Machine Learning Mini Project – Credit Card Fraud Detection

Group 2- Dov

Project Objective

This project aims to compare different models for detecting credit card fraud. Furthermore, I will optimize the model using hyperparameter tuning, feature engineering and ensemble techniques.

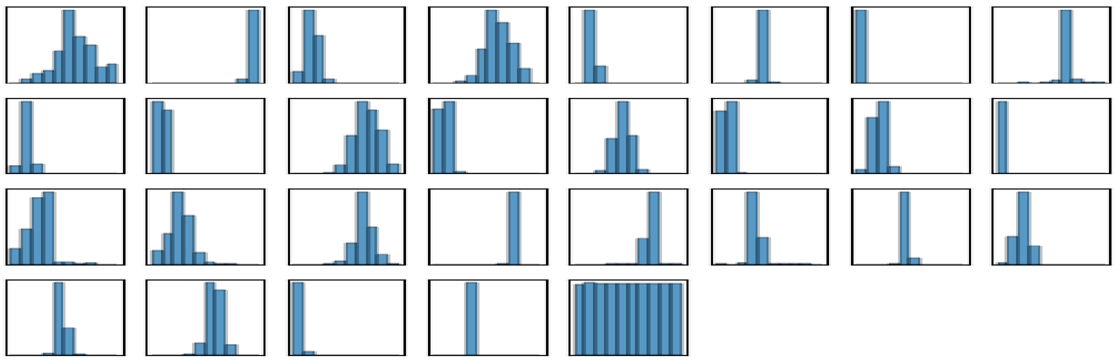
Dataset Description

The dataset was downloaded from Kaggle which is including over 569,000 credit card transaction records from European cardholders, consisting of 31 columns. Apart from id, amount and class (a binary label, where 1 indicates a fraud case and 0 indicates a normal case), there are 28 columns named V1 to V28. These are anonymized features representing various transaction attributes (e.g., time, location, etc.).

Data Preprocessing

No null values or duplicate row were found. The "id" column was dropped as it is not useful for analysis. I created X (which include all features except "Class") and y (which contains "Class" only). Further, 80% of the data is used for training and 20% for testing.

Since most features show a normal distribution, standardization was applied instead of normalization. Additionally, because the sample sizes of fraud cases and normal cases are balanced, resampling is not necessary.



Methodology

I utilized supervised learning classification methods. The models chosen for this project include Logistic Regression (LR), K Nearest Neighbors (KNN), SVM (Support Vector Machine) and Random Forest (RF) as they are common classification models.

For evaluation, I used accuracy, F1-score, precision and recall. Precision measures the rate of false positive cases, which can lead to unnecessary costs and time for banks to validate those cases. Recall assesses the rate of false negative cases, which can negatively impact customers and result in financial and reputational losses for banks. Accuracy and F1-sroce provide a balance between precision and recall.

Below table is the summary of the performance of 4 models:

	Accuracy	Precision	False Positive	Recall	False Negative	F1-score
LR	0.9651	0.9775	1252	0.9524	2714	0.9648
KNN	0.9979	0.9958	242	1.0000	0	0.9979
SVM	0.9971	0.9966	194	0.9975	141	0.9971
RF (Test data)	0.9999	0.9998	14	1.0000	0	0.9999
RF (Train data)	1.0000	1.0000	0	1.0000	0	1.0000

Overfitting

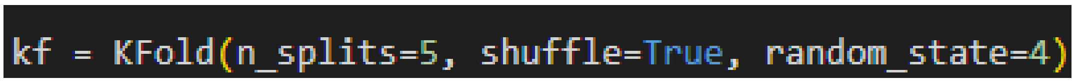
Although all models perform very well, especially Random Forest, overfitting may pose a potential issue for this high performance. To minimize the risk of overfitting, I employed performance comparisons between training and testing data, as well as K-Fold Cross-Validation.

1. ***Performance Comparison between Train and Test Data***

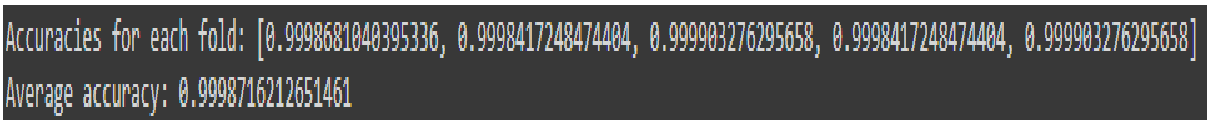
From the table above, the performance of RF is excellent for both training and testing datasets. This suggest that the model generalizes well to unseen data.

2. ***K-Fold Cross-Validation***

In the K-Fold Cross-Validation, I used 5 for n_splits and a different random_state for testing. An average accuracy of 0.9999 was obtained. This indicates that the model is robust and not likely overfitting.



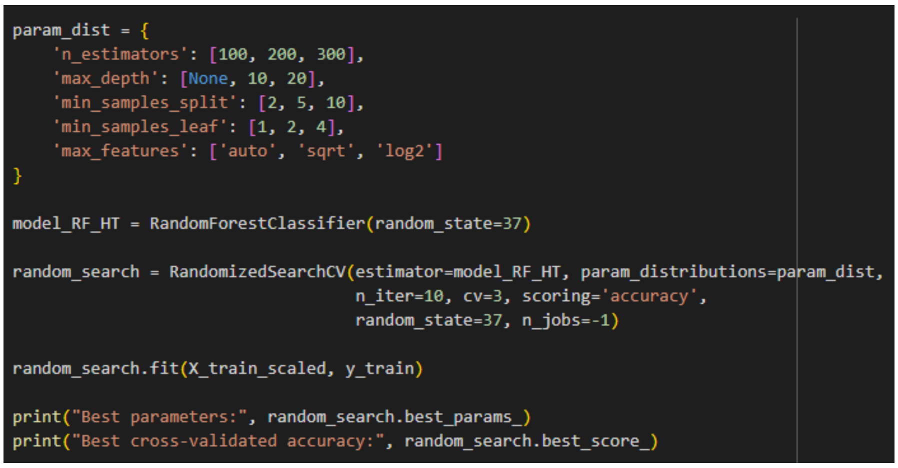
```
kf = KFold(n_splits=5, shuffle=True, random_state=4)
```



```
Accuracies for each fold: [0.9998681040395336, 0.9998417248474404, 0.999903276295658, 0.9998417248474404, 0.999903276295658]
Average accuracy: 0.999871621265l461
```

Hyperparameter Tuning

Given that the model demonstrates strong performance without significant overfitting, the next step is to optimize it though hyperparameter tuning. Since training the model can be time-consuming, I opted for 10 iterations of random search instead of grid search.



```
param_dist = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

model_RF_HT = RandomForestClassifier(random_state=37)

random_search = RandomizedSearchCV(estimator=model_RF_HT, param_distributions=param_dist,
                                   n_iter=10, cv=3, scoring='accuracy',
                                   random_state=37, n_jobs=-1)

random_search.fit(X_train_scaled, y_train)

print("Best parameters:", random_search.best_params_)
print("Best cross-validated accuracy:", random_search.best_score_)
```

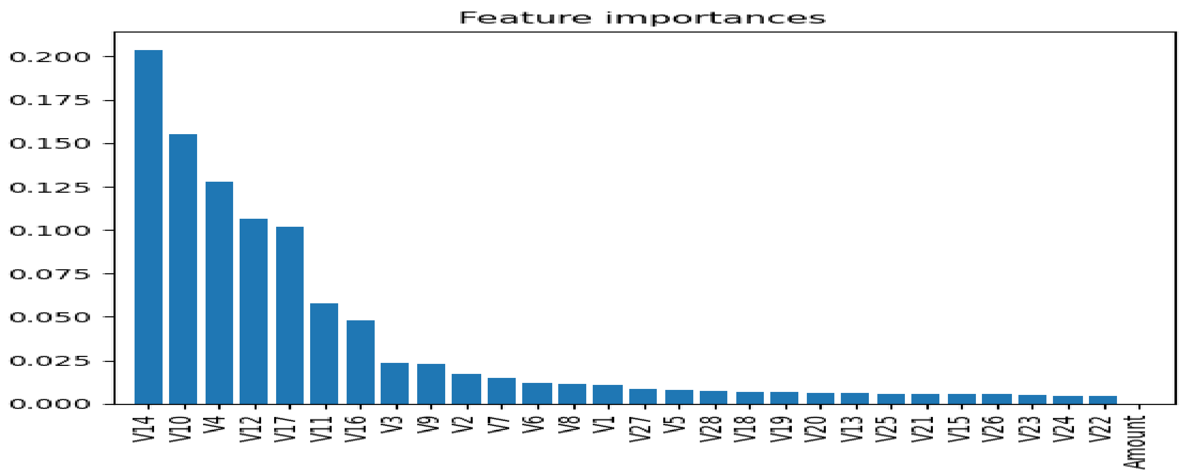
The final results:

'n_estimators': 100
'min_samples_split': 5
'min_samples_leaf': 1
'max_features': 'sqrt'
'max_depth': None

These parameters are similar to the default values, except for min_samples_split, which has been adjusted from the default of 1. The accuracy after tuning remains at 0.9999, which is identical to the accuracy before tuning. This may be due to the similar parameter values.

Feature Engineering – Feature Selection

After hyerparameter tuning, feature selection was applied to reduce the number of features, thereby simplifying the model. This can reduce the cost of training and running the model, as well as decrease the risk of overfitting and improve generalization.



To find the optimal balance between number of features and performance, six cut-off importance scores were evaluated: 0.006, 0.008, 0.01, 0.012, 0.02 and 0.024. The corresponding feature reduction resulted in 8, 14, 15, 18, 20 and 22 features, respectively. All configurations performed well, and the summary of results can be found in the table below.

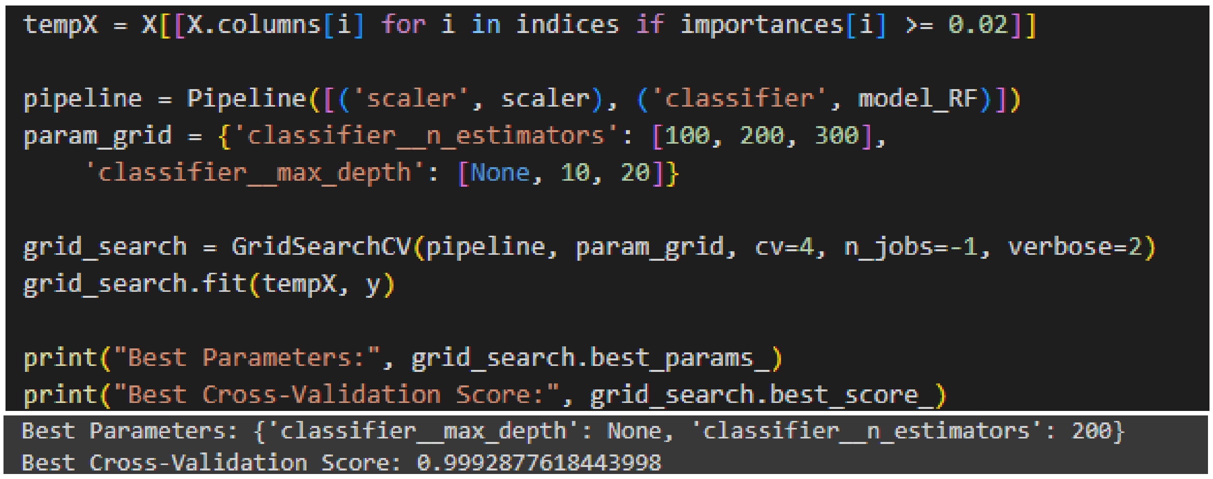
Compared to different cut off score, it seems 0.02 as cut off score is the best choice as it reduces 20 out of 29 features (69%) and get the highest accuracy.

Cut off Score	Feature Remained	Feature Reduced	Accuracy	False Positive	False Negative
0.006	21	8	0.9992	17	71
0.008	15	14	0.9985	19	156
0.01	14	15	0.9987	21	128
0.012	11	18	0.9973	34	274
0.02	9	20	0.9995	37	15
0.024	7	22	0.9992	49	45

Final K Fold & Hyperparameter Tuning

After feature selection, K-Fold Cross-Validation and hyperparameter tuning were conducted again to ensure generalization and find the new parameter settings. With 20 features removed, the training time of the model decreased significantly. Therefore, I opted for grid search instead of random search. However, to avoid long running times, I decided to adjust 2 parameters, each with 3 values. The parameters chosed were ‘n_estimators’ and ‘max_depth’, as they are the most important for the Random Forest model. Using 4 folds leads to a total 3 x 3 x 4 = 36 runs.

The best parameters are found as 'classifier__max_depth': None, and 'classifier__n_estimators': 200. The accuracy score is 0.9993.



```
tempX = X[[X.columns[i] for i in indices if importances[i] >= 0.02]]

pipeline = Pipeline([('scaler', scaler), ('classifier', model_RF)])
param_grid = {'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [None, 10, 20]}

grid_search = GridSearchCV(pipeline, param_grid, cv=4, n_jobs=-1, verbose=2)
grid_search.fit(tempX, y)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
Best Parameters: {'classifier__max_depth': None, 'classifier__n_estimators': 200}
Best Cross-Validation Score: 0.9992877618443998
```

Comparison with Original Model

	Number of features	% change	Accuracy	% change	False Positive	% in Total	False Negative	% in Total
Original Model	29	N/A	0.9999	N/A	14	0.02%	0	0
Final Model	9	-69%	0.9996	-0.03%	32	0.06%	13	0.02%

The final model utilizes 9 features (a reduction of 69%), but there is only a minimal performance drop of less than 0.1%.

Challenges and Limitations

1. **Model Selection, Hyperparameter Tuning & Feature Engineering**
   There are numerous methods to conduct analysis, hyperparameter tuning and feature engineering. Given my limited time and knowledge, there may be better approach for this project.
2. **Overfitting Problem**
   The performance of all models is nearly perfect, which may be indicative of overfitting. I need to implement various strategies to reduce the likelihood of overfitting.
3. **Unknown Meaning of Feature (V1 – V28)**
   Due to the privacy protection, no column names are provided for V1 to V28. This limitation prevents me from performing further feature engineering to transform them into more useful features.
4. **Data Source Limitation**
   Since the dataset comes from records of European cardholders, it may not generalize well to other regions, such as Asia.

Future Work

1. **Use Data from Different Countries**
   By utilizing data from various countries, I can create tailored made models for specific regions. Additionally, I can compare the patterns of fraudulent transactions across different areas and develop a generalized model applicable to all countries.
2. **Testing More Possibilities**
   I plan to explore more models, hyperparameters, feature combinations andm ensemble techniques in order to identify a better model with lower costs.

Conclusion

Though simple comparison, the best-performing model in this project is the Random Forest, followed by KNN, SVM and finally Logistic Regression (which still performs very well).

Performance comparison between training and testing data, along with K-Fold Cross-Validation, were employed to ensure high generalization and low probability of overfitting.

After multiple rounds of K-Fold Cross-Validation, hyperparameter tuning and feature selection, the Random Forest model was tuned to maintain high accuracy (with less than a 0.06% performance drop) while reducing the features count by 69% (from 29 features to 9 features). This is satisfactory outcome, as it also saves time and cost for training and running the model.

Interesting the amount of transaction was identified as the least important feature in the feature importance analysis. This finding may warrant further exploration in future work.

Reference

https://scikit-learn.org/stable/modules/cross_validation.html

https://www.kaggle.com/code/willkoehrsen/intro-to-model-tuning-grid-and-random-search

https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023