**Description of design choice:**

**Isolation Forest Algorithm :**

One of the newest techniques to detect anomalies is called Isolation Forests. The algorithm is based on the fact that anomalies are data points that are few and different. As a result of these properties, anomalies are susceptible to a mechanism called isolation.

Im using this method here as this method is highly useful and is fundamentally different. It introduces the use of isolation as a more effective and efficient means to detect anomalies than the commonly used basic distance and density measures. Moreover, this method is an algorithm with a low linear time complexity and a small memory requirement. It builds a good performing model with a small number of trees using small sub-samples of fixed size, regardless of the size of a data set.

This method seems apt for this test case scenario as the typical machine learning methods tend to work better when the patterns they try to learn are balanced, meaning the same amount of good and bad behaviors are present in the dataset.

How Isolation Forests Work

The Isolation Forest algorithm isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The logic argument goes: isolating anomaly observations is easier because only a few conditions are needed to separate those cases from the normal observations. On the other hand, isolating normal observations require more conditions. Therefore, an anomaly score can be calculated as the number of conditions required to separate a given observation.

The way that the algorithm constructs the separation is by first creating isolation trees, or random decision trees. Then, the score is calculated as the path length to isolate the observation.

**Local Outlier Factor(LOF) Algorithm:**

The LOF algorithm is an unsupervised outlier detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outlier samples that have a substantially lower density than their neighbors. the lof seems pretty reasonable algorithm for predicting the outliers.

The number of neighbors considered, (parameter n\_neighbors) is typically chosen 1) greater than the minimum number of objects a cluster has to contain, so that other objects can be local outliers relative to this cluster, and 2) smaller than the maximum number of close by objects that can potentially be local outliers. In practice, such informations are generally not available, and taking n\_neighbors=20 appears to work well in general.

**Performance evaluation of the model:**

Isolation Forest detected 73 errors versus Local Outlier Factor detecting 97 errors

Isolation Forest has a 99.74% more accurate than LOF of 99.65% which is a pretty good result

When comparing error precision & recall for these 2 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 %

So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.

**Discussion of future work**

In the future, we can improve this model accuracy by using a different dataset with more data and outliers.

This data can be fed into an artificial intelligence algorithms or neural networs which with the help of deep learning can help produce better results.

This might be at the expense of computational time and efforts it can be expensive to train on a larger dataset for fraud detection.

**The source code used to create the pipeline**

import numpy as np

import pandas as pd

import sklearn

import scipy

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import classification\_report,accuracy\_score

from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor

from sklearn.svm import OneClassSVM

from pylab import rcParams

rcParams['figure.figsize'] = 14, 8

RANDOM\_SEED = 42

LABELS = ["Normal", "Fraud"]

data = pd.read\_csv('creditcard.csv',sep=',')

data.head()

data.info()

data.isnull().sum()

count\_classes = pd.value\_counts(data['Class'], sort = True)

count\_classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(2), LABELS)

plt.xlabel("Class")

plt.ylabel("Frequency")

fraud = data[data['Class']==1]

normal = data[data['Class']==0]

print(fraud.shape,normal.shape)

fraud.Amount.describe()

normal.Amount.describe()

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Amount per transaction by class')

bins = 50

ax1.hist(fraud.Amount, bins = bins)

ax1.set\_title('Fraud')

ax2.hist(normal.Amount, bins = bins)

ax2.set\_title('Normal')

plt.xlabel('Amount ($)')

plt.ylabel('Number of Transactions')

plt.xlim((0, 20000))

plt.yscale('log')

plt.show();

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Time of transaction vs Amount by class')

ax1.scatter(fraud.Time, fraud.Amount)

ax1.set\_title('Fraud')

ax2.scatter(normal.Time, normal.Amount)

ax2.set\_title('Normal')

plt.xlabel('Time (in Seconds)')

plt.ylabel('Amount')

plt.show()

data1= data.sample(frac = 0.1,random\_state=1)

data1.shape

data.shape

Fraud = data1[data1['Class']==1]

Valid = data1[data1['Class']==0]

outlier\_fraction = len(Fraud)/float(len(Valid))

print(outlier\_fraction)

print("Fraud Cases : {}".format(len(Fraud)))

print("Valid Cases : {}".format(len(Valid)))

import seaborn as sns

#Getting correlations of each features in dataset

corrmat = data1.corr()

top\_corr\_features = corrmat.index

plt.figure(figsize=(20,20))

#plot heat map

g=sns.heatmap(data[top\_corr\_features].corr(),annot=True,cmap="RdYlGn")

#Creating independent and Dependent Features

columns = data1.columns.tolist()

# Filtering the columns to remove data we do not need

columns = [c for c in columns if c not in ["Class"]]

# Storing those variable that we are predicting

target = "Class"

# Defining a random state

state = np.random.RandomState(42)

X = data1[columns]

Y = data1[target]

X\_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))

#Balancing the dataset using SMOTE

from imblearn.over\_sampling import SMOTE

smote = SMOTE(sampling\_strategy = 'minority')

X, Y = smote.fit\_resample(X, Y)

Y.value\_counts()

print(X.shape)

print(Y.shape)

classifiers = {

"Isolation Forest":IsolationForest(n\_estimators=100, max\_samples=len(X),

contamination=outlier\_fraction,random\_state=state, verbose=0),

"Local Outlier Factor":LocalOutlierFactor(n\_neighbors=20, algorithm='auto',

leaf\_size=30, metric='minkowski',

p=2, metric\_params=None, contamination=outlier\_fraction)

}

n\_outliers = len(Fraud)

for i, (clf\_name,clf) in enumerate(classifiers.items()):

#Fiting data and tagging outliers

if clf\_name == "Local Outlier Factor":

y\_pred = clf.fit\_predict(X)

scores\_prediction = clf.negative\_outlier\_factor\_

else:

clf.fit(X)

scores\_prediction = clf.decision\_function(X)

y\_pred = clf.predict(X)

#Reshaping prediction values 0 for Valid txn, 1 for Fraud ones

y\_pred[y\_pred == 1] = 0

y\_pred[y\_pred == -1] = 1

n\_errors = (y\_pred != Y).sum()

# Run Classification Metrics

print("{}: {}".format(clf\_name,n\_errors))

print("Accuracy Score :")

print(accuracy\_score(Y,y\_pred))

print("Classification Report :")

print(classification\_report(Y,y\_pred))