RESEARCH PAPER

CREDIT CARD FRAUD DETECTION SYSTEM

Raghav Sukhwal
Shivam Sharma
Dhruv Kumar
Department of Computer Science
Application Sharda University Noida, Uttar Pradesh, India

Mr Amit Kumar
Department Of Computer Science Engineering
Sharda University Noida,
Uttar Pradesh, India

***Abstract*— For clients to avoid being charged for items they did not buy, credit card companies must be able to identify fraudulent credit card transactions. To overcome such challenges, Data Science and Machine Learning might be applied. This study uses Credit Card Fraud Detection to show how machine learning can be used to model a data collection. The Credit Card Fraud Detection Issue includes modelling previous credit card transactions using information from transactions that turned out to be fraudulent.**
The model is then applied to assess the likelihood of fraud in a new transaction. Our objective is to eliminate erroneous fraud classifications while detecting all fraudulent transactions. Credit card fraud detection is an excellent illustration of classification. During this process, we focused on analysing and pre-processing large data sets as well as implementing multiple anomaly detection methods.

I. INTRODUCTION

A credit card allows the individual identified on it to charge products or services to his account, for which he will get regular invoices. It also carries identity information such as a signature or a photograph. The information on the card is currently read by automated teller machines (ATMs), store readers, banks, and online internet banking systems.
They have a unique card number, which is important. Both the plastic card's physical security and the privacy of the credit card number are necessary for its security.

A major rise in fraudulent activity has been brought on by the quick growth in credit card transactions. Credit card fraud is defined as theft or fraud that uses a credit card as a false source of funds during a transaction.

Statistical approaches and a range of data mining algorithms are frequently utilised to address this fraud detection issue. The bulk of credit card fraud detection systems employ artificial intelligence, meta learning, and pattern matching. Evolutionary algorithms called genetic algorithms search for more effective fraud detection techniques. The creation of an effective and secure electronic payment system is given top attention in order to establish whether a transaction is fraudulent or not. This essay will examine credit card fraud and methods of detection. When someone uses another person's credit card for their own personal use without the owner's knowledge, it is considered credit card fraud. In such cases, it is exploited by fraudsters up until its available credit is depleted. We therefore need a solution that lowers the total credit card limit, which is more susceptible to fraud. Also, a genetic algorithm improves its results over time. The creation of an effective and secure electronic payment system is given top priority for detecting fraud.

II. LITERATURE REVIEW

A fraudulent act is one that is illegal or criminal and is done with the intention of obtaining money or other benefits. It is a deliberate action carried out in contravention of a law, regulation, or policy with the aim of obtaining illegal financial benefit.

A number of publicly accessible literatures on anomaly or fraud detection in this domain have already been published. A thorough study conducted by Clifton Phua and his colleagues revealed that data mining applications, automated fraud detection, and adversarial detection are some of the techniques employed in this field. Unusual methods, based on network reconstruction algorithm that allows creating representations of the divergence of one instance from a reference group, have been effective on medium-sized online transactions. One such method is hybrid data mining/complex network classification algorithm. No algorithm can reliably predict if a transaction is fake, making fraud detection a challenging task.

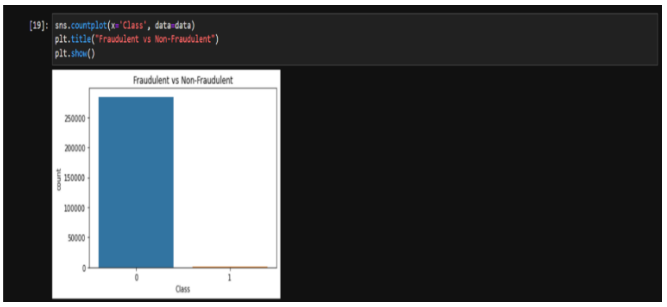
The following are characteristics of a good fraud detection system:

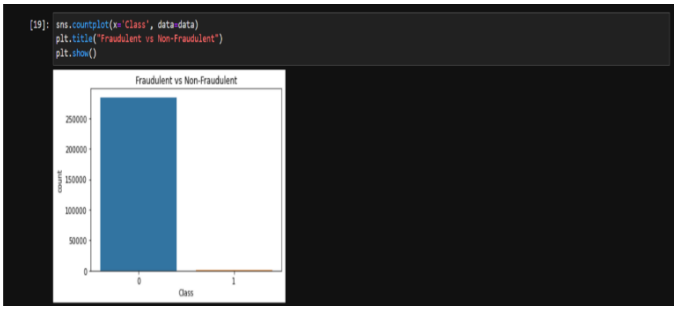
- It is important to appropriately identify the frauds.
- Frauds need to be found out right away.
- An honest transaction shouldn't be labelled as fraudulent.

III. PROBLEM IDENTIFICATION

The project's objective is to use machine learning algorithms to predict fraudulent credit card transactions. From both the bank's and the customer's perspectives, this is essential. The banks cannot afford to let fraudsters steal their clients' money. Since the bank is accountable for the fraudulent transactions, every fraud results in a

The dataset includes transactions made by cardholders of European credit cards over the course of two days in September 2013. The dataset is much skewed, with frauds making up 0.172% of all transactions in the positive class. When creating the model, we must be mindful of the data imbalance and use a variety of algorithms to get the optimum model.





IV. FUNCTIONALITY

To detect and prevent credit card fraud, a number of machine learning and deep learning algorithms have been employed, including logistic regression, naive bays, decision trees, SVM, and random forest. By categorizing them into useful groups, this study surveys the machine learning algorithms used for credit card transaction fraud detection. Based on accuracy, precision, recall, etc., a thorough comparison of different methods is also made. In the end, thorough insights.

On the basis of training data, the Classification algorithm is a Supervised Learning technique that is used to categories new observations. A programmer that does classification divides new observations into several classes or groups, such as Yes or No, 0 or 1, etc., after learning from the dataset or observations provided. Targets, labels, or categories can all be used to describe classes. The outcome variable of classification, in contrast to regression, is a category rather than a value. The classification algorithm uses labelled input data, which means that it has input and the associated output, as it is a supervised learning technique.

Classification Algorithms

- Decision Tree Classifier

A supervised learning method called a decision tree can be used to tackle classification and regression issues, although it is most frequently used to solve classification issues.

Supervised Machine Learning techniques like decision trees involve continuously segmenting the data based on a particular parameter. Decision nodes and leaves are the two components that can be used to explain the tree.

- Logistic Regression

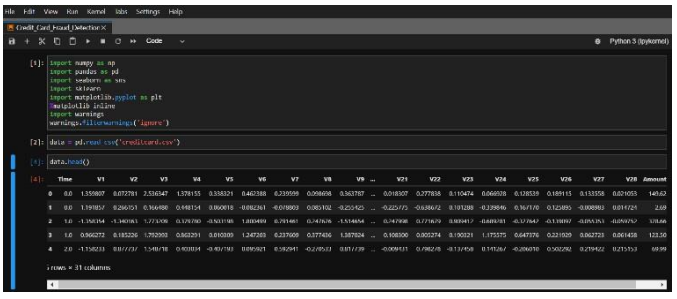
A given collection of independent variables are utilized to predict the categorical dependent variable using the supervised learning technique known as logistic regression. In a categorical dependent variable, the output is predicted via logistic regression. As a result, the result must be a discrete or categorical value. Either Yes or No, 0 or 1, etc., are possible. So rather than providing an exact result of 0 or 1, it provides probabilistic values that are in the range of 0 and 1.

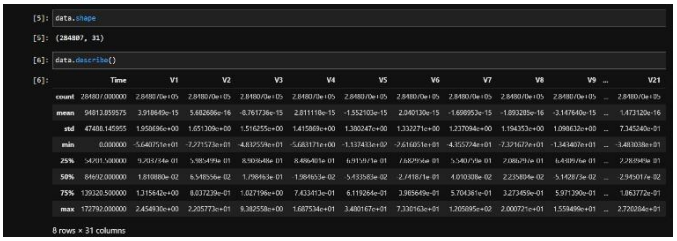
- Random Forest Classifier

A popular supervised machine learning technique for solving classification and regression issues is random forest. Using the average for regression and the majority vote for classification, it builds decision trees from a variety of samples. The Random Forest Algorithm's ability to handle data sets with both continuous and categorical variables, as in regression and classification, is one of its key features. It offers better outcomes for classification challenges.

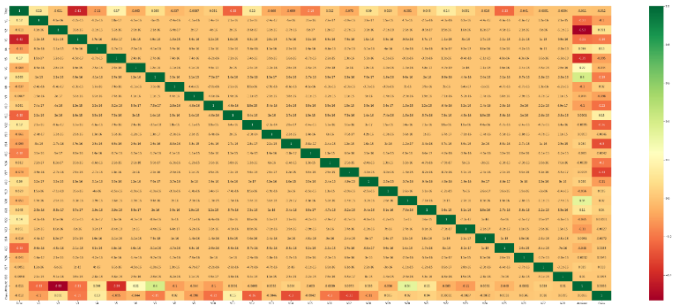
- XGBoost Classifier

A supervised machine learning approach used for structured and tabular data is called XGBoost classifier. Both approaches fall under the umbrella of supervised machine learning. A gradient boosted decision tree implementation created for speed and performance is called XGBoost. An extreme gradient boost algorithm is XGBoost. Thus, it is a large machine learning method with numerous components. Large, intricate datasets are compatible with XGBoost. An ensemble modelling method is XGBoost.

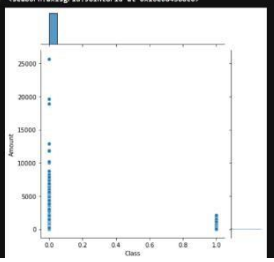




```
[15]: cormat = data.corr()
      top_corr_features = cormat.index
      plt.figure(figsize=(30,20))
      g = sns.heatmap(data[top_corr_features].corr(),annot=True, cmap="RdYlGn")
```



```
[23]: sns.jointplot('Class','Amount',data=data)

[23]: <seaborn.axisgrid.JointGrid at 0x16e0843dc0>
```

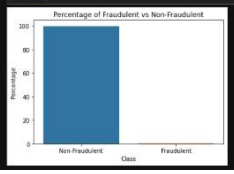


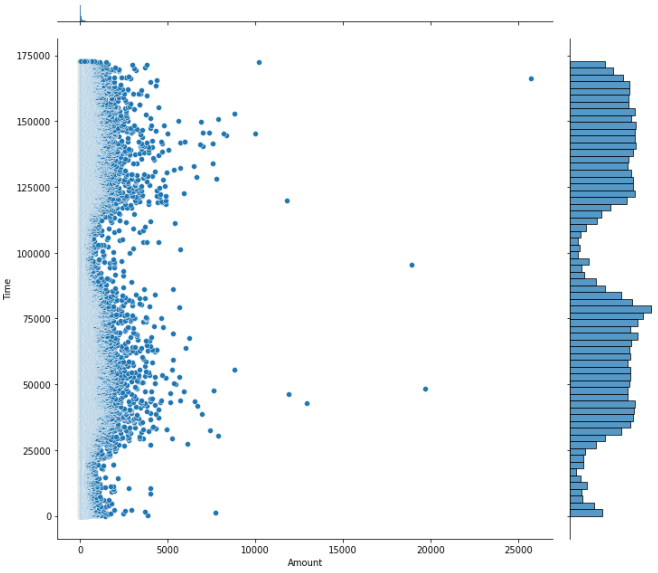
```
[25]: plt.figure(figsize=(10,10))
      sns.jointplot(x=data.Amount, y=data.Time, height=10)
      plt.show()
      sns.despine()

      <Figure size 720x720 with 0 Axes>
```

As can be seen, some of our predictors seem to be connected to the class variable. Despite this, there don't seem to be many meaningful links for so many different variables.

```
[19]: sns.countplot(x='Class', data=data)
      plt.title("Fraudulent vs Non-Fraudulent")
      plt.show()
```

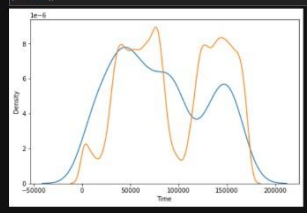

```
[20]: fraud_percentage = {'Class':['Non-Fraudulent', 'Fraudulent'], 'Percentage':[normal_share, fraud_share]}
      data_fraud_percentage = pd.DataFrame(fraud_percentage)
      sns.barplot(x='Class', y='Percentage', data=data_fraud_percentage)
      plt.title("Percentage of Fraudulent vs Non-Fraudulent")
      plt.show()
```





We learned that there is no discernible relationship between class distribution and time.

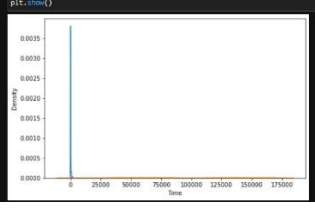
```
[21]: plt.figure(figsize=(8,5))
      ax = sns.distplot(fraud['Time'], label='fraudulent', hist=False)
      ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
      plt.show()
```

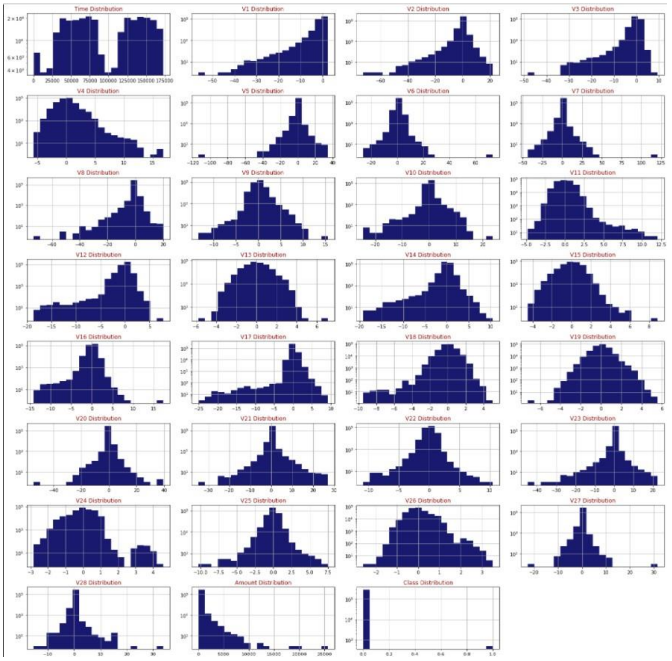



```
[26]: def draw_hist(dataframe, features, rows, cols):
          fig = plt.figure(figsize=(20,20))
          for i, feature in enumerate(features):
              ax=fig.add_subplot(rows,cols,i+1)
              dataframe[feature].hist(bins=20,ax=ax,facecolor='midnightblue')
              ax.set_title(feature+ " Distribution",color='DarkRed')
              ax.set_yscale('log')
          fig.tight_layout()
          plt.show()

      draw_hist(data,data.columns,8,4)
```

Distribution of classes with amount and time :- We came to know that fraud is maximum for only lower amount whereas for higher amount fraud is very less.

```
[22]: plt.figure(figsize=(8,5))
      ax = sns.distplot(fraud['Amount'], label='fraudulent', hist=False)
      ax = sns.distplot(normal['Time'], label='non-fraudulent', hist=False)
      plt.show()
```





```
[10]: import matplotlib.gridspec as gridspec
      plt.clf()
      pca_features = data.columns[1:28]
      plt.figure(figsize=(16,28*4))
      gs = gridspec.GridSpec(28, 1)
      for i, col in enumerate(data[pca_features]):
          ax = plt.subplot(gs[i])
          sns.distplot(data[col][data.Class == 0], bins=50, label='Valid Transaction', color='green')
          sns.distplot(data[col][data.Class == 1], bins=50, label='Fraudelent Transaction', color='red')
          ax.set_xlabel('')
          ax.set_title('Histogram of feature: ' + str(col),fontsize=15)
          plt.legend(loc='best',fontsize=12)
      plt.show()

      <Figure size 432x288 with 0 Axes>
```

Split & Train Model

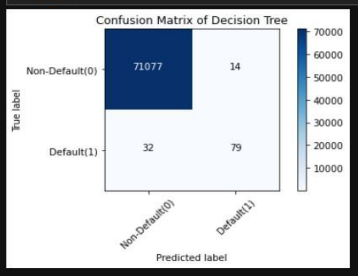
```
[30]: from sklearn.model_selection import train_test_split, GridSearchCV
      from sklearn import metrics
      from sklearn import tree

[31]: X = data.drop('Class', axis = 1).values
      y = data['Class'].values

[32]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 1)
```

Confusion Matrices

- DecisionTreeClassifier

```
[33]: from sklearn.metrics import confusion_matrix
      from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
      from sklearn.metrics import accuracy_score

[34]: DT = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
      DT.fit(X_train, y_train)
      dt_yhat = DT.predict(X_test)

[35]: print("Accuracy = {}".format(accuracy_score(y_test, dt_yhat)))

      Accuracy = 0.9993539507317211

[36]: confusion_matrix(y_test, dt_yhat, labels = [0, 1])

[36]: array([[71077,    14],
             [   32,    79]], dtype=int64)

[37]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

          fmt = '.2f' if normalize else 'd'
          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
              plt.text(j, i, format(cm[i, j], fmt),
                       horizontalalignment = 'center',
                       color = 'white' if cm[i, j] > thresh else 'black')

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
```

```
[39]: import itertools
      tree_matrix = confusion_matrix(y_test, dt_yhat, labels = [0,1])
      tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                           classes = ['Non-Default(0)','Default(1)'],
                                           normalize = False, title = 'Decision Tree')
      plt.savefig('tree_cm_plot.png')
      plt.show()
```

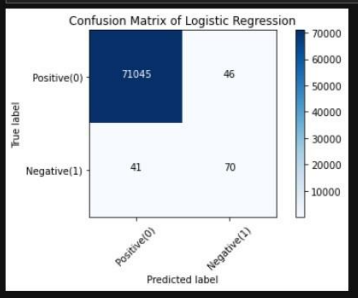



```
[40]: confusion_tree = confusion_matrix(y_test, dt_yhat, labels = [0, 1])

[41]: TP = confusion_tree[1,1]
      TN = confusion_tree[0,0]
      FP = confusion_tree[0,1]
      FN = confusion_tree[1,0]

[42]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9993539507317211
      Precision :  0.8494623655913979
      Sensitivity :  0.7117117117117117
```

- Logistic Regression

```
[43]: from sklearn.linear_model import LogisticRegression

[44]: lr = LogisticRegression()
      lr.fit(X_train, y_train)
      lr_yhat = lr.predict(X_test)

[45]: print("Accuracy = {}".format(accuracy_score(y_test, lr_yhat)))

      Accuracy = 0.9987781242099941

[46]: confusion_matrix(y_test, lr_yhat, labels = [0, 1])

[46]: array([[71045,    46],
             [   41,    70]], dtype=int64)

[47]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

          fmt = '.2f' if normalize else 'd'
          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
              plt.text(j, i, format(cm[i, j], fmt),
                       horizontalalignment = 'center',
                       color = 'white' if cm[i, j] > thresh else 'black')

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
```

```
[48]: lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0,1])
      lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                         classes = ['Positive(0)','Negative(1)'],
                                         normalize = False, title = 'Logistic Regression')
      plt.savefig('lr_cm_plot.png')
      plt.show()
```

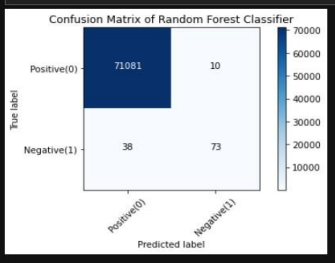



```
[49]: confusion_lr = confusion_matrix(y_test, lr_yhat, labels = [0, 1])

[50]: TP = confusion_lr[1,1]
      TN = confusion_lr[0,0]
      FP = confusion_lr[0,1]
      FN = confusion_lr[1,0]

[51]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))

      Accuracy :  0.9987781242099941
      Precision :  0.603448275862069
      Sensitivity :  0.6306306306306306
```

- RandomForest Classifier

```
[52]: from sklearn.ensemble import RandomForestClassifier

[53]: rf = RandomForestClassifier(max_depth = 4)
      rf.fit(X_train, y_train)
      rf_yhat = rf.predict(X_test)

[54]: print("Accuracy = {}".format(accuracy_score(y_test, rf_yhat)))
      Accuracy = 0.9993258616331002

[55]: confusion_matrix(y_test, rf_yhat, labels = [0, 1])

[55]: array([[71081,    10],
             [   38,    73]], dtype=int64)

[56]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

          fmt = '.2f' if normalize else 'd'
          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
              plt.text(j, i, format(cm[i, j], fmt),
                       horizontalalignment = 'center',
                       color = 'white' if cm[i, j] > thresh else 'black')

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
```

```
[57]: rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0,1])
      rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                         classes = ['Positive(0)','Negative(1)'],
                                         normalize = False, title = 'Random Forest Classifier')
      plt.savefig('rf_cm_plot.png')
      plt.show()
```

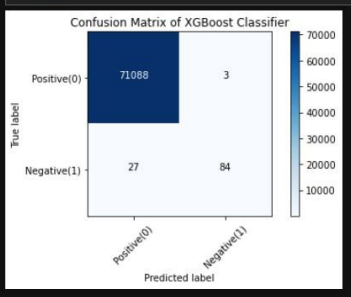



```
[58]: confusion_rf = confusion_matrix(y_test, rf_yhat, labels = [0, 1])

[59]: TP = confusion_rf[1,1]
      TN = confusion_rf[0,0]
      FP = confusion_rf[0,1]
      FN = confusion_rf[1,0]

[60]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))
      Accuracy :  0.9993258616331002
      Precision :  0.8795180722891566
      Sensitivity :  0.6576576576576577
```

- XGBoost Classifier

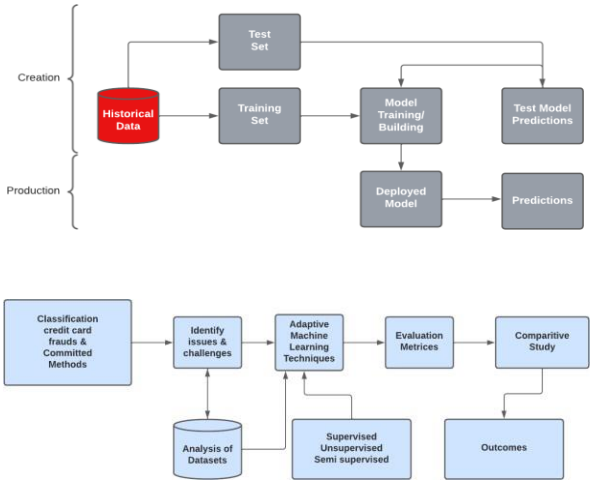
```
[61]: from xgboost import XGBClassifier

[62]: xgb = XGBClassifier(max_depth = 4)
      xgb.fit(X_train, y_train)
      xgb_yhat = xgb.predict(X_test)
      [22:25:40] WARNING: ..\src\learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric
      if you'd like to restore the old behavior.

[63]: print("Accuracy = {}".format(accuracy_score(y_test, xgb_yhat)))
      Accuracy = 0.9995786635206876

[64]: confusion_matrix(y_test, xgb_yhat, labels = [0, 1])

[64]: array([[71088,     3],
             [   27,    84]], dtype=int64)

[65]: def plot_confusion_matrix(cm, classes, title, normalize = False, cmap = plt.cm.Blues):
          title = 'Confusion Matrix of {}'.format(title)
          if normalize:
              cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

          plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
          plt.title(title)
          plt.colorbar()
          tick_marks = np.arange(len(classes))
          plt.xticks(tick_marks, classes, rotation = 45)
          plt.yticks(tick_marks, classes)

          fmt = '.2f' if normalize else 'd'
          thresh = cm.max() / 2.
          for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
              plt.text(j, i, format(cm[i, j], fmt),
                       horizontalalignment = 'center',
                       color = 'white' if cm[i, j] > thresh else 'black')

          plt.tight_layout()
          plt.ylabel('True label')
          plt.xlabel('Predicted label')
```

```
plt.savefig('xgb_cm_plot.png')
plt.show()
```




```
[68]: confusion_xgb = confusion_matrix(y_test, xgb_yhat, labels = [0, 1])
      confusion_xgb

[68]: array([[71088,     3],
             [   27,    84]], dtype=int64)

[69]: TP = confusion_xgb[1,1]
      TN = confusion_xgb[0,0]
      FP = confusion_xgb[0,1]
      FN = confusion_xgb[1,0]

[70]: print("Accuracy : ", (TP+TN)/float(TP+TN+FN+FP))
      print("Precision : ", TP/float(TP+FP))
      print("Sensitivity : ", TP/float(TP+FN))
      Accuracy :  0.9995786635206876
      Precision :  0.9655172413793104
      Sensitivity :  0.7567567567567568
```

BLOCK DIAGRAM & DESCRIPTION



V. RESULTS

We checked for the accuracy scores after fitting out training data into the model and we received an accuracy of 99.95 percent which is better that the accuracy received from other classification algorithms.

Algorithms	Accuracy	Precision	Sensitivity
Decision Tree Classifier	99.935%	84.946%	71.171%
Logistic Regression	99.877%	60.344%	63.063%
Random Forest	99.932%	87.951%	65.765%
XG Boost Classifier	99.957%	96.551%	75.675%

VI. CONCLUSION AND FUTURESCOPE

Without a question, credit card fraud is a form of criminaldeception. This article outlined the most frequent types offraud, as well as how to detect them, and examined recentresearch findings in the field. This paper also includes a detailed explanation of how machine learning can be usedto improve fraud detection findings, as well as the algorithm, pseudocode, explanation, and experimentation results. While the method achieves a precision of over 99.6%, when only a tenth of the data set is taken into account, it only achieves a precision of 28%. When the complete dataset is given into the system, however, the precision increases to 33%. Due to the massive imbalance, such a high percentage of accuracy is to be expected.

VII. REFERENCES

[1] https://en.wikipedia.org/wiki/Logistic_regression

[2] https://stackoverflow.com/questions/22721060/matplotlib-unexpected-gridspec- behavior

[3] https://www.sciencedirect.com/science/article/pii/S187705092030065X

[4] https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive- models/notebook