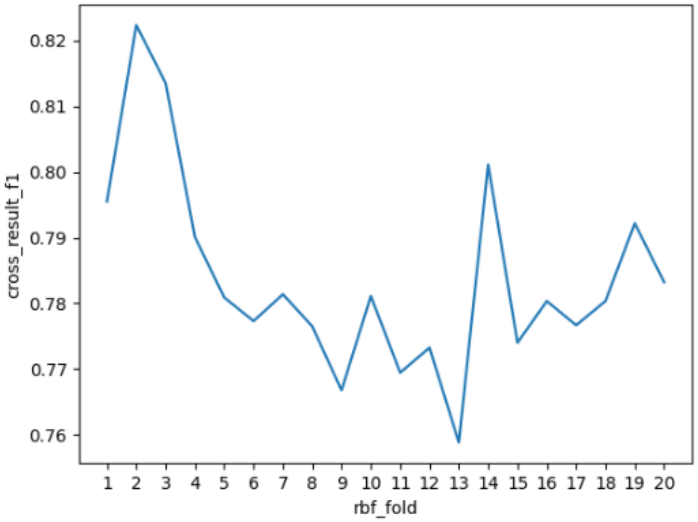
When building a model, in order to improve the result of it, we must tune some parameters. The main parameters in SVM selected by us to tune are ‘C’, ‘gamma’ and ‘kernel’.

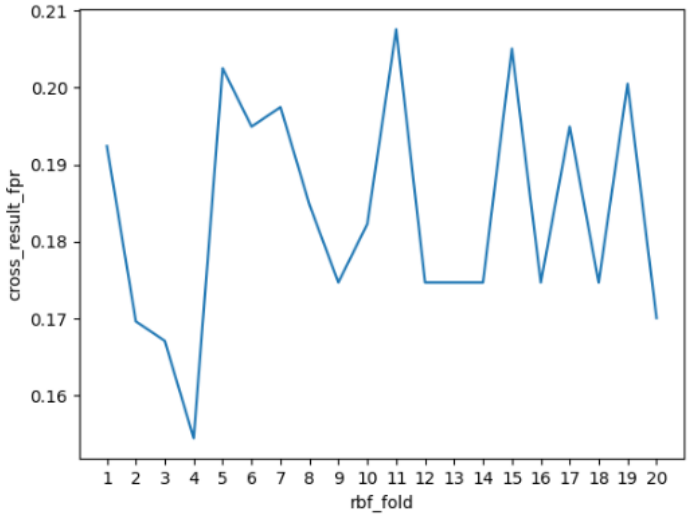
Notice that as tuning parameters should be the function in SVM.py only and should not tune parameters every time the client need to see the result of different model, so we implement tuning parameters in if __name__ == "__main__" and run it in SVM.py only.

C: penalty to balance maximizing margin and minimizing error.
gamma: define the range of influence for a single training sample. When it is large, the influence by the support vector will be small, which means that the support vector only affects the small range around itself, so the decision boundary will be more complex and easier to overfit.
Kernel: choose a way to transfer data into a higher dimensional.

The way to tune the parameters is GridSearchCV. As kernel ‘linear’ cannot appear with parameter ‘gamma’, we separate it from other kernels in param_grid.
The result of best parameters combination is C=1, gamma=0.01,kernel= ‘rbf’.

Model Result:





According to the above result, the average f1 score of SVM is 0.79214416 and the average FPR score is 0.17070675.

Strengths and weaknesses of SVM model:

There are some advantages in using SVM:

- model works very well when the dataset has a clear margin of separation
- effective in high-dimensional spaces especially when the number of attributes is greater than that of records.
- in the process only use support vectors to decide the hyperplane, that is, use a small part of the training set, so it is memory efficient.

However, it also has many limitations:

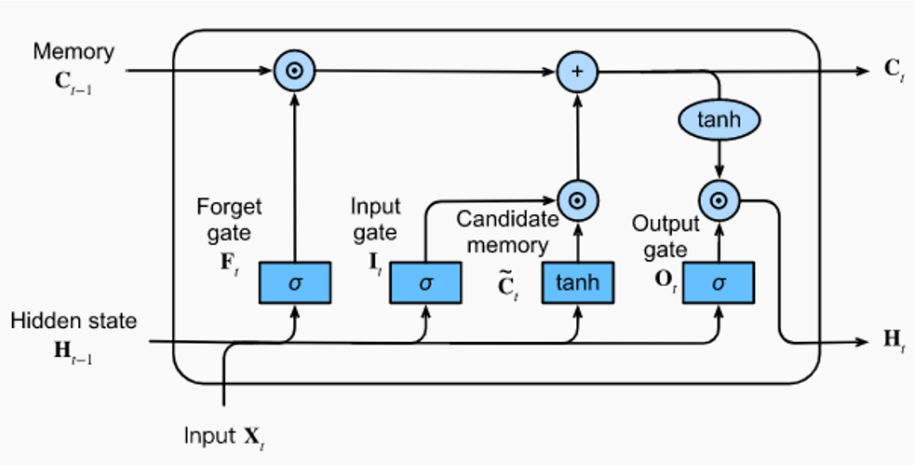
- As it only uses a small part of training set to decide the hyperplane, it will be sensitive to noise.
- It is important to choose the right parameters including suitable kernels to have a better performance SVM model, but sometimes this will be computationally intensive.
- When handling large datasets, SVM needs to solve a quadratic optimization problem, so it always takes a long time to build on large datasets.

Potential areas for improvement:

- extensive parameter tuning: other parameter tuning methods like random search and bayesian optimization can be used to find parameters having better performance.
- try to use ensemble learning which means that it could use bagging or boosting to combine the predictions of different models.
- feature selection and feature engineering can be done to remove the attributes irrelative or redundant. Or create some new features which could better describe data characteristics.

RNN(LSTM):

Approach and introduce:



A traditional RNN consists of a series of repeated circuits made up of many neurons, each of which takes input, produces an output, and then uses that output as the input for the subsequent neuron. However, processing relatively long sequences limits the accuracy and precision, making it challenging to obtain acceptable performance. Fraud detection is the process of recognizing and evaluating unusual behavior or trends in order to potentially uncover fraudulent activity. Fraud detection may be efficiently handled by LSTM, a type of neural network that is able to capture the long-term dependency connection of sequence data. Through input, forgetting, and output gates, the LSTM controls information flow and adjusts layer and unit states. Finally, the hidden layer state is the output of the next time step and is transmitted to the LSTM model.

Implement details:

We first serialize the transaction data and use it to build the LSTM, so that we can train the LSTM model that can predict potential fraud behavior, carry out further learning and analysis of the sequence, keep iterating, and finally judge whether there is fraud in this instance through the output of the testset, and return 0 or 1 to us.

We defined a function: **RNN_train(X_train, y_train, X_test, y_test).**
A function is developed to make RNN model training easier. It receives the characteristic data of the training and test sets (X_train and X_test) as well as the labels (y_train and y_test) as input. Using StandardScaler for feature scaling, the training and test sets must first be standardized. By standardizing the features to have a zero mean and unit variance, this phase aims to enhance the model's performance and convergence rate. Secondly, the sequence model is constructed. Two

LSTM layers, each with 64 cells, are added to the model. The first LSTM layer has return_sequences=True, which means it outputs the entire sequence, not just the output from the previous time step. The input shape is determined based on the feature size of the training kit. A dense layer of a single output unit and a signal activation function is then added to the model. Since the problem is binary and there are two possible outcomes (0 and 1), the signal activation function is appropriate. For optimization,Adam is chosen as the optimizer, and binary_crossentropy is chosen as the loss function for this model. The model is trained by the fit() method. To match the input shape of the model, the training data is reformatted accordingly.

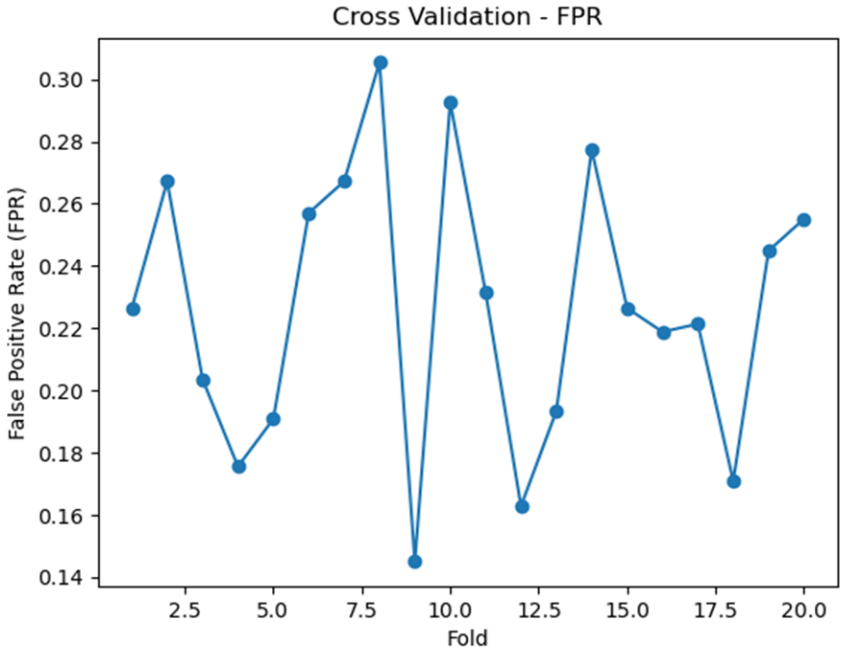
Then, we used a trained model to predict the test set and utilized the np.round() method to convert the result to continuous prediction by using the binary classification process. Then we use the confusion matrix to determine the components, which are true positives (tp), true negatives (tn), false positives (fp), and false negatives (fn). Lastly, the function will return the F1 score and the false negatives (fn).

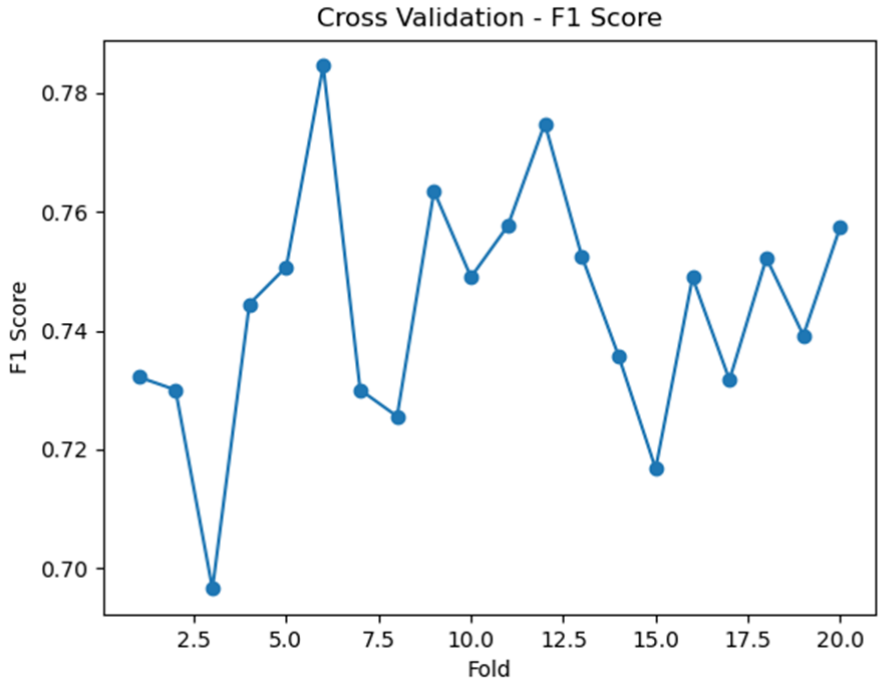
Additionally,To train the LSTM, we also created a function called **rnn_fit(X, part, cv)**, which was invoked in main.py. Three inputs are required for the function to function: X (data sets with features and labels), part (% of training sets), and cv (number of folds in cross-validation). The train_test_split function, which calculates the percentages of the training sets depending on the portion's parameters, is then used to split the data sets into X_train and X_test. The label columns from the training and test sets should then be deleted, and they should be stored in the y_train and y_test variables. The training and test sets' features are thermally encoded using OneHotEncoder. A process called db.one_hot is used, depending on whether the parameter Settings requires fitting a single thermal encoder (ohe).

Evaluation Metric:

Preface: Due to the limited computing resources, it is impossible to show the true capability of LSTM. Theoretically, LSTM can make its own hierarchy more complex, so as to better train and predict the data. However, due to the limitations of our equipment and resources, we only defined a relatively simple model. So it's not particularly impressive in terms of the forecast results. But given enough resources, LSTM must be a very powerful predictive model for our task.

When I choose the portion of non-fraud rate is 0.08% and the fraud portion is 80%. Also, choose the k of the k-fold is 20. We can get the figures about the FPR and F1 score:





According to the above figures: the average FPR is 0.2256 and the average F1 score is 0.74365. We can see that for the relatively simple LSTM, there are some limitations in the accuracy of prediction.

Optimize suggestion:

If you want to increase the performance of LSTM model, you can adjust the following options:

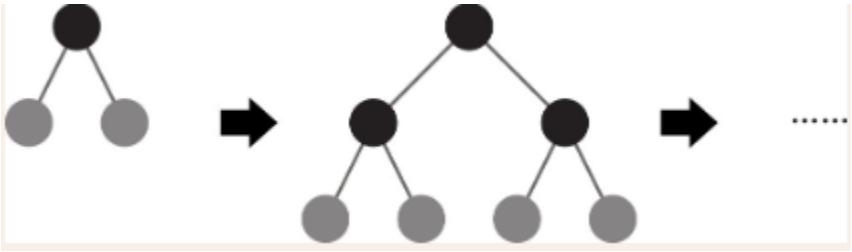
1. Increase the amount of data: If feasible, getting more training data may help model performance. For instance: we can grow the fraud instances and decrease the non-fraud instances (0.03 non-fraud and 0.8 fraud instances). Another feasible method is to combine the variant and base datasets to increase the overall instances.
2. Model adjusting and tuning the LSTM model's hyperparameters: The LSTM model's hyperparameters are tuned via grid search, random search, or Bayesian optimization. To increase model performance, experiment with the learning rate, batch size, number of iterations, and other hyperparameters. Such as increasing/decreasing the number of hidden layers, adjusting the size of hidden layers, and how they are connected. For example, we can add more hidden layers to improve the processing ability of LSTM model. At the same time, we can also add or drop the number of neurons. Also, when we create the optimizer, we can add a learning rate and adjust it to get a better result.
3. Model integration: Consider integrating multiple LSTM models, such as through integration techniques such as voting, averaging, or stacking, to leverage the strengths of different models to improve overall performance.

LightGBM:

Introduction:

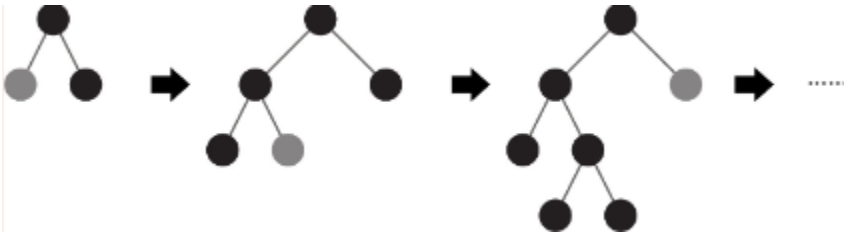
LightGBM (Light Gradient Boosting Machine) is a new member of the Boosting algorithm, developed by Microsoft. It uses the negative gradient of the loss function as a residual approximation of the current decision tree to fit a new decision tree. Its main idea is to use weak classifiers (decision trees) to iteratively train to get the optimal model, which has the advantages of good training, not easy to overfit, distributed support, and quick deal with massive data.

Most of the decision tree algorithms use a level-wise growth strategy, i.e., the leaf nodes in the same layer are split together every time, as shown in the figure below, but some leaf nodes have lower split gain, so splitting will increase the overhead.



Level-wise Growth Strategy

LightGBM algorithm uses a leaf-wise growth strategy, each time in the current leaf nodes to find the largest split gain leaf nodes for splitting, rather than all nodes that are split, as shown in the figure below, to improve the accuracy. However, the leaf-wise strategy is prone to overfitting when the sample size is small, and the LightBGM algorithm can prevent overfitting by limiting the depth of the tree with the parameter max_depth.



Leaf-wise Growth Strategy

Model Building:

After we have performed data reading and simple preprocessing, we can both introduce the LightGBM classification model LGBMClassifier to start the prediction model construction. The initial modeling uses default values for all parameters, and after the model is built, the test set of data is predicted, and the first 5 prediction results are shown below.

predict value	actual value
0	0
0	0
0	1
1	1
0	0

1 -fraud
0-non fraud

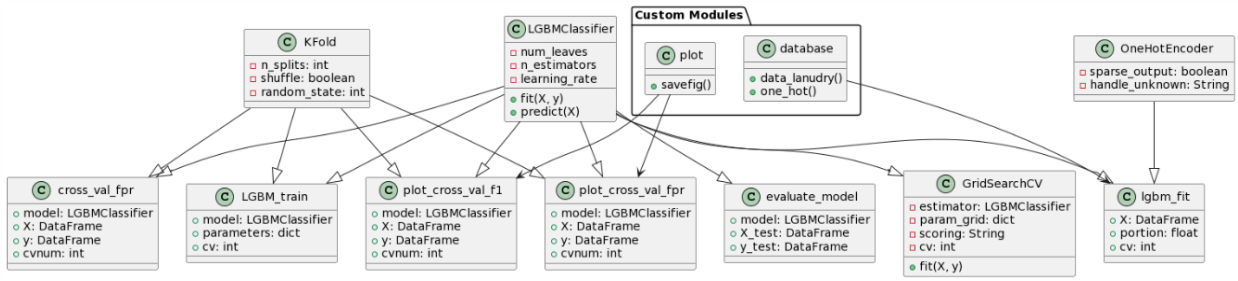
The LGBMClassifier, in essence, predicts not the exact 0's and 1's classification, but the probability of a sample belonging to a certain classification, the following are the probabilities of the first five predicted outcomes:

```
          0         1
[[0.89032957 0.10967043]
 [0.55876072 0.44123928]
 [0.86518846 0.13481154]
 ...
 [0.69277437 0.30722563]
 [0.7734531  0.2265469 ]
 [0.83953123 0.16046877]]
```

1 -fraud
0-non fraud

We used the score() function that comes with the LGBMClassifier to see the prediction accuracy of the model. Our calculated model accuracy score is 0.8157, i.e., the model accuracy reason 81.57%.

The class diagram of the code implementation of the model is as follows:



Model Tuning:

A. GridSearch: There are many parameters of the LightGBM classification model, here the following parameters are selected for tuning. Because the LightGBM model uses a leaf-wise growth strategy and has a large amount of training data, the common parameter used to regulate the complexity of the tree is:

- num_leaves instead of the maximum depth parameter of the tree, max_depth. The range of candidate values is [10,15,31].

- n_estimators: indicates the number of weak learners, whose default is 100. The range of candidate values is [10,20,30].

- learning_rate: Weight reduction factor for each weak learner, whose default is 0.1. Taking a smaller value means that more iterations and more soft learners are needed to achieve a certain number of misclassifications or learning effects. The range of candidate values is [0.05,0.1,0.2].

We then used GridSearch for parameter tuning and cross-validated the model 10 times using the F1 value as the model evaluation criterion. The final optimal values of the parameters obtained are {'learning_rate': 0.2, 'n_estimators': 30, 'num_leaves': 15}. That is, for the data in this case, the model predicts best when the weight reduction factor of the weak learner is set to 0.1, the maximum number of iterations is set to 20, and the maximum number of leaf nodes of the decision tree is set to 15.

The accuracy score at this point is 0.8162. We want to do secondary optimization by increasing the tuning parameter range, but because GridSearch is too time-consuming to run, we choose Optuna, which is lightweight and more powerful.

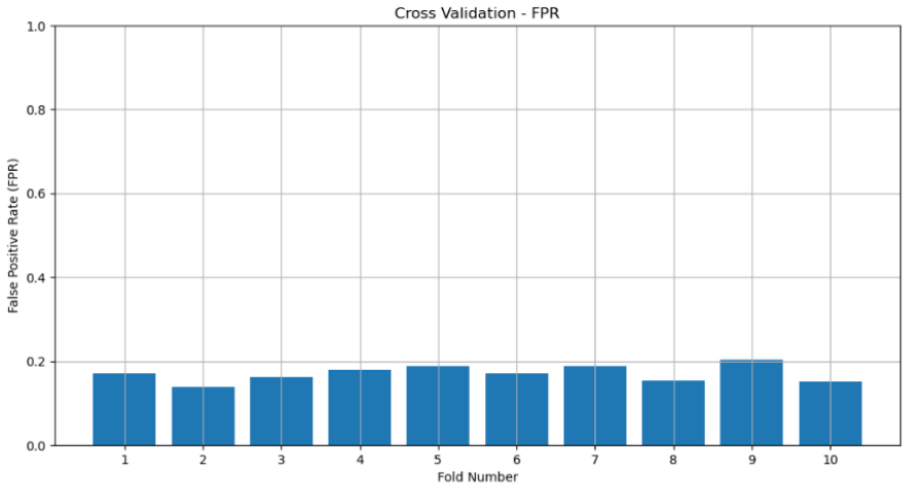
B. Optuna: The optimization process will perform 300 trials, each trying a different combination of hyperparameters, and guide subsequent searches based on the results of the F1 score. The selection interval for hyperparameters is shown in the table below.

Hyperparameter Name	Type	Range	Description
n_estimators	Integer	10 to 10,000	The number of trees, or iterations, in the model.
max_depth	Integer	3 to 30	The maximum depth of a tree, limiting growth to prevent overfitting.
learning_rate	Float	0.02 to 0.2	The learning rate, controlling the step size at each iteration while moving toward a minimum of a loss function.
num_leaves	Integer	10 to 100	The maximum number of leaves per tree.
boosting_type	Categorical	'gbdt', 'goss'	The type of boosting algorithm to use: Gradient Boosted Decision Trees ('gbdt') or Gradient-based One-Side Sampling ('goss').
min_data_in_leaf	Integer	5 to 200	The minimum number of samples per leaf node.
max_bin	Integer	100 to 500	The maximum number of bins that feature values will be bucketed in.
enable_bundle	Boolean	True, False	A flag to enable or disable bundling (grouping) of features to improve efficiency.

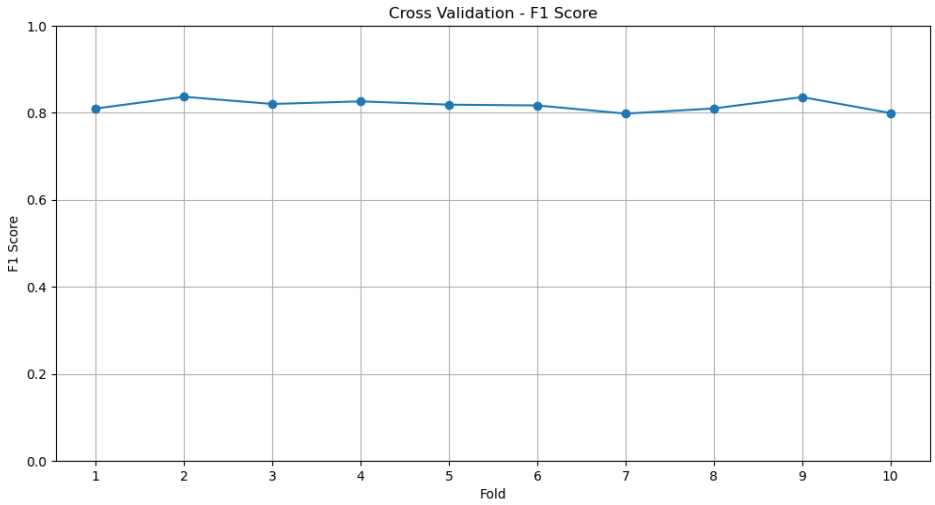
The final optimal values of the parameters obtained are {'n_estimators': 385, 'max_depth': 5, 'learning_rate': 0.03695584373736057, 'num_leaves': 10, 'boosting_type': 'goss', 'min_data_in_leaf': 32, 'max_bin': 392, 'enable_bundle': True}.

Evaluation Metric:

We cross-validated the model 10 times to evaluate the model, and the following plots show the changes in best FPR and best F1 scores.



Average False Positive Rate across 10 folds: 0.1717



Average F1 Score across 10 folds:0.8281

The results before and after the optimization are shown in the table below. The accuracy score, average FPR and average F1 score are all improved compared to the pre-tuning period, which indicates that the model's prediction has become better.

	accuracy score	average FPR	average F1 score
Before tuning	0.8157	0.1892	0.7842
After GridSearch tuning	0.8162 ↑	0.1857 ↓	0.7908 ↑
After Optuna tuning	0.8215 ↑ ↑	0.1717 ↓↓	0.8281 ↑ ↑

Optimize suggestion:

- According to the feature_importances_party that comes with the LGBMClassifier, the original features are ranked in terms of importance, and the features with low importance are reduced to improve the convergence time of the model and the convergence result.

Conclusion:

Model comparison:

	F1 score	FPR score
Random forest	0.7794	0.1636
Logistic Regression	0.7823	0.1745
SVM	0.7921	0.1707
RNN	0.7437	0.2256
LGBM	0.8281	0.1717

In this project, we tested the bank account fraud dataset and used several machine learning models: Logistic Regression, SVM, Random Forest, RNN, and LGBM. After experiments and evaluations, we came to the following conclusions:

-The Random Forest model can reduce the risk of overfitting a single decision tree by integrating the prediction results of multiple decision trees, improving the accuracy of the overall model, and obtaining more stable and reliable prediction results. It is also able to better model nonlinear relationships by capturing interactions between features through the splitting process of decision trees. However, the step of feature importance evaluation is missing, so the score in the f1-score is not high.

-The main advantage of the Logistic Regression model in the project is its simplicity and interpretability. Based on the principle of linear regression, it can provide clear coefficient interpretation when there are few features and there is a linear relationship between the features and the target variable. However, because it cannot capture the nonlinear relationships and complex feature interactions in the project, its performance is mediocre.

-The SVM model performs well when dealing with high-dimensional data and non-linear problems, and has also achieved good results for bank account fraud detection. It maps samples into high-dimensional space through kernel functions, which not only can handle nonlinear problems, but also can handle more high-dimensional data effectively than the random forest model, especially in this project.
In bank account fraud detection, fraud is usually a minority category and unbalanced sample distribution is common. SVM uses techniques such as undersampling or oversampling to deal with sample imbalance problems and improve the detection ability of rare fraud events. Therefore, in addition to taking the longest to train, it performs best among our models.

-The RNN model is a deep learning model suitable for sequence data processing, which can capture the temporal dependencies in time series data. Since fraudulent behaviors are usually temporally correlated, RNN can predict current fraudulent behaviors by memorizing previous states. However, because the RNN model lacks more data samples and a more complex network architecture to further improve performance when processing project tasks, its performance is not very good.

-The LGBM model achieves high accuracy and recall rates in fraud detection through the gradient-boosting decision tree method. Due to the lack of computing resources and the long optimization time, the Optuna optimization process was performed for only 300 trials, so the best combination of parameters may not have been found, the model gets only a minor boost.

Code base: see appendix B.

Further improvements:

- Model integration: try to use ensemble learning which means that it could use bagging or boosting to combine the predictions of different models.
- Feature engineering: feature selection and feature engineering can be done to remove the attributes irrelative or redundant. Or create some new features which could better describe data characteristics.
- Hyperparameter tuning: Optimize the performance and generalization ability of the model by adjusting the model's hyperparameters
- Increase the amount of data: If feasible, getting more training data may help model performance, like combining the variant and base datasets to increase the overall instances. But this also requires higher computing power, which is a condition that we are currently difficult to meet.

Appendix:

Appendix A: Label for csv file

Each dataset with labels:

- **fraud_bool:** Fraud label (1 if fraud, 0 if legit)
- income: Annual income of the applicant in quantiles. Ranges between [0, 1].
- **name_email_similarity:** Metric of similarity between email and applicant's name. Higher values represent higher similarity. Ranges between [0, 1].
- **prev_address_months_count:** Number of months in the previous registered address of the applicant, i.e. the applicant's previous residence, if applicable. Ranges between [−1, 380] months (-1 is a missing value).
- **current_address_months_count:** Months in the currently registered address of the applicant. Ranges between [−1, 406] months (-1 is a missing value).
- **customer_age:** Applicant's age in bins per decade (e.g, 20-29 is represented as 20).
- **days_since_request:** Number of days passed since application was done. Ranges between [0, 78] days.
- **intended_balcon_amount:** Initial transferred amount for application. Ranges between [−1, 108].
- **payment_type:** Credit payment plan type. 5 possible (anonymized) values.
- **zip_count_4w:** Number of applications within the same zip code in last 4 weeks. Ranges between [1, 5767].
- **velocity_6h**: Velocity of total applications made in last 6 hours i.e., average number of applications per hour in the last 6 hours. Ranges between [−211, 24763].
- **velocity_24h**: Velocity of total applications made in last 24 hours i.e., average number of applications per hour in the last 24 hours. Ranges between [1329, 9527].
- **velocity_4w**: Velocity of total applications made in last 4 weeks, i.e., average number of applications per hour in the last 4 weeks. Ranges between [2779, 7043].
- **bank_branch_count_8w**: Number of total applications in the selected bank branch in last 8 weeks. Ranges between [0, 2521].
- **date_of_birth_distinct_emails_4w**: Number of emails for applicants with the same date of birth in the last 4 weeks. Ranges between [0, 42].
- **employment_status**: Employment status of the applicant. 7 possible (anonymized) values.
- **credit_risk_score**: Internal score of application risk. Ranges between [−176, 387].
- **email_is_free**: Domain of application email (either free or paid).
- **housing_status**: Current residential status for applicant. 7 possible (anonymized) values.
- **phone_home_valid**: Validity of provided home phone.
- **phone_mobile_valid**: Validity of provided mobile phone.
- **bank_months_count**: How old is the previous account (if held) in months. Ranges between [−1, 31] months (-1 is a missing value).

- **has_other_cards**: If the applicant has other cards from the same banking company.
- **proposed_credit_limit**: Applicant's proposed credit limit. Ranges between [200, 2000].
- **foreign_request**: If the origin country of the request is different from the bank's country.
- **source**: Online source of application. Either browser(INTERNET) or mobile app (APP).
- **session_length_in_minutes**: Length of user session in banking website in minutes. Ranges between [−1, 107] minutes
- **device_os**: Operative system of device that made a request. Possible values are: Windows, Macintox, Linux, X11, or other.
- **keep_alive_session**: User option on session logout.
- **device_distinct_emails_8w**: Number of distinct emails in banking websites from the used device in the last 8 weeks. Ranges between [0, 3].
- **device_fraud_count**: Number of fraudulent applications with used devices. Ranges between [0, 1].
- **month**: Month where the application was made. Ranges between [0, 7].

Appendix B:

https://github.com/COMP-7409A-Group12/Fraud-detection-on-BAFsuite-of-datasets.git

Reference:

Jesus, S., Pombal, J., Alves, D., Cruz, A. F., Saleiro, P., Ribeiro, R. P., Gama, J., & Bizarro, P. (2022, November 25). *Bank Account Fraud Dataset Suite (NeurIPS 2022)*. Kaggle. Retrieved November 28, 2023, from https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022?select=Variant+V.csv