Fraud Detection using Logistic Regression

December 8, 2022

[1]:
```python
import pandas as pd

df = pd.read_csv('data/card_data.csv')
```

[2]:
```python
df.head()
```

[2]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

[3]:
```python
df['Class'].value_counts()
```

[3]:
```
0    284315
1       492
Name: Class, dtype: int64
```

[4]:
```python
df = df.sample(frac = 1, random_state = 1)
df = df.reset_index(drop = True)
```

```python
df.head()
```

[4]:
```
          Time        V1        V2        V3        V4        V5        V6  \
0  119907.0 -0.611712 -0.769705 -0.149759 -0.224877  2.028577 -2.019887
1   78340.0 -0.814682  1.319219  1.329415  0.027273 -0.284871 -0.653985
2   82382.0 -0.318193  1.118618  0.969864 -0.127052  0.569563 -0.532484
3   31717.0 -1.328271  1.018378  1.775426 -1.574193 -0.117696 -0.457733
4   80923.0  1.276712  0.617120 -0.578014  0.879173  0.061706 -1.472002

         V7        V8        V9  …       V21       V22       V23       V24  \
0  0.292491 -0.523020  0.358468  … -0.075208  0.045536  0.380739  0.023440
1  0.321552  0.435975 -0.704298  … -0.128619 -0.368565  0.090660  0.401147
2  0.706252 -0.064966 -0.463271  … -0.305402 -0.774704 -0.123884 -0.495687
3  0.681867 -0.031641  0.383872  … -0.220815 -0.419013 -0.239197  0.009967
4  0.373692 -0.287204 -0.084482  … -0.160161 -0.430404 -0.076738  0.258708

        V25       V26       V27       V28  Amount  Class
0 -2.220686 -0.201146  0.066501  0.221180    1.79      0
1 -0.261034  0.080621  0.162427  0.059456    1.98      0
2 -0.018148  0.121679  0.249050  0.092516    0.89      0
3  0.232829  0.814177  0.098797 -0.004273   15.98      0
4  0.552170  0.370701 -0.034255  0.041709    0.76      0

[5 rows x 31 columns]
```

[5]:
```python
as_np = df.to_numpy()
index = int(len(as_np) * .92)

X_train, y_train = as_np[:index, :-1], as_np[:index, -1]
X_test, y_test = as_np[index:, :-1], as_np[index:, -1]

(X_train.shape, y_train.shape), (X_test.shape, y_test.shape)
```

[5]:
```
(((262022, 30), (262022,)), ((22785, 30), (22785,)))
```

[6]:
```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

X_test[0]
```

[6]:
```
array([ 0.14097956,  0.53955733, -1.15153973, -0.47041404,  0.57191953,
       -0.85362208, -0.27419086, -0.03159233, -0.25697594,  2.43387034,
       -0.75622807, -0.03956163, -1.77401948,  2.41251471,  1.26340856,
```

```
       -0.24099657,  0.16722599,  0.24463032,  0.56588687, -0.53397987,
        1.02654979,  0.4423462 ,  0.47928779, -0.49696321, -0.14435544,
       -0.64466426,  0.08065479, -0.24695714,  0.11458447,  1.64640304])
```

[7]:
```python
from sklearn.linear_model import LogisticRegression

model = LogisticRegression().fit(X_train, y_train)
test_predictions = model.predict(X_test)

pd.value_counts(test_predictions)
```

[7]:
```
0.0    22757
1.0       28
dtype: int64
```

[8]:
```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, test_predictions, labels = [0, 1])
disp = ConfusionMatrixDisplay(confusion_matrix = cm,
                              display_labels = ['Not Fraud', 'Fraud'])
disp.plot()
```

[8]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x173812d02b0>
```

[9]:
```
tn, fp, fn, tp = cm.ravel()

s = '''
True negatives: {0}
False Positives: {1}
False Negatives: {2}
True Positives: {3}'''.format(tn, fp, fn, tp)

print(s)
```

```
True negatives: 22741
False Positives: 8
False Negatives: 16
True Positives: 20
```

[10]:
```
# accuracy = (tp+tn)/(tp+tn+fp+fn)

def accuracy(tn, fp, fn, tp):
    return ((tp+tn)/(tp+tn+fp+fn))

"Accuracy: {0}".format(accuracy(tn, fp, fn, tp))
```

[10]:
```
'Accuracy: 0.9989466754443713'
```

[11]:
```
# recall = sensitivity = true positive rate = tp/(tp+fn)

def tpr(tn, fp, fn, tp):
    return (tp / (tp + fn))

"True positive Rate: {0}".format(tpr(tn, fp, fn, tp))
```

[11]:
```
'True positive Rate: 0.5555555555555556'
```

[12]:
```
# false negative rate = fn/(tp+fn)

def fpr(tn, fp, fn, tp):
    return (fn / (tp + fn))

"False Negative Rate: {0}".format(fpr(tn, fp, fn, tp))
```

[12]:
```
'False Negative Rate: 0.4444444444444444'
```

[13]:
```
# specificity = true negative rate = tn/(tn+fp)

def tnr(tn, fp, fn, tp):
    return (tn / (tn + fp))
```

```
"True Negative Rate: {0}".format(tnr(tn, fp, fn, tp))
```

[13]: 'True Negative Rate: 0.9996483361906018'

[14]:
```
# false positive rate = fp/(tn+fp)

def fpr(tn, fp, fn, tp):
    return (fp / (tn + fp))

"False Positive Rate: {0}".format(fpr(tn, fp, fn, tp))
```

[14]: 'False Positive Rate: 0.00035166380939821533'

[15]:
```
# precision = positive predictive value = tp/(tp+fp)

def ppv(tn, fp, fn, tp):
    return (tp / (tp + fp))

"Positive Predictive Value: {0}".format(ppv(tn, fp, fn, tp))
```

[15]: 'Positive Predictive Value: 0.7142857142857143'

[16]:
```
# negative predictive value = tn/(tn+fn)

def npv(tn, fp, fn, tp):
    return (tn / (tn + fn))

"Negative Predictive Value: {0}".format(npv(tn, fp, fn, tp))
```

[16]: 'Negative Predictive Value: 0.9992969196291251'

[17]:
```
# balanced accuracy = (tpr+tnr)/2

def balanced_accuracy(tn, fp, fn, tp):
    return (tpr(tn, fp, fn, tp) + tnr(tn, fp, fn, tp)) / 2

"Balanced Accuracy: {0}".format(balanced_accuracy(tn, fp, fn, tp))
```

[17]: 'Balanced Accuracy: 0.7776019458730787'

[18]:
```
# f1 score = 2 x (precision x recall)/(precision + recall)

def f1(tn, fp, fn, tp):
    p = ppv(tn, fp, fn, tp)
    r = tpr(tn, fp, fn, tp)
    return (2*p*r)/(p+r)

"F1 Score: {0}".format(f1(tn, fp, fn, tp))
```

[18]: 'F1 Score: 0.6250000000000001'

[19]:
```
probabilities = model.predict_proba(X_test)[:, 1]

probabilities
```

[19]:
```
array([0.00012139, 0.0003532 , 0.00030147, …, 0.00473659, 0.00017273,
       0.00171865])
```

[20]:
```
pd.value_counts(probabilities > 0.5)
```

[20]:
```
False    22757
True        28
dtype: int64
```

[21]:
```
import numpy as np

thresholds = np.linspace(0, 1, num = 2000).astype(np.float16)

thresholds
```

[21]:
```
array([0.000e+00, 5.002e-04, 1.000e-03, …, 9.990e-01, 9.995e-01,
       1.000e+00], dtype=float16)
```

[22]:
```
all_predictions = np.array([(probabilities > t).astype(int) for t in␣
 ↪thresholds])

all_predictions.shape
```

[22]: (2000, 22785)

[23]:
```
all_predictions[-4]
```

[23]: array([0, 0, 0, …, 0, 0, 0])

[24]:
```
pd.value_counts(all_predictions[-4])
```

[24]:
```
0    22776
1        9
dtype: int64
```

[25]:
```
confusion_matrices = [confusion_matrix(y_test, predictions) for predictions in␣
 ↪all_predictions]
tn_fp_fn_tps = [cm.ravel() for cm in confusion_matrices]

tprs = [tpr(tn, fp, fn, tp) for tn, fp, fn, tp in tn_fp_fn_tps]
fprs = [fpr(tn, fp, fn, tp) for tn, fp, fn, tp in tn_fp_fn_tps]
```

[26]:
```python
import plotly.express as px

px.scatter(x = fprs, y = tprs, color = thresholds, labels = dict(x = 'False␣
↪Positive Rate', y = 'True Positive Rate', color = 'Threshold'), title = 'ROC␣
↪Curve')
```

[27]:
```python
from sklearn.metrics import auc

auc(fprs, tprs)
```

[27]: 0.9810064911278153

[28]:
```python
# classification report

from sklearn.metrics import classification_report

print(classification_report(y_test, test_predictions, labels = [0, 1],␣
↪target_names = ['Not Fraud', 'Fraud']))
```

```
              precision    recall  f1-score   support

   Not Fraud       1.00      1.00      1.00     22749
       Fraud       0.71      0.56      0.63        36

    accuracy                           1.00     22785
   macro avg       0.86      0.78      0.81     22785
weighted avg       1.00      1.00      1.00     22785
```

[29]:
```python
# matthews correlation coefficient = (tp x tn) - (fp x fn) / sqroot((tp + fp) x␣
↪(tp + fn) x (tn + fp) x (tn + fn))
import math as mt

def mcc(tn, fp, fn, tp):
    return (((tp*tn) - (fp*fn))/mt.sqrt((tp + fp)*(tp + fn)*(tn + fp)*(tn +␣
↪fn)))

"Matthews Correlation Coefficient: {0}".format(mcc(tn, fp, fn, tp))
```

[29]: 'Matthews Correlation Coefficient: 0.6294313746803477'

[ ]: