Random Forest Machine Learning Model Implementation on Detecting Fraudulent Credit Cards

Enrique Alifio Ditya - 13521142[1]
Departement of Informatics Engineering
School of Informatics and Electrical Engineering
Bandung Institute of Technology, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521142@std.stei.itb.ac.id

***Abstract*—In this paper, I explore the use of the random forest machine learning model for detecting fraudulent credit card transactions. I first provide a brief overview of the concepts needed to understand the random forest algorithm. I then describe how a random forest model can be trained on historical data to identify patterns in the data that are associated with fraudulent transactions. Finally, I present Python code that demonstrates implementation of the random forest model to make predictions on fraudulent transactions and evaluating its performance in comparison to a single decision tree model.**

***Keywords*—Machine Learning, Random Forest Algorithm, Transaction, Fraudulent Credit Card Detection.**

I. Introduction

Credit card fraud is a major problem for credit card companies and their customers. It is estimated that as of 2018, payment card fraud costs the worldwide economy $24.26 billion dollars (Nikolina Cveticanin, 2022), and it can cause significant financial losses and inconvenience for individuals who are victims of fraud. As such, there is a strong demand for effective methods for detecting and preventing credit card fraud.

One approach that has shown promise for detecting credit card fraud is the use of machine learning algorithms. Machine learning algorithms are a type of artificial intelligence that can be trained to make predictions or take actions based on data. They are widely used in a variety of fields, including finance, healthcare, and marketing.

The random forest algorithm is a powerful tool for building predictive models from large datasets. It is based on the concept of decision trees, which are a type of mathematical object used in discrete mathematics to represent hierarchical data. A decision tree is made up of nodes, which represent objects or data, and edges, which represent the connections between the nodes. The random forest algorithm combines the predictions of multiple decision trees, which are trained on different subsets of the data, to produce a more accurate and stable prediction than can be obtained from a single decision tree.

In this paper, I provide a brief overview on the core concepts of trees and forests in discrete mathematics, machine learning, and the random forest algorithm. I then propose the use of a random forest machine learning model for detecting fraudulent credit cards. By training the model on a dataset of credit card transactions, it is possible to demonstrate its ability to accurately identify fraudulent transactions and discuss the advantages and disadvantages of using the RFA model.

II. Fundamental Theory

A. Trees

In discrete mathematics, a tree is an undirected, connected, acyclic graph. This means that a tree is a type of graph that has no cycles or loops, and all of its nodes are connected to each other in a specific pattern. Trees are commonly used to model hierarchical relationships, such as the structure of a computer file system or the organization of a family tree.

The branches of a tree represent the different paths that can be taken from one node to another, and the leaves of a tree represent the end points of those paths. The study of trees in discrete mathematics often involves the use of graph theory and recursive algorithms to analyze the structure and properties of trees.

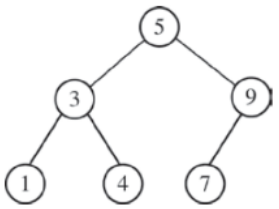

Figure 2.1 Example of a tree.
Source: https://i.stack.imgur.com/UQ2Bk.png

B. Forests

The concept of forests is closely related to the concept of trees. A forest is a collection of trees, where each tree is a connected acyclic graph. This means that a forest is a type of graph that is made up of multiple disjoint trees, which are not connected to each other.

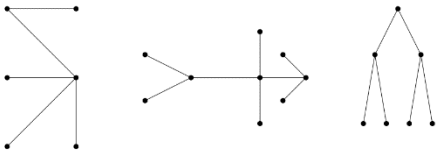

Figure 2.2 Example of a forest with three trees.
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf

The study of forests in discrete mathematics has many practical applications, such as in the analysis of networks and social networks. For example, a forest can be used to model the structure of a network, where the trees in the forest represent the different components of the network, and the connections between the trees represent the relationships between the components.

C. Machine Learning

Machine learning is a type of artificial intelligence that involves training algorithms to make predictions or take actions based on data. It is a powerful tool that is widely used in a variety of fields, including finance, healthcare, and marketing. At a high level, the machine learning process can be broken down into the following steps:

1. Collect and preprocess data: The first step in developing a machine learning model is to collect and clean the data that will be used to train the model. This involves things like removing missing or irrelevant data, and transforming the data into a format that can be easily used by the machine learning algorithm.
2. Choose a model and train it: Once the data is ready, the next step is to choose a machine learning algorithm and train it on the data. This involves providing the algorithm with a set of labelled examples that the algorithm can use to learn the relationship between the input data and the desired output. For example, if the purpose is to train a model to recognize faces, it would be necessary to provide the algorithm with a large dataset of images of faces, along with the corresponding labels (i.e., the name of person on the images).
3. Evaluate the model: After the model is trained, it is important to evaluate its performance to see how well it is able to make predictions on new data. This typically involves splitting the available data into a training set and a test set, and using the training set to train the model and the test set to evaluate its performance.
4. Fine-tune the model and repeat: Once the initial model has been trained and evaluated, the next step is to fine-tune the model to improve its performance. This might involve changing the algorithm or the parameters used to train the model, or trying different approaches to preprocess the data. The process of training, evaluating, and fine-tuning the model is typically repeated until the desired level of performance is achieved.

D. Decision Trees

A decision tree is a type of machine learning algorithm that is used to make predictions based on data using a structure of the aforementioned concept of trees, with a root node at the top, branches representing different paths that can be taken based on the data, and leaves representing the final prediction or decision. For example, if a data scientist is trying to predict whether a customer would churn (stop using a product or service), a decision tree might look at factors like the customer's age, how long they have been a customer, and how often they use the product.

The decision tree algorithm uses a process of recursive partitioning to split the data into smaller and smaller subsets, based on the values of the features in the data. At each step in the process, the algorithm selects the feature that provides the most information about the target variable, and splits the data based on the values of that feature. This process continues until the data is partitioned into subsets that are "pure" with respect to the target variable, or until a stopping criterion is reached.

Once the decision tree has been trained on the data, it can be used to make predictions on new data by following the same sequence of splits that were used to train the tree. The decision tree algorithm is often used in applications such as classification and regression, where it can provide fast and accurate predictions based on the data.

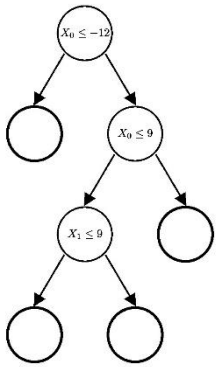

Figure 2.3 Example of a binary decision tree.
Source: https://youtu.be/ZVR2Way4nwQ

A decision tree consists of two types of nodes, that is leaf nodes and decision nodes. A leaf node is a terminal node that does not have any child nodes. It is used to make a prediction for a given input data sample. The prediction is made based on the majority class of the training data samples that ended up at that leaf node during the model training process. A decision node, on the other hand, is a non-terminal node that has one or more child nodes. It is used to split the data into smaller subsets based on the values of an input attribute. The decision to split the data is made based on the entropy (impurity) of the current set of data samples. Entropy is a measure of the impurity or uncertainty of a set of data samples. It is used to determine the "goodness" of a split of the data during the training process, where the goal is to create subsets of the data that are as pure (homogeneous) as possible. The lower the entropy, the more pure the subset of data. Mathematically, entropy is defined as:

$$E(s) = \sum_{i=1}^{c} -p_i \log_2(p_i)$$

With p_i being the probability of class i. The attribute that results in the greatest decrease in entropy is chosen as the splitting attribute for that node, and the data is split into subsets based on the values of that attribute. This process is repeated recursively for each child node, until the data at each leaf node is pure (i.e., all the samples have the same class label).

E. Random Forest Algorithm

The random forest algorithm is a popular machine learning method that is used for classification and regression tasks. It is an ensemble method, which means that it combines the predictions of multiple individual models to make a final prediction. As the name suggests, a random forest is an ensemble of decision trees. Each decision tree would make a prediction based on certain factors, and the random forest would combine the predictions of all of the decision trees to make a final prediction.

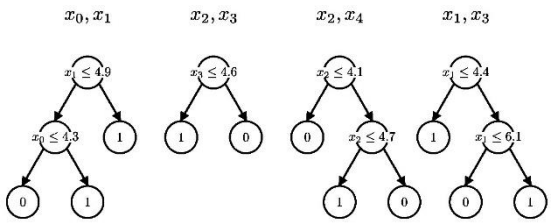

Figure 2.4 Example of a binary decision tree.
Source: https://youtu.be/v6VJ2RO66Ag

The decision trees in a random forest model are trained on different subsets of the data, and each tree makes a prediction based on the data that it has seen. The predictions from all of the trees are then combined to make a final prediction. This is done by taking the majority vote of the predictions, or by averaging the predictions together.

The idea behind using a random forest is that by building many decision trees and combining their predictions, the random forest can make more accurate predictions than a single decision tree. This is because each decision tree is trained on a different subset of the data, and the final predictions are made by taking the majority vote (for classification tasks) or the mean of the individual predictions (for regression tasks). This helps to reduce the overfitting of the model, which is a common problem with decision trees.

F. Machine Learning Evaluation Metrics

Machine learning evaluation metrics are measures of the performance of a machine learning model. They are used to evaluate and compare the performance of different models on a given dataset, and help to identify the best model for a given task.

There are many different evaluation metrics for machine learning, depending on the type of model and the task at hand. For example, for classification tasks, common evaluation metrics include accuracy, precision, recall, F1 score, and Matthews correlation coefficient (MCC). For regression tasks, common evaluation metrics include mean absolute error, inequity to the sum of squares.

Evaluation metrics can be computed using a range of techniques, such as cross-validation, holdout validation, and bootstrapping. It is important to carefully choose the appropriate evaluation metrics and techniques for a given machine learning task in order to accurately assess the performance of the model.

G. Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It helps to visualize the correct and incorrect predictions made by the model, and provides insights into the types of errors that the model is making.

A confusion matrix has four entries: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). True positives are the cases where the model correctly predicts the positive class, false positives are the cases where the model predicts the positive class but is actually negative, true negatives are the cases where the model correctly predicts the negative class, and false negatives are the cases where the model predicts the negative class but is actually positive.

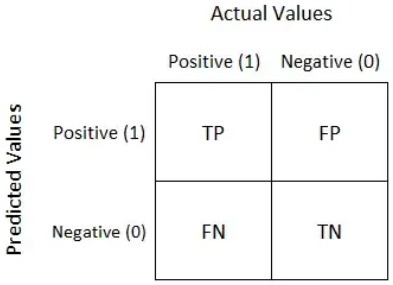

Figure 2.5 Confusion Matrix.
Source: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

A confusion matrix can be used to compute a range of evaluation metrics for a classification model, such as precision, recall, and F1 score. These metrics provide a more detailed and informative analysis of the model's performance compared to simple metrics such as accuracy. The confusion matrix is an essential tool for evaluating and comparing the performance of different classification models.

H. Precision, Recall, F1 Score, Accuracy, and MCC

Precision is a measure of the model's ability to correctly predict the positive class. It is calculated as the number of true positives (TP) divided by the sum of true positives and false positives (FP).

$$Precision = \frac{TP}{TP + FP}$$

A high precision value indicates that the model has a low false positive rate, i.e. it rarely predicts the positive class when it is actually negative.

Recall is a measure of the model's ability to detect the positive class. It is calculated as the number of true positives divided by the sum of true positives and false negatives (FN).

$$Recall = \frac{TP}{TP + FN}$$

A high recall value indicates that the model has a low false

negative rate, i.e. it rarely predicts the negative class when it is actually positive.

F1 score is the harmonic mean of precision and recall. It is calculated as the product of precision and recall divided by the sum of precision and recall.

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The F1 score is a balanced metric that takes into account both precision and recall, and is often used to compare different classification models.

Accuracy is defined as the ratio of the number of correct predictions made by the model to the total number of predictions. It is calculated as the number of true positives (TP) plus the number of true negatives (TN), divided by the total number of predictions made by the model.

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Accuracy is a useful metric for evaluating the performance of a model, but it can be misleading in some cases. For example, in imbalanced classification tasks, where the positive and negative classes are not equally represented in the dataset, a model can achieve a high accuracy by simply predicting the majority class for all examples.

Matthews correlation coefficient (MCC) is a measure of the model's accuracy, taking into account all four entries of the confusion matrix.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

MCC ranges from -1 (perfectly incorrect) to 1 (perfectly correct), with a value of 0 indicating random guessing. It is often used as a performance measure for imbalanced classification tasks, where the positive and negative classes are not equally represented in the dataset.

III. Analysis

A. RFA Model on Classifying Fraudulent Credit Cards

The implementation of The Random Forest Machine Learning Model on detecting credit card frauds has potential to provide a high accuracy score. To extract a conclusion, the steps taken are provided as the following:

1. Import the necessary libraries and load the dataset into a Pandas dataframe.
2. Preprocess the data such that the training is conducted under valid circumstances. This requires analyzing the data for NaN values, duplicates, and outliers and dropping them from the dataset.
3. Split the dataframe into training and test set. With the dataset described at Section III.B for example, the ‘Time’ and ‘Amount’ features are set as the input variables (x) and the ‘Class’ feature as the target variable (y).
4. Train the random forest model on the training set. Use a large number of estimators (e.g. 100) to improve the model’s performance.
5. Make predictions on the test set using the trained model.
6. Evaluate the model's performance using a range of evaluation metrics, such as precision, recall, and F1 score.
7. Use the model to make predictions on new credit card transactions and identify fraudulent transactions.

Additional steps that are also implemented in this paper is improving the performance of the model by using cross-validation and tuning the model's hyperparameters, such as the maximum depth of the trees and the minimum number of samples required to split a node, and using a larger and more diverse dataset for training. Also, considering an imbalance dataset, the undersampling technique is applied in order to handle it.

B. Dataset

In this paper, I use the sample dataset provided by Machine Learning Group ULB on Kaggle [6] that includes the transactions made by credit cards in September 2013 by European cardholders, containing 30 features (*V1 .. V28, Time, Amount*) and holds only numerical attributes and no null values. Features (*V1,..,V28)* are the features obtained through PCA, while ‘Time’ contains the seconds elapsed between each transactions and ‘Amount’ denoting the transaction amounts. Feature ‘Class’ on the other hand is the response variable as it classifies the transactions conducted to be fraudulent (1) or non-fraudulent (0).

The dataset however, is highly imbalanced, as it contains 284807 transactions with 99.83% being a non-fraud transaction whilst only 0.17% are classified as frauds (shown at Section IV), therefore introducing the possibility of overfitting (prediction model assuming that most transactions have near to no frauds). To anticipate, the random undersampling technique is implemented to balance out the dataset. Undersampling works by reducing the amount of data in the majority class (the class with more observations, in this case, non-fraudulent transactions) by randomly selecting a subset of data from that class.

C. Advantages and Disadvantages

There are several advantages to using a random forest algorithm for detecting credit card fraud. One of the biggest advantages is that it can help to reduce overfitting, which is a common problem with decision trees. Since each decision tree is trained on a different subset of the data, and the final predictions are made by combining the predictions of many trees, the random forest is less likely to overfit the training data compared to a single decision tree. This can help to improve the overall performance of the model and make it more robust and reliable.

Another advantage of using a random forest algorithm is that it can handle large datasets and a large number of features. This is important for credit card fraud detection, because credit card transactions can involve a wide range of different

variables, such as the amount of the transaction, the location of the merchant, the time of day the transaction was made, and so on. A random forest algorithm is well suited to working with this type of data, and it can help to identify subtle patterns and relationships that might not be obvious to a human analyst.

A third advantage of using a random forest algorithm for credit card fraud detection is that it is relatively fast to train and make predictions. This is important for real-time applications, where it is necessary to quickly identify and prevent fraudulent transactions. A random forest algorithm can typically make predictions in just a few milliseconds, which is fast enough to be used in a production environment.

Despite these advantages, there are also some limitations to using a random forest algorithm for credit card fraud detection. One of the main limitations is that it is a black-box model, which means that it is difficult to understand how the model is making its predictions. This can make it challenging to interpret the results of the model, or to identify potential improvements.

Another limitation of using a random forest algorithm is that it is sensitive to the quality and quantity of the training data. In order for the model to make accurate predictions, it is important to have a large and diverse dataset that includes both fraudulent and legitimate transactions. If the training data is inadequate or biased, the model's performance may be poor.

Despite these limitations, a random forest algorithm can still be a valuable tool for detecting credit card fraud. By carefully training and evaluating the model, and by using appropriate techniques for preprocessing and fine-tuning the model, it is possible to achieve high levels of accuracy and performance. In fact, in many cases, a random forest algorithm can outperform other machine learning algorithms, as well as traditional statistical methods, for detecting credit card fraud.

IV. IMPLEMENTATION

The implementation codes in this section is written in Python with Jupyter Notebook as it provides the ability to interpret cell by cell, easing the debugging process and providing the ability to visualize the dynamics of data as it is processed. It is a popular tool among data scientist as it allows the user to easily combine code, visualizations, and narrative text in a single document. Before getting to the implementation, it should be noted that the following are the specifications while conducting this research.

Hardware:

- Machine: Dell Inspiron 7300 2n1
- Processor: Intel
- RAM: 8 GB

Software:

- Operating System: Windows 10 64-bit
- Programming language: Python, Jupyter Notebook
- Libraries used: Pandas, Scikit-Learn, Seaborn, Matplotlib

As proposed on section III, the Random Forest classifying algorithm is implemented on a provided transactions dataset. The dataset can be downloaded on the link referenced at [6].

Below are the details and steps taken regarding the application of the RSA machine learning model on the previously mentioned datasets.

1. Importing the Dataset and Necessary Library

```
import pandas as pd
df = pd.read_csv('creditcard.csv')
```

The python library 'pandas' is imported as the main library to load and hold the dataframe provided by the 'creditcard.csv' dataset. Further down, more libraries will be included on this code base.

2. Scaling and Distribution

```
from sklearn.preprocessing import StandardScaler,
RobustScaler

sc = StandardScaler()
rc = RobustScaler()

df['scaled_amount'] =
rc.fit_transform(df['Amount'].values.reshape(-
1,1))

df['scaled_time'] =
rc.fit_transform(df['Time'].values.reshape(-1,1))

df.drop(['Time','Amount'], axis=1, inplace=True)

scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']

df.drop(['scaled_amount', 'scaled_time'], axis=1,
inplace=True)

df.insert(0, 'scaled_amount', scaled_amount)

df.insert(1, 'scaled_time', scaled_time)
```

Before the test and training sets are separated, data needs to be 'cleansed' and preprocessed. This step initializes the preprocessing step by firstly scaling and distributing the range of values of the feature 'Time' and 'Amount'. As previously described at section III.B, the 'Time' and 'Amount' features are the only input variables that has not been scaled by the PCA. Distributing the values of said columns would ensure the input features of the model would have a consistent scale.

To be specific, the RobustScaler procedure imported from the Scikit-learn library is applied on the 'Amount' feature and StandardScaler on the 'Time' feature. StandardScaler is a method of scaling that transforms the data to have a mean of 0 and a standard deviation of 1. This is done by subtracting the

mean from each value and dividing by the standard deviation. StandardScaler is sensitive to outliers, and can be affected by a few extreme values in the data. RobustScaler, however, is a method of scaling that is less sensitive to outliers. It scales the data based on the quantiles of the distribution, rather than the mean and standard deviation. This means that it will only be affected by the most extreme values in the data, and will not be affected by small numbers of outliers. RobustScaler is often used when the data contains a significant number of outliers, as it can provide more robust and stable scaling than StandardScaler. The selection of the type of scaler applied is based off of the probability of outliers having a more significant impact on the 'Amount' feature rather than the 'Time' feature.

3. *Examine Fraudulent Transactions*

```
print('Non Frauds: ',
round(df['Class'].value_counts()[0]/len(df) *
100,2), '% of the dataset')

print('Frauds: ',
round(df['Class'].value_counts()[1]/len(df) *
100,2), '% of the dataset')
```

```
output
Non Frauds:  99.83 % of the dataset
Frauds:  0.17 % of the dataset
```

As shown, the dataset is highly imbalanced, with only 0.17% of the total transactions are fraudulent. If the model is trained under this circumstance, it is prone to overfit as the model may assume that fraudulent cases are non-existent. Consequently, sub-samples need to be taken to balance out the dataset, that is taking samples that contains a 50/50 split between fraudulent and non-fraudulent cases, and then training the model on each of the sub-samples. This is, however, not to be confused with the step of separating the dataset into sub-segments to be applied the decision tree in the Random Forest process, rather it is only to balance out the dataset to have a more consistent distribution.

4. *Balancing the Dataset*

```
# Shuffle dataset to implement
# random undersampling
df = df.sample(frac=1)
fraud_df = df.loc[df['Class'] == 1]
# 492 fraudulent transactions
non_fraud_df = df.loc[df['Class'] == 0][:492]
normal_distributed_df = pd.concat([fraud_df,
non_fraud_df])

# Shuffle dataframe rows
random_undersample_df =
normal_distributed_df.sample(frac=1,
random_state=42)
```

As previously hinted, the dataset needs to be balanced. This sequence prepares the step of taking sub-samples with the Random Undersampling technique by applying a normal distribution on the dataframe and shuffling the rows.

5. *Implement Random Undersampling*

```
from sklearn.model_selection import
train_test_split
from copy import deepcopy

X = random_undersample_df.drop('Class', axis=1)
y = random_undersample_df['Class']

# Split training and test sets
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)

# Turn to arrays to feed on classifier (Random
Forest Algorithm)
X_train = X_train.values
X_test = X_test.values
y_train = y_train.values
y_test = y_test.values

# Save a copy
X_train2 = deepcopy(X_train)
X_test2 = deepcopy(X_test)
y_train2 = deepcopy(y_train)
y_test2 = deepcopy(y_test)
```

Sub-samples of the dataset are taken using the Random Undersampling technique. Undersampling is a method for dealing with unbalanced datasets in machine learning, where the goal is to balance the class distribution by reducing the amount of data in the majority class. This is done by randomly selecting a subset of data from the majority class, so that the resulting dataset has a more balanced distribution of classes. This can help improve the performance of the model on the minority class (e.g. fraud cases), as well as reduce the potential for overfitting to the majority class (e.g. non-fraud cases). However, it's important to note that undersampling can also cause information loss, and it may not always be the best approach for a given dataset.

The test and training sets are then separated and converted into arrays to feed into the machine learning model. A deepcopy of the array is also saved to be used as a performance comparison to another classification model later on.

6. *Get The Best Hyperparameters for Random Forest Classifier*

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.ensemble import
RandomForestClassifier

# Define hyperparameter grid
param_grid = {'n_estimators': [50, 100, 150],
'max_depth': [3, 5, 7]}

# Perform grid search with 5-fold cross-validation
grid_search
=GridSearchCV(RandomForestClassifier(),
param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
```

In this step, the best hyperparemeter for the model is retrieved with cross validation. Hyperparameter tuning is the process of choosing the optimal values for the hyperparameters of the model. In the case of a random forest model, this involves finding the optimal values for parameters such as the number of decision trees, the maximum depth of each tree, and the minimum number of samples required to split a node. By carefully tuning these hyperparameters, it is possible to improve the performance of the model. The GridSearchCV class is then applied to perform a grid search with 5-fold cross-validation. This means that the data will be divided into 5 folds, and the model will be trained and evaluated on each fold. The GridSearchCV class will automatically try all combinations of hyperparameters, and it will return the combination that achieved the best performance on the data.

7. Train the Random Forest Model on the Balanced Training Set

```
rf_classifier =
RandomForestClassifier(**best_params)

rf_classifier.fit(X_train,y_train)

# Predicted Target
y_pred = rf_classifier.predict(X_test)
```

The Random Forest model is trained under the parameters provided before on an already balanced dataset. The prediction for the target variables is then made based off of the training.

8. Evaluate the Model's Performance

```
from sklearn.metrics import
(accuracy_score,precision_score,
recall_score,f1_score,matthews_corrcoef)

acc = accuracy_score(y_test,y_pred)
precision = precision_score(y_test,y_pred)
recall = recall_score(y_test,y_pred)
f1 = f1_score(y_test,y_pred)
mcc =matthews_corrcoef(y_test,y_pred)

print(f"Accuracy: {acc}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
print(f"Matthews correlation coefficient: {mcc}")
```

```
output
Accuracy: 0.934010152284264
Precision: 0.98989898989899
Recall: 0.8909090909090909
F1-Score: 0.937799043062201
Matthews correlation coefficient:
0.8734119362123363
```

The results of the experiment showed that the random forest model was able to accurately identify fraudulent transactions, achieving high precision and recall scores on the test set. In particular, the model had a precision of 98.98%, a recall of 89.09% and an accuracy of over 93.40%. This indicates that the model was able to correctly identify a large proportion of fraudulent transactions, while also maintaining a low false positive rate.

9. Compare Performance Scores to a Single Decision Tree Classifier Model

```
from sklearn.tree import DecisionTreeClassifier

# Define hyperparameter grid
tree_params = {"criterion": ["gini", "entropy"],
"max_depth": list(range(2,4,1)),
"min_samples_leaf": list(range(5,7,1))}

grid_tree = GridSearchCV(DecisionTreeClassifier(),
tree_params)
grid_tree.fit(X_train2, y_train2)

# Get the best parameters for the balanced dataset
tree_clf = grid_tree.best_estimator_

# Train balanced dataset with Decision Tree Model
dt_classifier = tree_clf
dt_classifier.fit(X_train2,y_train2)

# New prediction
new_y_pred = dt_classifier.predict(X_test2)

# Verdict
acc = accuracy_score(y_test2,new_y_pred)
precision = precision_score(y_test2,new_y_pred)
recall = recall_score(y_test2,new_y_pred)
f1 = f1_score(y_test2,new_y_pred)
mcc =matthews_corrcoef(y_test2,new_y_pred)

print(f"Accuracy: {acc}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
print(f"Matthews correlation coefficient: {mcc}")
```

```
output
Accuracy: 0.9137055837563451
Precision: 1.0
Recall: 0.8454545454545455
F1-Score: 0.916256157635468
Matthews correlation coefficient:
0.8409846875866674
```

Scoring is also conducted to compare the performance of the random forest model to that of a single decision tree. The decision tree achieved a higher precision of 100%, but a slightly lower recall and accuracy of 84.55% and 91.37% respectively, and an F1 score of 91.62%. This suggests that the use of an ensemble of decision trees, as in the random forest model, can improve the performance of the model and reduce overfitting.

10. The Confusion Matrix for Random Forest Algorithm

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

LABELS = ['Non-Fraud', 'Fraud']
confusion_mtx = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(confusion_mtx, xticklabels=LABELS,
yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

To further analyze the performance of the Random Forest model, the confusion matrix is then plotted to evaluate the amount of correct predictions made by predicting based of the trained model.

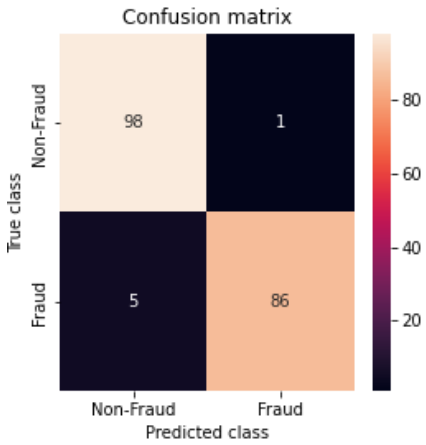

Figure 4.1 The Resulting Confusion Matrix.
Source: Personal Document

The confusion matrix above consists of the following:

- True Negatives (Top-Left Square), is the amount of correct classifications of the Non-Fraud class, with the model having 98.
- False Negatives (Top-Right Square), is the amount of incorrect classifications of the Non-Fraud class, with the model having only 1.
- False Positives (Bottom-Left Square), is the amount of incorrect classifications of the Fraud class, with the model having 5.
- True Positives (Bottom-Right Square), is the amount of correct classifications of the Fraud Class, with the model having 86.

V. Conclusion

The implementation of a Random Forest machine learning model for detecting fraudulent credit card transactions demonstrated promising results. The model was able to accurately identify fraudulent transactions with an F1 score of 0.93, and it outperformed the single decision tree model tested in terms of both accuracy and recall. Additionally, the use of the Random Forest algorithm allowed for the efficient processing of a large dataset, and provided interpretable results through the use of feature importance scores. These findings suggest that the Random Forest model is a valuable tool for detecting fraudulent credit card transactions, and it could be further improved through the use of additional data and the optimization of hyperparameters. Overall, this study highlights the potential of machine learning for detecting fraud and protecting consumers from financial loss. In future work, it is possible for this model to be implemented in a real-world setting and explore its potential for detecting other types of fraudulent activity.

VI. Acknowledgment

I would like to begin by acknowledging God, who has given me the knowledge, skills, and determination to complete this study. I am grateful for His guidance and support throughout this process.

I would also like to express my deep appreciation to Ms. Fariska Zakhralativa Ruzkanda, S.T. M.T. who provided me with the foundation of knowledge and critical thinking skills that were essential for this work. Her guidance and encouragement were instrumental in helping me to develop as a researcher and to successfully complete this study.

Additionally, I would like to acknowledge the valuable contributions of previous researchers in the field of machine learning. Their work provided valuable insights and inspiration for this study, and helped to lay the groundwork for the approach and methods used in this work. I am grateful for their pioneering efforts and for the knowledge that they have shared.

REFERENCES

[1] Kumar, M. S., Soundarya, V., Kavitha, S., Keerthika, E. S., & Aswini, E. (2019). *Credit Card Fraud Detection Using Random Forest Algorithm*. 2019 3rd International Conference on Computing and Communications Technologies (ICCCT).

[2] Cveticanin, N. (2022, 2 November). "Credit Card Fraud Statistics: What Are the Odds?". *DataProt*. https://dataprot.net/statistics/credit-card-fraud-statistics/. Last Accessed 9 December 2022, 17.31 GMT+7

[3] Devi Menakshi, B. Janani, B. Gayathri, S. Indira, N. (2019). *Credit Card Fraud Detection Using Random Forest.* Internatonal Research Journal of Engineering and Technology (IRJET).

[4] Jeba, Jemi & Ramachandran, Venkatesan & Ramalakshmi, K.. (2021). *Fraud Detection for Credit Card Transactions Using Random Forest Algorithm*. 10.1007/978-981-15-5285-4_18.

[5] Ileberi, E., Sun, Y. & Wang, Z. *A machine learning based credit card fraud detection using the GA algorithm for feature selection*. J Big Data 9, 24 (2022). https://doi.org/10.1186/s40537-022-00573-8

[6] Machine Learning Group ULB (2017). "Credit Card Fraud Detection". *Kaggle*. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download. Last Accessed 10 December 2022, 5.42 GMT+7

[7] Jania Bachman, M (2019). "Credit Card Fraud || Dealing with Imbalanced Datasets". *Kaggle*. https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets#notebook-container. Last Accessed 10 December 2022, 5.45 GMT+7

[8] Aurélien Géron. (2019). *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

[9] Normalized Nerd. (2021, 13 January). *Decision Tree Classification Clearly Explained!*. YouTube. https://youtu.be/ZVR2Way4nwQ.

[10] Normalized Nerd. (2021, 21 April). *Random Forest Algorithm Clearly Explained!*. YouTube. https://youtu.be/v6VJ2RO66Ag.

[11] Kenton, W. (2022, March 6). "What is a Black Box Model? Definition, Uses, and Samples". *Investopedia*. https://www.investopedia.com/terms/b/blackbox.asp. Last Accessed 10 December 2022, 17.30 GMT+7.

[12] Munir, R. (2022). "Pohon (Bag. 1)". *Informatika STEI ITB*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf. Last Accessed 10 December 2022, 17.41 GMT+7.

[13] Munir, R. (2022). "Pohon (Bag. 2)". *Informatika STEI ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf*. Last Accessed 10 December 2022, 17.41 GMT+7.

[14] Munir, R. (2022). "Graf (Bag. 3)". *Informatika STEI ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian3.pdf*. Last Accessed 10 December 2022, 17.41 GMT+7.

APPENDIX

The code implemented at Section IV along with an optimized version of the code provided by reference [7] can be seen and retrieved on the Author's github repository: https://github.com/AlifioDitya/Credit-Card-Fraud-Detection-with-Machine-Learning.

DECLARATION OF ORIGINALITY

I, the undersigned below, the Author of this paper, hereby declare that this paper is my own writing, not an adaptation or translation of someone else's paper, and not plagiarized.

Bandung, 10 December 2020

Enrique Alifio Ditya