Finance_AI Capstone Project-Copy1

November 6, 2022

1 Project Task: Week 1

2 Exploratory Data Analysis (EDA):

1. Perform an EDA on the Dataset.
    - Check all the latent features and parameters with their mean and standard deviation. Value are close to 0 centered (mean) with unit standard deviation
    - Find if there is any connection between Time, Amount, and the transaction being fraudulent.
2. Check the class count for each class. It's a class Imbalance problem.
3. Use techniques like undersampling or oversampling before running Naïve Bayes, Logistic Regression or SVM.
    - Oversampling or undersampling can be used to tackle the class imbalance problem
    - Oversampling increases the prior probability of imbalanced class and in case of other classifiers, error gets multiplied as the low-proportionate class is mimicked multiple times.
4. Following are the matrices for evaluating the model performance: Precision, Recall, F1-Score, AUC-ROC curve. Use F1-Score as the evaluation criteria for this project.

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
# Import Essential libraries

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import␣
↪accuracy_score,classification_report,confusion_matrix,roc_auc_score,roc_curve,f1_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import Sequential
from keras.layers import Dropout,Dense,Reshape,BatchNormalization
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
```

[3]:
```
# Import the train dataset
train_df = pd.read_csv('train_data.csv')
train_df.head()
```

[3]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0   38355.0  1.043949  0.318555  1.045810  2.805989 -0.561113 -0.367956
1   22555.0 -1.665159  0.808440  1.805627  1.903416 -0.821627  0.934790
2    2431.0 -0.324096  0.601836  0.865329 -2.138000  0.294663 -1.251553
3   86773.0 -0.258270  1.217501 -0.585348 -0.875347  1.222481 -0.311027
4  127202.0  2.142162 -0.494988 -1.936511 -0.818288 -0.025213 -1.027245

         V7        V8        V9  …       V21       V22       V23       V24  \
0  0.032736 -0.042333 -0.322674  … -0.240105 -0.680315  0.085328  0.684812
1 -0.824802  0.975890  1.747469  … -0.335332 -0.510994  0.035839  0.147565
2  1.072114 -0.334896  1.071268  …  0.012220  0.352856 -0.341505 -0.145791
3  1.073860 -0.161408  0.200665  … -0.424626 -0.781158  0.019316  0.178614
4 -0.151627 -0.305750 -0.869482  …  0.010115  0.021722  0.079463 -0.480899

        V25       V26       V27       V28  Amount  Class
0  0.318620 -0.204963  0.001662  0.037894   49.67      0
1 -0.529358 -0.566950 -0.595998 -0.220086   16.94      0
2  0.094194 -0.804026  0.229428 -0.021623    1.00      0
3 -0.315616  0.096665  0.269740 -0.020635   10.78      0
4  0.023846 -0.279076 -0.030121 -0.043888   39.96      0

[5 rows x 31 columns]
```

[4]:
```
# Import the test dataset
test_df = pd.read_csv('test_data.csv')
test_df.head()
```

[4]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  113050.0  0.114697  0.796303 -0.149553 -0.823011  0.878763 -0.553152
1   26667.0 -0.039318  0.495784 -0.810884  0.546693  1.986257  4.386342
2  159519.0  2.275706 -1.531508 -1.021969 -1.602152 -1.220329 -0.462376
3  137545.0  1.940137 -0.357671 -1.210551  0.382523  0.050823 -0.171322
4   63369.0  1.081395 -0.502615  1.075887 -0.543359 -1.472946 -1.065484

         V7        V8        V9  …       V20       V21       V22       V23  \
```

```
0  0.939259 -0.108502  0.111137  … -0.042711 -0.335776 -0.807853 -0.055940
1 -1.344891 -1.743736 -0.563103  …  0.926255 -1.377003 -0.072200 -0.197573
2 -1.196485 -0.147058 -0.950224  … -0.408289 -0.193271 -0.103533  0.150945
3 -0.109124 -0.002115  0.869258  … -0.199280  0.157994  0.650355  0.034206
4 -0.443231 -0.143374  1.659826  …  0.059880  0.224157  0.821209 -0.137223

        V24       V25       V26       V27       V28  Amount
0 -1.025281 -0.369557  0.204653  0.242724  0.085713    0.89
1  1.014807  1.011293 -0.167684  0.113136  0.256836   85.00
2 -0.811083 -0.197913 -0.128446  0.014197 -0.051289   42.70
3  0.739535  0.223605 -0.195509 -0.012791 -0.056841   29.99
4  0.986259  0.563228 -0.574206  0.089673  0.052036   68.00

[5 rows x 30 columns]
```

[5]:
```python
# Import the test hidden dataset
test_hidden_df = pd.read_csv('test_data_hidden.csv')
test_hidden_df.head()
```

[5]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  113050.0  0.114697  0.796303 -0.149553 -0.823011  0.878763 -0.553152
1   26667.0 -0.039318  0.495784 -0.810884  0.546693  1.986257  4.386342
2  159519.0  2.275706 -1.531508 -1.021969 -1.602152 -1.220329 -0.462376
3  137545.0  1.940137 -0.357671 -1.210551  0.382523  0.050823 -0.171322
4   63369.0  1.081395 -0.502615  1.075887 -0.543359 -1.472946 -1.065484

         V7        V8        V9  …       V21       V22       V23       V24  \
0  0.939259 -0.108502  0.111137  … -0.335776 -0.807853 -0.055940 -1.025281
1 -1.344891 -1.743736 -0.563103  … -1.377003 -0.072200 -0.197573  1.014807
2 -1.196485 -0.147058 -0.950224  … -0.193271 -0.103533  0.150945 -0.811083
3 -0.109124 -0.002115  0.869258  …  0.157994  0.650355  0.034206  0.739535
4 -0.443231 -0.143374  1.659826  …  0.224157  0.821209 -0.137223  0.986259

        V25       V26       V27       V28  Amount  Class
0 -0.369557  0.204653  0.242724  0.085713    0.89      0
1  1.011293 -0.167684  0.113136  0.256836   85.00      0
2 -0.197913 -0.128446  0.014197 -0.051289   42.70      0
3  0.223605 -0.195509 -0.012791 -0.056841   29.99      0
4  0.563228 -0.574206  0.089673  0.052036   68.00      0

[5 rows x 31 columns]
```

[6]:
```python
# Find out the shape of the dataset
print('Train Dataset Shape :-',train_df.shape)
print('Test Dataset Shape :-',test_df.shape)
print('Test Hidden Dataset Shape :-',test_hidden_df.shape)
```

```
Train Dataset Shape :- (227845, 31)
Test Dataset Shape :- (56962, 30)
Test Hidden Dataset Shape :- (56962, 31)
```

[7]:
```python
# Combine the train and test hidden dataset
dataset = pd.concat([train_df,test_hidden_df])
print('Shape of the combine dataset :',dataset.shape)
dataset.head()
```

```
Shape of the combine dataset : (284807, 31)
```

[7]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0   38355.0  1.043949  0.318555  1.045810  2.805989 -0.561113 -0.367956
1   22555.0 -1.665159  0.808440  1.805627  1.903416 -0.821627  0.934790
2    2431.0 -0.324096  0.601836  0.865329 -2.138000  0.294663 -1.251553
3   86773.0 -0.258270  1.217501 -0.585348 -0.875347  1.222481 -0.311027
4  127202.0  2.142162 -0.494988 -1.936511 -0.818288 -0.025213 -1.027245

         V7        V8        V9  …       V21       V22       V23       V24  \
0  0.032736 -0.042333 -0.322674  … -0.240105 -0.680315  0.085328  0.684812
1 -0.824802  0.975890  1.747469  … -0.335332 -0.510994  0.035839  0.147565
2  1.072114 -0.334896  1.071268  …  0.012220  0.352856 -0.341505 -0.145791
3  1.073860 -0.161408  0.200665  … -0.424626 -0.781158  0.019316  0.178614
4 -0.151627 -0.305750 -0.869482  …  0.010115  0.021722  0.079463 -0.480899

        V25       V26       V27       V28  Amount  Class
0  0.318620 -0.204963  0.001662  0.037894   49.67      0
1 -0.529358 -0.566950 -0.595998 -0.220086   16.94      0
2  0.094194 -0.804026  0.229428 -0.021623    1.00      0
3 -0.315616  0.096665  0.269740 -0.020635   10.78      0
4  0.023846 -0.279076 -0.030121 -0.043888   39.96      0

[5 rows x 31 columns]
```

[8]:
```python
# Find out types of data
dataset.dtypes
```

[8]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
```

```
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

3 Project Task: Week 1

4 Exploratory Data Analysis (EDA):

5 1. Perform an EDA on the Dataset.

- Check all the latent features and parameters with their mean and standard deviation. Value are close to 0 centered (mean) with unit standard deviation
- Find if there is any connection between Time, Amount, and the transaction being fraudulent.

[9]:
```python
# Find out the dataset is missing or not.
print("Is null value present is the dataset :- ",dataset.isna().sum().any())
print('\n',dataset.isna().sum())
```

```
Is null value present is the dataset :-  False

 Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
```

```
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

[10]:

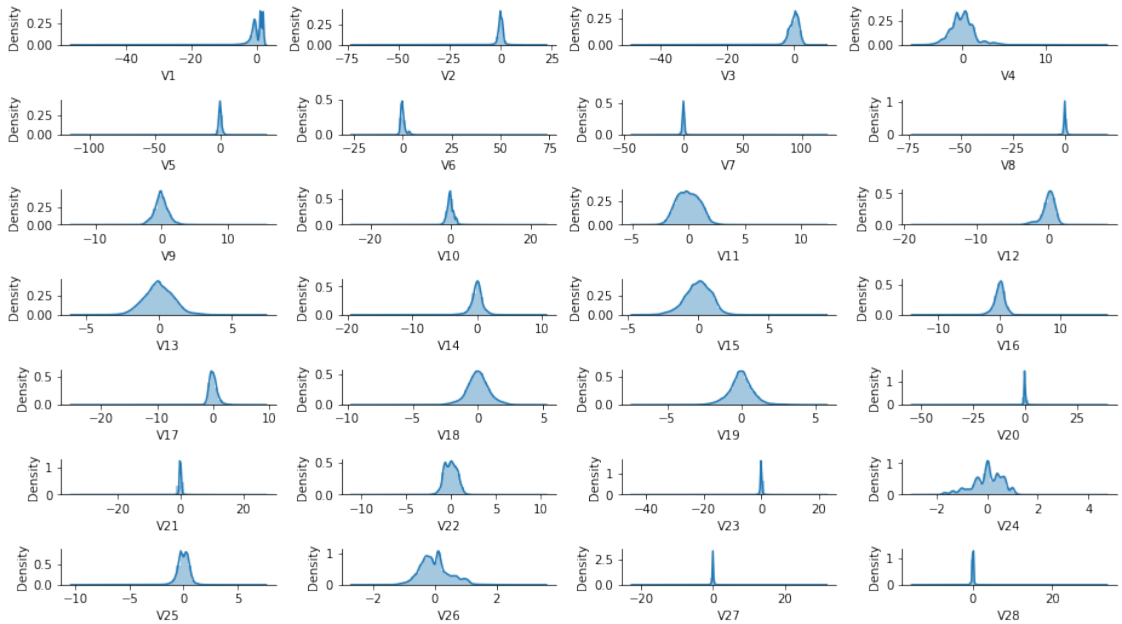
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 284807 entries, 0 to 56961
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
dtypes: float64(30), int64(1)
memory usage: 69.5 MB
```

5.0.1 Check all the latent features and parameters with their mean and standard deviation. Value are close to 0 centered (mean) with unit standard deviation

[11]:
```
feature = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',␣
 ↪'V12', 'V13', 'V14', 'V15', 'V16', 'V17',
            'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26',␣
 ↪'V27', 'V28']

plt.figure(figsize=(13,10))
n = 1
for f in feature:
    plt.subplot(10,4,n)
    sns.distplot(dataset[f],kde=True)
    sns.despine()
    n = n+1
plt.tight_layout()
plt.show()
```



[12]:

```
for f in feature:
    print('Features',f,'Mean is',round(dataset[f].mean(),3),'and Standard␣
 ↪Deviation is',round(dataset[f].std(),3))
```

```
Features V1 Mean is 0.0 and Standard Deviation is 1.959
Features V2 Mean is 0.0 and Standard Deviation is 1.651
Features V3 Mean is -0.0 and Standard Deviation is 1.516
Features V4 Mean is 0.0 and Standard Deviation is 1.416
Features V5 Mean is 0.0 and Standard Deviation is 1.38
Features V6 Mean is 0.0 and Standard Deviation is 1.332
Features V7 Mean is -0.0 and Standard Deviation is 1.237
Features V8 Mean is 0.0 and Standard Deviation is 1.194
Features V9 Mean is -0.0 and Standard Deviation is 1.099
Features V10 Mean is 0.0 and Standard Deviation is 1.089
Features V11 Mean is 0.0 and Standard Deviation is 1.021
Features V12 Mean is -0.0 and Standard Deviation is 0.999
Features V13 Mean is 0.0 and Standard Deviation is 0.995
Features V14 Mean is 0.0 and Standard Deviation is 0.959
Features V15 Mean is 0.0 and Standard Deviation is 0.915
Features V16 Mean is 0.0 and Standard Deviation is 0.876
Features V17 Mean is -0.0 and Standard Deviation is 0.849
Features V18 Mean is 0.0 and Standard Deviation is 0.838
Features V19 Mean is 0.0 and Standard Deviation is 0.814
Features V20 Mean is 0.0 and Standard Deviation is 0.771
Features V21 Mean is 0.0 and Standard Deviation is 0.735
Features V22 Mean is -0.0 and Standard Deviation is 0.726
Features V23 Mean is 0.0 and Standard Deviation is 0.624
```

```
Features V24 Mean is 0.0 and Standard Deviation is 0.606
Features V25 Mean is 0.0 and Standard Deviation is 0.521
Features V26 Mean is 0.0 and Standard Deviation is 0.482
Features V27 Mean is -0.0 and Standard Deviation is 0.404
Features V28 Mean is -0.0 and Standard Deviation is 0.33
```

5.0.2 Find if there is any connection between Time, Amount, and the transaction being fraudulent.

[13]:
```
dataset[['Time','Amount','Class']].corr()
```

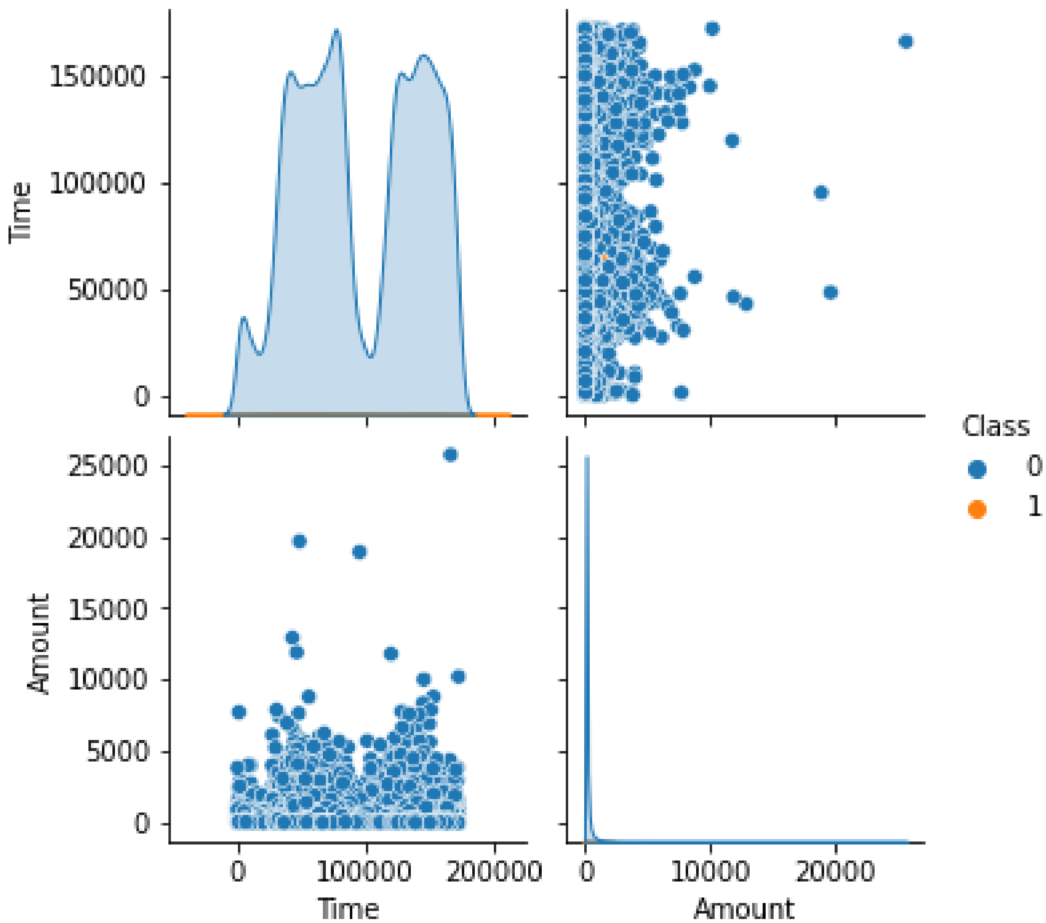
[13]:

	Time	Amount	Class
Time	1.000000	-0.010596	-0.012323
Amount	-0.010596	1.000000	0.005632
Class	-0.012323	0.005632	1.000000

- From the above result we can confirm that there is no any connection between Time, Amount and the transaction being fraudulent.

[14]:
```
sns.pairplot(dataset.
 ↪reset_index(drop=True),x_vars=['Time','Amount'],y_vars=['Time','Amount'],kind='scatter',hue
```

[14]:
```
<seaborn.axisgrid.PairGrid at 0x1348c44fd90>
```



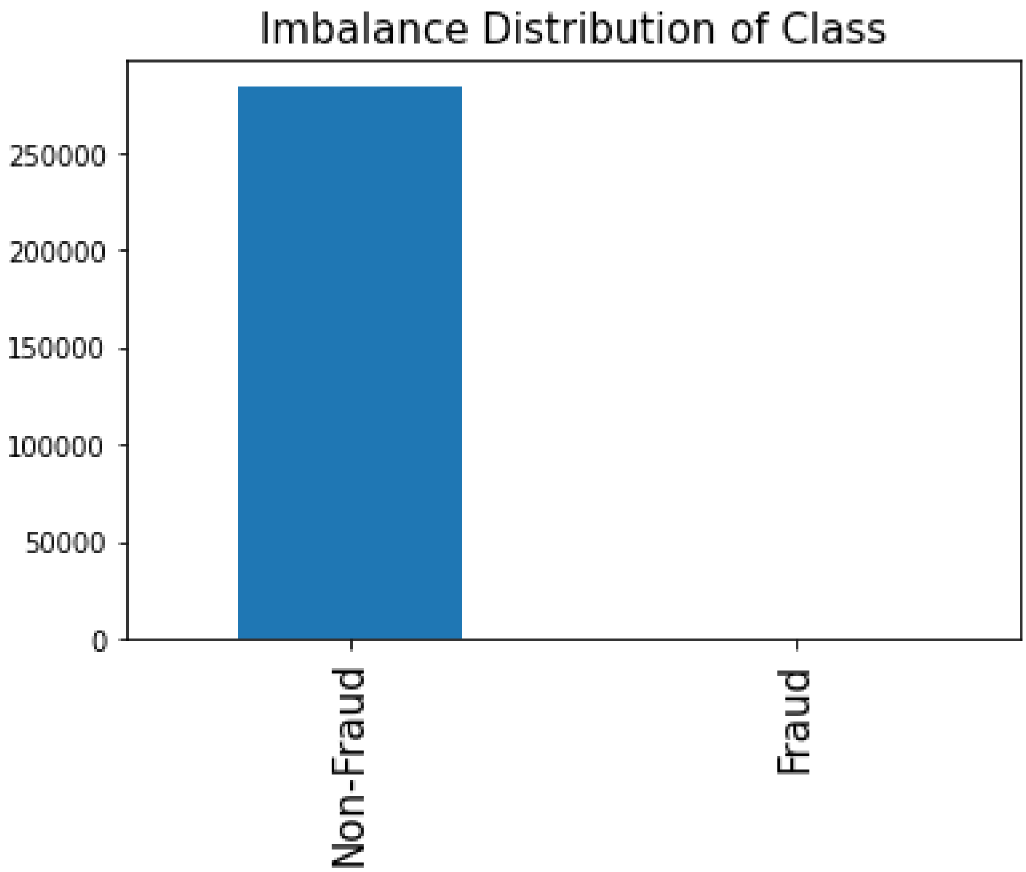
6 2. Check the class count for each class. It's a class Imbalance problem.

[15]:
```
dataset['Class'].value_counts()
```

[15]:
```
0    284315
1       492
Name: Class, dtype: int64
```

- The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset represents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

[16]:
```
df = dataset.copy()
df.Class.replace((0,1),('Non-Fraud','Fraud'),inplace=True)
df.Class.value_counts().plot(kind='bar')
plt.title('Imbalance Distribution of Class',fontsize=15)
plt.tick_params(axis='x', which='major', labelsize=15)
plt.show()
```



7 3. Use techniques like undersampling or oversampling before running Naïve Bayes, Logistic Regression or SVM.

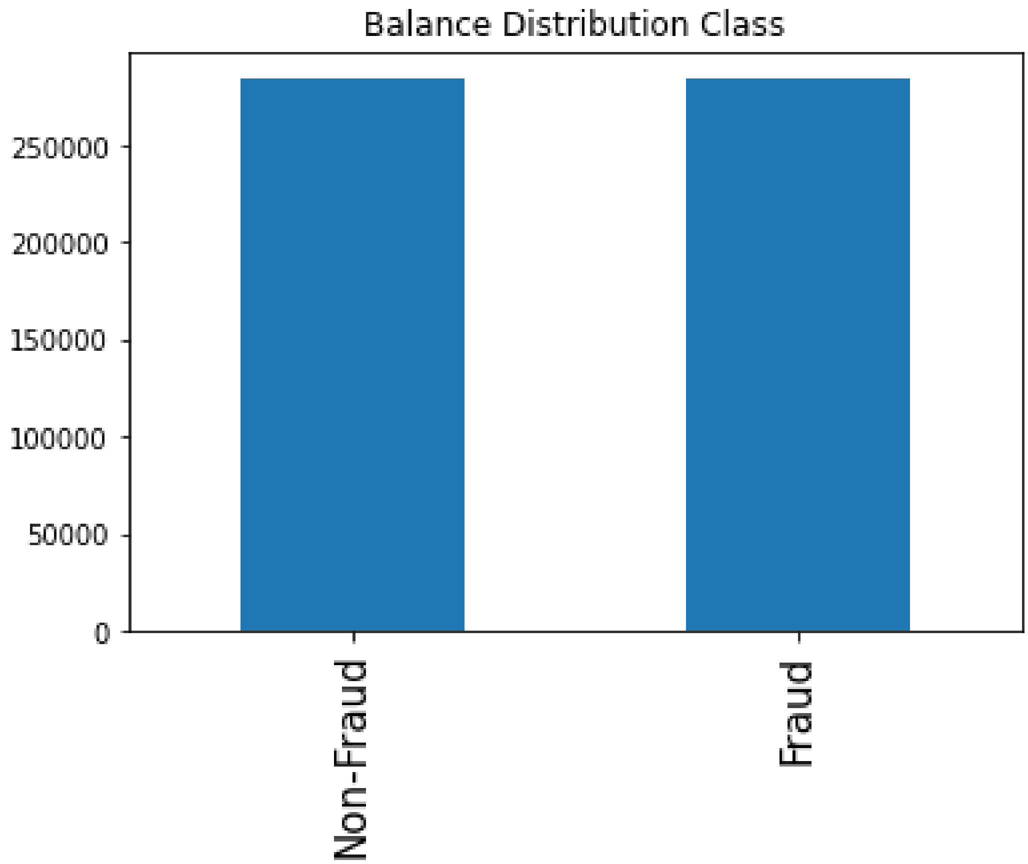
- Oversampling or undersampling can be used to tackle the class imbalance problem
- Oversampling increases the prior probability of imbalanced class and in case of other classifiers, error gets multiplied as the low-proportionate class is mimicked multiple times.

7.0.1 Oversampling

[17]:
```
count_0,count_1 = dataset.Class.value_counts()

class_1 = dataset[dataset['Class'] == 1]
class_0 = dataset[dataset['Class'] == 0]

class_1_over = class_1.sample(count_0,replace=True)

dataset_over = pd.concat([class_0,class_1_over],axis=0)
print('Before Oversampling')
print(dataset.Class.value_counts())
print('\n')
print('Random Under Sampling:')
print(dataset_over.Class.value_counts())
```

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


Random Under Sampling:
0    284315
1    284315
Name: Class, dtype: int64
```

[18]:
```
# Convert the Class

df_over = dataset_over.copy()
df_over.Class.replace((0,1),('Non-Fraud','Fraud'),inplace=True)
df_over.Class.value_counts().plot(kind='bar')
plt.title('Balance Distribution Class')
plt.tick_params(axis='x', which='major', labelsize=15)
```



7.0.2 Undersampling

[19]:
```
count_0,count_1 = dataset.Class.value_counts()

class_1 = dataset[dataset['Class'] == 1]
class_0 = dataset[dataset['Class'] == 0]

class_0_under = class_0.sample(count_1,replace=True)

dataset_under = pd.concat([class_1,class_0_under],axis=0)
print('Before Oversampling')
print(dataset.Class.value_counts())
print('\n')
print('Random Under Sampling:')
print(dataset_under.Class.value_counts())
```

```
Before Oversampling
0    284315
1       492
```

```
Name: Class, dtype: int64


Random Under Sampling:
1    492
0    492
Name: Class, dtype: int64
```

[20]:
```
# Convert the Class

df_under = dataset_under.copy()
df_under.Class.replace((0,1),('Non-Fraud','Fraud'),inplace=True)
df_under.Class.value_counts().plot(kind='bar')
plt.title('Balance Distribution Class')
plt.tick_params(axis='x', which='major', labelsize=15)
```

[21]:
```
# Over sample data scaling

data = dataset_over.drop(columns=['Class']).values
scaler = StandardScaler()
```

```
dataset_over_scale = scaler.fit_transform(data)
```

[22]:
```
xtrain_over,xtest_over,ytrain_over,ytest_over =␣
↪train_test_split(dataset_over_scale,dataset_over['Class'],test_size=0.2,
                                                    random_state=41)
```

[23]:
```
print(xtrain_over.shape)
print(xtest_over.shape)
print(ytrain_over.shape)
print(ytest_over.shape)
```

```
(454904, 30)
(113726, 30)
(454904,)
(113726,)
```

8 Modeling Techniques:

5. Try out models like Naive Bayes, Logistic Regression or SVM. Find out which one performs the best
6. Use different Tree-based classifiers like Random Forest and XGBoost.

- Remember Tree-based classifiers work on two ideologies: Bagging or Boosting
- Tree-based classifiers have fine-tuning parameters which takes care of the imbalanced class. Random-Forest and XGBboost.

7. Compare the results of 1 with 2 and check if there is any incremental gain.

9 5. Try out models like Naive Bayes, Logistic Regression or SVM. Find out which one performs the best

9.1 Modeling with Oversampled dataset

9.1.1 Bernoulli Naive Bayes with oversampled dataset

[24]:
```
bnb_model_over = BernoulliNB()
bnb_model_over.fit(xtrain_over,ytrain_over)
```

[24]: BernoulliNB()

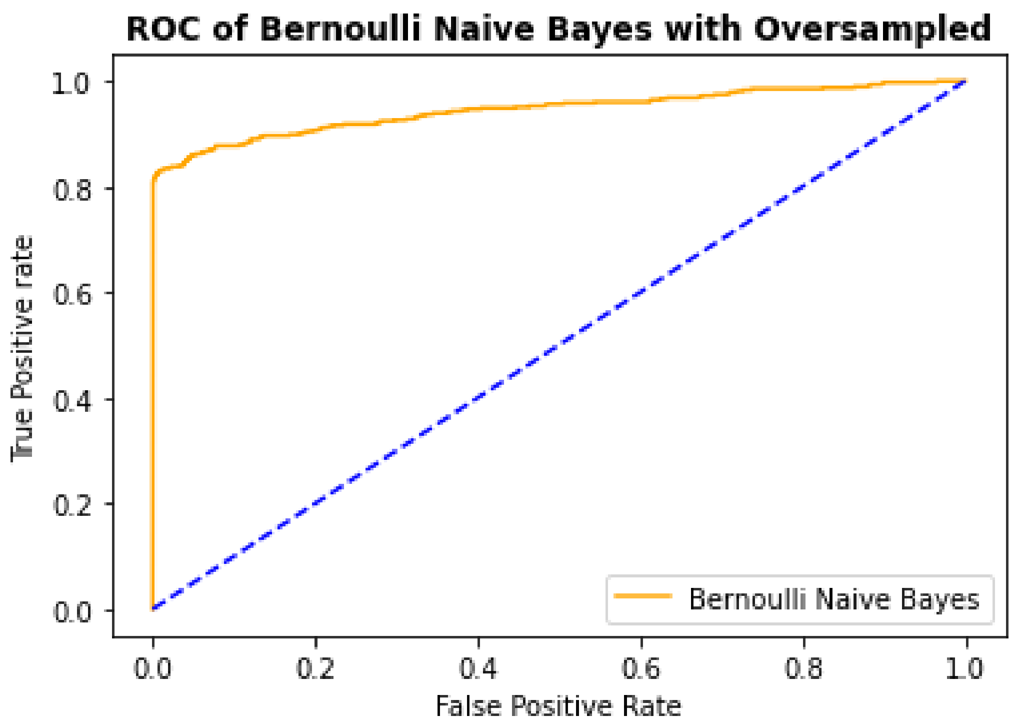
[25]:
```
ypred = bnb_model_over.predict(xtest_over)
bnb_acc_over = accuracy_score(ytest_over, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(bnb_acc_over))
print("\nClassification report : \n", classification_report(ytest_over, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest_over, ypred))
print("\nTrain Data Score : ",bnb_model_over.score(xtrain_over,ytrain_over))
print("\nTest Data Score : ",bnb_model_over.score(xtest_over,ytest_over))
```

```
Accuracy on validation set: 91.1436

Classification report :
               precision    recall  f1-score   support

           0       0.85      0.99      0.92     56864
           1       0.99      0.83      0.90     56862

    accuracy                           0.91    113726
   macro avg       0.92      0.91      0.91    113726
weighted avg       0.92      0.91      0.91    113726


Confusion Matrix :
 [[56450   414]
 [ 9658 47204]]

Train Data Score :  0.9115637585072894

Test Data Score :  0.9114362590788386
```

9.1.2 Receiver operating characteristic of Bernoulli Naive Bayes with Oversampled dataset

[26]:
```
pred_prob = bnb_model_over.predict_proba(xtest_over)
fpr1,tpr1,threshold1=roc_curve(ytest_over,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest_over))]
P_fpr,p_tpr,_=roc_curve(ytest_over,random_prob)

# auc scores
auc_score1 = roc_auc_score(ytest_over, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='Bernoulli Naive Bayes')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Bernoulli Naive Bayes with Oversampled',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.9453125380404437
```



9.1.3 Logistic Regression with oversampled dataset

[27]:
```
lr_model_over = LogisticRegression()
lr_model_over.fit(xtrain_over,ytrain_over)
```

[27]:
```
LogisticRegression()
```

[28]:
```
ypred = lr_model_over.predict(xtest_over)
lr_acc_over = accuracy_score(ytest_over, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(lr_acc_over))
print("\nClassification report : \n", classification_report(ytest_over, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest_over, ypred))
print("\nTrain Data Score : ",lr_model_over.score(xtrain_over,ytrain_over))
print("\nTest Data Score : ",lr_model_over.score(xtest_over,ytest_over))
```

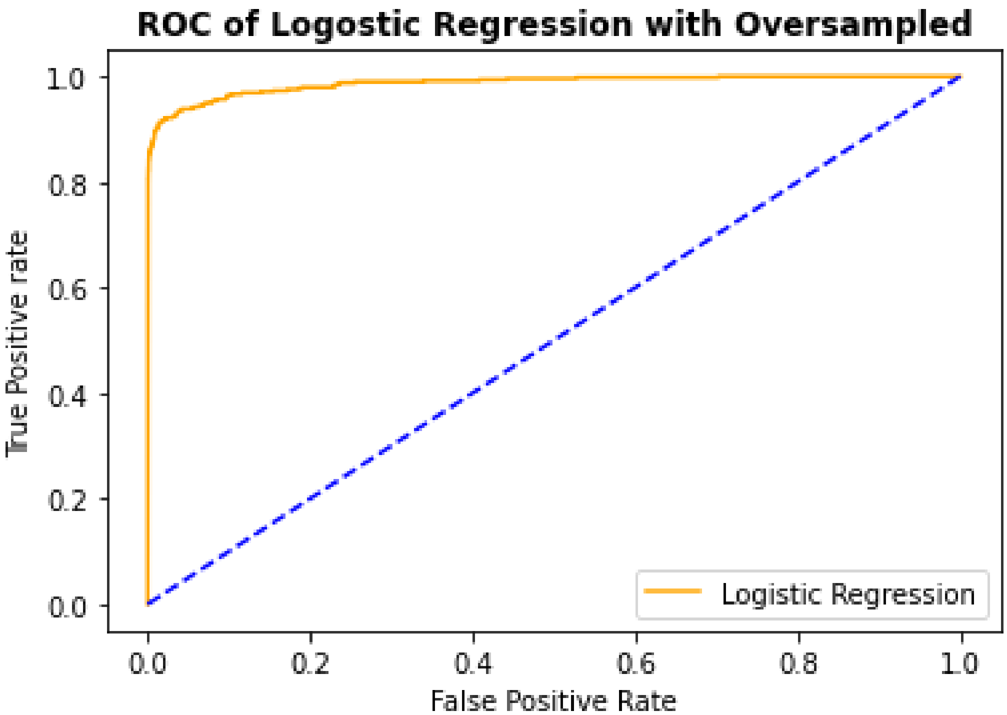
```
Accuracy on validation set: 94.9431

Classification report :
               precision    recall  f1-score   support

           0       0.93      0.98      0.95     56864
           1       0.98      0.92      0.95     56862
```

```
    accuracy                           0.95    113726
   macro avg       0.95      0.95      0.95    113726
weighted avg       0.95      0.95      0.95    113726


Confusion Matrix :
 [[55549  1315]
 [ 4436 52426]]

Train Data Score :  0.950402722332624

Test Data Score :  0.9494310887571883
```

9.1.4 Receiver operating characteristic for Logistic Regression with Oversampled dataset

[29]:
```
pred_prob = lr_model_over.predict_proba(xtest_over)
fpr1,tpr1,threshold1=roc_curve(ytest_over,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest_over))]
P_fpr,p_tpr,_=roc_curve(ytest_over,random_prob)

# auc scores
auc_score1 = roc_auc_score(ytest_over, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='Logistic Regression')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Logostic Regression with Oversampled',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.9871329793350258
```



9.1.5 Support Vector Machine with Oversampled dataset

[30]:
```
# %%time
# svm_over = SVC(class_weight='balanced',probability=True)
# svm_over.fit(xtrain_over,ytrain_over)
```

[31]:
```
# linear_pred = svm_over.predict(xtest_over)
```

[32]:
```
# svm_acc_over = accuracy_score(ytest_over, linear_pred)*100
# print ("\nAccuracy on validation set: {:.4f}".format(svm_acc_over))
# print("\nClassification report : \n", classification_report(ytest_over,␣
↪linear_pred))
# print("\nConfusion Matrix : \n", confusion_matrix(ytest_over, linear_pred))
# print("\nTrain Data Score : ",svm_over.score(xtrain_over,ytrain_over))
# print("\nTest Data Score : ",svm_over.score(xtest_over,ytest_over))
```

9.1.6 Receiver operating characteristic of Support Vector Machine with Oversampled dataset

[33]:
```
# pred_prob = svm_over.predict_proba(xtest_over)
# fpr1,tpr1,threshold1=roc_curve(ytest_over,pred_prob[:,1])

# # ROC curve for tpr=fpr
```

```
# random_prob=[0 for i in range (len(ytest_over))]
# P_fpr,p_tpr,_=roc_curve(ytest_over,random_prob)

# # auc scores
# auc_score1 = roc_auc_score(ytest_over, pred_prob[:,1])
# print(auc_score1)

# plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='SVM')
# plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
# plt.title('ROC of Support Vector Machine with Oversampled',weight='bold')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive rate')
# plt.legend(loc='best')
# plt.show()
```

9.2 Modeling with Undersampled

[34]:
```
# Under sample data scaling

data = dataset_under.drop(columns=['Class']).values
scaler = StandardScaler()
dataset_under_scale = scaler.fit_transform(data)
```

[35]:
```
xtrain_under,xtest_under,ytrain_under,ytest_under =␣
 ↪train_test_split(dataset_under_scale,dataset_under['Class'],
                                                                          ␣
 ↪test_size=0.2, random_state=41)
```

[36]:
```
print(xtrain_under.shape)
print(xtest_under.shape)
print(ytrain_under.shape)
print(ytest_under.shape)
```

```
(787, 30)
(197, 30)
(787,)
(197,)
```

9.2.1 Bernoulli Naive Bayes with Undersampled dataset

[37]:
```
bnb_model_under = BernoulliNB()
bnb_model_under.fit(xtrain_under,ytrain_under)
```

[37]: BernoulliNB()

[38]:
```
ypred = bnb_model_under.predict(xtest_under)
bnb_acc_under = accuracy_score(ytest_under, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(bnb_acc_under))
print("\nClassification report : \n", classification_report(ytest_under, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest_under, ypred))
print("\nTrain Data Score : ",bnb_model_under.score(xtrain_under,ytrain_under))
print("\nTest Data Score : ",bnb_model_under.score(xtest_under,ytest_under))
```

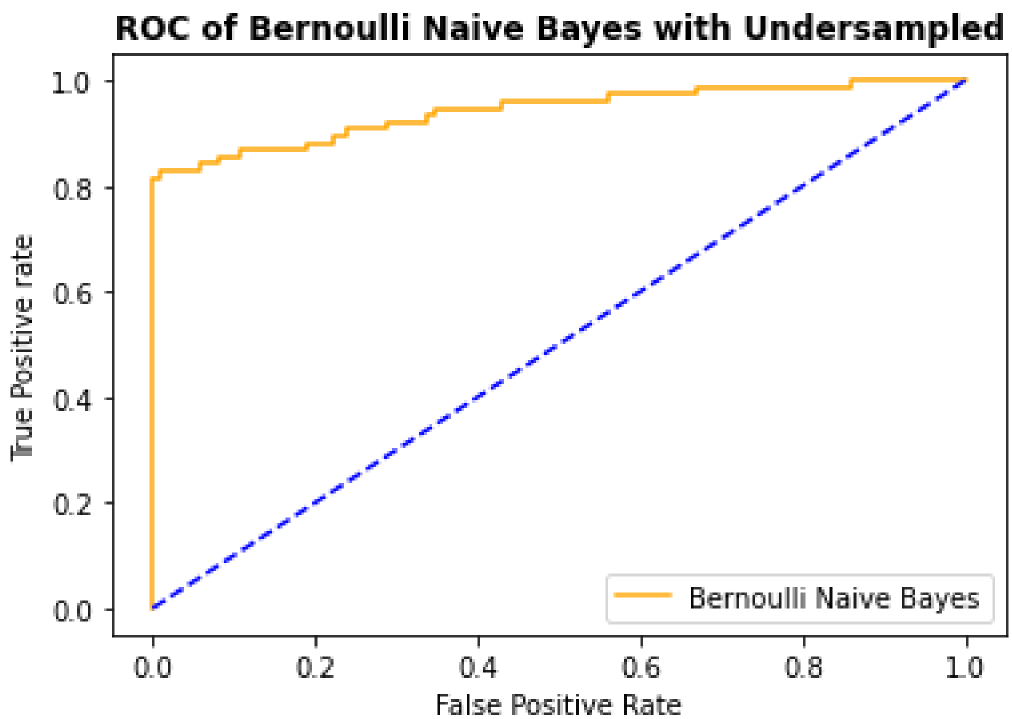
```
Accuracy on validation set: 92.8934

Classification report :
               precision    recall  f1-score   support

           0       0.90      1.00      0.95       121
           1       1.00      0.82      0.90        76

    accuracy                           0.93       197
   macro avg       0.95      0.91      0.92       197
weighted avg       0.94      0.93      0.93       197


Confusion Matrix :
 [[121   0]
 [ 14  62]]

Train Data Score :  0.9034307496823379

Test Data Score :  0.9289340101522843
```

9.2.2 Receiver operating characteristic of Bernoulli Naive Bayes with undersampled dataset¶

[39]:
```
pred_prob = bnb_model_under.predict_proba(xtest_under)
fpr1,tpr1,threshold1=roc_curve(ytest_under,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest_under))]
P_fpr,p_tpr,_=roc_curve(ytest_under,random_prob)

# auc scores
auc_score1 = roc_auc_score(ytest_under, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='Bernoulli Naive Bayes')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Bernoulli Naive Bayes with Undersampled',weight='bold')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

0.942040017398869



9.2.3 Logistic Regression with Undersampled dataset

[40]:
```
lr_model_under = LogisticRegression()
lr_model_under.fit(xtrain_under,ytrain_under)
```

[40]: LogisticRegression()

[41]:
```
ypred = lr_model_under.predict(xtest_under)
lr_acc_under = accuracy_score(ytest_under, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(lr_acc_under))
print("\nClassification report : \n", classification_report(ytest_under, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest_under, ypred))
print("\nTrain Data Score : ",lr_model_under.score(xtrain_under,ytrain_under))
print("\nTest Data Score : ",lr_model_under.score(xtest_under,ytest_under))
```

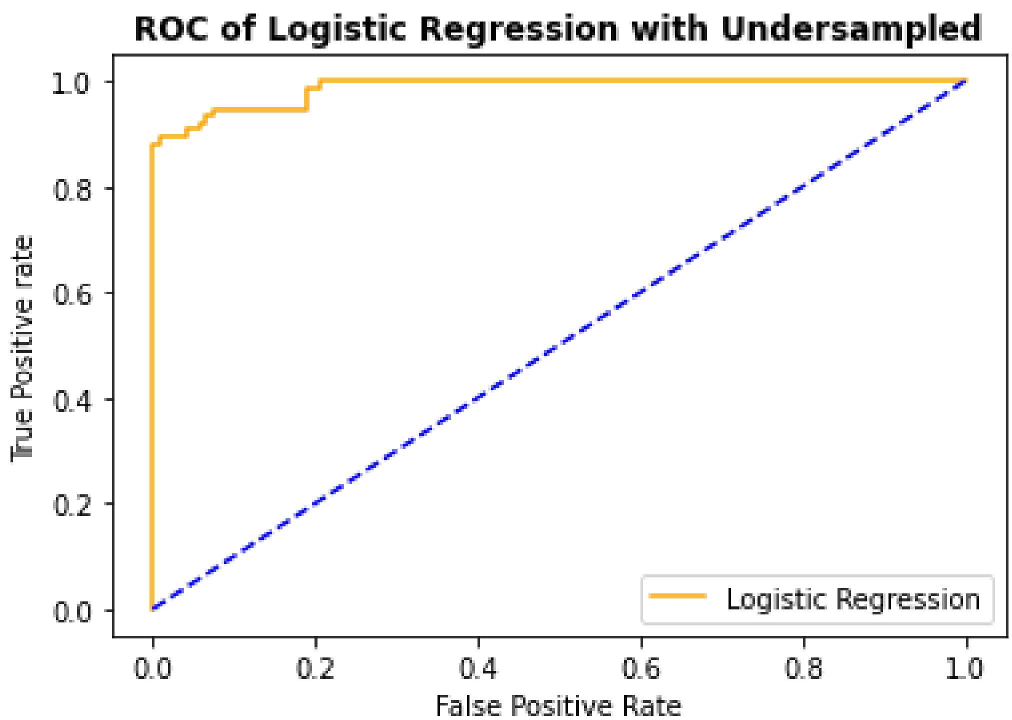
Accuracy on validation set: 94.9239

```
Classification report :
               precision    recall  f1-score   support

           0       0.94      0.98      0.96       121
           1       0.97      0.89      0.93        76

    accuracy                           0.95       197
   macro avg       0.95      0.94      0.95       197
weighted avg       0.95      0.95      0.95       197


Confusion Matrix :
 [[119   2]
 [  8  68]]

Train Data Score :  0.9491740787801779

Test Data Score :  0.949238578680203
```

9.2.4 Receiver operating characteristic of Logistic Regression with undersampled dataset¶

[42]:
```
pred_prob = lr_model_under.predict_proba(xtest_under)
fpr1,tpr1,threshold1=roc_curve(ytest_under,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest_under))]
P_fpr,p_tpr,_=roc_curve(ytest_under,random_prob)

# auc scores
auc_score1 = roc_auc_score(ytest_under, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='Logistic Regression')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Logistic Regression with Undersampled',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.9865158764680296
```



9.2.5 Support Vector Machine with Undersampled dataset

[43]:
```
%%time
svm_under = SVC(class_weight='balanced',probability=True)
svm_under.fit(xtrain_under,ytrain_under)
```

```
Wall time: 1.24 s
```

[43]:
```
SVC(class_weight='balanced', probability=True)
```

[44]:
```
svm_pred_under = svm_under.predict(xtest_under)
svm_acc_under = accuracy_score(ytest_under, svm_pred_under)*100
print ("\nAccuracy on validation set: {:.4f}".format(svm_acc_under))
print("\nClassification report : \n", classification_report(ytest_under,␣
 ↪svm_pred_under))
print("\nConfusion Matrix : \n", confusion_matrix(ytest_under, svm_pred_under))
print("\nTrain Data Score : ",svm_under.score(xtest_under,ytest_under))
print("\nTest Data Score : ",svm_under.score(xtest_under,ytest_under))
```

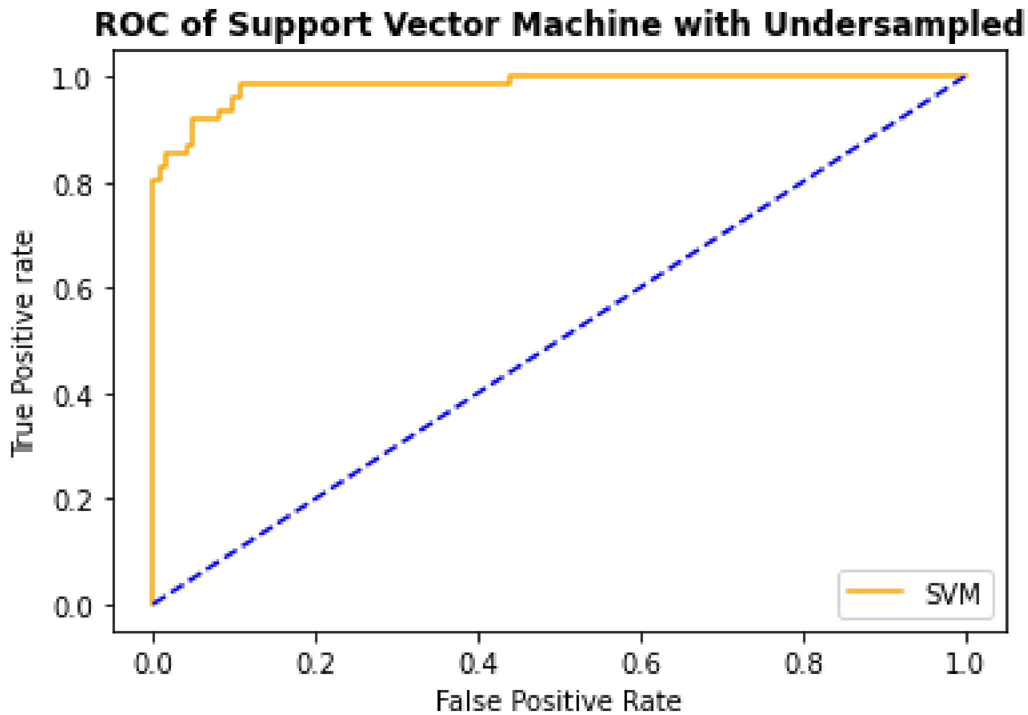
```
Accuracy on validation set: 92.8934

Classification report :
               precision    recall  f1-score   support
```

```
           0       0.91      0.98      0.94       121
           1       0.96      0.86      0.90        76

    accuracy                           0.93       197
   macro avg       0.94      0.92      0.92       197
weighted avg       0.93      0.93      0.93       197


Confusion Matrix :
 [[118   3]
 [ 11  65]]

Train Data Score :  0.9289340101522843

Test Data Score :  0.9289340101522843
```

[45]:

```
pred_prob = svm_under.predict_proba(xtest_under)
fpr1,tpr1,threshold1=roc_curve(ytest_under,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest_under))]
P_fpr,p_tpr,_=roc_curve(ytest_under,random_prob)

# auc scores
auc_score1 = roc_auc_score(ytest_under, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='SVM')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Support Vector Machine with Undersampled',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.983906046107003
```



10 6. Use different Tree-based classifiers like Random Forest and XGBoost.

- Remember Tree-based classifiers work on two ideologies: Bagging or Boosting
- Tree-based classifiers have fine-tuning parameters which takes care of the imbalanced class. Random-Forest and XGBboost.

[46]:
```
data = train_df.drop(columns=['Class']).values
scaler = StandardScaler()
train_data = scaler.fit_transform(data)
```

[47]:
```
xtrain,xtest,ytrain,ytest =␣
↪train_test_split(train_data,train_df['Class'],test_size=0.2,random_state=41)
```

[48]:
```
print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(182276, 30)
(182276,)
(45569, 30)
(45569,)
```

10.0.1 Random Forest Classifier

[49]:
```
%%time
rfc_model = RandomForestClassifier(n_estimators=400,random_state=11,class_weight='balanced')
rfc_model.fit(xtrain,ytrain)
```

```
Wall time: 18min 14s
```

[49]:
```
RandomForestClassifier(class_weight='balanced', n_estimators=400,
                       random_state=11)
```

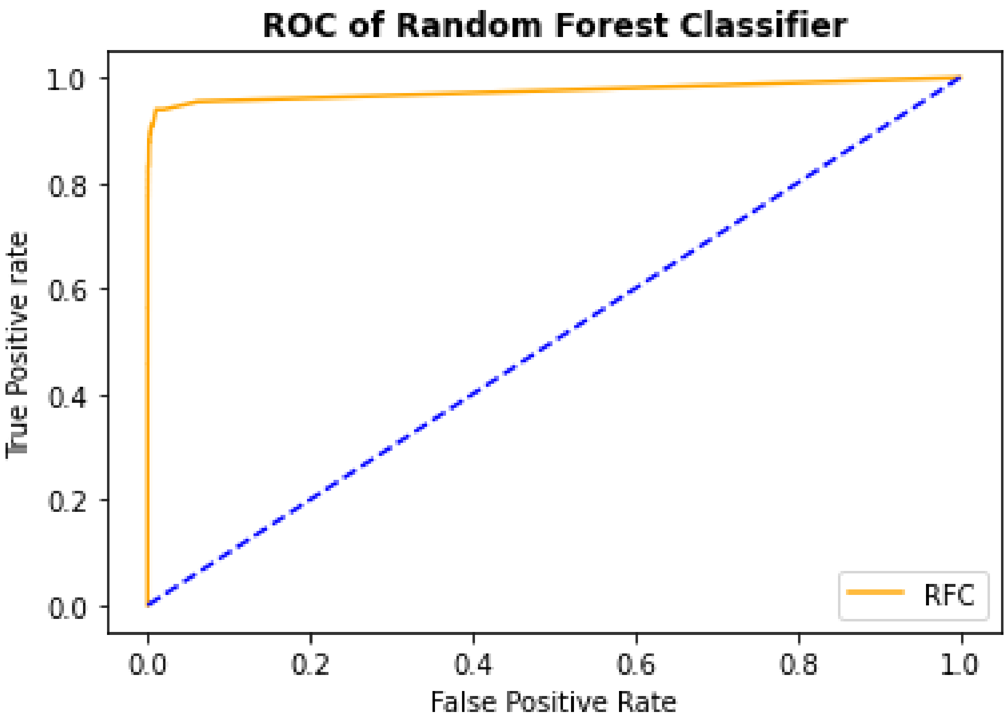
[50]:
```
ypred = rfc_model.predict(xtest)
rfc_acc = accuracy_score(ytest, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(rfc_acc))
print("\nClassification report : \n", classification_report(ytest, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest, ypred))
print("\nTrain Data Score : ",rfc_model.score(xtrain,ytrain))
print("\nTest Data Score : ",rfc_model.score(xtest,ytest))
```

```
Accuracy on validation set: 99.9561

Classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     45503
           1       0.94      0.74      0.83        66

    accuracy                           1.00     45569
   macro avg       0.97      0.87      0.92     45569
weighted avg       1.00      1.00      1.00     45569


Confusion Matrix :
 [[45500     3]
 [   17    49]]

Train Data Score :  1.0

Test Data Score :  0.9995611051372644
```

[51]:
```
pred_prob = rfc_model.predict_proba(xtest)
fpr1,tpr1,threshold1=roc_curve(ytest,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest))]
P_fpr,p_tpr,_=roc_curve(ytest,random_prob)
```

```
# auc scores
auc_score1 = roc_auc_score(ytest, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='RFC')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of Random Forest Classifier',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.9748245037456738
```



10.0.2 XGBClassifier

[52]:
```
%%time
xgb_model = XGBClassifier(n_estimators=1000,max_depth=6,scale_pos_weight=99)
xgb_model.fit(xtrain,ytrain)
```

```
Wall time: 8min 59s
Parser   : 112 ms
```

[52]:
```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=1000,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, …)
```

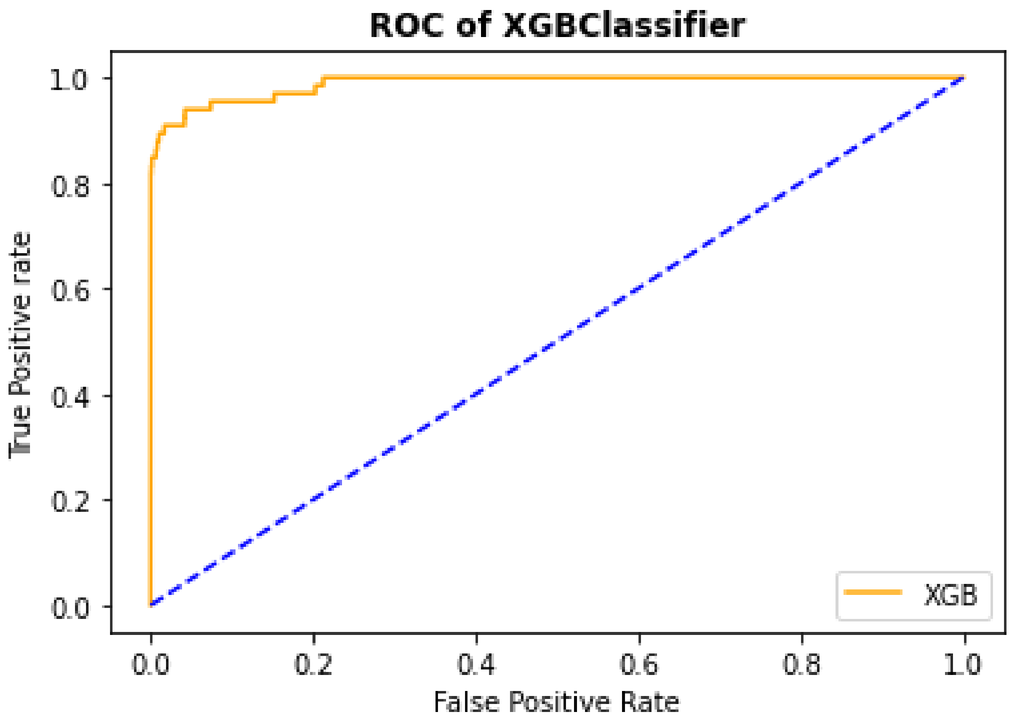
[53]:
```
ypred = xgb_model.predict(xtest)
xgb_acc = accuracy_score(ytest, ypred)*100
print ("\nAccuracy on validation set: {:.4f}".format(xgb_acc))
print("\nClassification report : \n", classification_report(ytest, ypred))
print("\nConfusion Matrix : \n", confusion_matrix(ytest, ypred))
print("\nTrain Data Score : ",xgb_model.score(xtrain,ytrain))
print("\nTest Data Score : ",xgb_model.score(xtest,ytest))
```

```
Accuracy on validation set: 99.9627

Classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     45503
           1       0.93      0.80      0.86        66

    accuracy                           1.00     45569
   macro avg       0.96      0.90      0.93     45569
weighted avg       1.00      1.00      1.00     45569


Confusion Matrix :
 [[45499     4]
 [   13    53]]

Train Data Score :  1.0

Test Data Score :  0.9996269393666747
```

[54]:
```
pred_prob = xgb_model.predict_proba(xtest)
fpr1,tpr1,threshold1=roc_curve(ytest,pred_prob[:,1])

# ROC curve for tpr=fpr
random_prob=[0 for i in range (len(ytest))]
P_fpr,p_tpr,_=roc_curve(ytest,random_prob)
```

```
# auc scores
auc_score1 = roc_auc_score(ytest, pred_prob[:,1])
print(auc_score1)

plt.plot(fpr1,tpr1,linestyle='-',color='orange',label='XGB')
plt.plot(P_fpr,p_tpr,linestyle='--',color='blue')
plt.title('ROC of XGBClassifier',weight='bold')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')
plt.legend(loc='best')
plt.show()
```

```
0.9883304397512253
```



11 7. Compare the results of 1 with 2 and check if there is any incremental gain.

[55]:
```
models = pd.DataFrame({
    'Models' : ['Bernaulli Naive Bayes Over-Sampled','Logistic Regression␣
↪Over-Sampled',
```

```
                'Bernaulli Naive Bayes Under-Sampled','Logistic Regression␣
↪Under-Sampled','SVM Under-Sampled',
                'Random Fosrest Classifier','XGB',],
   'Score' :␣
↪[bnb_acc_over,lr_acc_over,bnb_acc_under,lr_acc_under,svm_acc_under,rfc_acc,xgb_acc]})

round(models.sort_values(by='Score',ascending=False),2)
```

[55]:

	Models	Score
6	XGB	99.96
5	Random Fosrest Classifier	99.96
1	Logistic Regression Over-Sampled	94.94
3	Logistic Regression Under-Sampled	94.92
2	Bernaulli Naive Bayes Under-Sampled	92.89
4	SVM Under-Sampled	92.89
0	Bernaulli Naive Bayes Over-Sampled	91.14

- From above test its shows that the tree base (XGB classifier) models are best performed.

12 Project Task: Week 2

13 Applying ANN:

1. Use ANN (Artificial Neural Network) to predict Store Sales.
    - Fine-tune number of layers
    - Number of Neurons in each layers
    - Experiment in batch-size
    - Experiment with number of epochs. Check the observations in loss and accuracy
    - Play with different Learning Rate variants of Gradient Descent like Adam, SGD, RMS-prop
    - Find out which activation performs best for this use case and why?
    - Calculate RMSE
    - Check Confusion Matrix, Precision, Recall and F1-Score
2. Try out Dropout for ANN. How is it performed? Compare model performance with the traditional ML based prediction models from above.
3. Find the best setting of neural net that can be best classified as fraudulent and non-fraudulent transactions. Use techniques like Grid Search, Cross-Validation and Random search.

[56]:
```
model = Sequential()
model.add(Reshape((30,),input_shape=(30,)))
model.add(BatchNormalization())
model.add(Dense(100,activation='relu'))
```

```
model.add(Dropout(0.5))
model.add(BatchNormalization())
model.add(Dense(50,activation='relu'))
model.add(Dropout(0.3))
model.add(BatchNormalization())
model.add(Dense(1,activation='sigmoid'))
```

[57]:
```
# Adam optimizers

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(xtrain,ytrain,epochs=10,batch_size=512,validation_data=(xtest,ytest))

# Model Evaluation
adam_score = model.evaluate(xtest,ytest, batch_size=512)
print('\nTest loss : {:.4f}'.format(adam_score[0]*100))
print('\nTest accuracy : {:.4f}'.format(adam_score[1]*100))

y_classes_test = np.argmax(model.predict(xtest, verbose=0),axis=1)
# reduce the output to 1d array
yhat_classes_test = y_classes_test.reshape(-1,1)

print('\nConfusion Maxtrix :')
print(confusion_matrix(y_true=ytest,y_pred=yhat_classes_test))
print('\nClassification Report : ')
print(classification_report(y_true=ytest,y_pred=yhat_classes_test))
```

```
Epoch 1/10
357/357 [==============================] - 29s 17ms/step - loss: 0.2623 -
accuracy: 0.9104 - val_loss: 0.0246 - val_accuracy: 0.9993
Epoch 2/10
357/357 [==============================] - 5s 13ms/step - loss: 0.0185 -
accuracy: 0.9983 - val_loss: 0.0100 - val_accuracy: 0.9993
Epoch 3/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0092 -
accuracy: 0.9989 - val_loss: 0.0074 - val_accuracy: 0.9994
Epoch 4/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0063 -
accuracy: 0.9991 - val_loss: 0.0059 - val_accuracy: 0.9993
Epoch 5/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0057 -
accuracy: 0.9992 - val_loss: 0.0050 - val_accuracy: 0.9992
Epoch 6/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0053 -
accuracy: 0.9991 - val_loss: 0.0047 - val_accuracy: 0.9992
Epoch 7/10
357/357 [==============================] - 6s 16ms/step - loss: 0.0048 -
accuracy: 0.9992 - val_loss: 0.0054 - val_accuracy: 0.9993
```

```
Epoch 8/10
357/357 [==============================] - 4s 12ms/step - loss: 0.0047 -
accuracy: 0.9991 - val_loss: 0.0050 - val_accuracy: 0.9993
Epoch 9/10
357/357 [==============================] - 5s 13ms/step - loss: 0.0048 -
accuracy: 0.9991 - val_loss: 0.0050 - val_accuracy: 0.9993
Epoch 10/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0043 -
accuracy: 0.9992 - val_loss: 0.0050 - val_accuracy: 0.9993
90/90 [==============================] - 1s 6ms/step - loss: 0.0050 - accuracy:
0.9993

Test loss : 0.4955

Test accuracy : 99.9320

Confusion Maxtrix :
[[45503     0]
 [   66     0]]

Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     45503
           1       0.00      0.00      0.00        66

    accuracy                           1.00     45569
   macro avg       0.50      0.50      0.50     45569
weighted avg       1.00      1.00      1.00     45569
```

[58]:

```
# Stochastic Gardient Descent optimizers

model.compile(optimizer='sgd',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(xtrain,ytrain,epochs=10,batch_size=512,validation_data=(xtest,ytest))

# Model Evaluation
sgd_score = model.evaluate(xtest,ytest, batch_size=512)
print('\nTest loss : {:.4f}'.format(sgd_score[0]*100))
print('\nTest accuracy : {:.4f}'.format(sgd_score[1]*100))

y_classes_test = np.argmax(model.predict(xtest, verbose=0),axis=1)
# reduce the output to 1d array
yhat_classes_test = y_classes_test.reshape(-1,1)

print('\nConfusion Maxtrix :')
print(confusion_matrix(y_true=ytest,y_pred=yhat_classes_test))
```

```python
print('\nClassification Report : ')
print(classification_report(y_true=ytest,y_pred=yhat_classes_test))
```

```
Epoch 1/10
357/357 [==============================] - 9s 14ms/step - loss: 0.0042 -
accuracy: 0.9992 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 2/10
357/357 [==============================] - 4s 12ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 3/10
357/357 [==============================] - 4s 12ms/step - loss: 0.0041 -
accuracy: 0.9992 - val_loss: 0.0048 - val_accuracy: 0.9993
Epoch 4/10
357/357 [==============================] - 5s 13ms/step - loss: 0.0044 -
accuracy: 0.9991 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 5/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0043 -
accuracy: 0.9992 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 6/10
357/357 [==============================] - 5s 13ms/step - loss: 0.0041 -
accuracy: 0.9992 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 7/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0046 -
accuracy: 0.9992 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 8/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0043 -
accuracy: 0.9992 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 9/10
357/357 [==============================] - 7s 20ms/step - loss: 0.0040 -
accuracy: 0.9993 - val_loss: 0.0048 - val_accuracy: 0.9993
Epoch 10/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0041 -
accuracy: 0.9992 - val_loss: 0.0048 - val_accuracy: 0.9993
90/90 [==============================] - 1s 5ms/step - loss: 0.0048 - accuracy:
0.9993

Test loss : 0.4810

Test accuracy : 99.9298

Confusion Maxtrix :
[[45503     0]
 [   66     0]]

Classification Report :
              precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00     45503
           1       0.00      0.00      0.00        66

    accuracy                           1.00     45569
   macro avg       0.50      0.50      0.50     45569
weighted avg       1.00      1.00      1.00     45569
```

[59]:
```
# RMSProp optimizer

model.
 ↪compile(optimizer='rmsprop',loss='binary_crossentropy',metrics=['accuracy'])
model.fit(xtrain,ytrain,epochs=10,batch_size=512,validation_data=(xtest,ytest))

# Model Evaluation
rms_score = model.evaluate(xtest,ytest, batch_size=512)
print('\nTest loss : {:.4f}'.format(rms_score[0]*100))
print('\nTest accuracy : {:.4f}'.format(rms_score[1]*100))

y_classes_test = np.argmax(model.predict(xtest, verbose=0),axis=1)
# reduce the output to 1d array
yhat_classes_test = y_classes_test.reshape(-1,1)

print('\nConfusion Maxtrix :')
print(confusion_matrix(y_true=ytest,y_pred=yhat_classes_test))
print('\nClassification Report : ')
print(classification_report(y_true=ytest,y_pred=yhat_classes_test))
```

```
Epoch 1/10
357/357 [==============================] - 11s 16ms/step - loss: 0.0045 -
accuracy: 0.9993 - val_loss: 0.0054 - val_accuracy: 0.9994
Epoch 2/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 0.0060 - val_accuracy: 0.9994
Epoch 3/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0039 -
accuracy: 0.9994 - val_loss: 0.0055 - val_accuracy: 0.9994
Epoch 4/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0043 -
accuracy: 0.9993 - val_loss: 0.0049 - val_accuracy: 0.9993
Epoch 5/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 0.0048 - val_accuracy: 0.9993
Epoch 6/10
357/357 [==============================] - 6s 17ms/step - loss: 0.0041 -
accuracy: 0.9994 - val_loss: 0.0059 - val_accuracy: 0.9994
Epoch 7/10
357/357 [==============================] - 5s 14ms/step - loss: 0.0040 -
```

```
accuracy: 0.9993 - val_loss: 0.0051 - val_accuracy: 0.9994
Epoch 8/10
357/357 [==============================] - 5s 15ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 0.0052 - val_accuracy: 0.9994
Epoch 9/10
357/357 [==============================] - 6s 16ms/step - loss: 0.0041 -
accuracy: 0.9993 - val_loss: 0.0045 - val_accuracy: 0.9994
Epoch 10/10
357/357 [==============================] - 6s 16ms/step - loss: 0.0038 -
accuracy: 0.9993 - val_loss: 0.0048 - val_accuracy: 0.9994
90/90 [==============================] - 1s 6ms/step - loss: 0.0048 - accuracy:
0.9994

Test loss : 0.4756

Test accuracy : 99.9364

Confusion Maxtrix :
[[45503     0]
 [   66     0]]

Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     45503
           1       0.00      0.00      0.00        66

    accuracy                           1.00     45569
   macro avg       0.50      0.50      0.50     45569
weighted avg       1.00      1.00      1.00     45569
```

13.0.1 Observation:-

[60]:
```
models = pd.DataFrame({
    'Models' : ['Bernaulli Naive Bayes Over-Sampled','Logistic Regression␣
↪Over-Sampled',
                'Bernaulli Naive Bayes Under-Sampled','Logistic Regression␣
↪Under-Sampled','SVM Under-Sampled',
                'Random Fosrest␣
↪Classifier','XGB','Adam_optimizer_NN','SGD_optimizer_NN','RMSProp_optimizer_NN'],
    'Score' :␣
↪[bnb_acc_over,lr_acc_over,bnb_acc_under,lr_acc_under,svm_acc_under,rfc_acc,xgb_acc,
              adam_score[1]*100,sgd_score[1]*100,rms_score[1]*100]})

round(models.sort_values(by='Score',ascending=False),2)
```

[60]:

	Models	Score
6	XGB	99.96
5	Random Fosrest Classifier	99.96
9	RMSProp_optimizer_NN	99.94
7	Adam_optimizer_NN	99.93
8	SGD_optimizer_NN	99.93
1	Logistic Regression Over-Sampled	94.94
3	Logistic Regression Under-Sampled	94.92
2	Bernaulli Naive Bayes Under-Sampled	92.89
4	SVM Under-Sampled	92.89
0	Bernaulli Naive Bayes Over-Sampled	91.14

14 3. Find the best setting of neural net that can be best classified as fraudulent and non-fraudulent transactions. Use techniques like Grid Search, Cross-Validation and Random search.

14.0.1 Cross Validation:-

[61]:
```
#Extract features and label
X_train=train_df.drop(columns=['Class']).values
y_train=train_df['Class'].values

X_test=test_hidden_df.drop(columns=['Class']).values
y_test=test_hidden_df['Class'].values
print('the shape of X_train and  y_train: ', X_train.shape, y_train.shape)
print('the shape of X_test and  y_test: ', X_test.shape,y_test.shape)
```

```
the shape of X_train and  y_train:  (227845, 30) (227845,)
the shape of X_test and  y_test:  (56962, 30) (56962,)
```

[62]:
```
COLUMN_NAMES = ["Approach","Model Name","F1 Scores","Range of F1 Scores","Std␣
↪Deviation of F1 Scores"]
df_model_selection = pd.DataFrame(columns=COLUMN_NAMES)

def model_traintest_CV(model_obj, model_name, approach, n_splits, X, y):
    global df_model_selection

    skf = StratifiedKFold(n_splits, random_state=12,shuffle=True)

    weighted_f1_score = []

    for train_index, test_index in skf.split(X,y):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        model_obj.fit(X_train,y_train,epochs=10,batch_size=32,verbose=0)
```

```
        # predict  classes
        yhat_classes_train = np.argmax(model_obj.predict(X_train,␣
↪verbose=0),axis=1)
        yhat_classes_test = np.argmax(model_obj.predict(X_test,␣
↪verbose=0),axis=1)

        # reduce the output to 1d array
        yhat_classes_train = yhat_classes_train.reshape(-1, 1)
        yhat_classes_test = yhat_classes_test.reshape(-1, 1)

        #Accuracy Score for train Data
       ␣
↪#accuracy_score_train=accuracy_score(y_true=y_train,y_pred=yhat_classes_train)

        #test_ds_predicted = model_obj.predict( X_test )

        weighted_f1_score.append(round(f1_score(y_true=y_test,␣
↪y_pred=yhat_classes_test , average='weighted'),2))

    sd_weighted_f1_score = np.std(weighted_f1_score, ddof=1)
    range_of_f1_scores = "{}-{}".
↪format(min(weighted_f1_score),max(weighted_f1_score))
    df_model_selection = pd.concat([df_model_selection,
                                    pd.
↪DataFrame([[approach,model_name,sorted(weighted_f1_score),
                                                             ␣
↪range_of_f1_scores,sd_weighted_f1_score]], columns =COLUMN_NAMES) ])
```

[63]:
```
%%time
X=X_train
y=y_train
n_splits=5
approach='Neural Network'
model_obj=model
model_name='Neural Network'
model_traintest_CV(model_obj, model_name, approach, n_splits, X, y)
df_model_selection
```

```
Wall time: 27min 59s
```

[63]:
```
         Approach      Model Name                     F1 Scores  \
0  Neural Network  Neural Network  [1.0, 1.0, 1.0, 1.0, 1.0]

  Range of F1 Scores  Std Deviation of F1 Scores
0            1.0-1.0                         0.0
```

[64]:

```
%%time
# Now lets try to get the Scores using StratifiedKFold Cross Validation for␣
↪Neural Network
#Initialize the algo
model_obj=model
X=X_train
y=y_train

#Initialize StratifiedKFold Method
kfold = StratifiedKFold(n_splits, random_state=12,shuffle=True)

#Initialize For Loop
i=0
for train,test in kfold.split(X,y):
    i = i+1
    X_train,X_test = X[train],X[test]
    y_train,y_test = y[train],y[test]

    model_obj.fit(X_train,y_train,epochs=10,batch_size=32,verbose=0)
    # predict  classes
    yhat_classes_train = np.argmax(model_obj.predict(X_train, verbose=0),axis=1)
    yhat_classes_test = np.argmax(model_obj.predict(X_test, verbose=0),axis=1)

    # reduce the output to 1d array
    yhat_classes_train = yhat_classes_train.reshape(-1, 1)
    yhat_classes_test = yhat_classes_test.reshape(-1, 1)

    test_f1_score=round(f1_score(y_true=y_test, y_pred=yhat_classes_test ,␣
↪average='weighted'),2)
    train_f1_score=round(f1_score(y_true=y_train, y_pred=yhat_classes_train ,␣
↪average='weighted'),2)

    print("Train f1-Score: {}, Test f1-score: {}, for Sample Split: {}".
↪format(train_f1_score,test_f1_score,i))
```

```
Train f1-Score: 1.0, Test f1-score: 1.0, for Sample Split: 1
Train f1-Score: 1.0, Test f1-score: 1.0, for Sample Split: 2
Train f1-Score: 1.0, Test f1-score: 1.0, for Sample Split: 3
Train f1-Score: 1.0, Test f1-score: 1.0, for Sample Split: 4
Train f1-Score: 1.0, Test f1-score: 1.0, for Sample Split: 5
Wall time: 36min 43s
Parser   : 519 ms
```

[65]:

```
%%time
#Lets extract the Train and Test sample for split 2
kfold = StratifiedKFold(n_splits=5,random_state=1,shuffle=True)
i=0
```

```
for train,test in kfold.split(X,y):
    i = i+1
    if i == 2:
        X_train,X_test,y_train,y_test = X[train],X[test],y[train],y[test]

model_obj.fit(X_train,y_train,epochs=10,batch_size=32,verbose=0)
```

```
Wall time: 7min 16s
Compiler : 112 ms
Parser   : 4.44 s
```

[65]: <keras.callbacks.History at 0x13491a7eb50>

[66]:
```
%%time
X_test=test_hidden_df.drop(columns=['Class']).values # Unseen data
y_test=test_hidden_df['Class'].values                # Unseen data

# predict  classes
yhat_classes_train = np.argmax(model_obj.predict(X_train, verbose=0),axis=1)
yhat_classes_test = np.argmax(model_obj.predict(X_test, verbose=0),axis=1)

# reduce the output to 1d array
yhat_classes_train = yhat_classes_train.reshape(-1,1)
yhat_classes_test = yhat_classes_test.reshape(-1,1)

#Accuracy Score for train Data
accuracy_score_train=accuracy_score(y_true=y_train,y_pred=yhat_classes_train)
#Accuracy Score for test Data
accuracy_score_test=accuracy_score(y_true=y_test,y_pred=yhat_classes_test)
print('Train Accuracy score is: {} and Test Accuracy score is: {}'.
 ↪format(accuracy_score_train,accuracy_score_test))

#Confusion Matrix  for train Data
cm=confusion_matrix(y_true=y_train,y_pred=yhat_classes_train)
print('Confusion Matrix for Train Data \n',cm)

#Confusion Matrix Report for Test Data
cm=confusion_matrix(y_true=y_test,y_pred=yhat_classes_test)
print('Confusion Matrix for Test Data \n',cm)

#Classification Report for Train Data
cr=classification_report(y_true=y_train,y_pred=yhat_classes_train)
print('Classification Report for Train Data \n',cr)

#Classification Report for Test Data
cr=classification_report(y_true=y_test,y_pred=yhat_classes_test)
print('Classification Report for Test Data \n',cr)
```

```
Train Accuracy score is: 0.9982718514779785 and Test Accuracy score is:
0.9982795547909132
Confusion Matrix for Train Data
 [[181961      0]
 [   315      0]]
Confusion Matrix for Test Data
 [[56864     0]
 [   98     0]]
Classification Report for Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    181961
           1       0.00      0.00      0.00       315

    accuracy                           1.00    182276
   macro avg       0.50      0.50      0.50    182276
weighted avg       1.00      1.00      1.00    182276

Classification Report for Test Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.00      0.00      0.00        98

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962

Wall time: 1min 36s
Parser   : 492 ms
```

- After cross validation of neural network, there is no significant change in the performance of the model.

15 Anomaly Detection:

4. Implement anomaly detection algorithms.
   - Assume that the data is coming from a single or a combination of multivariate Gaussian
   - Formalize a scoring criterion, which gives a scoring probability for the given data point whether it belongs to the multivariate Gaussian or Normal Distribution fitted in (a)
   - Inference and Observations:
5. Visualize the scores for Fraudulent and Non-Fraudulent transactions.
6. Find out the threshold value for marking or reporting a transaction as fraudulent in your anomaly detection system.

7. Can this score be used as an engineered feature in the models developed previously? Are there any incremental gains in F1-Score? Why or Why not?

8. Be as creative as possible in finding other interesting insights.

[67]:
```
train_df_1 = train_df
test_hidden_df_1 = test_hidden_df
```

[68]:
```
fraud = train_df_1[train_df_1['Class'] == 1]
valid = train_df_1[train_df_1['Class'] == 0]

outlier_fraction = len(fraud)/float(len(valid))
print(outlier_fraction)
print("Fraud Cases : {}".format(len(fraud)))
print("Valid Cases : {}".format(len(valid)))
```

```
0.0017322412299792043
Fraud Cases : 394
Valid Cases : 227451
```

[69]:
```
state = np.random.RandomState(42)
xtrain = train_df_1.drop(columns=['Class'])
ytrain = train_df_1['Class']
xtest = test_hidden_df_1.drop(columns=['Class'])
ytest = test_hidden_df_1['Class']
```

[70]:
```
print('the shape of xtrain and  ytrain: ', xtrain.shape, ytrain.shape)
print('the shape of xtest and  ytest: ', xtest.shape,ytest.shape)
```

```
the shape of xtrain and  ytrain:  (227845, 30) (227845,)
the shape of xtest and  ytest:  (56962, 30) (56962,)
```

[71]:
```
model_isolation =␣
↪IsolationForest(n_estimators=200,max_samples=len(xtrain),contamination=outlier_fraction,
                                random_state=state)
model_isolation.fit(xtrain)
```

[71]:
```
IsolationForest(contamination=0.0017322412299792043, max_samples=227845,
                n_estimators=200,
                random_state=RandomState(MT19937) at 0x1348D2CB340)
```

[72]:
```
ypred = model_isolation.predict(xtest)
ypred[ypred == 1] = 0
ypred[ypred == -1] = 1
```

[73]:
```
print('\nConfusion Matrix : ')
print(confusion_matrix(ytest,ypred))
print('\nClassification Report : ')
```

```
print(classification_report(ytest,ypred))
print('\nAccuracy Score : ')
print(accuracy_score(ytest,ypred))
```

```
Confusion Matrix :
[[56794    70]
 [   69    29]]

Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.29      0.30      0.29        98

    accuracy                           1.00     56962
   macro avg       0.65      0.65      0.65     56962
weighted avg       1.00      1.00      1.00     56962


Accuracy Score :
0.997559776693234
```

[74]:
```
prediction = pd.DataFrame(data=ypred,columns=['predicted_class'])
isolation_pred = pd.concat([test_hidden_df_1,prediction],axis=1)

n_errors = (ypred != ytest).sum()
print('Number of missclassification error for the data is: {}'.format(n_errors))

# Predict the scores
test_scores = model_isolation.decision_function(xtest)
test_scores = pd.DataFrame(data=test_scores,columns=['scores'])
iforest_pred = pd.concat([isolation_pred,test_scores],axis=1)
```

```
Number of missclassification error for the data is: 139
```

[75]:
```
iforest_pred.head()
```

[75]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  113050.0  0.114697  0.796303 -0.149553 -0.823011  0.878763 -0.553152
1   26667.0 -0.039318  0.495784 -0.810884  0.546693  1.986257  4.386342
2  159519.0  2.275706 -1.531508 -1.021969 -1.602152 -1.220329 -0.462376
3  137545.0  1.940137 -0.357671 -1.210551  0.382523  0.050823 -0.171322
4   63369.0  1.081395 -0.502615  1.075887 -0.543359 -1.472946 -1.065484

         V7        V8        V9  …       V23       V24       V25       V26  \
0  0.939259 -0.108502  0.111137  … -0.055940 -1.025281 -0.369557  0.204653
```

```
1 -1.344891 -1.743736 -0.563103  … -0.197573  1.014807  1.011293 -0.167684
2 -1.196485 -0.147058 -0.950224  …  0.150945 -0.811083 -0.197913 -0.128446
3 -0.109124 -0.002115  0.869258  …  0.034206  0.739535  0.223605 -0.195509
4 -0.443231 -0.143374  1.659826  … -0.137223  0.986259  0.563228 -0.574206

        V27       V28  Amount  Class  predicted_class    scores
0  0.242724  0.085713    0.89      0                0  0.190594
1  0.113136  0.256836   85.00      0                0  0.176348
2  0.014197 -0.051289   42.70      0                0  0.183428
3 -0.012791 -0.056841   29.99      0                0  0.193321
4  0.089673  0.052036   68.00      0                0  0.185338

[5 rows x 33 columns]
```

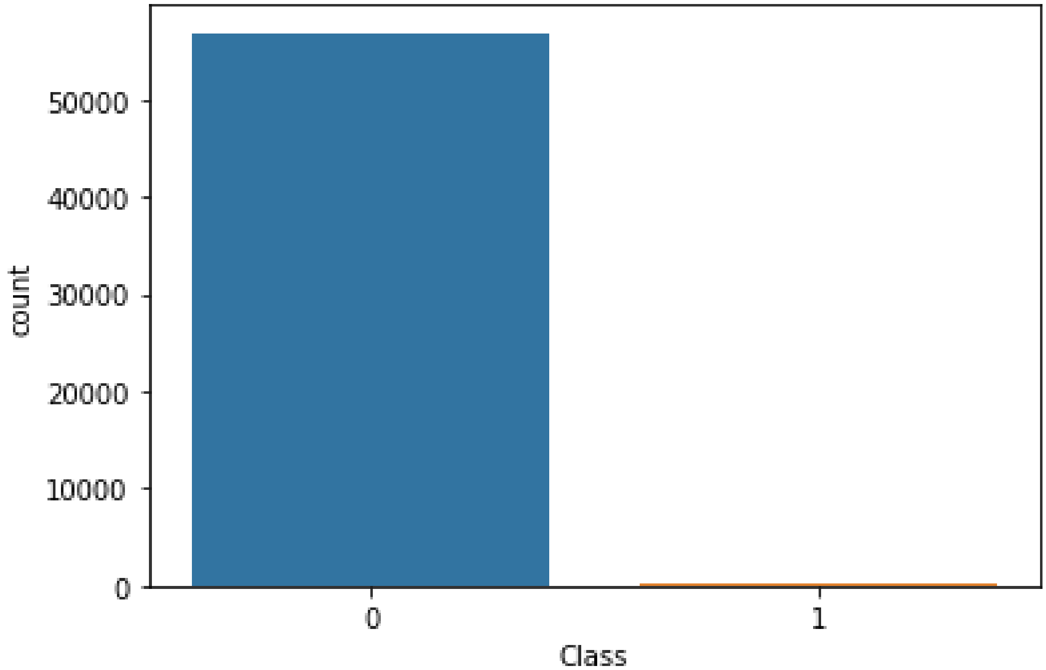
[76]:
```
predicted_class = xgb_model.predict(xtest)
predicted_class = pd.DataFrame(data=predicted_class,columns=['predicted_class'])
xgb_pred = pd.concat([test_hidden_df_1,predicted_class],axis=1)
xgb_pred.head()
```

[76]:
```
       Time        V1        V2        V3        V4        V5        V6  \
0  113050.0  0.114697  0.796303 -0.149553 -0.823011  0.878763 -0.553152
1   26667.0 -0.039318  0.495784 -0.810884  0.546693  1.986257  4.386342
2  159519.0  2.275706 -1.531508 -1.021969 -1.602152 -1.220329 -0.462376
3  137545.0  1.940137 -0.357671 -1.210551  0.382523  0.050823 -0.171322
4   63369.0  1.081395 -0.502615  1.075887 -0.543359 -1.472946 -1.065484

         V7        V8        V9  …       V22       V23       V24       V25  \
0  0.939259 -0.108502  0.111137  … -0.807853 -0.055940 -1.025281 -0.369557
1 -1.344891 -1.743736 -0.563103  … -0.072200 -0.197573  1.014807  1.011293
2 -1.196485 -0.147058 -0.950224  … -0.103533  0.150945 -0.811083 -0.197913
3 -0.109124 -0.002115  0.869258  …  0.650355  0.034206  0.739535  0.223605
4 -0.443231 -0.143374  1.659826  …  0.821209 -0.137223  0.986259  0.563228

        V26       V27       V28  Amount  Class  predicted_class
0  0.204653  0.242724  0.085713    0.89      0                0
1 -0.167684  0.113136  0.256836   85.00      0                0
2 -0.128446  0.014197 -0.051289   42.70      0                0
3 -0.195509 -0.012791 -0.056841   29.99      0                0
4 -0.574206  0.089673  0.052036   68.00      0                0

[5 rows x 32 columns]
```

[77]:
```
sns.countplot(x=xgb_pred.Class)
xgb_pred.Class.value_counts()
```

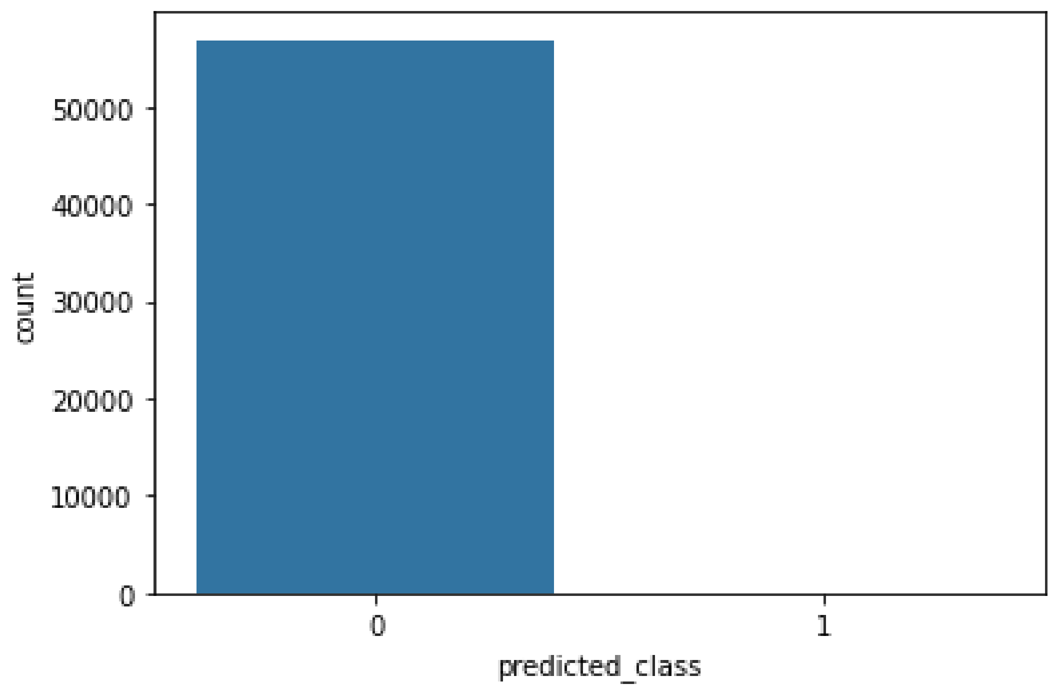
[77]:
```
0    56864
1       98
```

```
Name: Class, dtype: int64
```



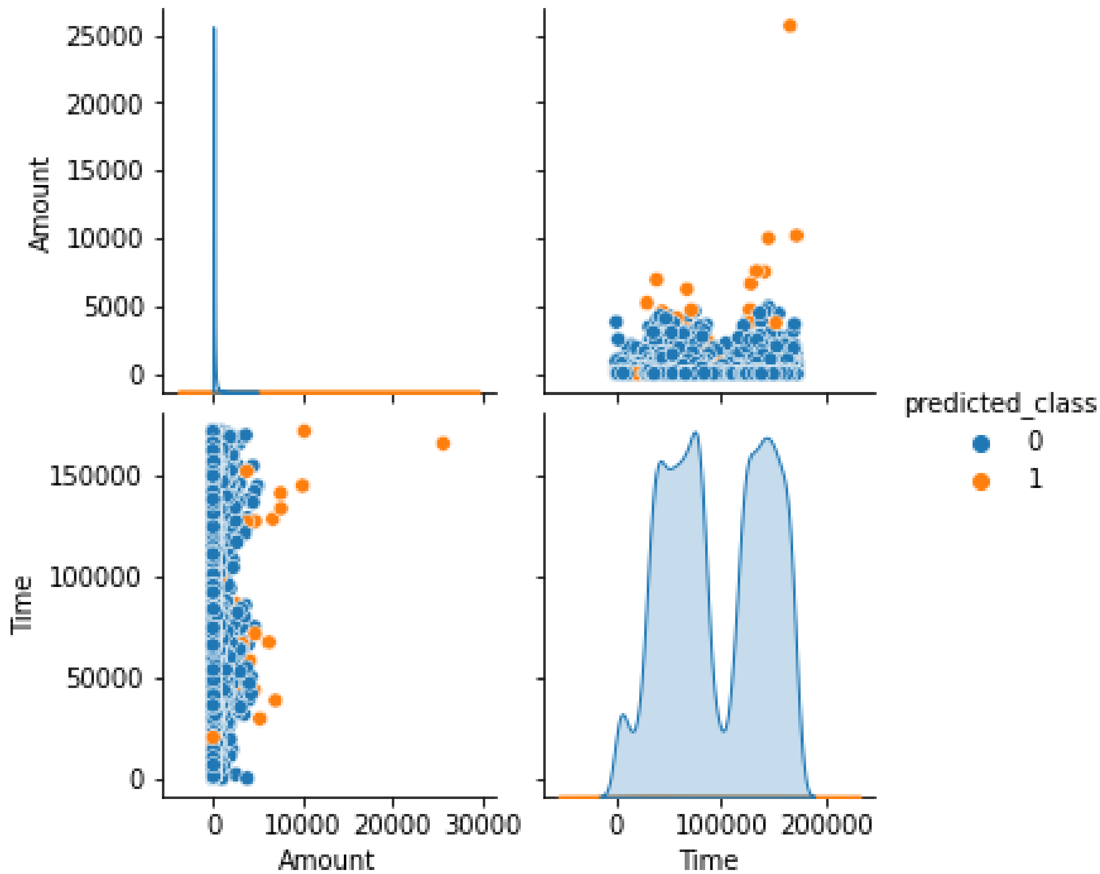
[78]:
```
sns.countplot(x=xgb_pred.predicted_class)
xgb_pred.Class.value_counts()
```

[78]:
```
0    56864
1       98
Name: Class, dtype: int64
```



[79]:
```
sns.
 ↪pairplot(iforest_pred,x_vars=['Amount','Time'],y_vars=['Amount','Time'],kind='scatter',hue=
```

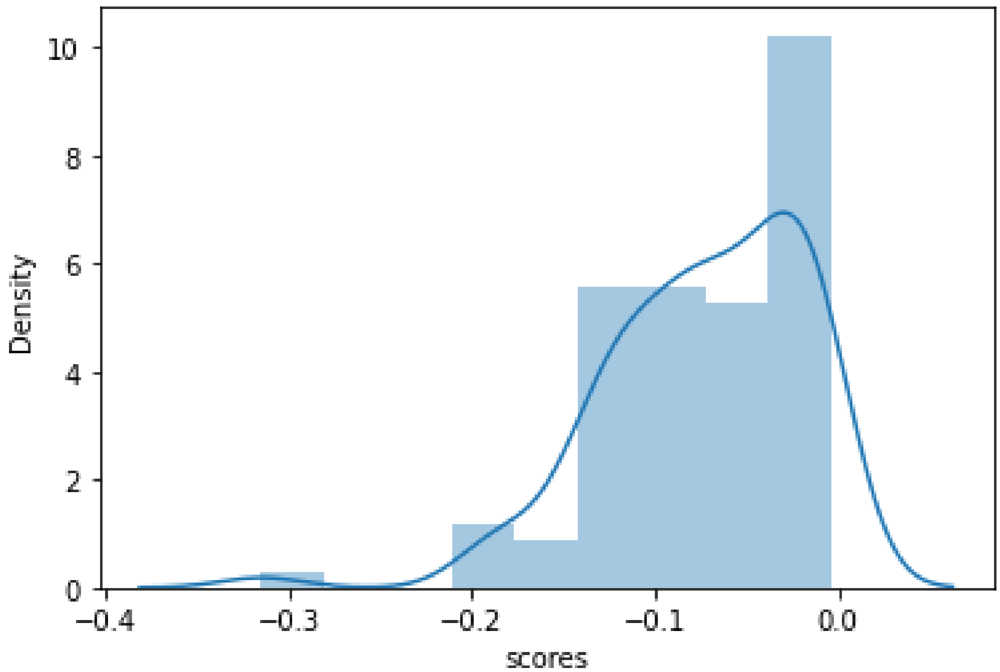
[79]: <seaborn.axisgrid.PairGrid at 0x1348ca83160>



[80]:

```
sns.distplot(iforest_pred[iforest_pred.predicted_class == 1].scores)
```

[80]: <AxesSubplot:xlabel='scores', ylabel='Density'>

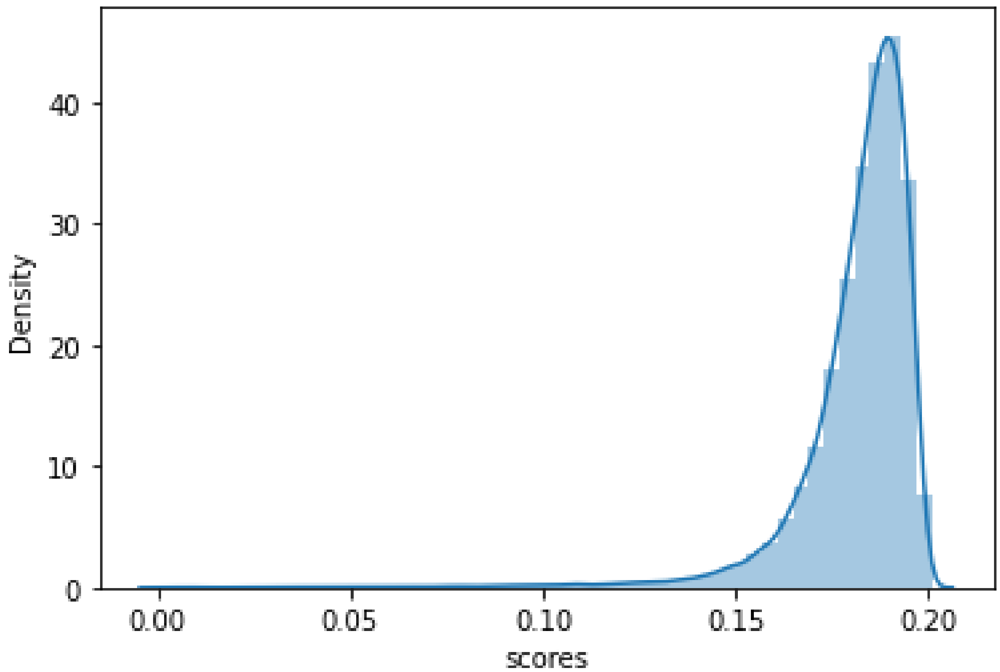


[81]:
```
iforest_pred[iforest_pred.predicted_class == 1].scores.describe()
```

[81]:
```
count    99.000000
mean     -0.073114
std       0.055304
min      -0.315070
25%      -0.111606
50%      -0.070257
75%      -0.027971
max      -0.003711
Name: scores, dtype: float64
```

[82]:
```
sns.distplot(iforest_pred[iforest_pred.predicted_class == 0].scores)
```

[82]:
```
<AxesSubplot:xlabel='scores', ylabel='Density'>
```



[83]:
```
iforest_pred[iforest_pred.predicted_class == 0].scores.describe()
```

[83]:
```
count    56863.000000
mean         0.181402
std          0.016287
min          0.000979
25%          0.177182
50%          0.185220
75%          0.190788
max          0.200706
Name: scores, dtype: float64
```

[84]:
```
pd.set_option('display.max_columns', 50)
pd.set_option('display.max_rows', 200)
iforest_pred[iforest_pred.predicted_class == 1]
```

[84]:
```
          Time         V1         V2         V3         V4         V5  \
40    146140.0  -6.133493   1.371835  -5.770578  -2.384282  -9.622621
550    32610.0  -7.189655   6.978507  -3.642243  -0.125557  -1.950690
653   141812.0 -34.614374 -29.145460 -14.985962   7.677798  -8.846632
1847   28143.0 -27.143678  15.365804 -28.407424   6.370895 -20.087878
2271  171234.0 -16.224299  12.730564 -12.841065  -1.141867  -8.986583
3060   55614.0  -7.347955   2.397041  -7.572356   5.177819  -2.854838
3801   41237.0 -10.281784   6.302385 -13.271718   8.925115  -9.975578
4098  160444.0 -28.623353 -19.262983 -13.042666  11.027770 -14.710239
```

5225	133010.0	-23.103244	-23.866465	-5.699313	6.503369	11.184636
5395	10784.0	-9.791064	8.261750	-2.524941	-0.896418	-2.430637
6562	26961.0	-23.237920	13.487386	-25.188773	6.261733	-17.345188
6873	18359.0	-4.071799	4.929765	-9.523349	5.791918	-4.772561
7021	127598.0	-9.110393	-15.447182	-6.708418	0.115435	-10.138617
7811	86903.0	-18.606386	-26.557869	0.622018	11.229502	11.471213
8359	66762.0	-25.206885	-33.066689	-10.207116	8.228752	-0.863502
8992	52439.0	-15.128164	-4.759922	-4.388698	2.968675	1.412581
10056	29601.0	-5.715306	-0.094700	-7.856475	2.259661	-23.611865
10098	157529.0	-6.811662	7.863451	-6.335487	0.748775	-2.001254
10329	127224.0	-9.410703	-15.782232	-0.646987	3.533611	9.087979
10593	94364.0	-15.192064	10.432528	-19.629515	8.046075	-12.838167
11322	6986.0	-4.397974	1.358367	-2.592844	2.679787	-1.128131
11331	53236.0	-17.161218	-2.768536	-5.205376	3.746952	0.023957
11618	18714.0	-10.356863	8.684640	-3.006555	-0.910952	-2.836169
13344	71898.0	-19.827846	-15.711131	-7.387595	7.604936	-9.141652
13627	24120.0	-11.918763	8.626111	-15.895755	6.038810	-9.897807
15033	102671.0	-4.991758	5.213340	-9.111326	8.431986	-3.435516
15273	135810.0	-22.322051	-22.208926	-8.997418	3.396521	1.155982
15641	51557.0	-11.071124	9.681705	-6.415495	-0.201135	-4.919017
16206	55578.0	-12.188710	10.541128	-7.346069	-0.233647	-5.634183
16782	12093.0	-4.696795	2.693867	-4.475133	5.467685	-1.556758
19438	154188.0	-17.678628	-25.041752	-3.780019	7.928090	17.471828
20290	100298.0	-22.341889	15.536133	-22.865228	7.043374	-14.183129
20569	22690.0	-8.595584	7.160228	-13.211950	5.915241	-7.744634
21337	58342.0	-5.143560	-2.771759	-5.682932	2.981850	-7.459633
21355	53054.0	-21.743551	-27.983104	-2.154281	4.795170	9.604996
22484	85285.0	-7.030308	3.421991	-9.525072	5.270891	-4.024630
22767	26525.0	-3.156608	-2.895171	3.300964	1.550562	16.160824
23988	64509.0	-18.913955	-22.977901	-0.076753	6.140323	13.183533
25017	26556.0	-19.179826	11.817922	-21.919174	6.086236	-14.708845
26225	24735.0	-14.575410	9.802337	-18.043109	6.136942	-11.623105
26589	84204.0	-1.927453	1.827621	-7.019495	5.348303	-2.739188
26780	24532.0	-13.911336	9.508117	-17.506256	6.112442	-11.191653
26968	55311.0	-6.159607	1.468713	-6.850888	5.174706	-2.986704
28331	93920.0	-12.381048	8.213022	-16.962530	7.116091	-9.772826
28996	41305.0	-12.980943	6.720508	-13.455636	8.698610	-11.479552
28999	20931.0	-16.367923	9.223692	-23.270631	11.844777	-9.462037
29604	152036.0	-4.320609	3.199939	-5.799736	6.502330	0.378479
31088	36520.0	-24.465549	-5.667223	-12.157678	6.108698	-4.898742
31138	171200.0	-15.103308	11.874957	-11.907335	-1.110679	-8.266879
31455	29650.0	-7.895380	7.769963	-3.839012	0.738865	-3.361851
31881	44423.0	-33.017174	-39.818310	-1.445971	10.739659	28.762671
31896	93824.0	-3.632809	5.437263	-9.136521	10.307226	-5.421830
32514	105064.0	-16.654005	-16.746795	1.308989	5.486626	18.310857
32549	8808.0	-4.617217	1.695694	-3.114372	4.328199	-1.873257
32690	43851.0	-12.665062	-8.632804	-8.318791	1.875694	-22.064519

33476	141546.0	-13.396920	-19.230653	-9.042012	5.678408	-21.577019
33548	93860.0	-10.850282	6.727466	-16.760583	8.425832	-10.252697
34450	93853.0	-5.839192	7.151532	-12.816760	7.031115	-9.651272
36642	41508.0	-32.273470	17.930550	-32.454198	6.555152	-23.236403
38276	71861.0	-5.397276	-3.869015	-5.555717	1.738352	-19.832791
39921	85285.0	-6.713407	3.921104	-9.746678	5.148263	-5.151563
40295	152058.0	-3.576362	3.299436	-7.460433	7.783634	-0.398549
40416	149898.0	-23.107768	-21.381940	-12.195878	6.238771	0.390471
41530	121466.0	-7.339008	-3.880399	-7.440497	0.005757	-4.022370
41533	168987.0	-25.672101	-30.913347	-2.943712	5.330375	11.886791
42307	53160.0	-11.629920	10.111411	-6.880781	-0.217390	-5.276595
43355	148806.0	-33.669917	-47.429676	-7.198018	10.055906	29.016124
43398	83921.0	-10.359314	-1.072950	0.642733	-0.976677	1.376521
43973	4429.0	-5.043472	4.015914	1.999942	-3.470182	1.755487
44257	127463.0	-11.732650	-5.571658	-6.692965	1.753141	0.583680
44475	137494.0	-27.007997	-23.923599	-9.514275	4.870419	-2.447970
44503	23698.0	-10.589953	8.038901	-14.822157	5.989558	-9.035866
44641	21046.0	-16.917468	9.669900	-23.736443	11.824990	-9.830548
45215	98282.0	-15.634705	-17.753670	1.741845	4.519862	19.180525
45660	57957.0	-29.516123	-33.204192	-4.424643	9.684232	9.000517
46402	138416.0	-26.054765	-37.154221	-4.707242	7.258823	14.929359
46495	55460.0	-11.746656	-10.410737	-1.273259	5.588220	-2.582046
47329	172273.0	-9.030538	-11.112584	-16.233798	3.592021	-40.427726
47401	67676.0	-7.103082	-1.706830	-9.726332	1.657590	-31.356750
47483	38763.0	-14.711825	-23.250844	-7.631400	5.975826	-15.615302
48689	166198.0	-35.548539	-31.850484	-48.325589	15.304184	-113.743307
49333	79617.0	-24.316924	-20.949142	-11.058967	9.144180	-10.495812
49716	36225.0	-23.420359	-6.614989	-11.359362	6.110039	-4.295862
50384	41203.0	-8.426814	6.241659	-9.946470	8.199614	-8.213093
50447	93879.0	-13.086519	7.352148	-18.256576	10.648505	-11.731476
51249	18675.0	-12.339603	4.488267	-16.587073	10.107274	-10.420199
51504	26833.0	-20.532751	12.373989	-23.009003	6.144821	-15.587296
52067	128317.0	-17.015895	-18.501723	-2.965763	5.989228	7.811563
53009	11080.0	-2.125490	5.973556	-11.034727	9.007147	-1.689451
53073	128701.0	-19.780626	-25.663628	-10.865410	6.046025	-16.459773
53361	145283.0	-21.532478	-34.704768	-8.303035	10.264175	3.957175
53559	152443.0	-13.878774	-15.126570	-6.760480	8.225483	-10.438514
53930	52997.0	-34.148234	18.902453	-33.680984	6.648835	-23.669726
54937	100384.0	-23.579440	-25.377991	-2.677573	4.577916	6.617594
55218	78773.0	-9.323036	8.154362	-5.743131	1.136134	-4.440984
55254	133971.0	-10.950173	-13.359133	-10.664755	1.157565	-28.363785
56219	86570.0	-36.510583	-40.938048	-5.377986	11.474590	11.066946
56494	17044.0	-1.686955	4.081249	-7.770538	5.565240	-3.701602
56787	20474.0	-10.931437	9.092123	-3.473866	-0.920861	-3.176341

	V6	V7	V8	V9	V10	V11	\
40	13.470790	-0.795775	-18.592913	-2.399328	-4.778083	-0.869835	

550	7.518234	-14.834807	-34.535000	-2.837788	-5.139350	-0.485479
653	5.571214	13.167616	-4.717060	6.091319	4.347796	3.054225
1847	-4.666313	-18.709479	17.903574	-3.722279	-8.120962	4.419943
2271	-3.077067	-8.219623	10.713656	2.367518	4.536268	-3.962664
3060	-1.795239	-8.783235	0.437157	-3.740598	-8.332863	5.763189
3801	-2.832513	-12.703253	6.706846	-7.078424	-12.805683	6.786058
4098	9.042659	12.143391	-2.818318	5.376427	6.556983	0.838451
5225	-6.196146	-6.909951	-1.785498	3.373336	2.718968	1.627024
5395	4.530167	-8.784593	-22.159063	3.130838	2.344039	1.854359
6562	-4.534989	-17.100492	15.374630	-3.845567	-8.511767	5.138547
6873	-3.307247	-8.513680	3.073999	-3.297625	-10.957551	9.540746
7021	6.888007	15.384417	-1.162533	0.019771	-5.613092	0.668532
7811	-7.800637	-5.679901	1.205994	-0.446864	1.224902	-1.348572
8359	-2.886105	5.847721	-0.854459	0.433062	-2.974644	-1.580459
8992	1.017313	-9.760064	-14.018265	-0.041771	-3.080965	-2.932109
10056	16.493227	21.437514	-10.623397	-0.968764	-1.999255	2.308866
10098	5.876819	-16.564487	-39.267378	-3.984087	-8.985126	-1.055392
10329	-6.680670	-11.496851	-0.892163	1.321719	1.331957	-0.094932
10593	-1.875859	-21.359738	-3.717850	-5.969782	-17.141514	5.902400
11322	-1.706536	-3.496197	-0.248778	-0.247768	-4.801637	4.895844
11331	2.404662	-12.883356	-18.418978	-1.154101	-4.084567	-3.139342
11618	4.487907	-9.105365	-21.725442	3.148706	2.419650	1.730069
13344	7.399126	9.743209	-3.525079	4.862849	4.748025	0.530565
13627	-3.814247	-12.155913	8.143381	-2.994954	-10.088539	8.107018
15033	-1.827565	-7.114303	3.431207	-3.875643	-6.868509	7.150625
15273	-0.716039	3.832328	-1.871467	-0.840216	2.367328	1.486199
15641	4.793588	-15.420812	-24.520275	-2.919213	-4.537718	-1.615380
16206	4.648717	-16.164717	-23.641430	-2.875288	-4.368064	-1.866130
16782	-1.549420	-4.104215	0.553934	-1.498468	-4.594952	5.275506
19438	-13.591286	-10.532208	1.480096	0.553701	1.036473	-1.345270
20290	-0.463145	-28.215112	-14.607791	-9.481456	-20.949192	4.739582
20569	-3.606638	-10.660939	6.034773	-3.121460	-10.443585	8.703025
21337	4.937310	3.528577	-7.914629	-0.785318	-1.082876	1.443159
21355	-7.501200	-6.703914	0.266555	3.966169	-1.757164	0.916624
22484	-2.865682	-6.989195	3.791551	-4.622730	-8.409665	6.309044
22767	-8.710536	-17.936966	0.492347	1.851809	3.333234	2.098537
23988	-9.551773	-4.389705	0.303038	1.211620	0.621515	0.528342
25017	-4.308888	-15.357952	12.857165	-3.999861	-8.928656	5.849293
26225	-3.978362	-13.350186	9.829463	-2.893536	-9.804246	7.630468
26589	-2.107219	-5.015848	1.205868	-4.382713	-8.337707	7.190306
26780	-3.937424	-13.051697	9.407979	-2.918901	-9.875331	7.749594
26968	-1.795054	-6.545072	2.621236	-3.605870	-8.122161	6.029033
28331	-3.666836	-16.147363	2.078706	-4.250657	-16.746044	7.425801
28996	-2.681519	-14.019291	8.218191	-7.930900	-12.695947	5.589362
28999	-2.450444	-16.925152	1.384208	-6.287736	-13.002709	9.691461
29604	-1.948246	-2.167860	-0.728207	-1.977238	-3.473411	4.569194
31088	0.551627	-11.558180	-13.956673	-1.181080	-2.985239	-2.175315

31138	-2.934550	-7.479686	9.835984	2.323339	4.367054	-3.712310
31455	7.473970	-18.287733	-41.484823	-4.532523	-8.333226	-1.019981
31881	-19.996349	-19.083907	-1.992887	5.000712	4.895647	2.027218
31896	-2.864815	-10.634088	3.018127	-4.891640	-11.235048	8.788784
32514	-9.149236	-13.199196	-2.250596	5.201975	6.465594	2.877062
32549	-0.989908	-4.577265	0.472216	0.472017	-5.576023	4.802323
32690	14.672360	23.863052	-6.156826	2.520556	-0.287792	3.142875
33476	12.128950	26.237722	-1.794955	-3.144266	-7.298829	-1.510263
33548	-4.192171	-14.077086	7.168288	-3.683242	-15.239962	8.030708
34450	-2.938427	-11.543207	4.843627	-3.494276	-13.320789	8.460244
36642	-4.487066	-22.030417	18.282168	-3.797563	-8.102120	3.492832
38276	14.996212	19.753950	-2.395134	-1.452323	-3.888765	1.468648
39921	-2.099389	-5.937767	3.578780	-4.684952	-8.537758	6.348979
40295	-1.968441	-3.110476	-0.328404	-1.574363	-2.497561	4.604170
40416	-0.572096	4.354560	-1.182947	1.798918	-0.731991	2.395601
41530	3.353394	1.755877	-13.411491	-0.139337	-2.024238	0.180527
41533	-8.036176	-1.273707	-1.444300	2.047691	1.271063	-0.630691
42307	4.721149	-15.792767	-24.080851	-2.897251	-4.452891	-1.740756
43355	-20.054615	-18.381781	1.799471	1.663887	2.705437	0.597329
43398	0.763807	4.427995	-4.531827	8.113152	15.236028	4.411621
43973	-0.376547	4.163343	-4.054582	9.272376	11.936393	4.354865
44257	1.106816	5.398594	-3.356352	5.031461	5.193795	3.527239
44475	1.893199	3.224013	-6.878781	2.190446	1.323663	1.809361
44503	-3.731684	-11.558339	7.300134	-3.045608	-10.230620	8.345358
44641	-2.514829	-17.290657	1.820408	-6.264903	-12.916636	9.567110
45215	-13.009119	-13.465448	-0.404634	3.955240	2.979077	2.345099
45660	-6.543026	-1.072169	-1.352590	4.133987	1.204729	0.913224
46402	-12.709475	-7.004254	1.323841	-1.056135	-1.344687	-1.391533
46495	5.319379	6.079441	-4.303163	4.475720	4.983061	0.657541
47329	23.917837	44.054461	-7.277778	-4.210637	-7.776435	0.214173
47401	20.379524	29.205868	-5.498667	-1.369680	-3.713841	2.212401
47483	8.060516	21.246173	-1.896157	-1.450057	-6.284888	-1.359855
48689	73.301626	120.589494	-27.347360	-3.872425	-12.005487	6.853897
49333	6.834122	11.217637	-3.847405	2.988551	3.306169	1.167683
49716	0.085414	-10.053042	-11.796355	-0.952343	-2.584077	-1.964753
50384	-2.522046	-11.643028	5.339500	-7.051016	-12.265324	7.047733
50447	-3.659167	-14.873658	8.810473	-5.418204	-13.202577	6.357227
51249	0.130670	-15.600323	-1.157696	-5.304631	-12.938929	8.805682
51504	-4.384491	-15.939003	13.696416	-3.948455	-8.789723	5.612347
52067	-4.440128	-1.905238	-1.938201	3.276087	3.752052	1.642748
53009	-2.854415	-7.810441	2.030870	-5.902828	-12.840934	12.018913
53073	9.410864	18.585729	-4.394882	-1.049021	-4.226812	0.462665
53361	-3.229695	-4.066768	-4.083971	0.554072	-2.166867	0.939705
53559	10.821301	17.911884	-4.748378	0.645546	2.506675	2.601101
53930	-4.472917	-24.419483	16.635979	-3.957251	-8.352964	3.146654
54937	-5.150745	0.733276	-2.699651	4.102659	3.191544	3.626705
55218	4.138796	-13.729788	-19.119754	-4.105163	-5.691141	-1.279083

55254	17.019934	30.897666	-4.699193	-2.450505	-5.854899	-0.660502
56219	-5.982594	0.068963	-3.918451	7.193327	4.173965	1.530781
56494	-3.124916	-7.848606	1.748217	-3.270511	-11.208723	10.002190
56787	4.401880	-9.494388	-21.283961	3.172836	2.506592	1.604644

	V12	V13	V14	V15	V16	V17 \
40	0.958136	-0.703806	2.096236	0.639514	1.898168	-0.008576
550	4.133542	-1.564512	4.959964	-0.504797	2.441416	1.088824
653	1.005763	3.110310	-2.610117	4.441177	5.021881	-1.679253
1847	-6.210941	1.063837	-5.843528	-0.108836	-5.606597	-11.756256
2271	3.780793	0.598057	6.122479	0.170371	1.911043	3.533903
3060	-8.707879	-1.716949	-9.577194	0.146369	-7.586491	-12.503931
3801	-13.064240	1.179525	-13.694873	0.951479	-10.954286	-20.583593
4098	0.162703	2.226394	-2.967565	2.413155	4.622756	-0.921336
5225	2.458631	1.442617	0.252405	2.411592	1.747117	-0.012168
5395	0.237107	0.536751	3.185343	-0.823733	0.460981	2.675160
6562	-7.220020	0.615793	-7.327222	-0.038632	-6.331515	-12.688858
6873	-14.745849	0.078563	-13.127971	-0.489528	-9.860339	-16.511143
7021	0.564606	1.001802	0.383306	-0.422669	0.060591	-0.816459
7811	-0.094460	1.162601	0.548696	2.386685	0.511524	-0.022486
8359	0.547727	1.736481	1.944042	1.659599	2.684656	-0.414598
8992	1.590904	-1.053961	2.869798	0.955476	3.620831	0.223949
10056	-1.217238	0.241034	-0.421127	2.774303	2.594527	-0.640602
10098	3.955275	-2.179125	3.414998	-0.449382	3.084409	3.775429
10329	2.456269	-0.601960	1.710716	-0.369696	-1.821681	1.090115
10593	-13.580147	-0.451407	-8.334763	-2.025145	-10.196334	-17.270985
11322	-10.912819	0.184372	-6.771097	-0.007326	-7.358083	-12.598419
11331	2.255504	-1.257249	4.278103	1.720240	3.664296	1.415112
11618	0.402577	0.600400	3.429128	-0.847263	0.565918	2.832621
13344	-0.451395	1.152050	-3.497825	1.646625	2.632091	-1.269864
13627	-12.671314	0.969084	-10.137194	-0.619098	-8.426839	-14.624485
15033	-10.262984	2.733085	-10.127525	-0.262784	-5.190271	-8.655711
15273	-0.260376	2.015803	-0.074573	1.936466	1.247803	0.454929
15641	4.063530	-2.673083	6.930377	-0.804670	3.306019	1.963295
16206	4.403899	-2.550216	7.421944	-0.854536	3.506065	2.283127
16782	-11.349029	0.374549	-8.138695	0.548571	-6.653594	-10.246755
19438	1.278301	-0.270974	2.964974	0.857710	-1.193961	0.858656
20290	-11.924955	-1.501411	-3.836781	-2.720329	-8.880106	-15.825136
20569	-13.537702	0.592035	-11.386927	-0.570694	-9.030880	-15.410250
21337	1.208242	-0.549410	2.168970	1.443285	1.403376	0.230770
21355	3.042575	0.439105	0.114088	-0.201688	0.417597	0.530785
22484	-8.576761	0.246747	-11.534046	-0.364265	-5.452495	-11.887570
22767	2.723363	-1.236551	1.274733	2.228561	-1.469456	2.117834
23988	0.322013	1.156260	-0.935398	2.586230	0.932908	-0.203844
25017	-8.261650	0.153829	-8.829359	0.008879	-7.070953	-13.629721
26225	-11.978350	1.270701	-9.137791	-0.657644	-7.943184	-13.996045
26589	-9.424844	-0.223293	-12.875494	-0.071918	-6.299961	-12.719207

26780	-12.151584	1.195298	-9.387624	-0.648015	-8.064117	-14.153147
26968	-9.225855	-1.546759	-10.309334	0.308062	-7.787326	-12.822177
28331	-15.564838	-0.426338	-14.029538	-1.681889	-11.133761	-15.833589
28996	-11.960866	1.538671	-9.887214	0.633979	-11.350244	-21.710188
28999	-13.886595	0.838361	-13.517072	-0.377911	-7.855681	-11.803815
29604	-9.321153	-1.592518	-14.266836	0.467777	-4.066209	-6.626968
31088	4.317589	-0.183346	6.614835	1.392499	3.609271	3.365197
31138	3.440374	0.474618	5.630943	0.219504	1.709901	3.214280
31455	4.846452	-2.494630	6.634483	-0.224544	2.527769	2.122321
31881	3.162415	1.761576	-0.113249	2.984261	3.462235	-1.898428
31896	-18.553697	-0.339533	-15.623187	-0.188979	-12.427961	-20.159047
32514	-1.205270	2.390260	-0.341697	1.453632	1.924755	-0.483108
32549	-10.833164	0.104304	-9.405423	-0.807478	-7.552342	-9.802562
32690	-1.058840	0.940757	-3.465003	1.548751	3.163488	-1.812174
33476	-2.553842	-0.159508	1.257466	0.811600	1.253093	-0.285696
33548	-16.060306	0.270530	-14.952981	-0.241095	-11.866731	-15.486990
34450	-17.003289	0.101557	-14.094452	0.747031	-12.661696	-18.912494
36642	-4.748331	1.366199	-3.689181	-0.266118	-4.606720	-10.515507
38276	-1.204629	-0.795373	-0.471132	-0.051714	2.132198	-0.183123
39921	-8.681609	0.251179	-11.608002	-0.351569	-5.363566	-11.939092
40295	-9.001915	-1.276324	-13.969471	1.256945	-4.491629	-5.969987
40416	0.800584	1.873505	-2.642888	2.945018	4.138251	1.998838
41530	0.129513	-3.095282	2.470850	-1.012295	2.416267	-0.819003
41533	-0.755941	1.965513	-2.259398	1.194688	3.806672	-1.539355
42307	4.233714	-2.611649	7.176161	-0.829604	3.406041	2.123211
43355	2.917803	1.916414	2.741847	1.256297	0.948582	1.021919
43398	-1.284914	0.014676	-6.866943	2.992556	-3.534117	-2.495000
43973	-3.393634	1.308787	-4.632961	0.091934	-1.396627	-2.648779
44257	-0.556252	0.182919	-6.066479	2.758091	0.585457	1.252735
44475	1.039601	2.123019	-0.801184	2.231677	4.675399	0.097998
44503	-13.017834	0.818270	-10.636994	-0.599780	-8.668558	-14.938748
44641	-13.717067	0.899541	-13.272965	-0.402260	-7.754094	-11.644603
45215	-0.636945	1.411889	1.644778	0.152244	0.533781	0.190818
45660	1.310252	2.033246	-2.358332	0.737625	3.318413	-1.253843
46402	0.864529	1.912425	2.247206	0.792110	1.471444	0.646400
46495	-1.182700	0.134645	-4.577094	0.913440	1.054120	-1.699405
47329	-4.499851	0.241005	0.537895	2.901938	2.326099	-0.402142
47401	-1.534510	0.816452	-1.187157	0.949474	4.594817	-1.560379
47483	-1.447966	0.608318	0.234851	-0.494890	1.236134	-0.883152
48689	-9.189418	7.126883	-6.795942	8.877742	17.315112	-7.173805
49333	-0.351420	2.058694	-1.511081	4.153006	4.263727	-1.630317
49716	3.902407	-0.025189	5.970329	1.497607	3.337165	3.070031
50384	-13.742953	0.821627	-14.107464	1.020471	-11.847887	-21.673987
50447	-15.531611	0.659695	-11.412330	-2.447576	-9.833743	-18.174617
51249	-13.556130	1.165464	-9.809882	0.369987	-9.505210	-17.542030
51504	-7.914422	0.307820	-8.328601	-0.006979	-6.824524	-13.316079
52067	0.514523	2.190782	-1.980082	3.654318	2.058084	-1.526250

53009	-17.769143	-0.431036	-19.214325	-0.962465	-10.266609	-15.503392
53073	-0.409360	1.242140	0.318114	0.814830	1.685414	-0.512275
53361	3.108922	0.808613	4.109779	3.017039	0.554018	1.174609
53559	-1.671463	1.279100	-2.889784	2.671856	1.840254	-1.443680
53930	-4.095033	1.280562	-2.751587	-0.382589	-4.258292	-10.032699
54937	-2.432376	4.469566	-1.671117	1.889619	4.549260	-1.518485
55218	3.897595	-2.285943	7.234249	-0.401481	2.284974	2.650781
55254	-4.023513	-0.154341	-0.969794	0.653798	4.448977	-1.044700
56219	0.579988	2.237871	-4.705675	2.179692	4.202611	-2.863922
56494	-15.144988	-0.213782	-13.780377	-0.459420	-10.053112	-17.098444
56787	0.575652	0.662976	3.676920	-0.870836	0.666121	2.993476

	V18	V19	V20	V21	V22	V23 \
40	1.589432	-2.654056	0.193376	-10.233407	3.008718	-0.828517
550	0.473675	-1.701352	5.755105	-7.937783	1.498444	3.146670
653	-3.138214	1.751576	-20.235060	-8.172558	-1.323059	-3.329551
1847	-4.714947	0.783578	1.703888	1.796826	-1.960974	-0.902247
2271	0.583581	-1.135103	1.285604	0.427780	0.263553	1.551892
3060	-4.375631	2.465195	0.073164	-0.175273	0.543325	-0.547955
3801	-7.517262	2.872354	-0.247648	2.479414	0.366933	0.042805
4098	-0.567711	3.076979	-15.251547	-6.511789	-0.416771	-10.288551
5225	-2.989766	0.902210	-12.738325	-4.324577	-2.267901	-19.056701
5395	0.484895	-0.869686	-4.561608	22.588989	-8.527145	3.642683
6562	-4.847382	1.020536	1.630787	1.769708	-1.691973	-1.045673
6873	-5.114520	2.001710	1.356943	1.503540	-0.331471	-0.023827
7021	0.683746	-2.086419	10.570799	2.860811	-0.289826	11.990865
7811	2.470568	0.382161	11.163671	3.260623	-0.609765	9.376974
8359	0.782699	-0.877670	6.750851	1.468403	-2.966968	-1.900022
8992	0.180541	0.038479	-7.348950	-10.738634	4.198538	7.441508
10056	-0.027115	1.638086	-10.637028	1.781773	-0.875350	-4.113185
10098	1.845297	-3.123514	9.547556	-18.603088	6.790452	3.114580
10329	0.848317	-1.414251	5.966793	2.368389	-7.417140	-27.215436
10593	-7.079096	1.517695	1.657476	-3.474097	1.765446	1.701257
11322	-5.131549	0.308334	-0.171608	0.573574	0.176968	-0.436207
11331	0.432699	1.206821	-6.406493	-13.322369	4.725713	7.382624
11618	0.508993	-0.904663	-4.543540	22.599543	-8.555808	3.740897
13344	-0.394877	2.592450	-10.237424	-4.593101	-0.110243	-8.572959
13627	-4.889655	1.515572	1.455436	1.556121	-0.956969	-0.379310
15033	-2.024443	1.560479	0.098132	1.189423	0.247858	0.294448
15273	-1.584154	1.066541	-11.617418	-4.166880	1.047062	-0.916381
15641	1.170643	-1.508353	7.338470	-13.950186	3.438142	2.945028
16206	1.214884	-1.578998	7.381936	-13.923111	3.372198	3.080121
16782	-4.191066	0.991486	-0.158971	0.573898	-0.080163	0.318408
19438	0.599888	-0.061230	7.301168	2.864273	-0.544440	0.179031
20290	-6.750425	-0.129188	4.100019	-9.110423	4.158895	1.412928
20569	-4.984761	1.722051	1.397814	1.531017	-0.686448	-0.204835
21337	-0.866507	-0.184667	-3.211296	-3.675875	-1.023120	-19.935025

21355	-0.652905	-0.438190	0.007731	-0.149312	-1.770710	-6.601208
22484	-3.563585	0.876019	0.545698	1.103398	-0.541855	0.036943
22767	-2.827846	-1.173909	-1.444388	0.831647	-4.120410	-26.751119
23988	0.041600	1.336309	-3.089815	-1.753333	1.330413	19.002942
25017	-4.958830	1.272091	1.572950	1.746802	-1.353149	-0.762965
26225	-4.814141	1.350683	1.497143	1.574778	-1.173176	-0.522423
26589	-3.740176	0.844060	2.172709	1.376938	-0.792017	-0.771414
26780	-4.832993	1.391892	1.486919	1.570180	-1.119134	-0.486481
26968	-4.367677	2.643984	-0.289830	1.061314	0.125737	0.589592
28331	-5.748533	2.271082	0.537795	0.167703	1.503413	-0.767755
28996	-8.859452	3.629714	-0.843303	2.549628	-0.532228	-0.235096
28999	-4.761026	0.618624	0.993585	-2.343674	1.004602	1.188212
29604	-1.437158	-0.215410	-0.263686	0.476660	0.434278	-0.136940
31088	-0.682418	0.199947	-5.448307	-11.489450	3.087889	-0.214251
31138	0.539777	-1.064368	1.244927	0.401829	0.330268	1.422984
31455	-0.495560	-3.323344	10.150611	-20.262054	5.805795	3.552843
31881	-1.383824	1.100056	-15.806476	-5.298210	-1.701734	-12.529524
31896	-6.888891	2.586093	1.354065	2.309880	0.978660	-0.096130
32514	-2.515823	-0.244454	-12.725574	-3.831422	-0.525765	-10.701228
32549	-4.120629	1.740507	-0.039046	0.481830	0.146023	0.117039
32690	-1.989247	0.627383	-13.684210	-5.733766	1.051820	-1.292170
33476	2.880040	-0.920825	16.436920	4.036760	-1.638596	17.606637
33548	-5.748652	4.130031	-0.646818	2.541637	0.135535	-1.023967
34450	-6.626975	4.008921	0.055684	2.462056	1.054865	0.530481
36642	-4.539148	0.332820	2.418001	0.581454	-1.931440	-0.895689
38276	-1.538705	-2.493639	-2.887304	-1.343165	0.052619	0.223988
39921	-3.583603	0.897402	0.135711	0.954272	-0.451086	0.127214
40295	-1.274666	1.147784	-0.181455	0.540731	0.719526	0.379249
40416	-0.368338	0.164615	-12.271605	-4.351209	0.140651	-1.108491
41530	-0.697158	-1.079556	-4.909465	-6.999382	1.955673	-9.361176
41533	-1.588081	2.841875	-11.312828	-4.777701	2.097348	20.803344
42307	1.192764	-1.543676	7.360210	-13.936646	3.405169	3.012580
43355	-1.660357	-0.666690	6.373293	2.476313	-1.536640	-3.806103
43398	-0.023393	-1.189589	1.031542	-3.553737	0.342812	-0.967060
43973	-0.501078	-1.238093	5.006974	-2.085362	-0.377101	-0.497721
44257	-0.354890	0.586068	-8.171636	-3.518915	1.284303	1.465411
44475	-3.474204	1.510600	-16.064850	-0.894563	-2.218606	-3.232467
44503	-4.927558	1.598092	1.433457	1.546427	-0.848811	-0.308660
44641	-4.741303	0.584626	0.996745	-2.336111	0.972755	1.241866
45215	-0.987806	1.455528	-7.039888	-2.791355	-1.884608	-6.469437
45660	-0.213730	4.801123	-18.292308	-6.823117	3.263958	18.946734
46402	0.265946	-0.790931	13.530721	4.293728	-2.389004	4.259335
46495	-0.369458	3.107986	-10.069499	-4.061621	1.044273	-4.415716
47329	1.257379	2.008145	2.454553	-0.269048	0.988144	7.040028
47401	-0.739444	-2.004658	-8.817441	-2.719498	0.876382	-2.041865
47483	2.745105	-2.251865	16.178535	4.413073	-0.491620	17.297845
48689	-1.968044	5.501747	-54.497720	-21.620120	5.712303	-1.581098

49333	-0.267261	1.994190	-11.301771	-4.307449	-1.244117	-13.118217
49716	-0.737788	0.394931	-6.026993	-10.224747	2.715479	-0.446999
50384	-7.784042	3.204309	0.563869	2.427460	0.692667	0.020305
50447	-7.269905	0.623797	-1.376298	2.761157	-0.266162	-0.412861
51249	-6.792638	2.069377	-0.085501	-2.089610	1.745315	1.376816
51504	-4.921612	1.188204	1.592754	1.754608	-1.466115	-0.856779
52067	0.319587	5.591971	-15.448986	-5.414098	3.688960	11.360879
53009	-5.494928	-0.410481	1.493775	1.646518	-0.278485	-0.664841
53073	2.270138	-0.017753	-0.044882	-0.617730	-3.400874	-1.185750
53361	0.601035	-4.353679	19.746453	5.198718	-7.331078	-32.828995
53559	-1.436928	2.030912	-9.197412	-4.801821	1.268041	7.810062
53930	-4.466360	0.078270	3.203513	-0.603248	-1.689064	-0.965298
54937	-1.082387	2.496750	-14.750416	-5.937090	2.698953	11.366755
55218	1.068561	-1.026947	4.444840	-10.464876	4.080214	2.287590
55254	-0.325435	-0.090950	0.966506	-1.023572	-0.331756	9.616935
56219	-0.132174	4.606007	-22.838548	-8.037544	3.258447	15.879421
56494	-5.366660	1.661719	1.299638	1.393142	-0.543608	-0.317855
56787	0.530599	-0.941133	-4.516221	22.614889	-8.593642	3.787713

	V24	V25	V26	V27	V28	Amount	Class	\
40	1.021715	-1.868893	1.133050	2.236081	-1.378523	2031.02	0	
550	-1.671991	0.284759	0.123243	-0.123494	0.580233	3.68	0	
653	0.452444	2.410307	0.580394	-1.363536	15.632689	1417.29	0	
1847	0.144011	2.024388	-0.204214	1.332153	0.385891	99.99	1	
2271	-0.142470	1.436109	-0.035261	0.524327	0.882698	3.82	0	
3060	-0.503722	-0.310933	-0.163986	1.197895	0.378187	0.83	1	
3801	0.478279	0.157771	0.329901	0.163504	-0.485552	118.30	1	
4098	1.315177	0.274091	0.189539	-6.007981	14.929133	2074.69	0	
5225	1.151152	-2.977135	0.481818	7.138240	-7.756345	593.48	0	
5395	-0.534120	0.489866	0.228191	1.152759	0.156205	10.76	0	
6562	0.143386	1.611577	-0.221576	1.481233	0.438125	99.99	1	
6873	-0.017298	-0.003155	-0.302673	2.074226	0.662986	89.99	0	
7021	-2.241583	-0.277317	-1.262543	-1.474150	0.499481	4738.79	0	
7811	-0.336511	1.852675	0.458192	-1.351597	0.535306	2476.70	0	
8359	0.133337	-0.843226	0.491554	1.956495	-4.709215	3307.14	0	
8992	-1.163202	1.156147	0.022968	3.416390	-2.723858	1.18	0	
10056	0.531372	-0.030760	0.429433	6.211230	-2.298957	5239.50	0	
10098	-0.432534	-0.821609	-0.377180	-0.264976	0.957384	1.00	0	
10329	1.639105	-5.785209	-1.299990	0.574072	1.545369	3804.63	0	
10593	0.381587	-1.413417	-1.023078	-2.634761	-0.463931	1.00	1	
11322	-0.053502	0.252405	-0.657488	-0.827136	0.849573	59.00	1	
11331	-1.159454	1.903526	0.386401	2.838684	-2.941030	1.18	0	
11618	-0.540228	0.573614	0.235687	1.132372	0.151097	13.98	0	
13344	-0.545888	-0.258922	0.221568	-3.085763	6.403049	1729.79	0	
13627	0.018155	0.668582	-0.281572	1.814857	0.576307	89.99	0	
15033	-0.548504	-0.174617	0.406703	-0.402339	-0.882886	0.00	1	
15273	0.173672	2.184576	0.871416	6.081676	-8.187460	212.00	0	

15641	-0.817178	-0.248115	-0.044372	-0.516305	0.272880	1.98	0
16206	-0.827462	-0.096758	-0.030586	-0.559027	0.269650	1.98	0
16782	-0.245862	0.338238	0.032271	-1.508458	0.608075	0.00	1
19438	1.322373	-1.400529	-0.670069	0.869406	-3.491490	145.52	0
20290	0.382801	0.447154	-0.632816	-4.380154	-0.467863	1.00	1
20569	0.001286	0.394912	-0.287440	1.931604	0.611406	89.99	0
21337	1.120075	-5.785255	-0.676433	4.139387	-0.798326	4111.00	0
21355	0.726929	-1.083765	-0.266994	3.833140	-6.098303	1359.76	0
22484	-0.355519	0.353634	1.042458	1.359516	-0.272188	0.00	1
22767	0.002922	-7.495741	-0.376964	1.811647	1.056891	8.94	0
23988	-0.021378	4.513681	0.897265	3.001875	-2.124633	2.28	0
25017	0.117028	1.297994	-0.224825	1.621052	0.484614	99.99	1
26225	0.031964	0.886971	-0.276769	1.723108	0.547535	89.99	0
26589	-0.379574	0.718717	1.111151	1.277707	0.819081	512.25	1
26780	0.028497	0.832399	-0.277975	1.745969	0.554760	89.99	0
26968	-0.568731	0.582825	-0.042583	0.951130	0.158996	0.83	1
28331	0.371951	-1.415639	-0.517022	-0.434621	0.292721	97.00	1
28996	0.673209	0.226598	-0.006168	-1.185696	-0.747361	59.68	1
28999	-1.047184	-0.035573	0.664900	2.122796	-1.416741	1.00	1
29604	-0.620072	0.642531	0.280717	-2.649107	0.533641	1.00	1
31088	1.163305	-0.529307	-0.073581	4.538395	-8.233983	1.00	0
31138	-0.132711	1.285922	-0.048866	0.566245	0.886527	3.82	0
31455	0.090460	-0.407827	-0.098209	-0.565530	0.724305	6.28	0
31881	2.057019	-1.166944	0.156381	10.507884	-4.127442	2310.00	0
31896	0.432377	-0.435628	0.650893	1.693608	0.857685	8.54	1
32514	-0.238222	-2.280457	2.680182	5.759754	-1.657244	102.24	0
32549	-0.217565	-0.138776	-0.424453	-1.002041	0.890780	1.10	1
32690	0.751013	0.759148	1.070889	2.412802	-0.853621	4627.10	0
33476	-1.106439	3.410742	-0.971861	-2.402525	0.626452	7541.70	0
33548	0.406265	0.106593	-0.026232	-1.464630	-0.411682	78.00	1
34450	0.472670	-0.275998	0.282435	0.104886	0.254417	316.06	1
36642	0.143893	2.343341	-0.211100	1.129057	0.377602	89.99	0
38276	-0.955293	0.023609	0.977815	3.053391	-2.129780	4726.30	0
39921	-0.339450	0.394096	1.075295	1.649906	-0.394905	252.92	1
40295	-0.616962	-0.442811	0.359841	-2.651825	0.422184	1.00	1
40416	-0.032221	0.082385	0.914932	5.480808	-8.464609	247.00	0
41530	1.365381	-3.574515	1.082744	3.949963	-0.496934	2717.00	0
41533	0.290511	5.826159	0.578662	6.987314	-3.666456	998.19	0
42307	-0.822321	-0.172436	-0.037479	-0.537669	0.271266	1.98	0
43355	-0.072090	-1.885646	-0.448436	0.765828	-1.794908	152.00	0
43398	-0.268445	1.533023	-0.615259	-2.546234	3.042055	2.37	0
43973	-0.129580	0.594289	0.243699	-0.131943	-2.501568	0.77	0
44257	-1.591631	1.066891	-0.481870	-3.237295	5.665833	513.46	0
44475	-0.431582	0.745492	-0.180440	3.296722	-1.070156	458.97	0
44503	0.011331	0.559247	-0.283945	1.861154	0.590515	89.99	0
44641	-1.051086	0.038009	0.672317	2.108471	-1.421243	1.00	1
45215	0.873380	-0.626474	0.554730	4.279139	-1.869447	5.35	0

45660	0.559350	5.521140	1.766634	8.708972	-5.688681	390.65	0
46402	0.731504	1.704721	-0.393755	-0.374638	-4.601959	2160.12	0
46495	-0.929446	0.180019	0.436373	-3.420042	6.649828	1580.56	0
47329	0.347693	2.520869	2.342495	3.478175	-2.713136	10199.44	0
47401	0.603670	0.321988	1.268208	5.868242	-4.071666	6239.54	0
47483	-0.503495	1.939396	-1.049141	-2.457912	0.623320	6950.51	0
48689	4.584549	4.554683	3.415636	31.612198	-15.430084	25691.16	0
49333	-0.227553	-1.708349	-0.310107	-5.388598	5.823423	1593.37	0
49716	1.183879	-0.526388	-0.065579	4.567813	-8.257218	1.00	0
50384	0.499809	0.467594	0.483162	1.195671	0.198294	88.23	1
50447	0.519952	-0.743909	-0.167808	-2.498300	-0.711066	30.31	1
51249	-0.554271	-1.610741	0.153725	1.212477	-1.869290	188.78	1
51504	0.125777	1.402587	-0.223755	1.574249	0.469201	99.99	1
52067	0.771200	2.274458	1.954516	6.507171	-4.075417	6.37	0
53009	-1.164555	1.701796	0.690806	2.119749	1.108933	1.00	1
53073	1.067914	0.195895	-0.402495	5.866955	-6.390338	6652.89	0
53361	0.118986	-8.696627	-1.778061	-0.519786	2.716716	10000.00	0
53559	-0.395547	1.002443	0.315886	2.336609	2.900826	3758.25	0
53930	0.131674	2.319682	-0.237727	0.964016	0.456214	1.00	0
54937	0.379077	4.828097	0.710186	3.772529	4.615165	779.86	0
55218	-0.513223	-0.919724	-0.556700	-2.652242	-0.140218	1.50	0
55254	0.504795	2.681358	-0.263591	3.414929	-2.455350	7583.32	0
56219	0.665994	7.519589	0.671345	-3.829039	22.620072	102.00	0
56494	-0.015527	1.178955	-0.205649	1.911967	0.950716	1.00	0
56787	-0.544336	0.644307	0.241288	1.106555	0.151340	9.98	0

	predicted_class	scores
40	1	-0.027755
550	1	-0.008717
653	1	-0.148085
1847	1	-0.086519
2271	1	-0.041258
3060	1	-0.005967
3801	1	-0.077947
4098	1	-0.139969
5225	1	-0.077530
5395	1	-0.037322
6562	1	-0.074142
6873	1	-0.054243
7021	1	-0.036494
7811	1	-0.036823
8359	1	-0.044829
8992	1	-0.014325
10056	1	-0.084567
10098	1	-0.112992
10329	1	-0.079310
10593	1	-0.141555

11322	1	-0.013302
11331	1	-0.070257
11618	1	-0.037539
13344	1	-0.033780
13627	1	-0.055662
15033	1	-0.022082
15273	1	-0.013180
15641	1	-0.031826
16206	1	-0.051728
16782	1	-0.005036
19438	1	-0.021674
20290	1	-0.172568
20569	1	-0.050838
21337	1	-0.004224
21355	1	-0.015272
22484	1	-0.014335
22767	1	-0.087774
23988	1	-0.025473
25017	1	-0.070863
26225	1	-0.070476
26589	1	-0.034864
26780	1	-0.070208
26968	1	-0.010726
28331	1	-0.117571
28996	1	-0.096662
28999	1	-0.110220
29604	1	-0.007625
31088	1	-0.085065
31138	1	-0.012516
31455	1	-0.105730
31881	1	-0.188230
31896	1	-0.117639
32514	1	-0.096640
32549	1	-0.009469
32690	1	-0.074585
33476	1	-0.144849
33548	1	-0.114272
34450	1	-0.106214
36642	1	-0.126800
38276	1	-0.032991
39921	1	-0.029157
40295	1	-0.003711
40416	1	-0.057998
41530	1	-0.012065
41533	1	-0.118588
42307	1	-0.040108
43355	1	-0.124634

```
43398               1 -0.028187
43973               1 -0.010135
44257               1 -0.006270
44475               1 -0.021204
44503               1 -0.050251
44641               1 -0.116630
45215               1 -0.019425
45660               1 -0.141320
46402               1 -0.073625
46495               1 -0.011979
47329               1 -0.188818
47401               1 -0.108425
47483               1 -0.117448
48689               1 -0.315070
49333               1 -0.098369
49716               1 -0.063258
50384               1 -0.084965
50447               1 -0.123608
51249               1 -0.127916
51504               1 -0.069015
52067               1 -0.072810
53009               1 -0.139539
53073               1 -0.089837
53361               1 -0.184853
53559               1 -0.059244
53930               1 -0.134571
54937               1 -0.092611
55218               1 -0.020347
55254               1 -0.117467
56219               1 -0.187192
56494               1 -0.074659
56787               1 -0.041819
```

[ ]: