Deteccao_de_fraude

April 4, 2023

1 Importando as bibliotecas

[1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import precision_score
from sklearn.model_selection import train_test_split
```

2 Importando os dados

A base de dados foi escolhido no Kaggle no seguinte link (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud)

[2]:
```
df = pd.read_csv('creditcard.csv')
```

2.1 Verificando os dados

Verificando se há dados faltantes

[3]:
```
df
```

[3]:
```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321   \
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
…            …          …          …         …         …         …
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

                V6         V7         V8         V9  …        V21       V22  \
0         0.462388   0.239599   0.098698   0.363787  … -0.018307  0.277838   \
1        -0.082361  -0.078803   0.085102  -0.255425  … -0.225775 -0.638672
2         1.800499   0.791461   0.247676  -1.514654  …  0.247998  0.771679
```

```
3       1.247203  0.237609  0.377436 -1.387024  … -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  … -0.009431  0.798278
…            …         …         …         …  …         …         …
284802 -2.606837 -4.918215  7.305334  1.914428  …  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  …  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  …  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  …  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  …  0.261057  0.643078

              V23       V24       V25       V26       V27       V28  Amount  \
0       -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1        0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2        0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3       -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4       -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
…             …         …         …         …         …         …       …
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
0           0
1           0
2           0
3           0
4           0
…       …
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]
```

O dataset foi tratado com um metodo de redução de dimensionalidade e alterou o nome da coluna para proteger os dados dos clientes

[4]:
```
df.isna().sum()
```

[4]:
```
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
```

Aqui estou inspecionando para ver o balanceamento dos dados

[5]:
```
df.Class.value_counts()
```

[5]:
```
Class
0    284315
1       492
Name: count, dtype: int64
```

Como podemos ver ela é extremamente desbalanceada. É necessário alguma tecnica de imblearn. Alguns metodos são:Gerar dados sinteticos, Oversampling ou undersampling para assim aumentar ou diminuir frequência de uma classe nesse notebook optei por oversampling com geração de dados sinteticos. Eu usei a função SMOTE que gera um oversampling com dados sinteticos.

[6]:
```
ys = df.Class.value_counts().loc[1] / df.Class.value_counts().sum()
```

[7]:
```
ys
```

[7]:
```
0.001727485630620034
```

3 Preparando os dados para o oversampling

[8]:
```
X = df.drop(columns = 'Class')
y = df.Class

X_train, X_val, y_train, y_val = train_test_split(X, y, stratify = df.Class)
```

Aqui importo a biblioteca imblearn e busco a função SMOTE

[9]:
```
from imblearn.over_sampling import SMOTE
```

A função SMOTE deve ser aplicada apenas na parte de treinamento dos dados para treinar o modelo o que é uma fraude ou não

[10]:
```
X_train_over, y_train_over = SMOTE().fit_resample(X_train, y_train)
```

Aqui import o modelo que desejo. Eu usei o RandomForestClassifier ao enves do modelo XGBoost-Classifier por um motivo ele é mais simples que o XGBoost e isso gera menos overfit. Como há poucos é possivel que ocorra um underfit ou overfit no XGBoost

[11]:
```
from sklearn.ensemble import RandomForestClassifier
```

[12]:
```
from sklearn.metrics import f1_score
```

Usei o f1_score pois ele é a média harmonica da precisão e recall assim tendo uma noção do balanceamento das métricas. Nesse projeto é mais importante que o recall seja alto para evitar falsos negativos. Quanto maior o recall mais fraudes são detectadas corretamente e menos falsos negativos ocorrem

[14]:
```
modelo = RandomForestClassifier()
modelo.fit(X_train_over, y_train_over)
pred = modelo.predict(X_val)
pred_proba = modelo.predict_proba(X_val)[:,1]
f1 = f1_score(y_val, pred)
f1
```

[14]:
```
0.8523206751054853
```

[15]:
```
from sklearn.metrics import recall_score
```

Aqui podemos ver que o modelo tem um recall de 82,11% o que torna muito superior a baseline de apenas chutar que não há nenhuma fraude

[16]:
```
resultado = recall_score(y_val, pred)
print(f'{resultado * 100:.2f}')
```

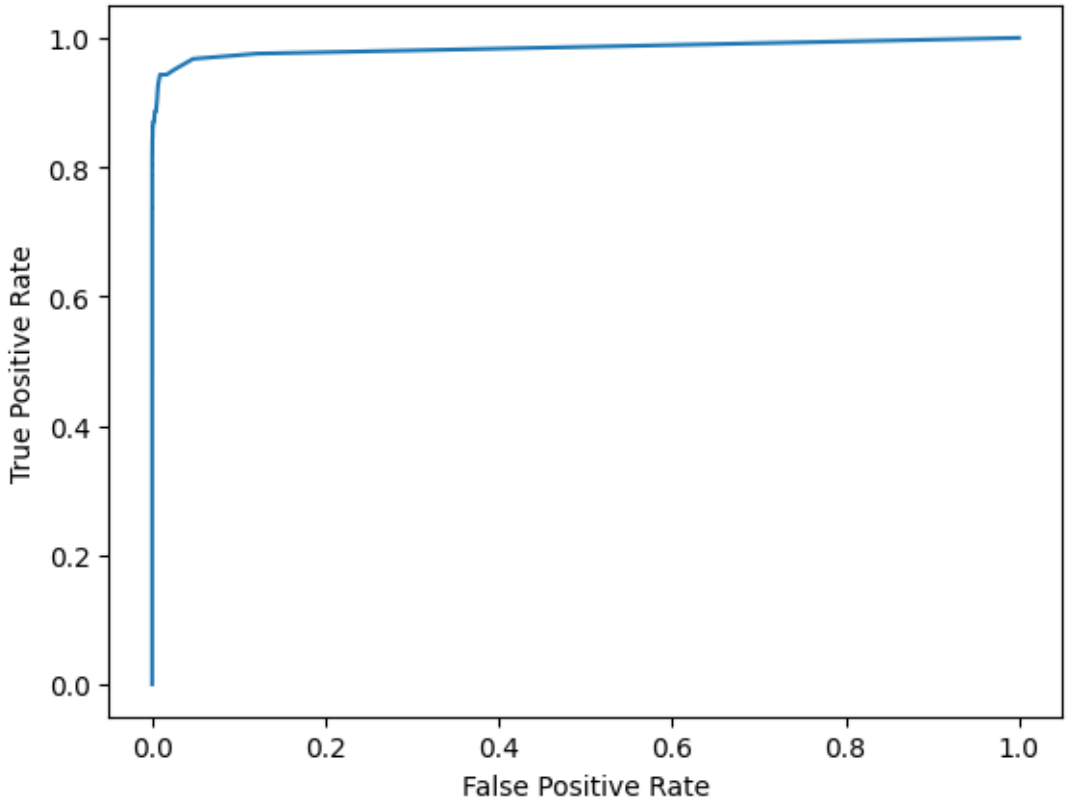
```
82.11
```

4 ROC e AUC score

[17]:
```
from scikitplot.metrics import plot_roc
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve
```

Aqui é usado o roc para visualizar qual é a melhor opção para o modelo. Nesse caso Quanto menor os falsos positivos melhor. Um falso positivo irá bloquear uma fraude de ocorrer.

[18]:
```
fpr, tpr, _ = roc_curve(y_val, pred_proba)
plt.plot(fpr, tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Podemos ver no gráfico que os verdadeiros positivos chegam bem alto quando se inicia um aumento nos falsos positivos. Mas nesse caso é importante minimizar falsos positivos

[19]:
```
from sklearn.metrics import roc_auc_score
```

[21]:
```
roc_auc_score(y_val, pred_proba)
```

[21]:
```
0.9844326426212812
```

[22]:
```
modelo_em_tunning = RandomForestClassifier(n_estimators = 300,␣
↪min_samples_split = 4)
modelo_em_tunning.fit(X_train_over, y_train_over)
preds = modelo_em_tunning.predict(X_val)
pred_proba = modelo_em_tunning.predict_proba(X_val)
roc_auc_score(y_val, pred_proba[:, 1])
```

[22]:
```
0.9870034681438278
```

Mesmo com um leve ajuste dos hiperparametros não houve um ganho substancial no f1_score. Apenas no AUC score

[23]:
```
f1_score(y_val, preds)
```

[23]:
```
0.8595744680851064
```

[24]:
```
recall_score(y_val, preds)
```

[24]:
```
0.8211382113821138
```

5 Conclusão

Com um método básico de imblearn temos um aumento substancial na nosa baseline. Considerando que ela era chutar que não há fraudes, com essa baseline temos uma precisão de 99,87% de precisão e 0% de recall tendo um f1 score de 0 por ter um recall de 0%. Com um tratamento adequado dos dados tem um ganho alto de f1 score e recall em 84,38% no f1 score e no recall de 81,30%. Com um ajuste de hiperparametros com gridsearch poderia ter resultados bem melhores ou até mesmo usando o XGBoost. Mas optei por simplicidade para não ocorrer overfit nem underfit