International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 06 | June - 2022 Impact Factor: 7.185 ISSN: 2582-3930

CREDIT CARD FRAUD DETECTION USING ISOLATION FOREST AND LOCAL OUTLIER FACTOR

RISHIKESHAN O V[1], SAKALA SAI KIRAN[2], PRASATH S[3], ANITHA M[4]

[1,2,3] UG Scholar, Department of CSE, Kingston College, Vellore-59
[4]Asst.Professor, Department of CSE, Kingston College, Vellore-59

Abstract - Nowadays, card fraud is increasing due to the prevalence of modern technology. Thus, Automatic systems to detect and prevent against card fraud are a significant tool in the financial industries battle against card crime. Machine learning and novelty detection techniques approaches are taken in Credit Card Fraud Detection field, since they are effective technologies and methodologies and easy to apply at the same time. It is used in order to reduce fraud activities. The main aim of this project was to create a program that detects and Identifies potentially fraudulent credit card transactions from a given data set, and evaluate its performance to be compared with other classifiers with evaluation metric.

The program was trained with the given data set, based on the some of the most popular Machine Learning and Deep Learning Classification algorithms which are; Random Forest, Isolation Forest and Neural Networks Algorithm Load Balancing and Feature Selection were maintained throughout the project, and it was implemented using Python programming language. However, there were no autonomous system that will be able to categorically define a transaction as fraud. The objective was to highlight those transactions that have a high probability of being fraudulent based on some criteria, known or otherwise learnt.

Key Words: isolation forest and local outlier factor, Pandas, Json, Matplotlib and seaborn.

1. INTRODUCTION

This chapter consists of, a preface that includes a brief background of credit card fraud detection topic, followed by aims and scope of the project, motivation, the main intended beneficiaries, and a listing of the report structure.

1.1 PREFACE

Nowadays, card fraud is increasing due to the prevalence of modern technology. As the card fraud losses total was around 567 million in 2015, and there was a 9% increase in 2016 with a loss of 618 million. Enterprises and public institutions must be well prepared to defend against such frauds that cause the loss of billions of dollars worldwide annually. Automatic systems to detect and prevent against card fraud are a significant tool in the financial industry's battle against card crime. However, it is not easy, or always possible to detect fraudulent patterns in transaction data by programmatic rules-based systems or inspection by fraud analysts; especially in large data sets. Credit card usage is increasing year on year, and the credit card and card payments market plays a huge role in today's economy. Predicted statistics by indicated that credit card payments will increase over the next decade. The predicted transactions growth is to 3.7 billion in 2026, from 2.8 billion in 2016. Which consequently indicated that economic growth is one of the significant drivers of the credit card volumes and values future. Moreover, credit card usage has well known advantages such as; simplicity of payment, access to credit, purchase guarantees and financial management to name a few, unfortunately these do not come without the risk of becoming a victim of fraudulent transactions. To reduce credit card fraud, it is important to use expert rules and statistical based models, such as rules based detection engines, machine learning and novelty detection techniques such as Clustering, Classification based and Nearest Neighbor, that can be used to distinguish potential and genuine fraud. Thus, classification is the one will be mainly used in order to solve this problem so it will be discussed in further details later. Moreover, effective technologies and methodologies that can detect fraud and illegal activities such as; money laundering have been delivered and applied. Machine learning algorithms have existing efficient fraudulent patterns (patterns will be discussed in detail at a later section), which typically found in data to be uncovered, or sometimes they do not, and predict possible fraudulent transactions, as it is a critical part of the fraud detection toolkit. A real or representative data set plays a significant role when it comes to build a system in machine learning as well. Specifically, to have a better fraud detection model there must be a large set of data to be used.

Figure 1: Preface image of project

1.2 AIM AND SCOPE

The main aims of this project are to establish a background in fraud detection novelty techniques and machine learning methodologies that will be used to complete this project, followed by detecting potentially fraudulent credit card transactions and classify both legitimate and fraudulent transactions from a given data set, where supervised and unsupervised Machine Learning Classification

Algorithms and Novelty Detection Techniques are used on training the models. In the given data set, each individual transaction is already allocated to a different known class, either for fraud or for legit. Finally, offering discussion, comparison between the techniques, and evaluation of the solution produced. However, it is important to remember during the whole project, that no autonomous system will be able to categorically define a transaction as fraud. The objective will be to highlight those transactions that have a high probability of being fraudulent based on some criteria, known or otherwise learnt. Correspondingly, the scope of this project focuses on creating a well-functioned software system model depending on known behaviors or feature variables. It allows the beneficiaries to inspect the performance or the accuracy of the various machine learning classification algorithms which can classify the transactions in training and testing sets into either fraud or legit in real time. The data given is highly unbalanced data set, therefore, the approach used in this project is to implement different machine learning algorithms, which are: Random Forest as there are many cases where it has been used successfully, Isolation Forest and Neural Networks.

2. RELATED WORKS

[1] The work done by the author, N.Kalaiselvi, S.Rajalakshmi, J.Padmavathi, Joyce B.Karthiga, "Credit Card Fraud Detection using learning to Rank Approach" in the year 2018.

An infrastructure build in the neural network platform is reliable to detect the fraudulence in credit card system for transaction. The issues resulting from the fraud in credit card transaction may involve a number of customers who drift their habits evolve and fraudsters who change their strategies over time. The vast majority of learning algorithms that have been proposed for fraud detection rely on assumptions that hardly hold in a real-world fraud-detection system. This includes the classification of data imbalance which is used to verify their hidden transaction, track the location of the fraudsters and to capture their image. The implementation of learning algorithm will precisely predict and rank alerts based on the scores allotted to each alert. Initially, we propose transaction blocking rule to ensure the security of the transaction. Secondly, we design scoring rules that involve pattern matching based on frequent data mining techniques. Finally, the machine is trained and updated with the dataset to timely investigate the transaction thereby earning credit card holder's satisfaction.

[2] The work done by the author, X. Yu, "Integrated Approach for Nonintrusive Detection of Driver Drowsiness", in the year 2012.

This project is the extension of Northland Advanced Transportation System Research Laboratory (NATSRL) FY 2008 and FY 2009 projects titled, "Real-time Nonintrusive Detection of Driver Drowsiness," which aims to develop a real-time, nonintrusive driver drowsiness detection system to reduce drowsiness-caused accidents. In our previous research, nonintrusive sensors for drivers' heart beat measurement were developed and implemented on the vehicle steering wheel. Heart rate variability (HRV) was analyzed from the heart beat pulse signals for the detection of driver drowsiness. Promising results were obtained. However, one of the major issues with the previous system is using only one parameter, Low-Frequency(LF)/High-frequency(HF) ratio of HRV, to access the driver's status, which has relative high variability and has different changing patterns for different drivers. In this project, we used multiple parameters for the drowsiness detection, including the LF/HF ratio, steering wheel reversal rate (SWRR) and steering wheel R2 (R2 = Pos2 +w2 , Pos is steering position and w is steering wheel moving speed), and EEG frequency band power ratio (Poweralpha+Powertheta)/ Power beta. Two-hour driving simulation tests were conducted on thirteen human subjects in a driving simulator. The driving simulation results show that large amplitude bursts of R2 value represent driving errors, which mostly happen after a significant decrease of SWRR. Also, bursts of the EEG ratio, which represent increased levels of drowsiness, happen after a fairly long-term decrease of ECG LF/HF ratio, as do the bursts of SWRR, which represents driving errors. The correlation above means that the information contained in these parameters time history can be used in drivers' drowsiness detection. Future work is needed to develop a complete and robust model to utilize these parameters to evaluate the drowsiness level of a driver during a long-time driving task.

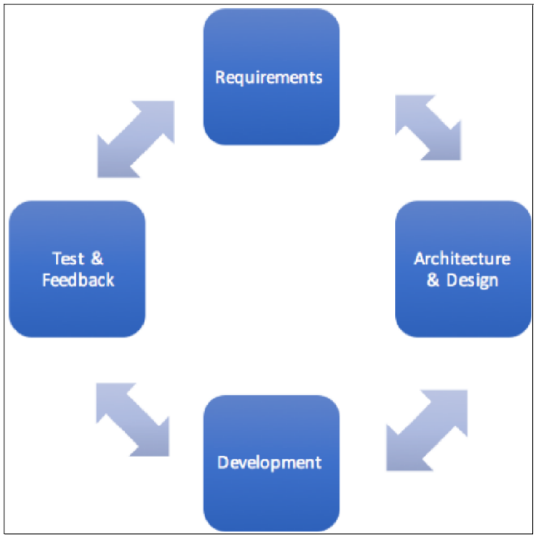
[3] The work done by the author, Sahil Dhankhad, Emad Mohammed, Behrouz Far, "Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study", in the year 2018.

The goal of data analytics is to delineate hidden patterns and use them to support informed decisions in a variety of situations. Credit card fraud is escalating significantly with the advancement of the modernized technology and become an easy target for fraudulent. Credit card fraud is a severe problem in the financial service and costs billions of a dollar every year. The design of fraud detection algorithm is a challenging task with the lack of real-world transaction dataset because of confidentiality and the highly imbalanced publicly available datasets. In this paper, we apply different supervised machine learning algorithms to detect credit card fraudulent transaction using a real-world dataset. Furthermore, we employ these algorithms to implement a super classifier using ensemble learning methods. We identify the most important variables that may lead to higher accuracy in credit card fraudulent transaction detection. Additionally, we compare and discuss the performance of various supervised machine learning algorithms exist in literature against the super classifier that we implemented in this paper.

3. METHODOLOGY

In order to tackle the software system that has to be developed and tested in the limited time given, a development methodology had to be followed. The development methodology ensures that developing and testing of the software system are managed, and at the end of the project a functioned prototype will be delivered. Thus, Agile software development strategy was chosen to be followed in this project, as shown in figure. Agile development strategy is a suitable method, as it focuses on incremental delivery,

continual planning and continual learning. It divides the project into iterations, small incremental builds. An advantage of the life cycle model is that the client will be able to see the working outcome straight after each iteration. This methodology was chosen in preference to Waterfall, for example, because if Waterfall was adopted then the implementation stage cannot take place unless the planning stage is done, and due to the fixed structure of the required steps. Which will not be suitable for this project as the timescale is not flexible at all, and the moderation of the requirements belong to this project is at a high risk of changing. However, in Agile any requirements modification at any stage of the process can be integrated even in late development process with no risking losing the entire work. Moreover, a way faster and successive iteration of a working software system will be produced regularly, and viewed by the client at a consistent pace, which results in client satisfaction with the rapid releases of the project. Lastly, after each iteration the client will be sending feedback that provide a huge opportunity to modify and improve the software in the upcoming iterations. Although Agile is a more teamwork methodology, planning is easier with a single developer. Most of the work came naturally, and I followed it to broke down the tasks that should be done, prioritizing the tasks based on feedback given by the supervisor in the meetings, set the intended goals and being flexible to change and modify along the project since the work was partitioned in weekly sprints.



Agile Development Strategy

Figure 2: Methodology Diagram

Five iterations have been utilized over the project timescale, objectives and deliverables of each iteration are listed below along with each iteration,

• First Iteration focused on implementing, loading and processing the data needed for each classifier.

– Deliverable from this iteration was preparing and splitting the training and testing sets to be ready to use in classification processes.

• Second Iteration focused on implementing the three separated classifiers from three different algorithms. Using the data set resulted from the first iteration allow the classifiers to be trained and tested.

– Deliverable from this iteration was producing three classifiers that are able tobe trained and tested.

• Third Iteration focused on the Load Balancing of both test and train sets.

– Deliverable from this iteration was balance the fraudulent and legitimate transactions to have a well balance model.

• Fourth Iteration focused on Feature Selection to train the model.

– Deliverable from this iteration was selecting the features manually or by using the best k features selection method.

• Fifth Iteration focused on a simple command-line interface application.

– Deliverable from this iteration was the user interface, in order to provide user interaction with the software, by containing the usage of the three previous iterations.

4. IMPLEMENTATION DETAILS

This chapter will discuss the implementation of the software system provided with the code. The main parts of the implemented code will be specified with the related tools used.

4.1 PROJECT STRUCTURE

This project was organized by splitting the code into separated small parts using Python modular programming, to produce a reliable, readable, and maintainable software system. The small parts are represented in classes, that can be collected later to produce a complete software system. Thus, the re-usability of the code is facilitated by the usage of the classifiers, and it ease the access for a specific functionality in the code. Thus, the final solution is implemented and structured in CCFD package that contains six different files demonstrated as follows,

• Random Forest: this classifier file represents a learning model based on Random Forest classification algorithm.

• Isolation Forest: this classifier file represents a learning model based on Isolation Forest classification algorithm.

• Neural Networks: this classifier file represents a learning model based on Neural Networks classification algorithm.

• Split Data: this file is responsible for load balancing, loading the historic data given by the user as an input, prepare them to be classified and split them into fraudulent and legitimate transactions.

• Evaluation: this file is responsible for model evaluation as it provides the evaluation techniques metric.

• Classification app: this file represents a simple command line application, that takes the inputs including the directory path of the file and the classifier from the user as arguments, classifies the data, loads and saves the final results of each classifier into models directory, along with the saved output pickled files.

4.2 DATA PREPARATION & PROCESSING

This section will mainly discuss how data set was prepared and processed to be used in each classifier.

4.2.1 DATA LOADING

Loading necessary data is the first step in the field of machine learning and its approaches. Since the data consists of historic labelled transactions in csv format, and it must be prepared for the classification algorithms, (built-in function) from csv module, read csv() was imported and called using pandas. The csv file directory will be taken as an argument from the user and will be converted into a Data Frame using pandas for further processing unit.

It is highly unbalanced data, whereas fraud and legitimate transactions are represented respectively, by approximately 2% and 98% as illustrated in the following code block,

```
Distribution of Legitimate (0) and Fraudulent (1):
0    284315
1       492
Name: Class, dtype: int64
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Figure 3: Data Distribution

The data set contains only numerical (continues) input variables resulted from a Principal Component Analysis (PCA) feature selection transformation in 28 major components out of the 30 components utilized in this project. The behavioral characteristics of the card is represented by, a variable of each profile usage represents the spending behaviors of the customer linked with hours, and days of the month, as well as geographical locations or the merchant type where the transaction happens. Therefore, to distinguish fraudulent activities these variables are used later. Unfortunately, the details and background information of the provided features cannot be specified due to confidentiality issues. The Time feature involves the seconds passed between each transaction and the first transaction occurrence. The Amount feature represents the amount of the transactions, and it has a relatively small mean of all the transactions made which is 88.3 as in the code shown below, and the Class feature is the response variable, as it distinguishes the transactions by labelling them into zeros and ones.

	Time	V1	...	Amount	Class
count	284807.000000	2.848070e+05	...	284807.000000	284807.000000
mean	94813.859575	1.165980e-15	...	88.349619	0.001727
std	47488.145955	1.958696e+00	...	250.120109	0.041527
min	0.000000	-5.640751e+01	...	0.000000	0.000000
25%	54201.500000	-9.203734e-01	...	5.600000	0.000000
50%	84692.000000	1.810880e-02	...	22.000000	0.000000
75%	139320.500000	1.315642e+00	...	77.165000	0.000000
max	172792.000000	2.454930e+00	...	25691.160000	1.000000

[8 rows x 31 columns]

Figure 4: Features descriptive statistics

4.2.2 TRAIN AND TEST SPLIT

Data set used for classifiers is usually split into training and test data sets. This process has been done to avoid over fitting problem detailed in the background section. Testing the accuracy of a classifier in its prediction of fraudulent transaction using the test set is essential, to know how well the model learned on the training set. Scikit-learn provides useful functions, fit() and predict(), for train and test the classifiers. Usually, a 70/30 or 80/20 split is used for train and test. However, a load balancing has been conducted in this project instead, and detailed in the next section. Moreover, the training and testing sets will be constructed at the end as dataset train, and dataset test, to be used and divided into transactions and labels for both training and test sets, as the code in figure shown below.

```
selected_features = list(dataset_train)[:-1]

X_train = dataset_train[selected_features]
y_train = dataset_train[['Class']]

X_test = dataset_test[selected_features]
y_test = dataset_test[['Class']]
```

Figure 5: Train and test sets

4.2.3 LOAD BALANCING

For this project, it can be seen from the above mentioned information that the data set is highly unbalanced. Attempting to train a model with the given data combination, will result in over fitting as the model will conveniently converge into classifying everything as legitimate. Consequently, the high accuracy of training sets will be resulted as the fraudulent transactions are so few and the outlier fraudulent transactions will never be detected. So, in order to balance the data, the number of zero instances which represent legitimate transactions, and one instances which represents fraudulent transactions used must be balanced. For test sets the amount was fixed to 42 instances for both zeros and ones so that it is could assess how the model classifies true positives and true negatives accurately. In regards to the training sets, the number of one instances was set to 450 as there was no other option, and the number of zeros was adjusted into the data combination chosen throughout different experiments. The number of zero labelled rows was randomly chosen and experiments were run a number of times for each data mixture in order to understand the fluctuation of accuracy due to the varied data samples. A sample of the instance distribution can be seen in figure shown below.

```
def split_data(n_zeros=1500, n_ones=450, n_zeros_test=42, n_ones_test=42, k=30, alg='Random Forest'):
```

Figure 6: Instance distribution in split data ()

4.2.4 FEATURE IMPORTANCE

As mentioned about features selection in the background section, feature importance is a significant quality in ensemble algorithms, specifically in random forests. It is being utilized in this project because it measures the relative importance of each feature in the prediction. The attribute feature importance was implemented in Random Forest class; it returns an array of numbers of each feature importance determining the splits. Therefore, a combination of Seaborn and Matplotlib libraries were used for a good visualization of the results in ascending order, the code is shown in figure below.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize = (7,3))
feat_importance = pd.DataFrame({'Feature': features_list[:-1], 'Feature Importance': clf.feature_importances_})
feat_importance = feat_importance.sort_values(by='Feature Importance', ascending=True)
graph = sns.barplot(x='Feature',y='Feature Importance', data=feat_importance)
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
graph.set_title('Features Importance using Random Forest', fontsize=7)
plt.show()
```

Figure 7: Feature importance

4.2.5 FEATURE SELECTION

Another important step of data preparation is the feature selection process. This data set provides 32 data headers (features). Experiments were run to choose the k best features of this feature set using SelectKBest() from Scikit-learn library. Choosing features can either be done manually by changing the k value of each classifier in each try, or by setting the k value into None which basically selects all features except the labels, and this option allows the classifier to be trained using the features with the highest scores. The code shown in figure below, also indicates that the method takes two attributes, f classify which represents the ANOVA F-value between a label or feature for the classification tasks, and the k value that represents the features.

```
if k is not None:
    sel = SelectKBest(f_classif, k=k)
    sel.fit(X_train, y_train)
    columns = sel.get_support()
    cols_indices = X_train.columns[columns]
    X_train = X_train[cols_indices]
    X_test = X_test[cols_indices]
    pickle.dump(cols_indices, open('models/indices_' + alg + '.pkl', "wb"))

return X_train, X_test, y_train, y_test, features_list
```

Figure 8: SelectKBest()

4.2.6 TRAINING AND EVALUATING THE MODEL

The selected features variable shown in figure below, in train and test split section, will contain the whole list of features except the labels represented by feature. The labels are assigned by both variables y train and y test. However, this will result in having the training set represented by X train and y train, and test represented by X test and y test. All the variables will be returned by the split data() method. To train the classifier and fit it into the training set the function fit() has been implemented in each model. This function requires passing the training set transactions and the labels as arguments. The predict() function was also used by the classifiers in the evaluation part to classify the received data points from unseen data, or a testing test in some cases.

4.2.7 PRE-PROCESSED DATA

Since random sampling is used in load balance the training data set, a random sample in training set is reproduced in each run of the code. However, for model optimization the same input data must be used, to avoid producing two different results. Moreover, the observed change in the accuracy is resulted from changing the parameters of the chosen algorithm not from the change of input data. So, pickle is used to save data in orderto ensure that the experiments done are reproducible, also to ensure that the input data has not been changed. This will lead to deterministic results, which allows running the same experiment with the same parameters twice and obtain exactly the same results. Since pickle is a persistence model, it was implemented in split data() method. Pickle created object files that will contain classifier models created previously, and it can be saved and loaded. In order to store an object externally, for a classifier object, pickle uses the pickle.dump() function to save the data. Whereas in loading the object , pickle.load() will be used to de-serialise the data. This allows the user to use a previously saved objectof the chosen classifier, if the read from pickle Boolean variable shown in figure1 is set to True, the code in figure2 will be executed to handle loading the saved classifiers for training and testing. Otherwise, False, will allow the system to create new objects for both training and testing, that can be used in similar further experiments.

```
read_from_pickle = False
if not read_from_pickle:
    data = pd.read_csv('creditcard.csv')
```

Pickle objects - set the value

```
else:
    # Skip re-train and re-test the data, read directly from the saved classifiers in pickle files
    dataset_train = pickle.load(open("train.pkl", "rb"))
    dataset_test = pickle.load(open("test.pkl", "rb"))
```

Pickle objects - loading file

Figure 9: Pickle Objects

4.3 CLASSIFIERS

For the purpose of this project, three types of classifiers have been created, and constructed as three separated classes that implement split data() function from Split Data.py and evaluate classification() function from Evaluation.py.Scikit-learn classifiers have been also imported where each classifier class implements the associated model of its learning algorithm.

Hence, RandomForestClassifier has been constructed from Scikit-learn to represent the model of Random Forest classifier. IsolationForest has been also con- structed for Isolation Forest classifier, and for Neural Networks Sequential model has been imported from Keras library.

4.3.1 RANDOM FOREST

Random Forest algorithm is the supervised classifier chosen to predict whether a transaction is legitimate or fraudulent. In order to classify fraudulent transactions by this classifier, RandomForestClassifier has been imported from Scikit-learn ensemble module along with pickle module. As shown in figure, the hyper-parameters chosen in the implemented code were, the n estimators, representing the tree numbers, and set to 100. Also, random state was set to 0 and the max depth was set to None. Additionally, both X-train and y train were used into fit() function to create the model.

```
#number of selected k features
k = None
X_train, X_test, y_train, y_test, features_list, data = split_data(k=k, alg='Random Forest')

clf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=0)
model = clf.fit(X_train, y_train.values.ravel())
```

Figure 10: Random Forest Classifier

4.3.2 ISOLATION FOREST

Isolation Forest is unsupervised classifier, created to be compared to the supervised and the deep learning algorithm implemented in this project. Along with pickle module,

IsolationForest was imported from Scikit-learn to create the model by training the data using fit() function that takes X train as a parameter. As shown in figure below, the hyper parameters chosen in this classifier were random state which was set to zero,and n estimators was set to 100.

4.3.3 NEUTRAL NETWORKS

To construct a Neural Network model, Sequential model API alongside a network of dense layers were imported from Keras library. Keras was also imported from Tensorflow as canbe seen in figure,

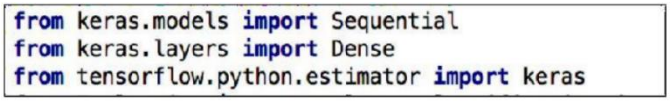


Figure 11: Imported Libraries

The first step was building the classifier using Sequential constructor that will have layers added to it using the function add () as shown in figure. The layers added are of type Dense, the 30 represents the output size, whereas 15 represents the output in the second Dense layer, and both can be changed as preference. The option relu, Rectified Linear Unit, was chosen as an activation function. Whereas, the last layer presents the predictions using sigmoid which is a smoother function to get more accurate probabilities for classification. The k best feature can also be utilized using neural networks if the features used in the k value are set to all, using the value None, that allows passing all features for the SelectKBest().

The method compile () of the Sequential model was used, and it requires three arguments to be passed, a loss function, an optimizer which set to Adamant a list of metrics. The three passed arguments are presented in figure below. Last step was fitting the model using fit() function which requires at least two arguments: input and target tensors, noting that only a single iteration of the training data will be performed when passing two arguments. In the case of this project, it is shown below that five arguments have been passed, X test and y test representing the training set, epochs which represents the number of batches, batch size which is the number of the samples passed, and verbose which is the output logging. The evaluation performance was calculated and printed out using the evaluate classification () function as what have been done with the other classifiers.

4.3.4 CLASSIFIERS EVALUTON

In order to evaluate the implemented classifiers, the evaluation method evaluate classification()function was imported from Evaluation.py. Scikit-learn Confusion Matrix, Accuracy, Precision and Recall classification metric functions have been imported in this class, and utilised as specified in the learning evaluation section in the background. Each evaluation function requires passing the classification algorithm along with the testing sets and the predictions. However, the prediction is calculated differently for each algorithm as illus-trated in figure shown below. For random forest, for example, it is declared to be used as the default, whereas in neural networks and isolation forests, -1 will be returned for outliers (Fraudulent), and 1 (Legitimate) for inlier outputs. Therefore, this alteration was done to ensure that classification outputs are consistent across all three algorithms and that the system will automatically trigger the chosen one. Prediction along with other necessary variables are used in the method. This method calculates and prints out the confusion matrix, and construct the accuracy, recall, and precision scores, that will be listed for each classifier results later on the Evaluation section.

Each evaluation metric with the function used from Scikit-learn, is listed below,

- Confusion Matrix: confusion matrix()
- Accuracy: accuracy score()
- Precision: precision score()
- Recall: recall score()

4.4 CLASSIFIERS COMMAND-LINE INTERFACE

A simple command-line interface classification application was utilized, as discussed in the specification and design part previously. It has been utilized for number of reasons. Firstly, it is user-friendly and easy to use, it provides different classification algorithms can be chosen by the user depending on the entered argument, it automatically generates help and usage messages as well, and provide issues errors when the user input invalid arguments. To support this solution, argparse python module was utilized, and the simple command line application has been implemented in classification app.py. The get args() function has ArgumentParser which is an object that holds all the needed information to parse a command line into python data types, and it has a list of arguments with different options available added using add argument() function, figure below shows the implemented code of this part. Moreover, the information is saved and used by parse args() function. The arguments will be added up using the accumulate () function. In order to run the application, the user must pass two arguments to the command line, the model of choice (ie. Random forest, Neural networks or Isolation forests) and the path of the file containing the data set to classify. Once the arguments are passed, the app chooses the corresponding serialized classification model as well as the selected feature indexes for it, which have been saved as serialized pickled objects after training. For this reason, it is important that any new data set, must follow the same structure as the training data set given.

5. SYSTEM DESIGN

A Unified Modelling Language (UML) is created to have a better understanding of the system. UML diagrams also provide the developer with a template as a guide used to construct the software system, and it present a visual representation of the software system and how it is intended to be. Therefore, this section will present diagrams that have beencreated and provided from UML below, the class diagram shown below in figure 3.1 presents the interaction of different classes in the software system, and the internal structure of it.

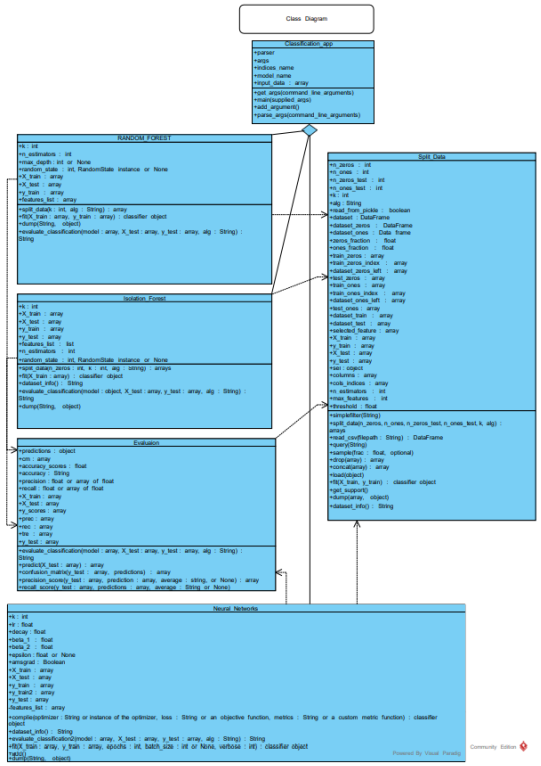


Figure 12: Class Diagram

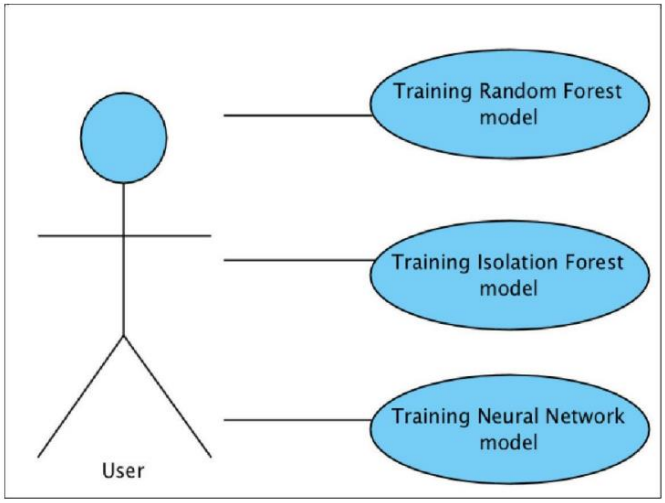


Figure 13: Use Case Diagram

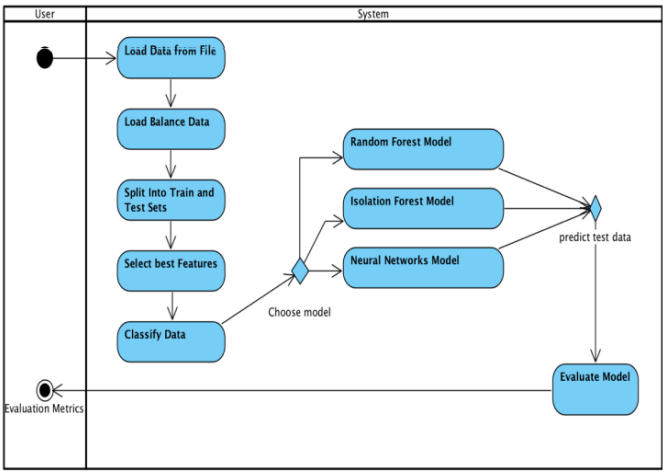


Figure 14: Activity Diagram of Software System

A brief explanation of the most important classes will be listed below,

• Evaluation: this class represents the main part of evaluating how good is a classifier model, this will be represented by providing the evaluation performance techniques, which are confusion matrix, accuracy, recall and precision.

• Split Data: this class is responsible for load and prepare the data set and split it into training and testing sets. By pre-processing the data and creates a Data Frame pandas, it allows loading the data into the system and provides a necessary representation of the data as features and labels to be used later in the software system by the classifiers. It is also important for load balance the fraudulent and legiti- mate transactions which helps to produce a better model. Also, information about the data sets will be provided including the distribution amount of fraudulent and legitimate transactions used in training and testing.

• Classification app: This class represents a simple command-line interface to allow the user to provide a data set, and choose a classifier alongside the arguments needed for the data splitting process to create a load balanced training and testing sets, to be used by the chosen classifier. And it represents the final results of the classifier. Moreover, the use case diagram shown in figure 3.2 represents a grouped set of possible interactions between the system and the user in the software system, where possible classifying options are illustrated. The three available options are Random Forest, Isolation Forest and Neural Networks. Furthermore, figure represents the approach involving dynamic aspects of the implemented software system as an activity diagram. The flowchart involves the steps taken which are; load the data from the given file, load balance the data, split data into training and test sets, select the best features, and finally classify the data using one of the available models. As mentioned previously, the models are Random Forest, Isolation Forest and Neural Networks. Finally, the predicted test set will be resulted and evaluated using the evaluation metrics. Additionally, this section provides all the functional requirements represented in an easy format to read and track.

6. RESULTS

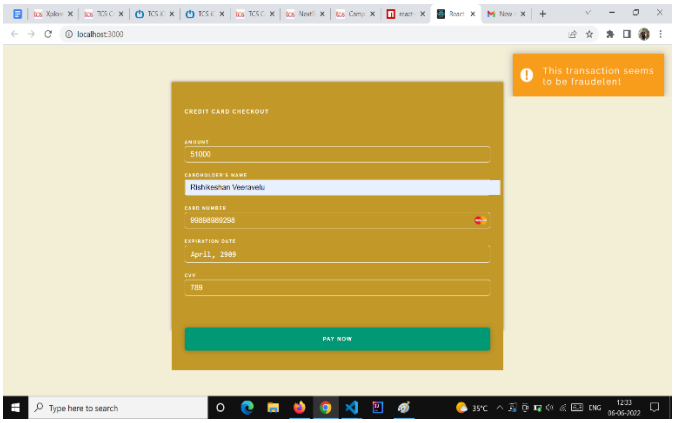


Figure 15: Fraudulent Check Screen

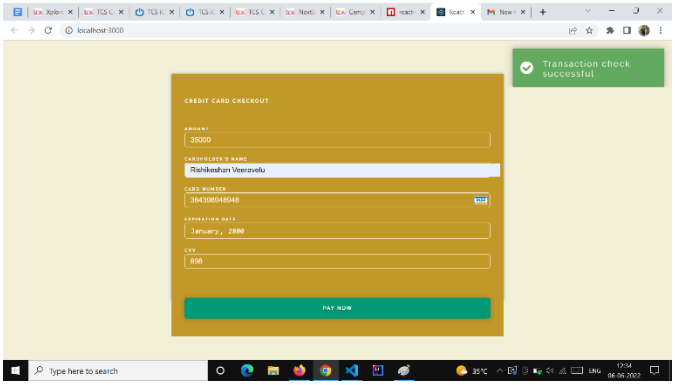


Figure 16: Proper Check Screen

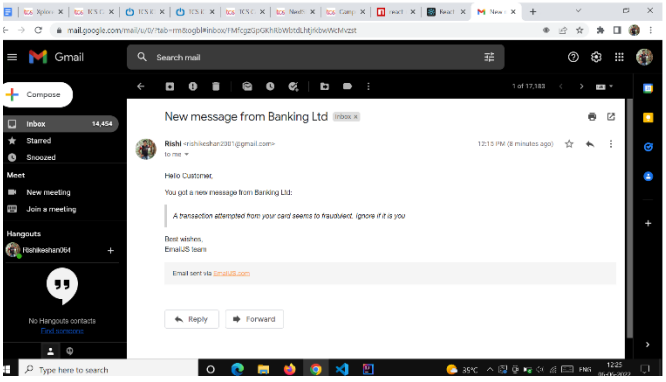


Figure 17: Indication of fraudulent through mail

7. CONCLUSIONS

To conclude, throughout this report, multiple tests were performed on the above mentioned implemented classifiers, in order to evaluate and find the best one that can classify credit card fraudulent and legitimate transactions using machine learning and deep learning techniques. This part was solved by three different models, and evaluated using the performance metrics including accuracy, precision and recall. In terms of accuracy, according to the results shown in the summary above, different experiments were utilized and resulted in high accuracy scores achieved by both random forest and neural networks models in different experiments. The lowest accuracy score achieved over the experiments was by Isolation Forest model. Whereas, as Random Forest achieved the highest. A simple command-line interface was also utilized to run the created engine and to evaluate each classifier independently, as the user chose. However, the accuracy of the output is linked to the quality of the training phase, which is in itself is no simple task, and yet the stated output is to a large extent inexplicable. Therefore, it cannot be decided which is most suitable classifier in both, solving the credit card fraud detection problem and predicting fraudulent transactions correctly as it depends on the data set given and the experiments applied. However, on the plus side unlocking the power of the CPU is done to continuously evaluate and re-evaluate data sets, to find things or achieve things that at this time there is no other way to do it. A very large simulated brain is used to identify patterns and then highlight them as useful results. This obviously opens up opportunities in areas such as speech recognition, big data analysis studies etc. The learning aspect here avoids the programmers of having to describe language to the computer for example, or to programmatically instruct a computer exactly what card fraud will look like based on the current knowledge.

ACKNOWLEDGEMENT

The authors would like to thank Ms. M Anitha for his suggestions and excellent guidance throughout the project period.

REFERENCES

[1]. Credit Card Fraud Detection using learning to Rank Approach", N.Kalaiselvi, S.Rajalakshmi, J.Padmavathi, Joyce B.Karthiga, 2018.

[2]. X. Yu, "Integrated Approach for Nonintrusive Detection of Driver Drowsiness," University of Minnesota Duluth October 2012 2012.

[3].Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study", Sahil Dhankhad, Emad Mohammed , Behrouz Far, 2018 IEEE International Conference on Information Reuse and Integration(IRI).

[4]. "A Novel approach for Credit Card Fraud Detection", Ayushi Agrawal, Shiv Kumar, Amit Kumar Mishra, 2015 2nd International Conference on Computing for Sustainable Global Development(INDIACom).

[5].Almarsoomi, H. and Kurnaz S. 2019. Credit Card Fraud Detection Us-ing Machine Learning Methodology. International Journal of ComputerScience and Mobile Computing,[Online] 8(3),pp.25760. Availableat:https://www.academia.edu/38608138/Credit_Card_Fraud_Detection_usin g_ Machine_Learning_Methodology_ [Accessed 9 Apr. 2019].

[6]. Asaithambi, S. 2018. Why, How and When to apply Feature Selection [Online].Available at:https://towardsdatascience.com/ why-how-and-when-toapply-feature-selection-e9c69adfabf2 [Accessed 23 Apr. 2019].

[7]. Breiman, L. 2001. Random Forests. Machine Learning, 45, 5-32 [Online]. Available at: http://dx.doi.org/10.1023/A:1010933404324 [Accessed 5 Apr. 2019].

[8]. Bronlee, J. 2017. How to Use the Keras Functional API for Deep Learning [Online]. Available at: https://machinelearningmastery.com/ keras-functionalapi-deep-learning/ [Accessed: 26 Apr. 2019]

[9]. Brown G. 2009. Ensemble learning. C. Sammut, G. Webb (Eds.) Encyclopedia of Machine Learning, Springer [Online].Available at: http://www.springer.com/ computer/artificial/book/978-0-387-30768-8. [Accessed: 7 Apr. 2019]