ML_models

January 3, 2021

1 ML model techniques

[1]:
```
#importing necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
dataframe=pd.read_csv(r"Financial/train_data.csv")
```

[3]:
```
dataframe.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]
```

2 Exploratory Data Analysis

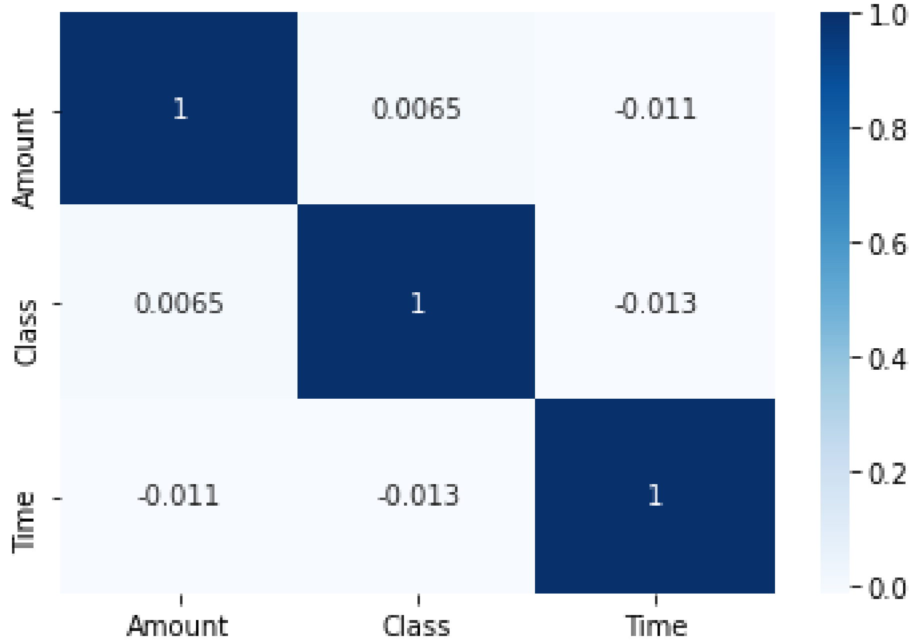
[4]:
```
#checking for standard deviation
dataframe.std()
```

[4]:
```
Time      47500.410602
V1            1.963028
V2            1.661178
V3            1.516107
V4            1.415061
V5            1.367074
V6            1.325341
V7            1.220384
V8            1.192648
V9            1.097367
V10           1.087268
V11           1.021904
V12           0.999581
V13           0.995449
V14           0.959575
V15           0.916011
V16           0.875795
V17           0.851222
V18           0.838685
V19           0.812614
V20           0.772535
V21           0.734187
V22           0.724544
V23           0.625165
V24           0.606012
V25           0.521348
V26           0.482314
V27           0.400286
V28           0.331184
Amount      248.100141
Class         0.041548
dtype: float64
```

[5]:
```
#checking for correlation between Amount,Class and Time
df=dataframe.iloc[:,-2:]
df=df.join(dataframe["Time"])
sns.heatmap(df.corr(),annot=True,cmap="Blues")
print(df.corr())
```

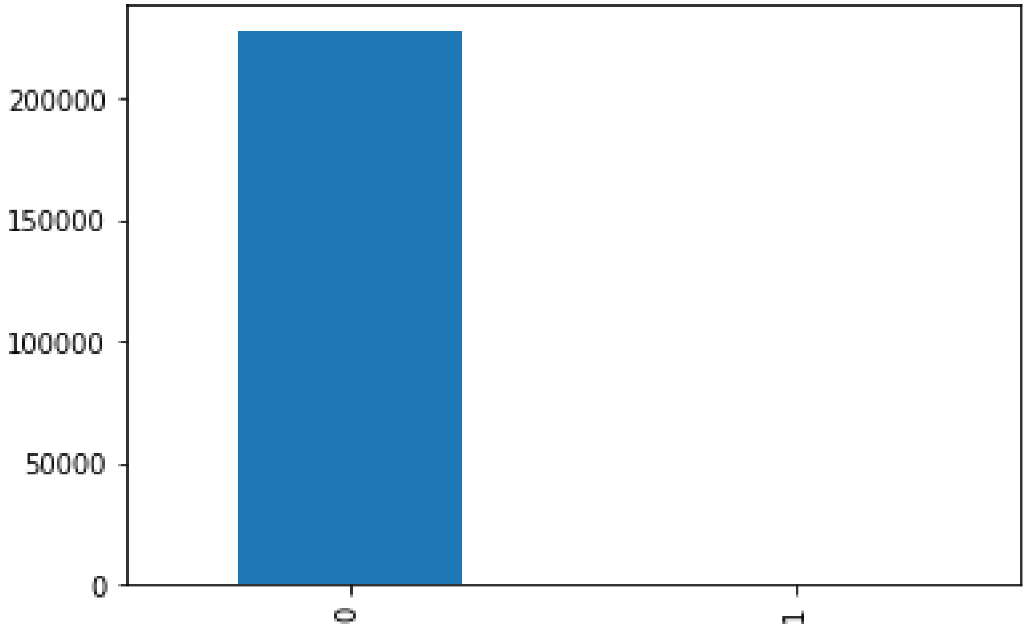
```
          Amount     Class      Time
Amount  1.000000  0.006506 -0.011328
Class   0.006506  1.000000 -0.012800
```

```
Time   -0.011328 -0.012800  1.000000
```



[6]:
```
#checking for type of dataset Balaned or Imbalanced
X_imb=dataframe.iloc[:,0:-1]
Y_imb=dataframe.iloc[:,-1]
Y_imb.value_counts().plot.bar()
print(Y_imb.value_counts())
```

```
0    227451
1       394
Name: Class, dtype: int64
```

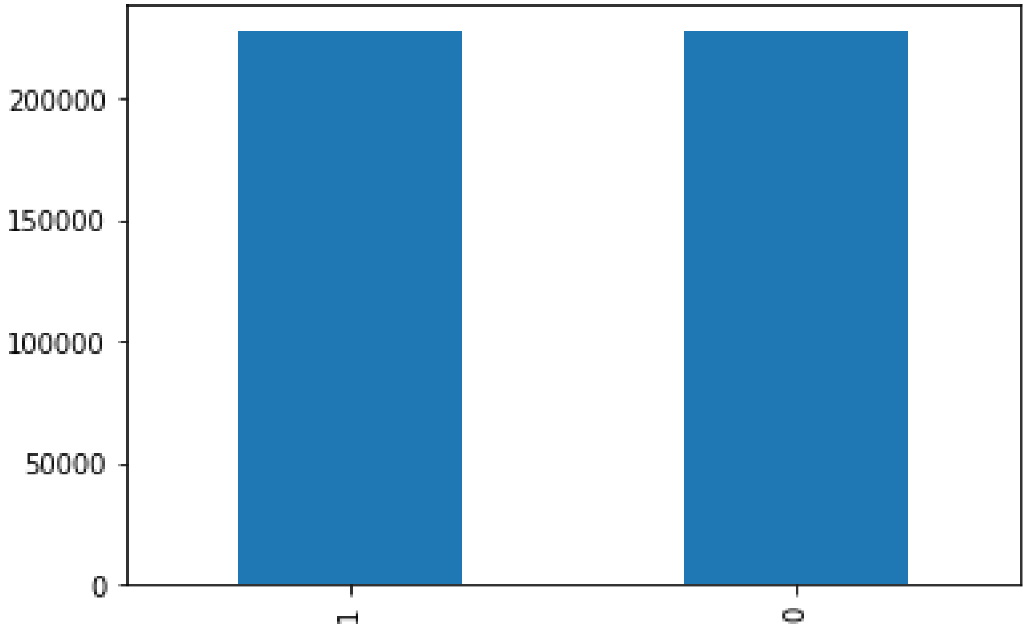


[7]:
```
#Upscaling dataset
from imblearn.over_sampling import SMOTE
smk=SMOTE()
X,Y=smk.fit_sample(X_imb,Y_imb)
print(print(X_imb.shape,Y_imb.shape))
print(X.shape,Y.shape)
print(Y.value_counts())
Y.value_counts().plot.bar()
```

```
Using TensorFlow backend.
```

```
(227845, 30) (227845,)
None
(454902, 30) (454902,)
1    227451
0    227451
Name: Class, dtype: int64
```

[7]: <AxesSubplot:>



3 Modelling Techniques

[8]:
```
#Importing Test Dataset
Test_data=pd.read_csv(r"Financial/test_data_hidden.csv")
X_train,Y_train=X,Y
X_test,Y_test=Test_data.iloc[:,0:-1],Test_data.iloc[:,-1]
```

[9]:
```
pd.DataFrame(X_train)
```

[9]:
```
                 Time          V1        V2         V3         V4         V5  \
0        38355.000000    1.043949  0.318555   1.045810   2.805989  -0.561113
1        22555.000000   -1.665159  0.808440   1.805627   1.903416  -0.821627
2         2431.000000   -0.324096  0.601836   0.865329  -2.138000   0.294663
3        86773.000000   -0.258270  1.217501  -0.585348  -0.875347   1.222481
4       127202.000000    2.142162 -0.494988  -1.936511  -0.818288  -0.025213
…                 …           …         …          …          …          …
454897   93883.390713  -12.776379  7.324013 -17.364823  10.376465 -11.444020
454898  139850.607115   -1.043213 -0.617534  -2.982536   0.413148   0.919739
454899  158699.950195   -5.762649 -6.854235  -5.428864   5.146874   4.522205
454900   93885.053560  -12.149664  7.059878 -17.187042   9.271551 -10.772584
454901   47303.846632   -0.652141  1.198492   0.887867   1.223396   0.899199

              V6         V7        V8        V9  …       V20       V21  \
0      -0.367956   0.032736 -0.042333 -0.322674  … -0.084556 -0.240105
```

```
1       0.934790  -0.824802  0.975890  1.747469  … -0.373759 -0.335332
2      -1.251553   1.072114 -0.334896  1.071268  … -0.039868  0.012220
3      -0.311027   1.073860 -0.161408  0.200665  …  0.382305 -0.424626
4      -1.027245  -0.151627 -0.305750 -0.869482  …  0.106592  0.010115
…           …         …         …         …  …         …         …
454897 -3.445067 -14.752594  8.045089 -5.562879  … -1.166358  2.466291
454898 -0.901843  -0.376107 -0.350463  0.075369  … -0.192415  1.393949
454899 -5.463446  -4.242289  0.864730 -1.155935  …  3.052142  1.431798
454900 -3.932420 -14.882749  7.138916 -5.061796  … -0.875352  2.339961
454901 -0.361432   0.652013  0.093373 -0.979738  … -0.137472  0.005726

             V22       V23       V24       V25       V26       V27       V28  \
0      -0.680315  0.085328  0.684812  0.318620 -0.204963  0.001662  0.037894
1      -0.510994  0.035839  0.147565 -0.529358 -0.566950 -0.595998 -0.220086
2       0.352856 -0.341505 -0.145791  0.094194 -0.804026  0.229428 -0.021623
3      -0.781158  0.019316  0.178614 -0.315616  0.096665  0.269740 -0.020635
4       0.021722  0.079463 -0.480899  0.023846 -0.279076 -0.030121 -0.043888
…           …         …         …         …         …         …         …
454897 -0.280985 -0.255514  0.544748 -0.682898 -0.210926 -2.234725 -0.627462
454898  1.829746  0.495892  0.054918 -0.518278 -0.104928  0.451686  0.083999
454899 -0.143725  0.814622 -0.278181 -1.038293 -0.679673  0.576835 -0.984923
454900  0.173664 -0.503640  0.537976 -1.032623 -0.243717 -1.815792 -0.436306
454901 -0.034321 -0.122503 -0.157211 -0.050876 -0.284419 -0.027712  0.072404

            Amount
0        49.670000
1        16.940000
2         1.000000
3        10.780000
4        39.960000
…              …
454897   31.648289
454898  185.907282
454899  286.496729
454900   20.452786
454901    3.766716

[454902 rows x 30 columns]
```

[10]:
```python
#creating a function to fit the models and display the results
def model_training(lists,X,Y,X_test,Y_test):
    from sklearn.linear_model import LogisticRegression
    from sklearn import svm
    from sklearn.naive_bayes import GaussianNB
    import xgboost
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import accuracy_score,f1_score,roc_auc_score
```

```
    Accuracy_score=[]
    Auc_score=[]
    F1_score=[]
    models=[]
    for k,v in lists.items():
        models.append(k)
        clf=v()
        #print(clf)
        clf.fit(X,Y)
        print("fitted",k)
        Y_pred=clf.predict(X_test)
        Accuracy_score.append(accuracy_score(Y_test,Y_pred))
        F1_score.append(f1_score(Y_test,Y_pred))
        Auc_score.append(roc_auc_score(Y_test,Y_pred))
    dic={'Models':models,"Accuracy":Accuracy_score,"f1_score":
↪F1_score,"AUC_score":Auc_score}
    return pd.DataFrame(dic)
```

[11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
import xgboost
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,f1_score,roc_auc_score
```

[49]:
```
#standardizing the upscaled balanced training data sets
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

[49]:
```
array([[ 1.13050000e+05,  1.14697289e-01,  7.96303317e-01, …,
         2.42724347e-01,  8.57125564e-02,  8.90000000e-01],
       [ 2.66670000e+04, -3.93177045e-02,  4.95783912e-01, …,
         1.13135574e-01,  2.56835743e-01,  8.50000000e+01],
       [ 1.59519000e+05,  2.27570609e+00, -1.53150791e+00, …,
         1.41970058e-02, -5.12892486e-02,  4.27000000e+01],
       …,
       [ 1.38634000e+05,  2.20686736e+00, -7.48559259e-01, …,
        -4.62001550e-02, -7.25858535e-02,  1.00000000e+00],
       [ 5.39070000e+04,  1.43057893e+00, -8.42353562e-01, …,
         3.70949313e-02,  2.91797404e-02,  3.00000000e+01],
       [ 6.63730000e+04, -7.79271230e+00,  5.59993720e+00, …,
         1.66156929e-01,  9.96801592e-02,  8.39000000e+00]])
```

[13]:
```
#creating dictionary of the models and training the models using the balanced␣
↪data set

#print(X_train.shape,Y_train.shape)
#print(X_test.shape,Y_test.shape)
model_list={'Logistic':LogisticRegression,"Naive Bayes":GaussianNB,'SVM':svm.
↪SVC,"Random Forest":RandomForestClassifier,'XGboost':xgboost.XGBClassifier}
result=model_training(model_list,X_train,Y_train,X_test,Y_test)
result
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
fitted Logistic
fitted Naive Bayes
fitted SVM
fitted Random Forest
fitted XGboost
```

[13]:

	Models	Accuracy	f1_score	AUC_score
0	Logistic	0.990643	0.241821	0.929101
1	Naive Bayes	0.976827	0.108108	0.896715
2	SVM	0.997683	0.538462	0.891881
3	Random Forest	0.999526	0.854054	0.902991
4	XGboost	0.999421	0.830769	0.913125

[18]:
```
# training the models and displaying the results using imbalanced and␣
↪non-standardized dataset
#Importing Test Dataset
Test_data=pd.read_csv(r"Financial/test_data_hidden.csv")
X_test_imb,Y_test_imb=Test_data.iloc[:,0:-1],Test_data.iloc[:,-1]
imb_list={"Random Forest":RandomForestClassifier,'XGboost':xgboost.
↪XGBClassifier}
result_imb=model_training(imb_list,X_imb,Y_imb,X_test_imb,Y_test_imb)
result_imb
```

```
fitted Random Forest
fitted XGboost
```

[18]:

	Models	Accuracy	f1_score	AUC_score
0	Random Forest	0.999456	0.820809	0.862210
1	XGboost	0.999508	0.839080	0.872423

[19]:

```
#comparing ensemble moedels with non-ensemble models with non-standardised␣
↪imbalanced dataset
model_list={'Logistic':LogisticRegression,"Naive Bayes":GaussianNB,'SVM':svm.
↪SVC,"Random Forest":RandomForestClassifier,'XGboost':xgboost.XGBClassifier}
result_imb=model_training(model_list,X_imb,Y_imb,X_test_imb,Y_test_imb)
result_imb
```

```
/usr/local/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:940:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
fitted Logistic
fitted Naive Bayes
fitted SVM
fitted Random Forest
fitted XGboost
```

[19]:

	Models	Accuracy	f1_score	AUC_score
0	Logistic	0.998964	0.677596	0.816124
1	Naive Bayes	0.992872	0.225191	0.797793
2	SVM	0.998280	0.000000	0.500000
3	Random Forest	0.999456	0.818713	0.857116
4	XGboost	0.999508	0.839080	0.872423