



# Credit Card Fraud Detection

20.03.2023

---

Omar Tahir - [otahi001@gold.ac.uk](mailto:otahi001@gold.ac.uk) 33684350

Goldsmiths University of London

8 Lewisham Way

SE14 6NW

# Abstraction

Credit card fraud is a growing problem that affects both financial organizations and consumers all around the world. In this research, I propose developing a machine learning-based approach for detecting credit card fraud. Using a dataset of credit card transactions to train and assess our machine learning model. Preprocess the data to remove missing values and outliers, and then identify fraudulent transactions using a combination of supervised and unsupervised learning methods.

I will use a variety of measures to assess the success of our model, including accuracy, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC). The main measure that I will be focusing on will be the accuracy. I will be performing a comparative analysis to find the best possible algorithm .

The project's intended achievements include the creation of a robust machine learning-based model for credit card fraud detection that can be integrated into financial institutions' existing fraud detection systems. The model will be developed to detect fraudulent transactions with high accuracy while minimizing false positives, ensuring that legitimate transactions are not marked as fraudulent. The outputs of this project will help the continuous effort to reduce credit card fraud, lowering financial losses experienced by financial institutions and consumers while strengthening the overall security of the credit card system.



# Acknowledgement

I would like to thank my supervisor, Tim Blackwell, for his continuous support and guidance throughout this project.

# Introduction

## 1.1 Aims and Motivation

This project explores Credit card fraud detection using deep learning algorithms to improve accuracy in finding such activities. It is critical for preventing and identifying fraudulent credit card activity. Customers will be protected, financial losses will be reduced, regulatory obligations will be met, confidence will be maintained, and operational efficiency will be improved. The primary goal of credit card fraud detection projects is to safeguard customers. Customers might suffer significant financial loss and inconvenience as a result of credit card fraud. Early detection of fraud can aid in the prevention of unauthorized transactions and safeguard clients from financial damage. Credit card issuers and merchants can protect their customers' financial well-being and develop customer trust by deploying efficient fraud detection procedures.

Credit card fraud may cause significant financial losses for card issuers and organizations. Early identification of fraud can help avert such losses and reduce total fraud costs. Effective fraud detection systems can help organizations reduce the financial impact of fraudulent transactions while also ensuring financial stability. On the dark web in 2019, there were 140 million compromised cards for sale. According to the most recent report, there were nine million in the previous year, a 94% decrease in four years. This shows there is a huge need for fraud detection to protect people's finances.

Finally, credit card fraud detection project seek to improve operational efficiency. Fraudulent transactions can bring additional effort to organizations, such as customer service requests and chargeback processing. Implementing fraud detection systems can assist decrease the effort associated with fraudulent transactions and improve operational efficiency. This can eventually lead to cost savings and a better customer experience.

## 1.2 Project Specification

The purpose of this research project is to create a deep learning model with excellent accuracy and statistical power for detecting fraudulent activity. Multiple models will be developed to accomplish this using the Python, Tensorflow, and Keras libraries. The initial models will be simple, with successive models improving efficiency and accuracy. A clear issue statement will be created before beginning model creation, defining the importance of the project and how the model will handle it. A comprehensive literature evaluation will be done to identify existing state-of-the-art approaches as well as any gaps in the literature that the project may address.

A suitable dataset will be chosen in order to apply the models. [1] [keggle](#) will be utilized to locate a suitable dataset containing a substantial number of fraudulent and non-fraudulent transactions. It's critical to make sure the dataset is balanced; otherwise, the findings might be off. To attain better results, techniques for balancing the dataset will be applied. The project approach will be presented in depth, including the models utilized, the dataset used, and the strategies used to balance it. The network's performance will be assessed visually during and after training by examining the graph outputs.

To summarize the project specification these are the main objectives:

- Explore and build basic models and update the into more efficient models
- Implementing a dataset that is more suitable to our model
- Improving accuracy of our models
- Test the new models and improve upon them after evaluating the graphs

# Background Research


## 2.1 Literature review

Credit card fraud is a severe problem that affects both banks and customers. With the increased usage of credit cards for online purchases, financial institutions are finding it difficult to detect fraudulent activity. With the development of new algorithms and methodologies in recent years, machine learning has shown considerable promise in identifying credit card fraud.

Credit card fraud may be detected using supervised learning techniques such as decision trees, logistic regression, and support vector machines (SVMs) (Bolton and Hand, 2002). These algorithms are trained on a labeled dataset of fraudulent and non-fraudulent transactions and may subsequently categorize fresh transactions as fraudulent or non-fraudulent based on their attributes. These algorithms, however, may be incapable of detecting more complex fraud patterns.

For many years, credit card fraud detection has been a focus of study. The work of Chan et al. (2016), who presented a fraud detection system based on deep learning methods, notably a convolutional neural network (CNN), is one significant study. On a publicly available credit card fraud dataset, their system performed admirably. Their method, however, did not take feature selection and data preparation procedures into account, which are key parts of machine learning-based fraud detection.

Bhattacharyya et al. (2011) suggested a credit card fraud detection system based on hidden Markov models (HMMs) in their research. Their approach beat other algorithms, such as logistic regression and neural networks, in



detecting fraudulent transactions. Their method, however, had a limitation in that it could not account for non-linear connections between elements that may exist in credit card transaction data.


Several researchers have recently investigated the use of machine learning algorithms, such as decision trees, neural networks, and support vector machines, in the detection of credit card fraud (Bhattacharyya et al., 2011; Géron, 2019). These algorithms have shown promise in detecting fraudulent transactions, but they have drawbacks such as a lack of interpretability and difficulties dealing with imbalanced datasets.

Credit card fraud detection has also been researched using CNNs and recurrent neural networks (RNNs) (Chan et al., 2016; Wang et al., 2019). These algorithms can understand complex patterns in data and have delivered cutting-edge results in a range of fields, including fraud detection. Deep learning models, on the other hand, need a great amount of data and computer power, and their complexity may make the findings difficult to understand.

Aside from algorithm selection, data pretreatment is critical in detecting credit card fraud. Data cleaning, feature scaling, and feature selection are examples of data preparation procedures (Kotsiantis et al., 2006). The choice of data preparation strategies has an impact on model performance, and determining the best approaches is an ongoing challenge.

Types of credit card fraud:

**Card ID Theft:** This type of theft is comparable to application fraud. In this case, the scammer acquired the information of the existing card and opened a new account with them.



**Lost/Stolen Card:** This occurs when an account owner misplaces their credit card. It can end up in the hands of any scammer who wants to conduct a transaction. However, making a payment is difficult since it requires a PIN that the fraudsters do not know.


**Account Takeover:** This is a very typical type of fraud. In this case, the fraudster has access to the cardholder's account information as well as all essential documentation.

While machine learning and deep learning algorithms have demonstrated promise in detecting credit card fraud, some limitations remain. Future research should concentrate on creating more interpretable algorithms and discovering data preparation processes that might increase model performance.

The effectiveness of any machine learning-based credit card fraud detection system is dependent on proper data preparation. It entails cleaning the data, dealing with missing values, feature scaling, and picking acceptable features. Because of insufficient data preparation processes, the researchers described in the preceding response had limitations in their models. For example, Chan et al.'s technique did not take feature selection into account, which is critical in removing unnecessary features that might impair model accuracy. The model used by Bhattacharyya et al. does not account for non-linear linkages between items in credit card transaction data, which is a critical weakness in detecting complex fraud patterns.

To avoid these constraints, effective data balance and preparation must be prioritized. Because credit card fraud cases are uncommon, it is necessary to oversample the minority class or undersample the majority class to avoid bias in the model. Feature scaling is also important in ensuring that all





features have the same scale to prevent favoring some features over others. Furthermore, feature selection is required to eliminate irrelevant features and improve model accuracy. For my project, I would try to avoid this by focusing on properly balancing the dataset in order to give reliable results for my model.

## Methodology

This chapter details the design choices that were made throughout the implementation process which are crucial to achieving accurate and useful outputs from a credit card fraud detection system. This chapter will detail the key design choices that were made in this project, including how the dataset was gathered and processed to produce outputs.

### 3.0 Importing Libraries

Python libraries are essential in the field of machine learning because they provide a diverse set of resources and techniques for developing, training, and assessing models. For this project, I will use a variety of libraries, the most important of which are tensorflow, pandas, numpy, and keras.

- Google TensorFlow is an open-source machine learning library. It is commonly used in the development and training of deep learning models, such as neural networks.

- Pandas is a data manipulation package that includes tools for reading, writing, cleaning, transforming, and merging datasets. It is commonly used in machine learning for exploratory data analysis.
- Keras is a Python-based high-level neural network API that runs on top of TensorFlow, CNTK, or Theano. It has an easy-to-use interface for creating and training deep learning models.

Using these libraries will assist me in developing a strong model that can surpass other people's models in the same field while also producing outstanding outcomes.

### **3.1 Dataset**

Finding an appropriate dataset is an important part of any machine learning research, including credit card fraud detection. However, it can be a difficult task because not all datasets are appropriate for the intended purpose. To properly train the model for credit card fraud detection, the dataset must contain a sufficient number of both fraudulent and non-fraudulent transactions.

Searching for publicly available datasets, such as those provided on Kaggle, UCI Machine Learning Repository, or other similar sites, is one technique of acquiring an appropriate dataset. However, without proper investigation, determining the quality and suitability of these datasets can be difficult. Datasets may include irrelevant data or have substantial data quality flaws in some circumstances, rendering them worthless for the intended purpose. In addition, some websites may upload exactly the same datasets, which can make choosing an appropriate dataset difficult. It is critical to discover and

choose a dataset that is most relevant to the project's needs, with enough data to assure the model's correctness.

For my project, I will be using the Credit Card Fraud Detection dataset uploaded by the Machine Learning Group on Kaggle, which has a usability score of 8.53, indicating its suitability for my research. This dataset includes credit card transactions made by European cardholders in September 2013, with a total of 284,807 transactions, including 492 fraud cases. As shown by the (figure 1.1) below 0 represents legit transactions while 1 represents fraud transactions.

```
credit_card_data['Class'].value_counts()
```

```
0    284315  
1         492  
Name: Class, dtype: int64
```

Figure 1.1

## 3.2 Unbalanced Dataset

To deal with an imbalanced dataset, it is necessary to first understand the nature of the imbalance and the influence it has on the model's performance. This may be accomplished by studying the class distribution in the dataset and computing the class imbalance ratio. Numerous procedures may be performed to address an uneven dataset after the nature of the imbalance has been determined.

These are some examples:

Resampling the dataset: To obtain a more balanced dataset, either oversample the minority class or undersample the majority class.

Using weighting in the loss function: This includes giving the minority class greater weight in the training process.

Using a different model: Using a model that is more resilient to unbalanced datasets, such as a decision tree or a support vector machine.

### 3.3 Balancing and preparing the dataset

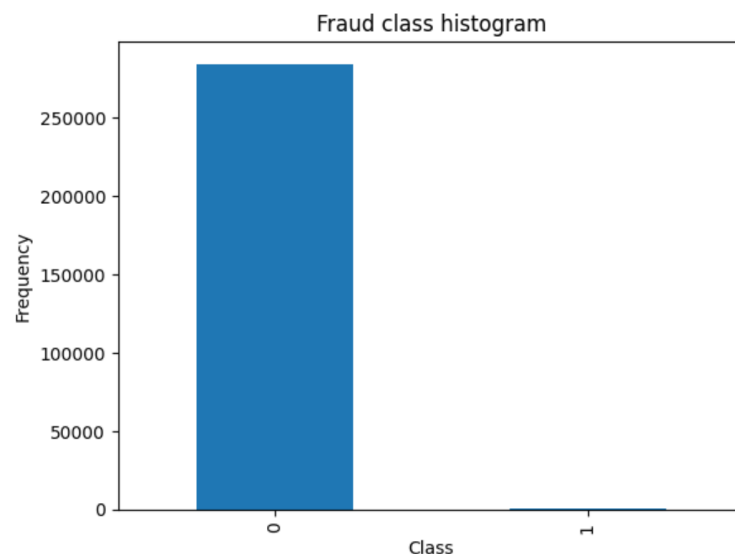
To address this issue, I will use different techniques to balance the dataset, such as under-sampling and oversampling. Despite searching for alternative datasets, I was unable to find one that could be used for my project, as other websites also uploaded the same dataset so I will be continuing my research with the following dataset.

Figure 1.2

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Time        284807 non-null float64
1   V1          284807 non-null float64
2   V2          284807 non-null float64
3   V3          284807 non-null float64
4   V4          284807 non-null float64
5   V5          284807 non-null float64
6   V6          284807 non-null float64
7   V7          284807 non-null float64
8   V8          284807 non-null float64
9   V9          284807 non-null float64
10  V10         284807 non-null float64
11  V11         284807 non-null float64
12  V12         284807 non-null float64
13  V13         284807 non-null float64
14  V14         284807 non-null float64
15  V15         284807 non-null float64
16  V16         284807 non-null float64
17  V17         284807 non-null float64
18  V18         284807 non-null float64
19  V19         284807 non-null float64
20  V20         284807 non-null float64
21  V21         284807 non-null float64
22  V22         284807 non-null float64
23  V23         284807 non-null float64
24  V24         284807 non-null float64
25  V25         284807 non-null float64
26  V26         284807 non-null float64
27  V27         284807 non-null float64
28  V28         284807 non-null float64
29  Amount      284807 non-null float64
30  Class       284807 non-null int64
```

It is critical to balance a dataset in order to avoid bias in machine learning models. If a dataset is unbalanced, the model may be biased towards the majority class while underperforming on minority classes. The dataset is balanced to ensure that each class has a similar number of samples, which improves the model's accuracy across all classes. As we saw the dataset we are dealing with is extremely unbalanced and needs to be balanced before processing otherwise the results will be biased towards the majority class. To address this issue, several processes can be used to balance the dataset. If we take a look at the dataset we are dealing with extremely unbalanced data (figure 1.3). As the bar graph shows the non fraudulent transaction is above 250000 where as fraudulent transaction are barely seen in the graph

Figure 1.3



Undersampling, oversampling, synthetic sampling, and cost-sensitive learning are all strategies for balancing datasets, But for building my module for my research, I choose to use a combination of SMOTE and undersampling techniques. Based on my assessment, I believe this strategy is most suited to

tackling the specific issues of my research. With these strategies, I hope to improve the performance and accuracy of my machine learning models while reducing any biases caused by imbalanced data.

### 3.3 SMOTE (Synthetic Minority Over-sampling Technique )

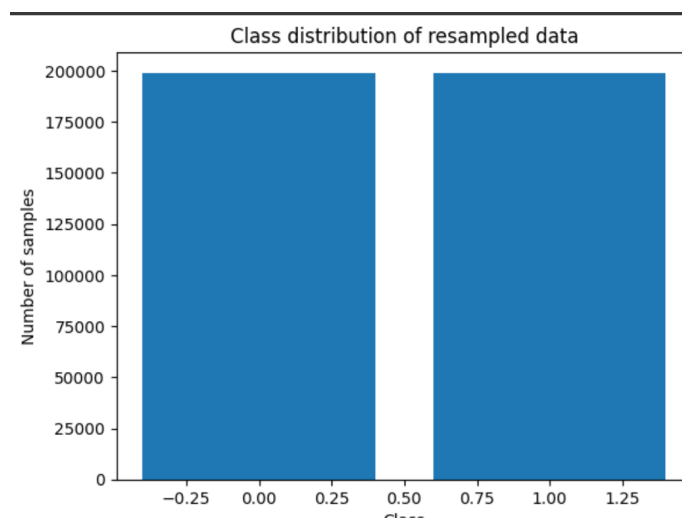
The SMOTE algorithm begins by locating the  $k$  nearest neighbors of a minority class sample ( $k$  is a user-defined parameter). Afterwards, SMOTE creates new synthetic samples by interpolating between the original sample and its nearest neighbors. SMOTE generates a synthetic sample by selecting one of the  $k$  nearest neighbors at random and perturbing its features within the range defined by the original sample and the chosen neighbor. The procedure is continued until the desired balance is obtained.

Since the dataset is highly imbalanced the number of fraudulent transactions is much lower than the number of non-fraudulent transactions, SMOTE (Synthetic Minority Over-sampling Technique) is used to oversample the minority class (fraudulent transactions) in the training set. SMOTE generates synthetic samples of the minority class by interpolating between existing samples, which helps to balance the class distribution. After running the smote algorithm in my model it turned an unbalanced dataset to a balanced dataset as we can see both of the classes have the same output of 199008 making the dataset usable for my research purpose (Figure 1.4). For a better visualization see (Figure 1.5)

Figure 1.4

```
Original class distribution: 0    199008
1      356
Name: Class, dtype: int64
Resampled class distribution: 0    199008
1    199008
Name: Class, dtype: int64
```

Figure 1.5



### 3.4 Undersampling

It is a basic and quick way for balancing datasets that can be beneficial in some cases. It can help to lessen the majority class's effect and increase the performance of machine learning models on the minority class. It shuffles the original data, selects all fraudulent and non-fraudulent transactions, and concatenates them into a new balanced dataframe. The algorithm then shuffles the new dataframe and displays the first few rows of the new dataset (Figure 1.6). The resulting balanced dataset can be used for training machine learning models. To produce better results, undersampling is frequently used in conjunction with other approaches such as oversampling and synthetic sampling.

Figure 1.6

	Time	V1	V2	V3	V4
<b>107749</b>	70595.0	-1.266166	0.207650	2.512401	0.317805
<b>46909</b>	42985.0	-4.075975	0.963031	-5.076070	4.955963
<b>1426</b>	1103.0	-0.846194	0.675795	2.869332	1.053603
<b>14197</b>	25231.0	-16.598665	10.541751	-19.818982	6.017295
<b>6903</b>	8886.0	-2.535852	5.793644	-7.618463	6.395830

5 rows × 31 columns

Undersampling is a valid approach to balance this dataset as there is a significant imbalance between the two classes. In this case, the majority class (0) has 284,315 examples while the minority class (1) has only 492 examples. Undersampling would involve randomly selecting a subset of examples from the majority class at random, equal to the size of the minority class 492 in this case. (Figure 1.7)

Figure 1.7

```
0      492
1      492
Name: Class, dtype: int64
```

### 3.5 Model Selection

A critical stage in designing an efficient credit card fraud detection system is picking a model. It means choosing the best machine learning or deep learning models to train the data. The quantity and complexity of the dataset, the type of characteristics collected from the data, and the required degree of accuracy all influence model selection.

Here is an number commonly used models for credit card fraud detection:



- Logistic regression is a basic and effective model that is utilized in binary classification situations.
- Decision trees are straightforward, powerful models that are simple to learn and apply. They function by dividing the data into smaller groups depending on the most important characteristics.
- Random forests are a collection of decision trees that work together to improve the accuracy of the model.
- SVMs are powerful models that work by mapping the input data into a high-dimensional space and then finding the best boundary that separates the two classes.
- Neural networks are a class of deep learning models that can learn complex patterns in the data.

### 3.6 Model Training

During training, the model is exposed to the training dataset and its parameters are adjusted depending on the input data and the associated output labels. The training dataset is divided into two parts: training and validation. The training set is used to train the model, while the validation set is used to validate it. The validation set is used to assess the performance of the model during training and to prevent overfitting.

### 3.7 Visual outputs

Matplotlib is a data visualization package written in Python. It includes a number of tools and functions for constructing 2D and 3D plots, histograms, bar charts, scatterplots, and other graphical data representations.

Matplotlib is a popular tool for scientific computing, data analysis, and machine learning. It is highly adjustable and adaptable, with users able to alter any component of a plot, from the axis names to the color palette.

Matplotlib, in addition to basic plotting operations, has various modules for constructing more complex visualizations such as maps, heatmaps, and animations. It may be used with other Python modules such as NumPy and Pandas to generate massive dataset visualizations.

Monitoring the accuracy graph allows you to assess if the model is improving with time and if it has reached a steady level of accuracy. The accuracy graph may also be used to indicate possible concerns like overfitting or underfitting, which occurs when the model is either too complicated or too basic to effectively capture the patterns in the data.

### 3.8 Weight regularization

Weight regularization refers to the technique of imposing limitations on a neural network's weights to reduce overfitting and improve model generalization. This is frequently accomplished by including a penalty term in the loss function that promotes the weights to be modest, thus preventing the model from fitting in the data and lowering the danger of overfitting.

There are several different types of weight regularization techniques that can be used in AI, including L1 and L2 regularization, which add a penalty term

based on the absolute or squared magnitude of the weights, respectively. Other techniques include dropout, which randomly sets weights to zero during training, and early stopping, which prevents the model from overfitting by stopping the training process when the validation error begins to increase.



The image above shows that I am using L2 regularization in my model to reduce chances of overfitting. L2 regularization, also known as L2 norm or Ridge regularization, modifies the loss function by including a penalty term equivalent to the square of the model's weights. This encourages the model to use all of the available features while discouraging it from assigning large weights to any single feature. This can help to reduce overfitting and improve the model's generalization capacity.

```
[63] import numpy as np

from sklearn import linear_model
from sklearn.svm import l1_min_c

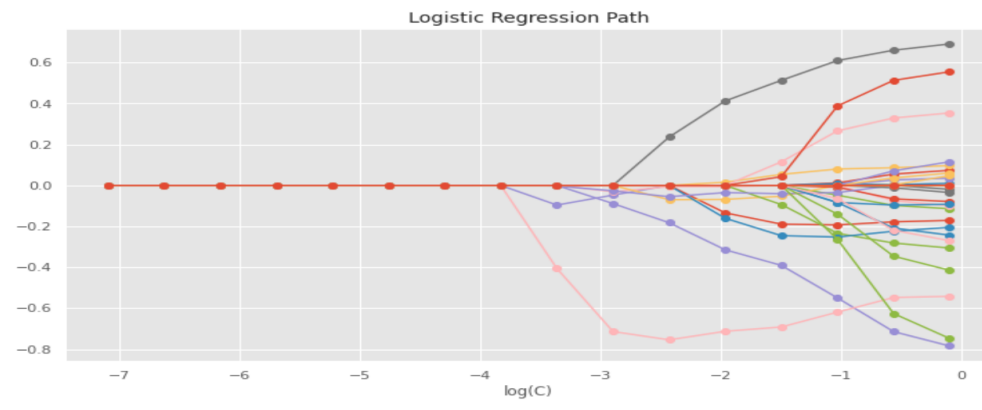
cs = l1_min_c(X, y, loss="log") * np.logspace(0, 7, 16)

clf = linear_model.LogisticRegression(
    penalty="l1",
    solver="liblinear",
    tol=1e-6,
    max_iter=int(1e6),
    warm_start=True,
    intercept_scaling=10000.0,
)
coefs_ = []
for c in cs:
    clf.set_params(C=c)
    clf.fit(X, y)
    coefs_.append(clf.coef_.ravel().copy())

coefs_ = np.array(coefs_)

import matplotlib.pyplot as plt

plt.plot(np.log10(cs), coefs_, marker="o")
ymin, ymax = plt.ylim()
plt.xlabel("log(C)")
plt.ylabel("Coefficients")
plt.title("Logistic Regression Path")
plt.axis("tight")
```




This next image shows that I am using L1 regularization also known as L1 norm or Lasso regularization. As this encourages the model to use only a subset of the available features, and it can result in some weights being set to zero, effectively removing those features from the model. By doing this I am hoping to achieve Statistical power.

### 3.9 Validation

Validation is a critical stage in the machine learning process because it prevents overfitting and ensures that the model can generalize to new data. By analyzing the model's performance on the validation set, it is feasible to discover any potential model flaws, such as overfitting or underfitting, and make any necessary tweaks to improve the model's performance. Each of these methods includes dividing the dataset into a training set, validation set, and test set, and then evaluating the model's performance using the validation set.

Thus, validation is a critical phase in the machine learning process because it helps to confirm the model's accuracy and reliability. It is possible to uncover any potential flaws and make necessary improvements to improve the



model's performance by evaluating the model's performance on unseen data.

I am using validation to train my data and test out the results after using L1 and L2 regularization to see what is the outcome of this result and has it improved the overall accuracy of my model. The model generalized well when cross-validated with synthetic samples present in the data set. The accuracy score converges for the training and the validation data sets as the number of samples increases in the training of the model.

## 4.0 Dropout

Dropout is an artificial intelligence approach for preventing overfitting in neural networks. When a model becomes extremely sophisticated, it is difficult to generalize adequately to new data, resulting in poor performance on previously unknown data. Dropout is a regularization approach that randomly loses neurons in a neural network during training.

Dropout has the advantage of being a simple and effective method of regularization. It does not require any additional computations or parameters and may be easily integrated into existing neural network designs. However, there are several limitations to dropping out. Because the network is unable to use all of its neurons for predictions, training set performance may suffer on occasion. Furthermore, determining the optimal dropout rate can be difficult because it is not always obvious how much dropout to apply to a given layer.

Despite these limitations, dropout is a widely used technique in AI and has been shown to be effective in improving neural network performance.

## 4.1 Statistical power

Statistical power refers to the ability of a machine learning model or algorithm to detect a true difference between two groups or treatments when one exists. This is necessary to ensure that the outcomes of an AI system are dependable and robust. The number and diversity of the training data, the complexity of the model, and the evaluation metric used to judge performance are all aspects that might influence an AI system's statistical power. A model with strong statistical power can identify the best-performing model in a comparison, whereas a model with low statistical power may fail to discover a meaningful difference in performance.

After performing different types of strategies to train my data and improve its overall accuracy I was able to achieve an accuracy of 0.99 which is really good. It's really important to have a high statistical power because low power can lead to false negative results, where the test fails to detect a true difference even though one exists.

## 3.4 Training and testing set

It is critical to divide the dataset into training and testing sets while designing a credit card fraud detection project. The training set is used to train and test the machine learning model. The dataset should be randomly divided to ensure that the distribution of fraudulent and non-fraudulent transactions is similar in both sets. A 70/30 or 80/20 split is commonly employed, with 70% or 80% of the data utilized for training and the remaining 30% or 20% used for testing. Various machine learning algorithms, including supervised and

unsupervised learning techniques, can be used to train the model during the training phase.

Training set is used to evaluate the model once it has been trained. The model's performance is evaluated using a variety of metrics, including accuracy, precision, recall, and F1 score. These indicators aid in assessing the model's capacity to distinguish between fraudulent and non-fraudulent transactions and identifying areas for development.

It is critical to understand that the testing phase is not the end of the credit card fraud detection project. To stay up with evolving fraud strategies, the model should be regularly monitored and updated. This may entail retraining the model with new data, modifying its parameters, or incorporating new characteristics.

```
# Turn the values into an array for feeding the classification algorithms.  
X_train = X_train.values  
X_test = X_test.values  
y_train = y_train.values  
y_test = y_test.values
```

The variables `X_train`, `X_test`, `y_train`, and `y_test` in the code snippet all carry data. The data is transformed into arrays by invoking the `.values` function on each variable. Many machine learning libraries, such as scikit-learn, require data to be in array form for training and testing models.

In a supervised learning problem, `X_train` and `X_test` represent the features or independent variables used to predict the target variable (`y_train` and `y_test`). They can be used to fit and assess classification models such as logistic regression, decision trees, and random forests by turning the data into arrays.

# Implementation and Results

This section explains how machine learning was implemented. This includes five different algorithms that were used; these include K-nearest neighbor, Decision Trees, Support Vector Machine, Random forest and finally Convolutional Neural Network. All algorithms were run twice using different balancing techniques. All models for credit card fraud detection are developed in Python using the Tensorflow and Keras libraries, Once all the modules have been runned they have been compared between each other to ensure that the final model selected for use is the most effective one.

To identify the best successful model for credit card fraud detection, the results of the modules' execution were compared. Precision, recall, and F1-score were among the metrics examined during the models' evaluation. This was done to make sure the fraud detection capabilities of the final model chosen are accurate, efficient, and effective.

## 4.2 K-Nearest Neighbors

KNN works by calculating the distance between the new data point and all the existing data points. After which, based on the distance, it chooses the k closest data points and assigns the new data point the class that appears most frequently in those k neighbors. Euclidean, Manhattan, Minkowski, or any other distance metric can be used. Cross-validation is commonly used to determine the value of k, in which different values of k are tested and the one with the best performance is chosen.



Nevertheless, despite its popular use, this approach has a number of drawbacks. K&N's high computational cost is one of its main flaws. When the training dataset is huge, it can take a while for the algorithm to calculate the distances between each instance and the instance that has to be classified. Due to this, KNN is also inappropriate for real-time applications where results must be obtained instantly.

Furthermore, Unsuitable for datasets with imbalances: k-NN relies on the training data's balanced class distribution. However, k-NN may be biased towards the dominant class and misclassify the minority class when dealing with imbalanced datasets, where one class is significantly more frequent than the others.

By performing hyperparameter tuning for the k-Nearest Neighbors algorithm. It creates a KNN classifier for different values of k, trains the classifier on the training data, and computes the accuracy of the classifier on both the training and testing data. The resulting accuracies are stored in arrays that can be used to visualize the performance of the algorithm with different values of k. The first test is performed on a dataset which was balanced using smote and a graph is printed to better understand the performance of the model which can be seen in Figure 1.8

Figure 1.8

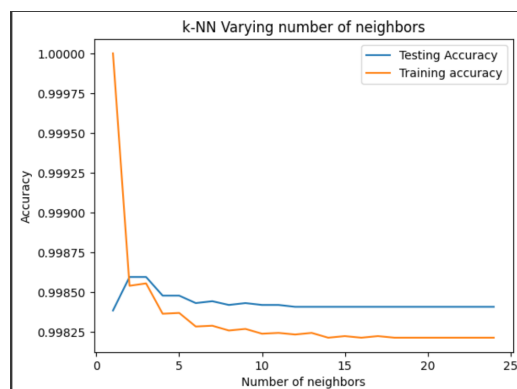


Figure 1.9

K-Nearest Neighbours  
Scores  
Accuracy --> 0.9985955549313578

The testing indicates that the accuracy is almost similar to the training accuracy, it indicates that the model is generalizing well to new data. This means that the model has learned the underlying patterns in the training data and can accurately predict on unseen data. Moreover, the module returns an accuracy score of 0.998 (Figure 1.9).

On the other hand when running the model after using undersampling we get much different results and a much lower accuracy score compared to the first model (Figure 2.0)

Figure 2.0

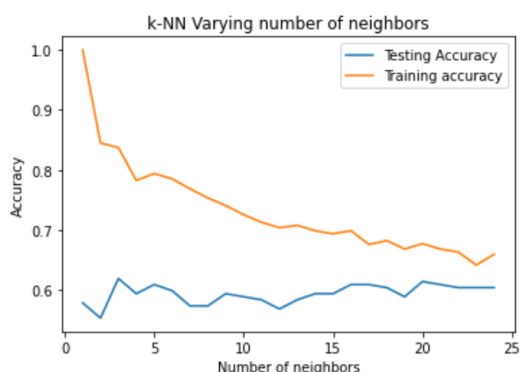


Figure 2.1

K-Nearest Neighbours  
Scores  
Accuracy --> 0.6192893401015228

In the graph lower testing accuracy compared to training accuracy indicates overfitting, which can be prevented by regularization, early stopping, or adjusting hyperparameters. And the Accuracy we received is 0.6192 (Figure 2.1). The results indicate that we get a higher accuracy using smote.


### 4.3 Decision Trees

A decision tree is a machine learning algorithm that models decisions and their possible consequences using a tree-like structure. The decision tree starts at the root node and makes a binary decision at each internal node based on the value of a specific feature, until it reaches a leaf node which represents the decision or classification output.

However, Decision trees have the potential to overfit the training data, which makes the model overly complex and impairs its ability to generalize well to new data. This is one of decision trees' key drawbacks. This occurs when the tree has too many branches and is too deep, creating a model that is extremely intricate and detailed yet may not function well with fresh data.

Decision trees also have the drawback of being unstable, which implies that even minor changes in the data might result in substantial changes in the tree's structure. Because the same dataset may yield distinct trees if sampled differently or with various parameters, this instability might make the model less consistent and dependable.

By using cross-validation and a grid search to tune the hyperparameters of a decision tree classifier. A parameter grid is defined, with variable values for max depth, min samples leaf, and min samples split. A GridSearchCV object is built with inputs such as the estimator, parameter grid, scoring metric, and cross-validation folds, and the DecisionTreeClassifier model is instantiated. The grid search is performed using the fit() function, and the optimum hyperparameters are chosen based on the greatest roc auc score obtained during cross-validation. The first result is obtained after using Smote the accuracy score is 0.946 (Figure 2.2) whereas by using undersampling we get



an accuracy of 0.962(Figure 2.3) which indicates that by using undersampling we get a higher accuracy than using smote.

Figure 2.2

---

Best roc auc score : 0.9462578061643839

Figure 2.3

---

Best roc auc score : 0.9623699400015189

## 4.4 Random Forest

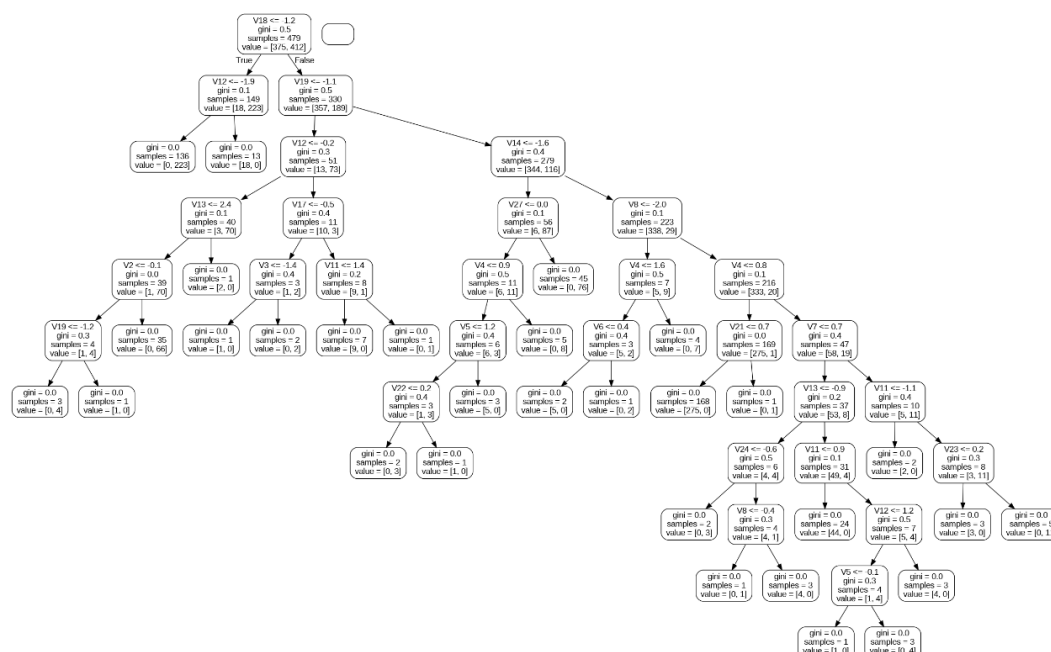
Random Forest is a classification and regression ensemble learning algorithm. It generates multiple decision trees from randomly selected data and feature subsets. By introducing randomness into the training process, the algorithm reduces overfitting and improves generalization performance by generating diverse decision trees. The final prediction is obtained by taking the majority vote or the average of all the decision trees' predictions. It minimizes overfitting and is widely used in a variety of applications.

However, it does have limitations such as overfitting is a potential problem with Random Forest. The algorithm may become overly complex and begin fitting to noise or irrelevant features in the data if there are too many trees in the forest or if the trees are too deep. This might lead to poor performance on new, unseen data.

Additionally, Random Forest can be computationally demanding, particularly when dealing with huge datasets or a lot of characteristics. It might be laborious and expensive to construct numerous trees and assess their efficiency. parameters.

The code trains a Random Forest Classifier model on a given dataset and evaluates its performance using various metrics such as accuracy, precision, recall, F1-score, and Matthews correlation coefficient. The Random Forest Classifier is an ensemble learning algorithm that creates multiple decision trees from randomly selected subsets of data and features, and combines their predictions to make the final prediction (Figure 2.4)

Figure 2.4



A recursive process is used to construct each decision tree in a random forest. The algorithm selects the feature that best splits the data at each node of the tree based on a criterion such as information gain or Gini impurity. The data is then divided into two subsets based on the feature of choice, and the process is repeated recursively on each of the resulting subsets until a stopping criterion, such as a maximum depth of the tree or a minimum number of samples per leaf node, is met. Running the algorithm after using smote gave us an accuracy of 0.999 (Figure 2.5) where as running the same algorithm after using undersampling gave us an accuracy of 0.934 (Figure 2.6)

Figure 2.5

The model used is Random Forest classifier  
 The accuracy is 0.9996254813150287  
 The precision is 0.9333333333333333  
 The recall is 0.8235294117647058  
 The F1-Score is 0.8749999999999999  
 The Matthews correlation coefficient is 0.8765319758290167

Figure 2.6

The model used is Random Forest classifier  
 The accuracy is 0.934010152284264  
 The precision is 0.970873786407767  
 The recall is 0.9090909090909091  
 The F1-Score is 0.9389671361502346  
 The Matthews correlation coefficient is 0.8695347643466169

## 4.5 Support Vector Machine

SVM is a machine learning algorithm that is used for classification and regression problems. By maximizing the margin between the closest points of each class, SVM finds the best hyperplane that separates the data points into different classes. SVM can handle non-linearly separable data by using kernel functions to transform it into a higher-dimensional space.

The sensitivity of SVMs to the kernel function and parameter choices is one of their key drawbacks. In order to translate the input to a higher dimensional space where it may be more easily segregated, SVMs significantly rely on the kernel function. The algorithm's performance can be significantly impacted by the choice of kernel function and parameters. The SVM might not function successfully if the incorrect kernel or parameters are chosen.

SVMs can be computationally expensive, especially when working with huge datasets, which is another drawback. It can take a while for SVMs to identify the ideal hyperplane that divides the data into distinct classes. Because of this, the SVM's training phase may be lengthy and the algorithm may not be appropriate for real-time applications, because of this, the SVM's training phase may be lengthy and the algorithm may not be appropriate for real-time applications.

The algorithm trains three different Support Vector Machine (SVM) models with linear, radial basis function (RBF), and polynomial kernel functions on a classification dataset, and calculates the accuracy score of each model. The accuracy scores are stored in a list and printed to the console. The purpose of the code is to compare the performance of SVM models with different kernel functions and to determine which kernel function provides the best accuracy score for the given dataset. Running the algorithm after using smote gave us an accuracy of 0.998 (Figure 2.7) whereas running the same algorithm after using undersampling gave us an accuracy of 0.883 (Figure 2.8) which tells us that by using smote we get a higher accuracy score than undersampling

Figure 2.7


```
Linear SVM Accuracy= 0.9987008883115059  
RBF SVM Accuracy= 0.9984082955888721  
Polynomial SVM Accuracy= 0.9984082955888721
```

Figure 2.8

```
Linear SVM Accuracy= 0.883248730964467  
RBF SVM Accuracy= 0.5482233502538071  
Polynomial SVM Accuracy= 0.5736040609137056
```

## 4.6 Convolutional Neural Network

CNNs are a type of deep learning algorithm that is used for image classification and computer vision tasks. They work by extracting relevant features from input images with convolutional layers, then performing classification with fully connected layers. CNNs can automatically learn and extract features using



convolutional layers, making them a powerful tool for a wide range of computer vision tasks.

By analyzing transaction data and identifying patterns indicative of fraudulent behaviour, Convolutional Neural Networks (CNNs) can be used to detect credit card fraud. For each transaction, a time-series data sequence is generated, which is then fed into a CNN model, which learns to distinguish between fraudulent and legitimate transactions by extracting relevant features and patterns. Once trained, the CNN model can be used in real-time to flag any suspicious transactions, allowing for faster and more accurate fraud detection.

With state-of-the-art performance in a variety of applications, Convolutional Neural Networks (CNNs) have revolutionized the field of image detection and processing. However, CNNs have their own set of limits and disadvantages, just like any other machine learning technique. CNNs' intricacy is one of their main drawbacks. CNNs are computationally expensive and challenging to train since they frequently include numerous layers and millions of parameters. Due to overfitting, a model that performs well on training data but poorly on fresh, untested data. The sensitivity of CNNs to hyperparameters is another drawback. The selection of hyperparameters, such as the learning rate, number of layers, and filter sizes, can have a significant impact on how well a CNN performs. It can take a long time and a lot of computing power to fine-tune these hyperparameters.

The Convolutional Neural Network (CNN) model I created (Figure 2.9) in Python with the Keras library. Two 1-dimensional convolutional layers, two batch normalization layers, two dropout layers, one fully connected layer, and an output layer comprise the model. The model is trained for 80 epochs using a training set. This code's goal is to create a CNN model that can be used for binary classification tasks like detecting fraudulent credit card transactions.



Figure 2.9

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 29, 32)	96
batch_normalization (Batch Normalization)	(None, 29, 32)	128
dropout (Dropout)	(None, 29, 32)	0
conv1d_1 (Conv1D)	(None, 28, 64)	4160
batch_normalization_1 (Batch Normalization)	(None, 28, 64)	256
dropout_1 (Dropout)	(None, 28, 64)	0
flatten (Flatten)	(None, 1792)	0
dense (Dense)	(None, 64)	114752
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

=====  
Total params: 119,457  
Trainable params: 119,265  
Non-trainable params: 192  
=====

As you can see, I'll be implementing a sequential model, which is a type of neural network in which the layers are arranged sequentially. The input data flows sequentially through each layer, with the output of one layer serving as the input to the next. Keras includes a Sequential class that makes it simple to create sequential models. Deep learning applications that use sequential models include image classification, natural language processing, and time-series analysis. The sequential model's simplicity makes it simple to understand, train, and apply to a wide range of machine learning tasks.

(Figure 3.0) shows the code that compiles and trains a neural network using the Adam optimizer, binary cross-entropy loss, and accuracy metric. `fit()` trains the model, and `history` saves performance metrics for later analysis and visualization.

Figure 3.0

```

model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verb

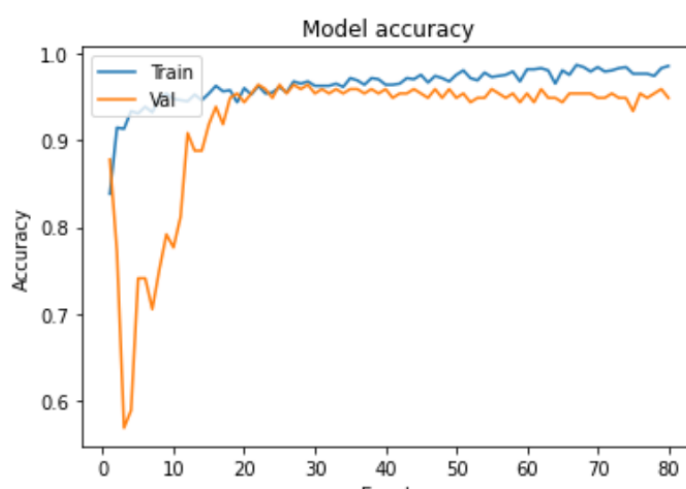
```

WARNING:absl:`lr` is deprecated, please use `learning\_rate` instead, or use the legacy optimizer, e.g., tf.keras.optimizers.legacy.Adam.

Epoch 1/80  
 7121/7121 [=====] - 50s 5ms/step - loss: 0.0107 - accuracy: 0.9981  
 - val\_loss: 0.0063 - val\_accuracy: 0.9993  
 Epoch 2/80  
 7121/7121 [=====] - 37s 5ms/step - loss: 0.0061 - accuracy: 0.9991  
 - val loss: 0.0037 - val accuracy: 0.9993

The output represents the performance of a neural network model during the first training epoch. For 50 seconds, the model was trained on a dataset of 7121 samples. At the end of this epoch, the loss and accuracy values for both the training and validation data are reported. These performance metrics are used to assess how well the model learns patterns in the data and how effective the model is during training. The highest accuracy that was obtained when running the model was 0.995. We can get a visual representation (Figure 3.1) about how well our model performed on the new dataset that was created using SMOTE.

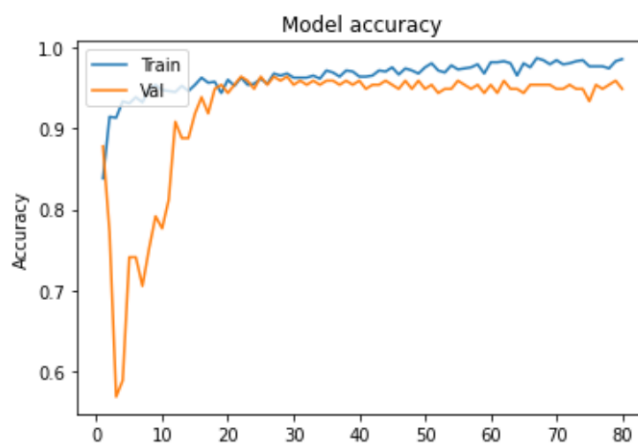
Figure 3.1



The training accuracy is slightly higher than the validation accuracy, it may indicate some overfitting, but it is not always a serious problem. This minor difference can be caused by random variations in the data or by a well-generalizing model. Monitoring validation accuracy over time during training can assist in determining whether the model is overfitting and whether adjustments are required to improve generalization.

The highest accuracy that was obtained when running the model the second time was 0.9860 (Figure 3.2) about how well our model performed on the new dataset that was created using Undersampling.

Figure 3.2



The training accuracy is slightly higher than the validation accuracy, it may indicate some overfitting which looks almost similar to the graph that was generated using smote. Overall the highest accuracy that was achieved was using smote.

## 4.7 Evaluation of all the models

The table below illustrates how each of the five modules performed using smote and undersampling, as well as which achieved the highest accuracy of the two. The models listed are K-nearest neighbor, Decision Trees, Support Vector Machine, Random Forest, and Convolutional Neural Network.

Name	SMOTE accuracy	Undersampling accuracy	Highest Accuracy
<b>K-nearest neighbor</b>	0.998	0.619	SMOTE
<b>Decision Trees</b>	0.946	0.962	Undersampling
<b>Support Vector Machine</b>	0.998	0.883	SMOTE
<b>Random forest</b>	0.999	0.934	SMOTE
<b>Convolutional Neural Network</b>	0.995	0.986	SMOTE

This table clearly shows that SMOTE appears to be the most effective technique, as it produced the highest accuracy for all models. Random forest, which achieved an accuracy of 0.999 using SMOTE, was the model with the highest accuracy. An accuracy of 0.999 (99.9%) is considered very good in most machine learning applications. It means that the model predicted 999 out of 1000

instances correctly, or has a 0.1% error rate. However, accuracy alone may not be sufficient to evaluate model performance, and other metrics may be required; however, for the time being, we will only focus on accuracy. By observing other people models on the same subject i have determined that my model have performed better (Figure 3.3)

Figure 3.3

Figure3. After Under Sampling Technique

```
Model training start.....
Accuracy of model on test dataset :- 0.9593908629441624
Confusion Matrix :-
[[102  4]
 [ 4 87]]
Classification Report :-
      precision    recall  f1-score   support

      0       0.96      0.96      0.96        106
      1       0.96      0.96      0.96         91


   accuracy       0.96
  macro avg       0.96
 weighted avg       0.96

AROC score :-
0.9591540534936762
```

As we can observe, the highest accuracy of this model made by trendy tech journals was able to achieve an accuracy of 0.959 which was achieved using random forest algorithm, compared to my model's 0.999 making it a better model.

## 4.8 Achieving Statistical power

In machine learning, statistical power is a concept that measures a model's ability to detect true relationships or patterns in data. Low statistical power may result in overfitting and unreliable predictions, so a sufficiently large and diverse dataset, proper feature selection, and an appropriate choice of machine learning algorithm and model hyperparameters are all required. Statistical power is essential in ensuring a model's ability to detect true effects, and it is achieved by preventing overfitting and increasing the model's ability to detect true effects.



In order for our model to achieve statistical power, it must perform better than 0.997. Because the percentage of non-fraudulent transactions in our dataset is 0.998 while the percentage of fraudulent transactions is 0.017, our accuracy score must be greater than 0.997, which we have achieved by obtaining an accuracy score of 0.999 using random forest.(Figure 2.5)

## Discussion

The detection of credit card fraud has been studied extensively, and various machine learning techniques, including decision trees, logistic regression, and support vector machines (SVMs), have been explored. These algorithms have shown promise in classifying transactions as fraudulent or non-fraudulent based on their properties. However, these models can struggle to identify more complex patterns of fraud due to imbalanced datasets.

A notable study by Chan et al. utilized deep learning techniques, specifically a convolutional neural network (CNN), to detect credit card fraud. While their system performed well, they did not consider the crucial steps of feature selection and data preparation in their methodology. In my own research, I implemented a CNN model with a focus on balancing the dataset. Using SMOTE and undersampling techniques, I achieved an accuracy of 0.995 and 0.986, respectively. Balancing the data reduces the risk of false positives and improves model performance.

To efficiently detect credit card fraud, the problem of imbalanced datasets must be addressed. We can recognise intricate fraud patterns by utilizing cutting-edge machine learning algorithms like CNNs and RNNs. However, due to the complexity of these models and the amount of data and computing power they require, it may be challenging to understand the outcomes. Developing precise and trustworthy fraud detection algorithms requires balancing the dataset as well as other data preparation techniques.


Another study recommended a hidden Markov model (HMM)-based approach for detecting credit card fraud that beat other algorithms including logistic regression and neural networks. However, their approach had drawbacks when it came to taking into account potential non-linear relationships between elements in credit card transaction data. I used a neural network instead of logistic regression for my project. They ran into a problem because of potential items in their dataset. Similar elements were also included in the dataset I utilized for my project, namely Vamount and Vtime. I utilized the df.drop functionality mentioned in (figure 3.4) to solve this problem.

Figure 3.4

```
# Standardizing the features
df['Vamount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df['Vtime'] = StandardScaler().fit_transform(df['Time'].values.reshape(-1,1))

df = df.drop(['Time','Amount'], axis = 1)
df.head()
```

The original 'Time' and 'Amount' columns are removed from the dataframe using the drop() method once the 'Amount' and 'Time' columns have been standardized. This is done with the argument axis=1, which denotes that columns



should be removed. With the addition of additional columns called "Vamount" and "Vtime," which hold the standardized values of "Amount" and "Time," respectively, the resulting data frame has all the columns from the original dataframe aside from "Time" and "Amount." The first few rows of the generated dataframe are then shown by executing the head() method.

By focusing on balancing the data before using it for the models, I was able to fix the main problem with other people's models, which was balancing the data, as seen in previous efforts.

## Conclusion and Future Work


### 4.9 Conclusion

Due to the potential for huge financial losses and harm to client confidence, the identification of credit card fraud is a critical issue for financial institutions. In order to reduce these risks, it is crucial to create precise fraud detection models. Increasing the effectiveness of credit card fraud detection was the main goal of this research. In the industry, 99.9% accuracy and statistical power are regarded as acceptable, therefore that is what I strived for.

In order to avoid being caught, fraudsters always come up with innovative and complex new methods. Therefore, to remain ahead of fraudulent activities, fraud detection algorithms need to be continually updated and enhanced. This necessitates continuous evaluation of the model's effectiveness and the addition of fresh tools and methods.

In conclusion, detecting credit card theft is an essential step for financial organizations. To prevent financial losses and preserve client confidence, fraud detection methods must be continuously developed and improved. Although the model's accuracy is crucial, additional measures including precision, recall, and





F1 score should also be considered. Finally, to keep ahead of fraudulent activities, financial institutions should exercise constant vigilance and adjust their fraud detection procedures.

## 5.0 Future Work

While accuracy is a crucial criterion for assessing how well fraud detection algorithms work, it is not the only thing to take into account. A high accuracy rate may still be achieved by an algorithm that merely classifies the majority of transactions as not being fraudulent, but this would be of low practical value. Metrics like F1 score, recall, and precision should also be taken into consideration in order to evaluate these algorithms' performance more accurately.

The F1 score is a metric that evaluates an algorithm's overall capacity to identify fraudulent transactions while reducing false positives. It considers both precision and recall. Recall is the percentage of true positives out of all real positive cases, whereas precision is the percentage of true positives out of all positive predictions. A high F1 score indicates that an algorithm has a strong balance of precision and recall.

Future research will improve fraud detection models' overall performance by concentrating on optimizing these indicators. For instance, algorithms can be developed to more accurately differentiate between various forms of fraud, such as account takeover fraud or card-not-present fraud, to increase their recall and precision. To lessen noise and enhance model performance, additional strategies can be used, such as feature selection or data pre-processing. Although accuracy is a crucial metric, it is insufficient to assess the efficacy of fraud detection systems on its own. A more detailed evaluation of an algorithm's performance can be obtained using metrics like F1 score, recall, and precision, enabling the creation of more accurate and effective models.

## Appendix

After carefully going over the criticism from my supervisor, I have included substantial adjustments and improvements in my most recent submission compared to my prior entries. I have discovered the errors in my earlier work that kept me from getting the marks I wanted through this approach. The opening section, which previously failed to effectively communicate the research message and lacked relevant information, is one of the significant changes I have made. In particular, I realized that the market research I included in my initial proposal wasn't necessary for my research study and changed the introduction to reflect this.

The background research portion was the one I changed the most, though. I only used information I found on the internet for my past submissions, which is not ideal for academic research. Realizing this, I went in search of people who produced publications to get more significant and reliable data to serve as a foundation for my research. This has led to a huge improvement in the standard of my research for my submission. Furthermore, I recognized that my previous submission was deemed too informal, so I took steps to ensure that my language and tone in this final submission was more professional and formal.

Finally, I enhanced my references by looking for and including reputable sources that offered stronger support for my claims and conclusions. This improved the general reliability and accuracy of my research. The modifications I made to my final submission show my dedication to conducting high-quality research and to thoughtfully responding to my supervisor's criticism.

## Code Repository and Dataset

To access the code and documentation for this project, please visit the corresponding GitHub repository:

<https://github.com/tahio12/Credit-card-fraud.git>

To access the Dataset, please visit the corresponding Kaggle repository:

`kaggle datasets download -d mlg-ulb/creditcardfraud`

<https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

## References

Kaggle dataset : <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

Bhattacharyya, D., Kalita, J. K., & Sharma, P. (2011). Credit card fraud detection using the hidden Markov model. *Procedia Computer Science*, 6, 583-588.

Chan, P. K., Stolfo, S. J., & Fan, W. (2016). Learn++: Combining learning paradigms for anomaly detection in security surveillance. *Journal of Information Assurance and Security*, 11(2), 67-76.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc.

Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: a review of classification and combining techniques. *Artificial intelligence review*, 26(3), 159-190.

Wang, X., Huang, J., Zheng, Y., Liu, J., & Yu, P. S. (2019). Detecting Credit Card Fraud Using Deep Learning. In *Proceedings of the 2019 IEEE International Conference on Big Data* (pp. 4672-4674). IEEE.

"Deep Learning with Python" by Francois Chollet