Fighting Fraud with Finesse: A Comparative Analysis of the Decision Tree and the Multi-layered Perceptron

Jasper John Adrada[1], Luis Miguel Razon[1], Kate Lynn Rojo[1]
and Kathleen Mae Romblon[1]

[1]De La Salle University – Laguna Campus
727V+352, LTI Spine Road, Laguna Blvd, Biñan, Laguna

Abstract. As credit card fraud is becoming a more common issue in society, the researchers aim to find solutions to determine whether a transaction is fraudulent or not through the use of machine learning. Comparing a decision tree learner and a multi-layered perceptron learner, it is determined that the decision tree learner will have better predictions of a fraudulent transaction as it performed the best in the performance metrics while also being the most resource-friendly and time-efficient to train.

Keywords: Machine learning, Decision tree, Multi-layered perceptron

1 Introduction

Credit card fraud is one of the significant problems faced by society and the financial industry. This problem can lead to financial losses for both customers and credit card companies. Detecting fraudulent transactions instantaneously is vital for the prevention of losses and maintaining the trust of customers. To address the said problem, the dataset chosen provides anonymized transaction data for a credit card company, and the main objective is to build a machine-learning model that can precisely predict fraudulent transactions.

The domain of the problem is related to the fraud of credit cards. Identity theft in the form of credit card fraud occurs when someone steals another person's credit card information without their consent in order to make purchases or withdraw money from their account. An instance of identity theft that involves gaining unauthorized access to an account with the intent of making purchases or withdrawals is the unauthorized use of a credit card [1]. The rapid development of online transactions and the ease with which personal information is available have made this problem increasingly common in the modern world. In the view of Hari Ravichandran, CEO and founder of digital safety company Aura, credit card fraud

continues to rise due to its profitability. As Ravichandran explained, scammers follow the money and see enormous value in credit card fraud and other schemes. The number of scams will continue to rise as long as scammers can commit them and profit from them [2].

Credit card fraud is becoming a common issue in society, which is why it is important to find solutions to minimize its occurrences. The information that is at risk in credit card fraud is not only the money information that the cardholder has, but also their personal information. Besides personal information getting stolen, the losses credit card fraud has amounted to can reach billions of dollars. In 2021, the global loss from credit card fraud was 32.34 billion dollars [3].

Credit Card Fraud is a big problem in online and offline transactions that can cause financial loss to both credit card companies and customers and is also inconvenient when dealing with it. With this, predicting fraudulent transactions may prevent this common type of fraud. There are several possible benefits if the Credit Card Fraud problem is solved. It will help prevent financial loss for both customers and credit card companies. Also, consumers will feel safer using a credit card for transactions and will most likely use it more, which will also benefit from the credit card networks as they will have more customers/cardholders. The aim of the paper is to find possible solutions to credit card fraud with the help of machine learning. A dataset containing data on credit card fraud is used to teach the machine learner so that it will be able to predict whether a transaction is fraudulent or not given the details of the transaction.

2 Credit Card Fraud Dataset

The Credit Card Fraud Dataset [4] contains the necessary features for a fraudulent transaction. The features include *distance_from_home*, which is the distance from the home where the transaction happened, *distance_from_last_transaction*, which is the distance from the last transaction that happened, *ratio_to_median_purchase_price*, which is the ratio of purchased price transaction to median purchase price, *repeat_retailer*, which indicates if the transaction happened from the same retailer, *used_chip*, which indicates if a credit card chip was used to complete the transaction, *used_pin_number*, which indicates whether or not a PIN number was used to complete the transaction, *online_order* to indicate whether or not the transaction was an online order, and finally, *fraud*, to indicate whether or not the transaction is fraudulent.

The label in the Credit Card Fraud Dataset indicates whether each transaction was fraudulent or not. In the dataset, there is a column named “fraud” that contains binary labels signifying if the transaction is fraudulent or not. If the transaction is not fraudulent the value is 0, else if the transaction is fraudulent the value is 1 (see Figure 2.1).

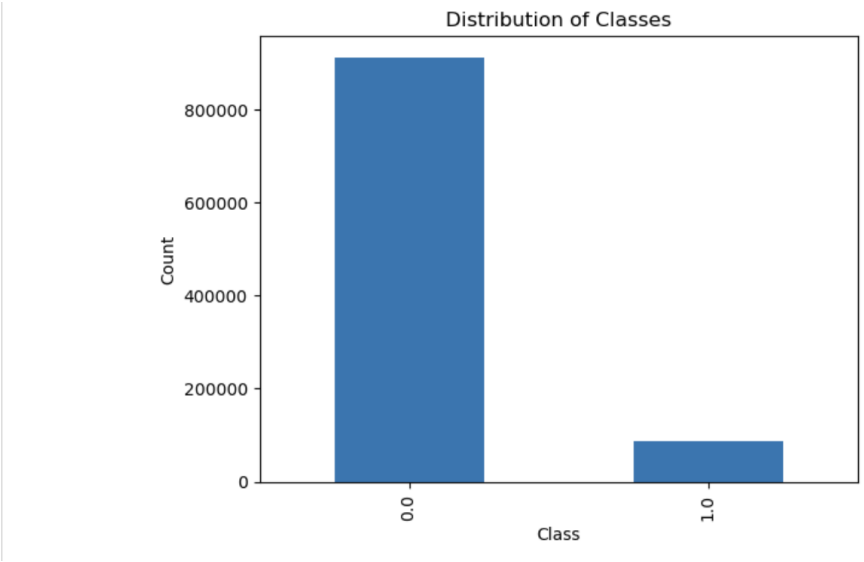

Fig. 2. *Fraud* label in the Credit Card Fraud Dataset

3 Methodology

The machine learning pipeline used to train and compare the dataset using different machine learning algorithms can be seen in Figure 3.1.

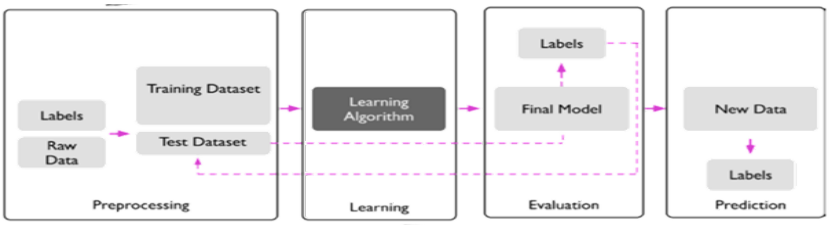

Fig. 3.1. Machine Learning Pipeline

3.1 Preprocessing Phase

This phase is responsible for gathering and processing the dataset. First, the columns from the file containing the dataset must be identified. In the case of the Credit Card Fraud dataset, it can be observed that the features that identify whether a credit card transaction is fraudulent or not can be found in the first columns while the label stating the conclusion can be found in the last column of the dataset (See Figure 3.2). The purpose of this process is to prevent errors or misidentification when the labels and the features need to be identified.

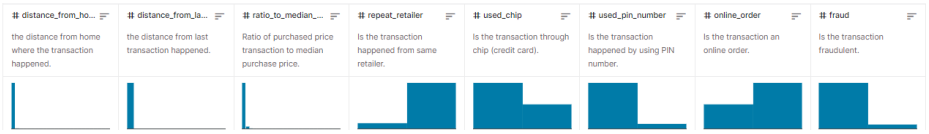

Fig. 3.2. Credit Card Fraud Dataset [4]

Once the appropriate columns were identified, the dataset was then separated into two tables, one table containing the features and the other containing the labels (See Figures 3.3 and 3.4). This process aids in training and comparing predictions with actual labels in the later phases of the pipeline.

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	used_chip	used_pin_number	online_order
0	57.877857	0.311140	1.945940	1.0	1.0	0.0	0.0
1	10.829943	0.175592	1.294219	1.0	0.0	0.0	0.0
2	5.091079	0.805153	0.427715	1.0	0.0	0.0	1.0
3	2.247564	5.600044	0.362663	1.0	1.0	0.0	1.0
4	44.190936	0.566486	2.222767	1.0	1.0	0.0	1.0

Fig. 3.3. Features of the dataset

	fraud
0	0.0
1	0.0
2	0.0
3	0.0
4	0.0

Fig. 3.4. Labels of the dataset

After the raw data has been processed and separated into their respective tables, the next step is to identify the training and test sets which will be used to train and test the model respectively. Scikit learn has a function called *train_test_split* which allowed the features and labels table to be separated into training and test sets fairly through the parameters indicated in Figure 3.5 which shuffled and allocated 20% of all the available data as the test set and the remaining 80% as the training set. This function also ensured fair distribution of classes among the labels where it acquired the 20% of the test data by randomly obtaining 20% of all the data that is flagged as *fraud/1* and 20% that is flagged as *not fraud/0* (See Figure 3.6).

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x_data,
    y_data,
    test_size=0.2,
    shuffle=True,
    random_state=42,
)
```

Fig. 3.5. *train_test_split* function

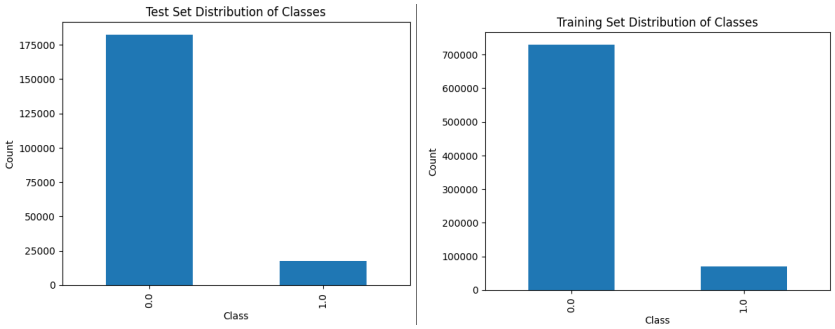

Fig. 3.6. Test & Training Sets Distribution of Classes

3.2 Learning

The learning phase would be responsible for training the models using the training set that was obtained in the previous phase. Two machine learning models will be used in this phase in order determine the strengths and weaknesses of both models by comparing their performance through different metrics.

Decision Tree. The first model that will be implemented is the Decision Tree model. It is a type of supervised machine learning that can be used for regression and classification tasks [5]. In the case of the dataset, the model is used specifically for classification since it only needs to identify certain features which would allow the model to conclude that the transaction is fraudulent. The generation of the decision tree was done with the aid of a package called *DecisionTreeClassifier* from Scikit Learn by calculating the nodes and values of the decision tree by taking in the training set, specifically the training features and labels table, while using hyperparameters that could be set by the user. When it is finished, it outputs the decision tree model with the best fit values and nodes which would be able to predict inputs (See Figure 3.7).

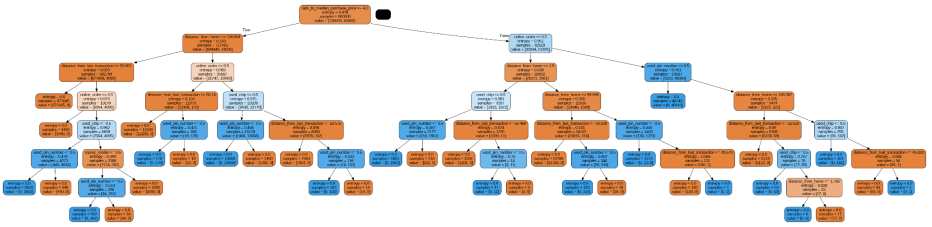

Fig. 3.7. Decision tree with max depth

The decision tree model implementation with the hyperparameters and the fitting of the training set can be seen in Figure 3.8. Hyperparameters required to create the decision tree include *max_depth* which is the maximum depth of the tree, and when adjusted, it might improve efficiency but at the cost of the accuracy if the depth is too low, *random_state* which controls the randomness estimator [6], and *criterion*, which could be set to "gini", "entropy", or "log_loss" which are the possible criteria for measuring the quality of the split of the decision tree.

```
tree_clf = DecisionTreeClassifier(max_depth = 20, random_state = 42, criterion = "entropy")
tree_clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=20, random_state=42)
```

Fig. 3.8. Decision Tree implementation in Python

Multi-layered Perceptron. The second model to be implemented would be the Multi-layered perceptron, which is another machine learning algorithm that also uses supervised learning. Due to the same reasons stated above, it was also used for classification to identify fraudulent credit card transactions. The multi-layered perceptron was implemented using the *MLPClassifier* package from Scikit Learn which calculates and tries to achieve convergence within the dataset by adjusting the weights of the inputs per epoch or iteration.

The multi-layered perceptron implementation with the hyperparameters and the fitting of the training set can be seen in Figure 3.9. The hyperparameters that would be used in training the model are *max_iter* which represents the highest number of epochs that the model can use, and lowering this might make the model train faster but might affect accuracy if the number of iterations is too low, and *random_state* which is similar to the hyperparameter of the same name in the decision tree section above.

```
#Multi-layer perceptron
from sklearn.neural_network import MLPClassifier
mlp800_clf = MLPClassifier(max_iter = 100, random_state = 21)
mlp800_clf.fit(x_train, y_train.values.ravel())
```

```
MLPClassifier(max_iter=100, random_state=21)
```

Fig. 3.9. Multi-layered perceptron implementation in Python

Cross-validation. Cross-validation was performed for both models using the *cross_val_score* function from the *model_selection* library of Scikit Learn (See Figure 3.10). The purpose of cross-validation is to ensure that the model does not overfit and only correctly predict values from the training set. The implementation takes in the model and the features and labels table from the dataset and outputs a score depending on the criteria set. Its hyperparameters are *cv*, which represents the amounts of folds used, where the number of scores returned corresponds to the number of folds, and *scoring*, which is the criteria used as the result, where it is set to "accuracy" so the output gives the accuracy of the model to the data.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_clf, x_data, y_data, cv = 3, scoring = "accuracy")
scores
```

Fig. 3.10. Cross validation score implementation

Performance metrics. In order to determine the best hyperparameters per model as well as compare and evaluate the performance of both models to determine the best one for the dataset, the following performance metrics will be used: *accuracy, precision, recall,* and *F1*.

The data needed for the metrics can be calculated from the confusion matrix that is accessible in the *confusion_matrix* method from the *metrics* library of Scikit Learn (See Figure 3.11). It is also accessible in the same library but with their own respective methods (See Figure 3.12).

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_test_pred)
cm
```

```
array([[182556,      1],
       [     2,  17441]])
```

Fig. 3.11. Confusion matrix

```
from sklearn.metrics import *
precision = precision_score(y_train, y_cross_predict, average = "macro")
accuracy = accuracy_score(y_train, y_cross_predict)
recall = recall_score (y_train, y_cross_predict, pos_label = 1)
f1 = f1_score(y_train, y_cross_predict, pos_label = 1)
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1: {f1}")
```

```
Precision: 0.999978855618342
Accuracy: 0.999985
Recall: 0.9998570611778159
F1: 0.9999142318028476
```

Fig. 3.12. Performance metrics

Hyperparameter optimization. The hyperparameters of the machine learning models will be optimized by slowly adjusting its values in order to find the lowest depth (for decision tree) and the lowest number of iterations (for multi-layered perceptron) that would result in the best results based on the performance metrics and cross validation scores mentioned above.

3.3 Evaluation

After the models are prepared, the next step is to use the models to predict the outcomes based on the features of the test set. This is done by using the *predict* method from each model which takes in the features of the test set as an input and outputs a table of the predicted labels which would be compared to the actual labels table from the test set. These comparisons would also result in the creation of a confusion matrix where the performance metrics of the model is calculated.

Based on the performance metrics gathered from predicting the test set, the model could be evaluated. If the performance metrics of the model produced low results, it means that it has low values across all of the metrics, it is a good indicator that the model is overfit especially if the cross evaluation produced good results. This would mean that readjustment of the hyperparameters in training the model is necessary.

4 Results and Discussion

4.1 Decision Tree

Based on the results below of the predictions of the Decision Tree, it shows that the performance of the model improves as the max_depth increases, with the highest accuracy of 0.999985 which is achieved at the max_depths of 7 to 12. While precision and recall have the highest value of 0.999965856 and 0.999885341, which is also from the *max_depth*s of 7 to 12. Meanwhile, the F1 also increased in precision and recall. As we can see, it maxes out at the *max_depth* of 7 which leads to an optimal

performance on both the training and test sets. Based on these results, it can be concluded that as the value of *max_depth* increases, it can lead to better performance of the decision tree.

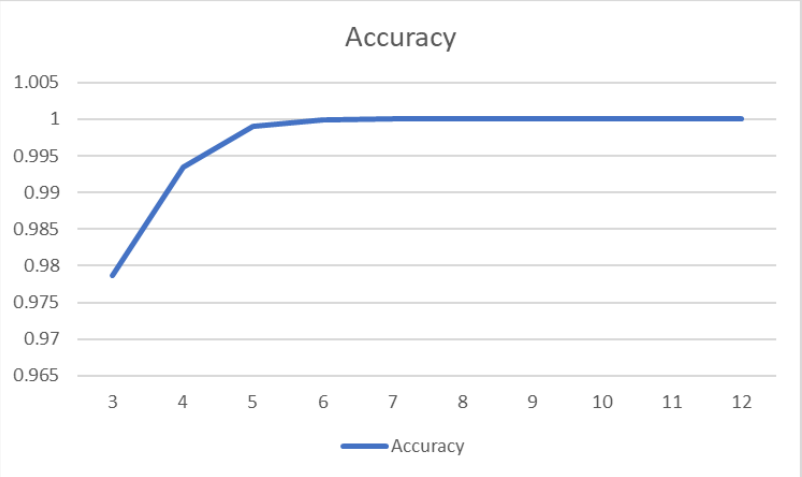

Fig. 4.1. Accuracy graph of the Decision Tree

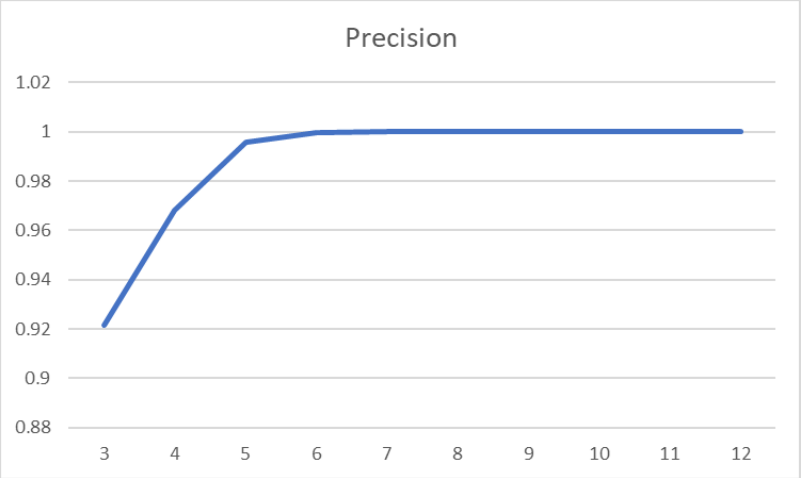

Fig. 4.2. Precision graph of the Decision Tree

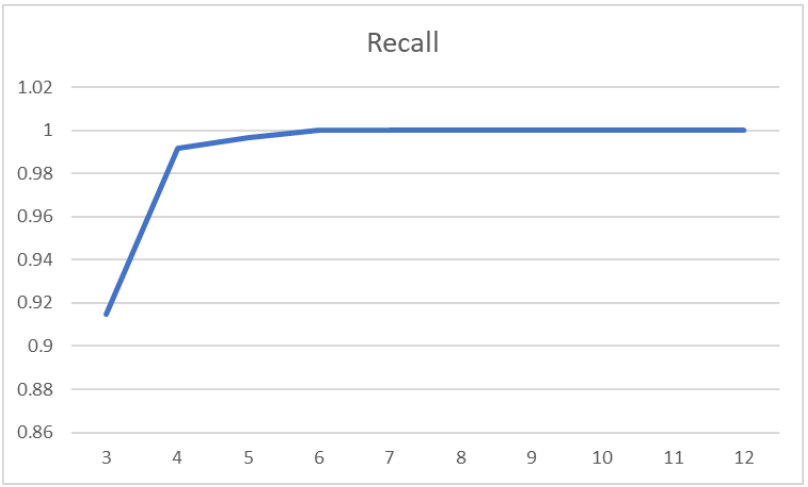

Fig. 4.3. Recall graph of the Decision Tree

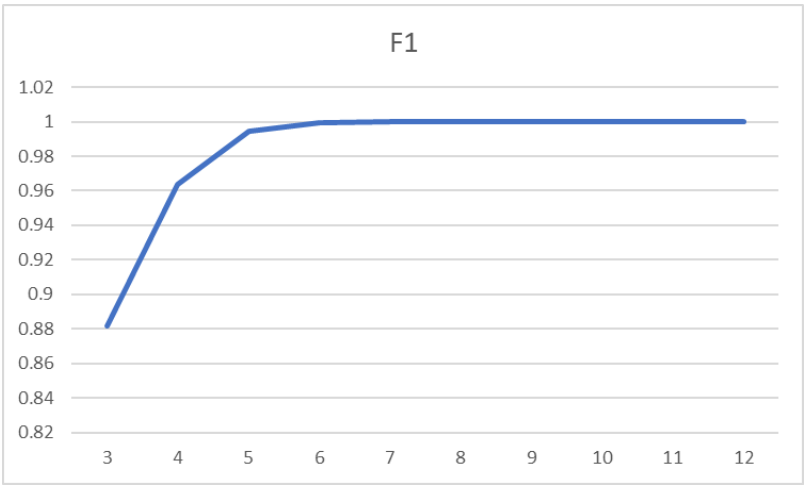

Fig. 4.4. F1 graph of the Decision Tree

4.2 Multi-layered Perceptron

Similar to the decision tree learner, the multi-layer perceptron learner is given a training set with a size of 800,000 data and a test set with a size of 200,000 data. The hyperparameters will be based on the number of iterations or epochs that the machine will test through, which the accuracy of the training and test sets will depend on.

The results of the predictions of the multi-layered perceptron can be seen in Figures 4.5 - 4.8. Based on the results taken, the performance of the model increases as the number of epochs also increases.

It is worth mentioning that the first epoch alone performed well in the performance metrics which started the model with 98.1% in accuracy, 94.8% in precision, 87.9% in recall, and 89.2% in F1. Based on the results of the first epoch, it can be hypothesized that the huge size of the training set contributed to the initial high performance of the model since it is training with a bigger variance of values which could have a positive impact on the accuracy of the decisions of the model in predicting new data.

Another hypothesis would be that the quality of the data is good with minimal noise in the huge dataset which helped with the performance of the model. However, it should be noted that the model was only able to reach convergence when the hyperparameter for maximum epochs was set to 100 but that could also be attributed to the size of the dataset which makes it near impossible to ensure that all the data is perfect without any sort of noise.

With each graph, it can be seen that the overall performance of the model peaks at around the third epoch. From the first to the third epoch, there was a 1% increase in accuracy, a 3.8% increase in precision, an 8.3% increase in recall, and a 7.5% increase in F1. The model only saw minor improvements in performance over the succeeding epochs, even in the 10 to 120 epoch range. Increasing the epochs by extreme amounts only impacted the performance of the model in small ways such as the F1 metric which got a 2% increase from the third epoch to the 120th, recall got a 1% increase, precision also achieved a 1% increase from the third epoch, and accuracy was the least affected of all with only a 0.2% increase in performance.

Although the performance of the model keeps on improving as the amount of epoch increases, it would be safe to conclude that the optimal hyperparameter for training a multi-layered perceptron model using the dataset that classifies fraudulent credit card transactions would be the third epoch because although the performance of the model increases alongside the maximum number of epochs, the effort of training the models makes the later epochs inefficient especially with the low increase in performance compared to the spike in performance which occurred during the third epoch.

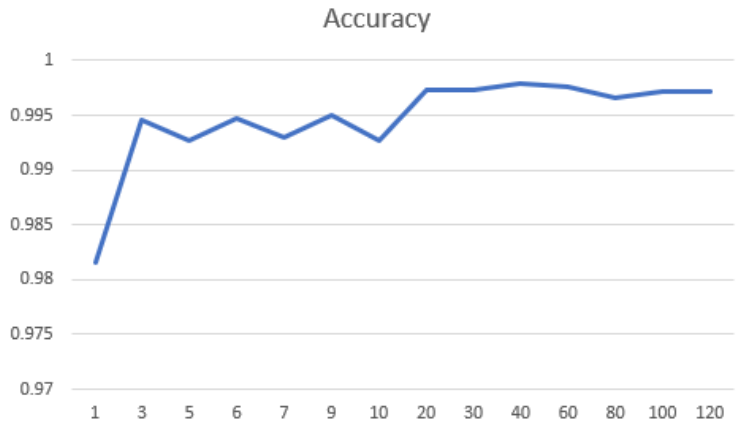

Fig. 4.5. Accuracy graph of Multi-layered Perceptron

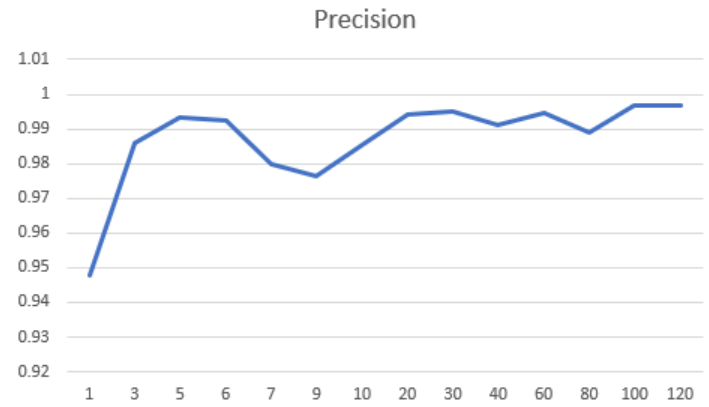

Fig. 4.6. Precision graph of Multi-layered Perceptron

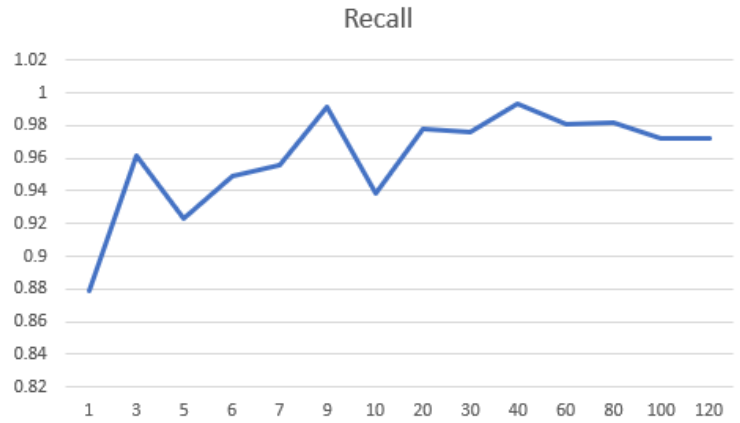

Fig. 4.7. Recall graph of Multi-layered Perceptron

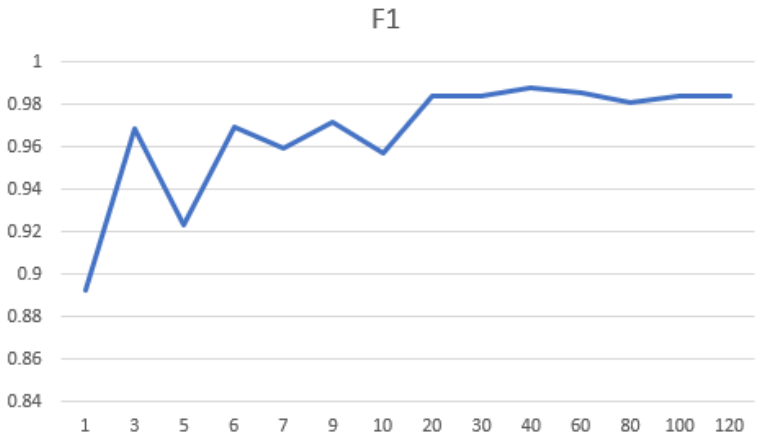

Fig. 4.8. F1 graph of Multi-layered Perceptron

4.3 Comparison of Machine Learning Algorithms

Comparing the best models of the decision tree and multi-layered perceptron models, it would be safe to assume that the decision tree is the superior model in terms of classification, specifically when training the Credit Card Fraud Dataset. As seen in Figures 4.9 and 4.10, the overall performance of the decision tree model is slightly better with improvements ranging from 0.5 to 2% in all the performance metrics compared to the performance of the multi-layered perceptron model.

The reason for the choice of model is also evident when comparing the confusion matrices of both the models since the basis for the difference in performance of the two models lies in their respective confusion matrices. In Figures 4.11 and 4.12, it can be seen that the multi-layered perceptron model has more false positives and false negatives compared to the decision tree model.

Another metric that should be considered is the resources required to train both models. In the previous section, it was seen that at some point, the performance of the multi-layered perceptron model matches that of the performance of the decision tree model. However, in order to get to that point, the maximum epoch hyperparameter of the multi-layered perceptron model must be set to the 40 to 120 epoch range, which heavily affects the time and resources at which the model will be trained, making the decision tree model more efficient as it can achieve a high performance with the least amount of time and resources.

```
#Accuracy of the prediction
from sklearn.metrics import *
precision = precision_score(y_test, y_test_pred, average = "macro")
accuracy = accuracy_score(y_test, y_test_pred)
recall = recall_score (y_test, y_test_pred, pos_label = 1)
f1 = f1_score(y_test, y_test_pred, pos_label = 1)
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1: {f1}")
```

```
Precision: 0.9999658558514717
Accuracy: 0.999985
Recall: 0.9998853408243995
F1: 0.9999140031532177
```

```
#create the confusion matrix of the prediction
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_test_pred)
cm
```

```
array([[182556,      1],
       [     2,  17441]], dtype=int64)
```

Fig. 4.9. Performance of the Decision Tree with a max depth of 7

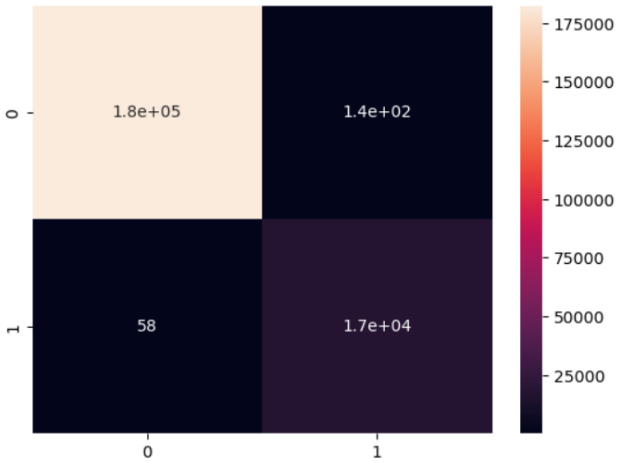

Fig. 4.10. Confusion Matrix of the Decision Tree with a max depth of 7

```
#Accuracy of the prediction
from sklearn.metrics import *
precision = precision_score(y_test, y_test_pred, average = "macro")
accuracy = accuracy_score(y_test, y_test_pred)
recall = recall_score (y_test, y_test_pred, pos_label = 1)
f1 = f1_score(y_test, y_test_pred, pos_label = 1)
print(f"Precision: {precision}")
print(f"Accuracy: {accuracy}")
print(f"Recall: {recall}")
print(f"F1: {f1}")
```

```
Precision: 0.9858749448899697
Accuracy: 0.994545
Recall: 0.9617038353494238
F1: 0.968505528131405
```

Fig. 4.11. Performance of the MLP model with a max epoch of 3

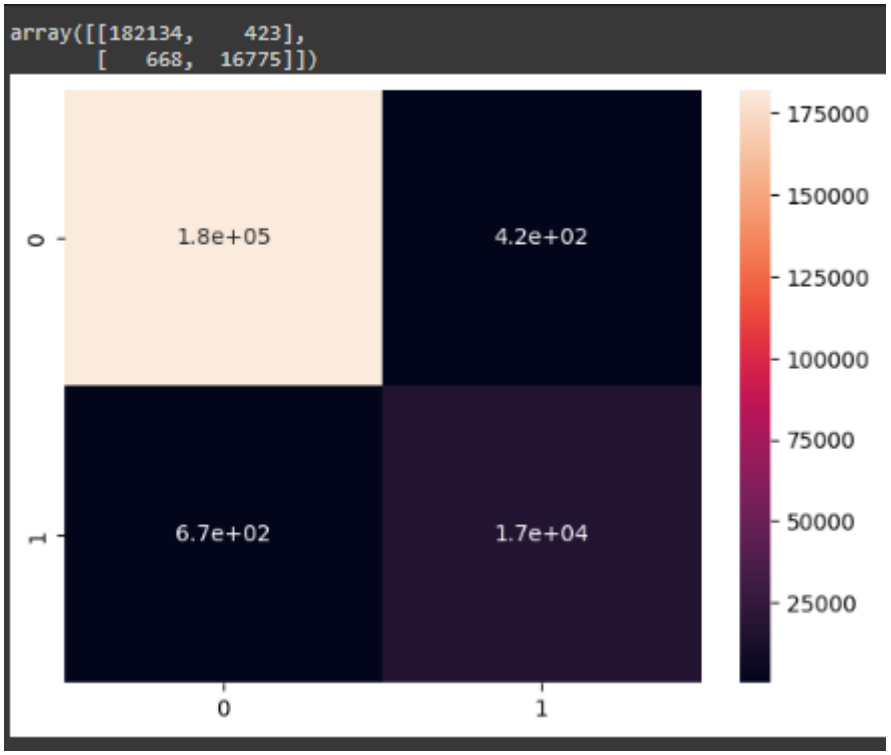

Fig. 4.12. Confusion Matrix of the MLP model with a max epoch of 3

5 Conclusion and Recommendations

This paper discusses about the issue of credit card fraud, which is one of the greatest issues that people in the financial world has been facing. The Credit Card Fraud Dataset has been used to train and test machine learning models: decision tree, and a multi-layered perceptron. Additionally, the models also covers the preprocessing phase, cross-validation, and hyperparameter optimization.

The best model that was chosen was the decision tree model, which achieved an accuracy of 0.999985 and precision and recall scores of 0.999965856 and 0.999885341, respectively with a max depth of 7 because it performed the best in the performance metrics while also being the most resource-friendly and time-efficient to train. The hyperparameter of the decision tree is its maximum depth which simplifies the tree, helping in efficiency but possibly deteriorate accuracy.

To improve the performance of the models, one suggestion would be to lessen the dataset since it contains well over 1 million data and find a way to lessen the noise of the dataset which would improve the accuracy of both models and it might allow the multi-layer perceptron to converge at a lower epoch so it could potentially have an advantage over the decision tree model.

References

1. Wex Definitions Team. (2022). *Credit Card Fraud*. Legal Information Institute. https://www.law.cornell.edu/wex/credit_card_fraud
2. *2023 Credit Card Fraud Report*. (2023, 01 23). Retrieved from Security.org: https://www.security.org/digital-safety/credit-card-fraud-report/
3. Nilson Report. (2022). Card fraud losses worldwide. *Nilsonreport.com*. https://nilsonreport.com/publication_newsletter_archive_issue.php?issue=1232
4. Dhanush Narayanan R. (2022). *Credit Card Fraud*. Kaggle.com. https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud?select=card_transdata.csv
5. IBM. (2023). *What is a Decision Tree*. Ibm.com. https://www.ibm.com/topics/decision-trees
6. R, D. N. (2022). Credit Card Fraud. Kaggle. Retrieved from https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud?select=card_transdata.csv
7. Scikit-learn. (2023). *sklearn.tree.DecisionTreeClassifier*. Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Appendix A. Contribution of Members

Table 1. Contribution of Members.

Name	Contributions
Adrada, Jasper John	Documentation, Report
Razon, Luis Miguel	Coding, Documentation, Report
Rojo, Kate Lynn	Documentation, Report
Romblon, Kathleen Mae	Documentation, Report