Fraud Detection

April 3, 2024

[1]:
```
#pip install imbalanced-learn
```

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

[3]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from imblearn.over_sampling import SMOTE
```

[4]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D,Conv2D, Flatten,␣
↪BatchNormalization, Dropout
```

[5]:
```
from sklearn.metrics import classification_report
```

[6]:
```
df = pd.read_csv('/Users/samantarana/Downloads/onlinefraud.csv')
```

[7]:
```
df.head()
```

[7]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1	0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	0

[8]:
```
df.info
```

[8]:
```
<bound method DataFrame.info of          step      type      amount     nameOrig
oldbalanceOrg  \
0           1   PAYMENT     9839.64  C1231006815      170136.00
1           1   PAYMENT     1864.28  C1666544295       21249.00
2           1  TRANSFER      181.00  C1305486145         181.00
3           1  CASH_OUT      181.00   C840083671         181.00
4           1   PAYMENT    11668.14  C2048537720       41554.00
…         …         …           …            …             …
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
…                   …           …               …               …        …
6362615            0.00   C776919290            0.00       339682.13        1
6362616            0.00  C1881841831            0.00            0.00        1
6362617            0.00  C1365125890        68488.84      6379898.11        1
6362618            0.00  C2080388513            0.00            0.00        1
6362619            0.00   C873221189      6510099.11      7360101.63        1

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
4                     0
…                   …
6362615               0
6362616               0
6362617               0
6362618               0
6362619               0

[6362620 rows x 11 columns]>
```

[9]:
```
df.isnull().sum()
```

[9]:
```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

[10]:
```
df.corr
```

[10]:
```
<bound method DataFrame.corr of          step      type      amount     nameOrig
oldbalanceOrg  \
0           1   PAYMENT     9839.64  C1231006815      170136.00
1           1   PAYMENT     1864.28  C1666544295       21249.00
2           1  TRANSFER      181.00  C1305486145         181.00
3           1  CASH_OUT      181.00   C840083671         181.00
4           1   PAYMENT    11668.14  C2048537720       41554.00
…       …       …          …           …             …
6362615   743  CASH_OUT   339682.13   C786484425      339682.13
6362616   743  TRANSFER  6311409.28  C1529008245     6311409.28
6362617   743  CASH_OUT  6311409.28  C1162922333     6311409.28
6362618   743  TRANSFER   850002.52  C1685995037      850002.52
6362619   743  CASH_OUT   850002.52  C1280323807      850002.52

         newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0             160296.36  M1979787155            0.00            0.00        0
1              19384.72  M2044282225            0.00            0.00        0
2                  0.00   C553264065            0.00            0.00        1
3                  0.00    C38997010        21182.00            0.00        1
4              29885.86  M1230701703            0.00            0.00        0
…                   …           …               …               …       …
6362615            0.00   C776919290            0.00       339682.13        1
6362616            0.00  C1881841831            0.00            0.00        1
6362617            0.00  C1365125890        68488.84      6379898.11        1
6362618            0.00  C2080388513            0.00            0.00        1
6362619            0.00   C873221189      6510099.11      7360101.63        1

         isFlaggedFraud
0                     0
1                     0
2                     0
3                     0
```

```
4                       0
…                     …
6362615                 0
6362616                 0
6362617                 0
6362618                 0
6362619                 0

[6362620 rows x 11 columns]>
```

[11]:
```
df.describe()
```

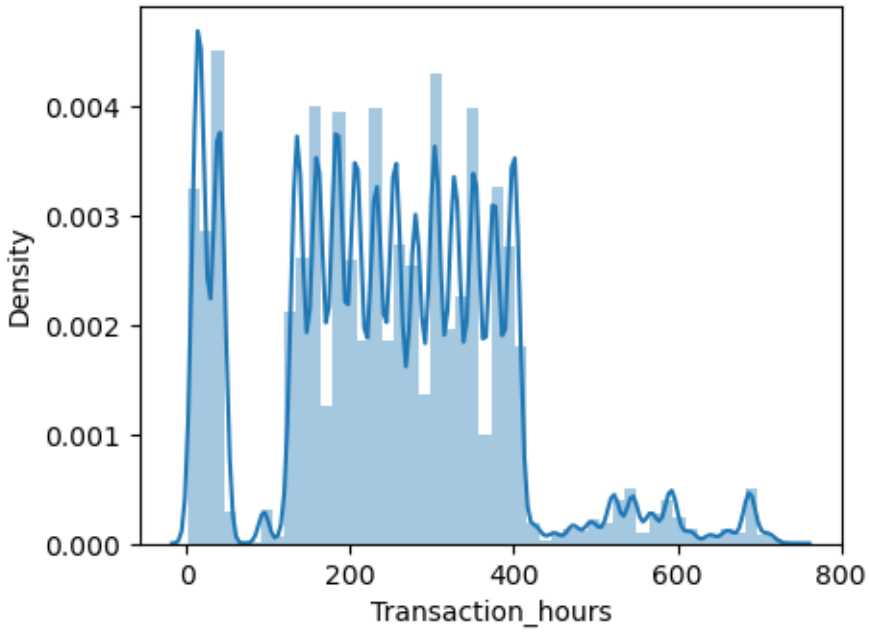
[11]:
```
                step        amount  oldbalanceOrg  newbalanceOrig  \
count  6.362620e+06  6.362620e+06   6.362620e+06    6.362620e+06
mean   2.433972e+02  1.798619e+05   8.338831e+05    8.551137e+05
std    1.423320e+02  6.038582e+05   2.888243e+06    2.924049e+06
min    1.000000e+00  0.000000e+00   0.000000e+00    0.000000e+00
25%    1.560000e+02  1.338957e+04   0.000000e+00    0.000000e+00
50%    2.390000e+02  7.487194e+04   1.420800e+04    0.000000e+00
75%    3.350000e+02  2.087215e+05   1.073152e+05    1.442584e+05
max    7.430000e+02  9.244552e+07   5.958504e+07    4.958504e+07

       oldbalanceDest  newbalanceDest       isFraud  isFlaggedFraud
count    6.362620e+06    6.362620e+06  6.362620e+06    6.362620e+06
mean     1.100702e+06    1.224996e+06  1.290820e-03    2.514687e-06
std      3.399180e+06    3.674129e+06  3.590480e-02    1.585775e-03
min      0.000000e+00    0.000000e+00  0.000000e+00    0.000000e+00
25%      0.000000e+00    0.000000e+00  0.000000e+00    0.000000e+00
50%      1.327057e+05    2.146614e+05  0.000000e+00    0.000000e+00
75%      9.430367e+05    1.111909e+06  0.000000e+00    0.000000e+00
max      3.560159e+08    3.561793e+08  1.000000e+00    1.000000e+00
```

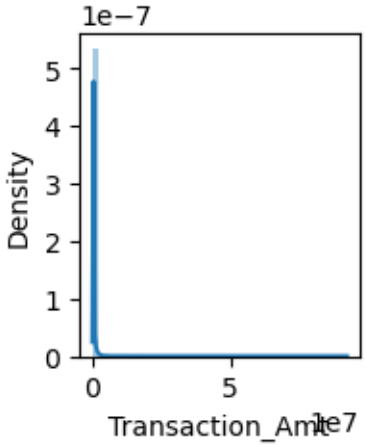
[12]:
```
df.columns = ['Transaction_hours', 'Type', 'Transaction_Amt', 'Sender',␣
 ↪'Sender_Bal_bfr', 'Sender_Bal_aftr', 'Receiver', 'Receiver_Bal_bfr',␣
 ↪'Receiver_Bal_aftr', 'isFraud', 'isFlaggedFraud']
```

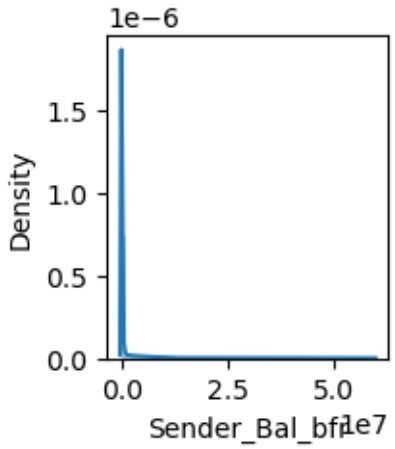
[13]:
```
features = ['Transaction_hours', 'Transaction_Amt', 'Sender_Bal_bfr',␣
 ↪'Sender_Bal_aftr', 'Receiver_Bal_bfr', 'Receiver_Bal_aftr']
```

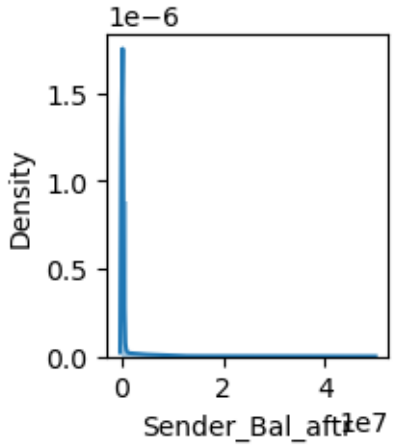
[14]:
```
#Plotting all numerical features for distribution check

plt.figure(figsize=(16,8))
warnings.filterwarnings('ignore')

for feature in features:
    plt.subplot(2,3,features.index(feature)+1)
    sns.distplot(df[feature])
```

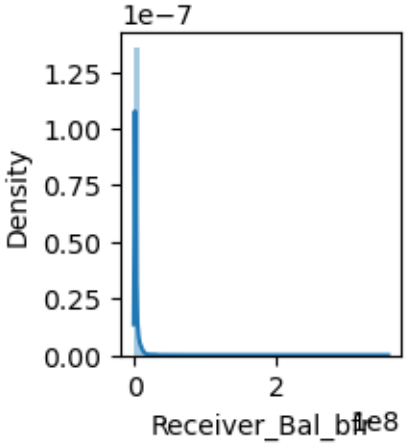
```
plt.show()
```

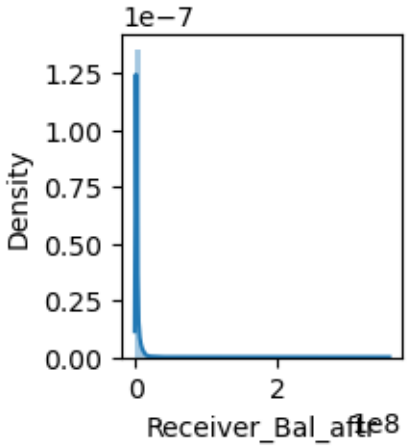

0.004
0.003
0.002
0.001
0.000
Density
0
200
400
600
800
Transaction_hours


1e−7
5
4
3
2
1
0
Density
0
5
1e7
Transaction_Amt


1e−6
1.5
1.0
0.5
0.0
Density
0.0
2.5
5.0
1e7
Sender_Bal_bfr


1e−6
1.5
1.0
0.5
0.0
Density
0
2
4
1e7
Sender_Bal_aft

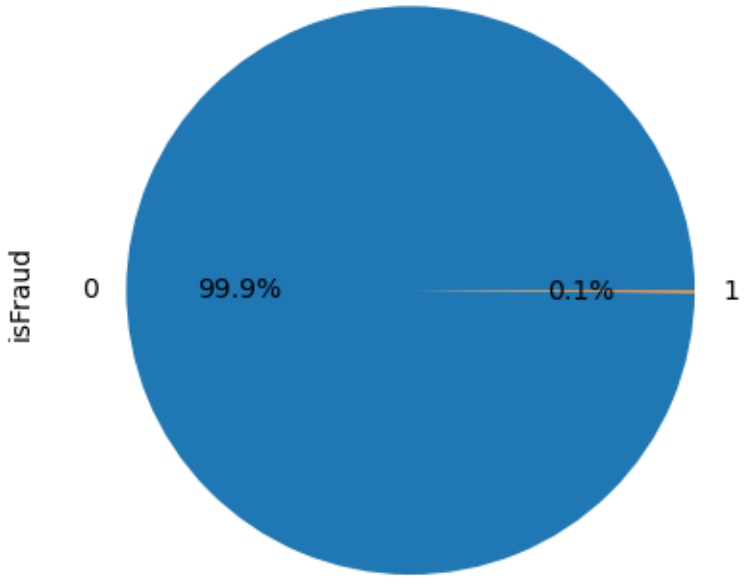

1e−7
1.25
1.00
0.75
0.50
0.25
0.00
Density
0
2
1e8
Receiver_Bal_bf



[15]:
```
df.isFraud.value_counts(normalize=True)
```

[15]:
```
0    0.998709
1    0.001291
Name: isFraud, dtype: float64
```

[16]:
```
df['isFraud'].value_counts().plot.pie(autopct='%1.1f%%');
```



[17]:
```
non_fraud = df[df['isFraud'] == 0]
fraud = df[df['isFraud'] == 1]
```

[18]:
```
df = df.drop(['isFlaggedFraud', 'Sender', 'Receiver'], axis = 1)
```

[19]:
```
df.columns
```

[19]:
```
Index(['Transaction_hours', 'Type', 'Transaction_Amt', 'Sender_Bal_bfr',
       'Sender_Bal_aftr', 'Receiver_Bal_bfr', 'Receiver_Bal_aftr', 'isFraud'],
      dtype='object')
```

[20]:
```
le = LabelEncoder()
df['Type'] = le.fit_transform(df['Type'])
```

[21]:
```
df.head()
```

[21]:

	Transaction_hours	Type	Transaction_Amt	Sender_Bal_bfr	Sender_Bal_aftr	Receiver_Bal_bfr	Receiver_Bal_aftr	isFraud
0	1	3	9839.64	170136.0	160296.36	0.0	0.0	0
1	1	3	1864.28	21249.0	19384.72	0.0	0.0	0
2	1	4	181.00	181.0	0.00	0.0	0.0	1
3	1	1	181.00	181.0	0.00	21182.0	0.0	1
4	1	3	11668.14	41554.0	29885.86	0.0	0.0	0

[22]:
```
X = df.drop(['isFraud'], axis = 1)
```

[23]:
```
Y = df.isFraud
```

[24]:
```
X.shape, Y.shape
```

[24]:
```
((6362620, 7), (6362620,))
```

[25]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.2,␣
↪random_state = 99)
```

[26]:
```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

[26]:
```
((5090096, 7), (1272524, 7), (5090096,), (1272524,))
```

[27]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[28]:
```
Y_train = Y_train.to_numpy()
Y_test = Y_test.to_numpy()
```

[29]:
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1])
```

[30]:
```
print("Before OverSampling, counts of label '1' : {}".format(sum(Y_train==1)))
print("Before OverSampling, counts of label '0' : {}".format(sum(Y_train==0)))
```

```
Before OverSampling, counts of label '1' : 6545
Before OverSampling, counts of label '0' : 5083551
```

[31]:
```
#SMOTE

smote = SMOTE(random_state = 42)
X_res, Y_res = smote.fit_resample(X_train, Y_train.ravel())

X_res = X_res.reshape(X_res.shape[0], X_res.shape[1],1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1],1)
```

[32]:
```
X_res.shape
```

[32]:
```
(10167102, 7, 1)
```

[33]:
```
print("After OverSampling, the shape of train_X: {}".format(X_res.shape))
print("After OverSampling, the shape of train_Y: {}".format(Y_res.shape))
```

```
After OverSampling, the shape of train_X: (10167102, 7, 1)
After OverSampling, the shape of train_Y: (10167102,)
```

[34]:
```
print("After OverSampling, counts of label '1' : {}".format(sum(Y_res==1)))
print("After OverSampling, counts of label '0' : {}".format(sum(Y_res==0)))
```

```
After OverSampling, counts of label '1' : 5083551
After OverSampling, counts of label '0' : 5083551
```

[35]:
```
model = Sequential()

#layers
model.add(Conv1D(filters=32, kernel_size = 2, activation='relu', input_shape =␣
↪X_res[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
```

```
model.add(BatchNormalization())
model.add(Dropout(0.2))
```

[36]:
```
#build ANN
model.add(Flatten())
model.add(Dense(64, activation ='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation = 'relu'))
```

[37]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 6, 32)             96

 batch_normalization (Batch  (None, 6, 32)             128
 Normalization)

 dropout (Dropout)           (None, 6, 32)             0

 conv1d_1 (Conv1D)           (None, 5, 64)             4160

 batch_normalization_1 (Bat  (None, 5, 64)             256
 chNormalization)

 dropout_1 (Dropout)         (None, 5, 64)             0

 flatten (Flatten)           (None, 320)               0

 dense (Dense)               (None, 64)                20544

 dropout_2 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 25249 (98.63 KB)
Trainable params: 25057 (97.88 KB)
Non-trainable params: 192 (768.00 Byte)
_________________________________________________________________
```

[38]:
```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[39]:
```
%%time

#fitting the model

history1 = model.fit(X_res, Y_res, epochs=1, validation_data=(X_test, Y_test))
```

```
317722/317722 [==============================] - 499s 2ms/step - loss: 0.1921 -
accuracy: 0.9570 - val_loss: 0.0862 - val_accuracy: 0.9699
CPU times: user 7min 20s, sys: 3min 3s, total: 10min 24s
Wall time: 8min 19s
```

[40]:
```
#import model
model12 = Sequential()

#layers
model12.add(Conv1D(filters=32, kernel_size = 2, activation='relu', input_shape␣
 ↪= X_res[0].shape))
model12.add(BatchNormalization())
model12.add(Dropout(0.2))

model12.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
model12.add(BatchNormalization())
model12.add(Dropout(0.2))
```

[41]:
```
#build ANN
model12.add(Flatten())
model12.add(Dense(64, activation ='relu'))
model12.add(Dropout(0.5))
model12.add(Dense(1, activation = 'relu'))
```

[42]:
```
# convert model weights to bfloat16 data type

policy = tf.keras.mixed_precision.Policy('mixed_bfloat16')
tf.keras.mixed_precision.set_global_policy(policy)
```

[43]:
```
X_train_bfloat = tf.convert_to_tensor(X_res, dtype = tf.bfloat16)
```

[44]:
```
X_train_bfloat.shape
```

[44]:
```
TensorShape([10167102, 7, 1])
```

[45]:
```
X_test_bfloat = tf.convert_to_tensor(X_test, dtype = tf.bfloat16)
```

[46]:
```
model12.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])
```

[47]:
```
%%time
```

```
#fitting the model

history2 = model12.fit(X_train_bfloat, Y_res, epochs=1, validation_data =␣
 ↪(X_test_bfloat, Y_test))
```

```
317722/317722 [==============================] - 2296s 7ms/step - loss: 0.1810 -
accuracy: 0.9608 - val_loss: 0.1313 - val_accuracy: 0.9275
CPU times: user 7min 28s, sys: 3min 5s, total: 10min 33s
Wall time: 38min 16s
```

0.0.1 With 92% of accuracy rate , we have established a working model for the frad detection !