Using PCA to Detect Credit Card Fraud

Author: Benjamin Tobio

Abstract

Credit card fraud is a growing concern for financial institutions and individuals alike. While advanced machine learning methods have recently become popularized to help identify possible fraudulent transactions, PCA can also be a quite useful method due to its easy implementation. While it may not turn out to be as accurate as advanced methods, PCA can be used as a stepping stone or baseline to build models off of and compare to. The goal of this project is to develop a PCA-based credit card fraud detection system in order to try and accurately classify transactions as either legitimate or fraudulent.

Introduction

For this project, I found a dataset on Kaggle with 1,000,000 theoretical credit card transactions. Each data point (transaction) includes seven features along with a binary variable that classifies the transaction as fraudulent or not; this is our target variable. Of the 1,000,000 transactions, 912,597 are classified as non-fraudulent and 87,403 are classified as fraudulent.

Using PCA, the seven features were broken down into a number of principal components. Using those principal components, the data was then transformed back into an "estimated" version of the original features. The difference between each transaction's original features and reconstructed features was then found and squared, we will call this the transaction's score. Finally, an optimized threshold was found and set on the transactions' scores to make predictions on whether or not furture transactions should be considered fraudulent (e.g. a transaction with a score below the threshold is predicted to be fraudulent).

Findings

Principal Component Selection

I first looked at the explained variance ratio in order to choose the number of principal components that the features were going to be broken down into. The results were as follows:

$$0.27480867, 0.27417948, 0.27361244, 0.06236582, 0.06228307, 0.02795915, 0.02479137$$

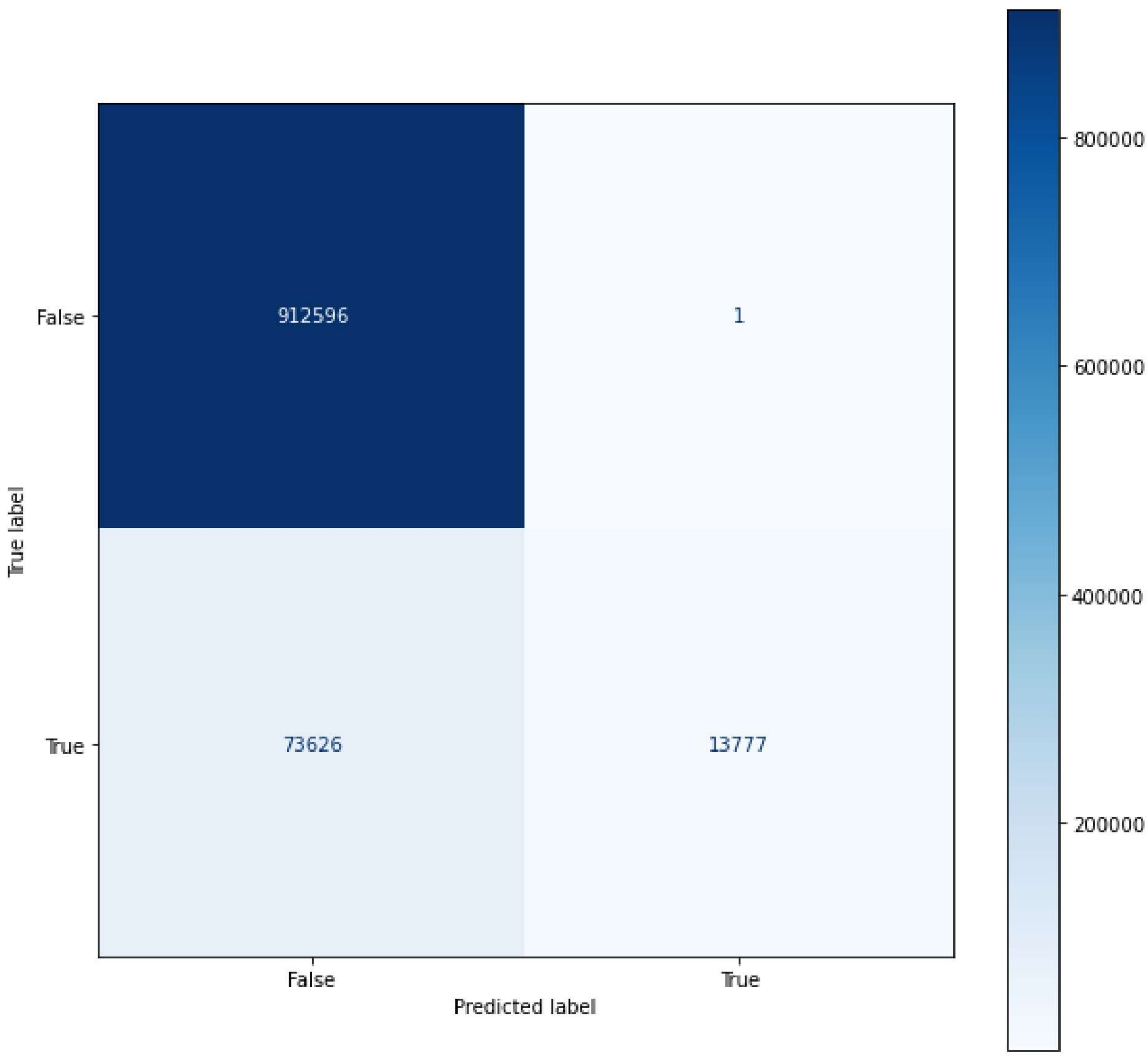
From these results, I decided to use three principal components when implementing PCA.

PCA Implementation and Threshold Optimization

After deciding on three principal components, PCA was implemented and the original features were reconstructed. The scores were then calculated and a threshold of .25847 was found to be optimal. If a transaction has a score below the threshold, it is predicted to be fraudulent. I found that after this threshold, the number of non-fraudulent transaction rapidly increases and would thus diminish the accuracy of the model.

Results

The following confusion matrix shows the model's predictions:



Conclusion

Based on the limited number of features, I believe that the results attained by the PCA model are actualy very good. While utilizing PCA only correctly classifies 13,777 fraudulent transactions out of 87,403 (15.76%), it only misclassifies one non-fraudulent transaction. Credit card companies obviously want to correctly identify as many fraudulent transactions as possible, but it is also important that they do not mis-classify too many non-fraudulent transactions as well. As mentioned before, while PCA may not be as accurate as machine learning models, it can be a valuable baseline to build models off of due to its ease of implementation.

References

https://www.kaggle.com/datasets/dhanushnarayananr/credit-card-fraud?resource=download
https://towardsdatascience.com/detect-anomalies-in-telemetry-data-using-principal-component-analysis-98d6dc4bf843
https://www.atmosera.com/blog/pca-based-anomaly-detection/

Code

In [ ]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDis
```

In [ ]:

```
df = pd.read_csv('card_transdata.csv')
df.head()
```

Out[ ]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	us
0	57.877857	0.311140	1.945940	1.0	
1	10.829943	0.175592	1.294219	1.0	
2	5.091079	0.805153	0.427715	1.0	
3	2.247564	5.600044	0.362663	1.0	
4	44.190936	0.566486	2.222767	1.0	

In [ ]:

```
df['fraud'].value_counts()
```

Out[ ]:

```
0.0    912597
1.0     87403
Name: fraud, dtype: int64
```

Data Scaling

In [ ]:
```
df_scaled = df.copy(deep=False)

cols_to_scale = ['distance_from_home', 'distance_from_last_transaction', 'ratio_to_medi

scaler = StandardScaler()
scaler.fit(df_scaled[cols_to_scale])
df_scaled[cols_to_scale] = scaler.transform(df_scaled[cols_to_scale])
df_scaled.head()
```

Out[ ]:

	distance_from_home	distance_from_last_transaction	ratio_to_median_purchase_price	repeat_retailer	us
0	0.477882	-0.182849	0.043491	1.0	
1	-0.241607	-0.188094	-0.189300	1.0	
2	-0.329369	-0.163733	-0.498812	1.0	
3	-0.372854	0.021806	-0.522048	1.0	
4	0.268572	-0.172968	0.142373	1.0	

Data Splitting

In [ ]:
```
X = df_scaled.drop('fraud', axis=1)
y = df_scaled['fraud']
```

PCA Explained Variance Ratio

In [ ]:
```
pca = PCA(n_components=7)
pca.fit_transform(X)
print(pca.explained_variance_ratio_)
```

```
[0.27480867 0.27417948 0.27361244 0.06236582 0.06228307 0.02795915
 0.02479137]
```

PCA Implementation

In [ ]:
```
pca = PCA(n_components=3)
X_pca = pd.DataFrame(pca.fit_transform(X), index = X.index)
X_restored = pd.DataFrame(pca.inverse_transform(X_pca), index = X_pca.index)
```

In [ ]:
```
def get_anomaly_scores(df_original, df_restored):
    loss = np.round(np.sum((np.array(df_original) - np.array(df_restored)) ** 2, axis=1
    loss = pd.Series(data=loss, index=df_original.index)
    return loss
```

In [ ]:
```
reconstruction_errors = get_anomaly_scores(X, X_restored)
check_df = pd.concat([reconstruction_errors, y], axis = 1).reset_index(drop=True)
```

```
check_df.columns = ['score', 'fraud']
check_df.sort_values('score').head()
```

Out[ ]:

	score	fraud
432691	0.18906	0.0
253435	0.25492	1.0
834905	0.25493	1.0
528776	0.25494	1.0
390707	0.25494	1.0

In [ ]:

```
check_df[check_df['score'] < .25847].groupby('fraud').count()
```

Out[ ]:

	score
fraud	
0.0	1
1.0	13777

In [ ]:

```
check_df['pred'] = np.where(check_df['score'] < .25847, 1.0, 0.0)
check_df.sort_values('score').head()
```

Out[ ]:

	score	fraud	pred
432691	0.18906	0.0	1.0
253435	0.25492	1.0	1.0
834905	0.25493	1.0	1.0
528776	0.25494	1.0	1.0
390707	0.25494	1.0	1.0

In [ ]:

```
print(classification_report(check_df['fraud'], check_df['pred']))
```

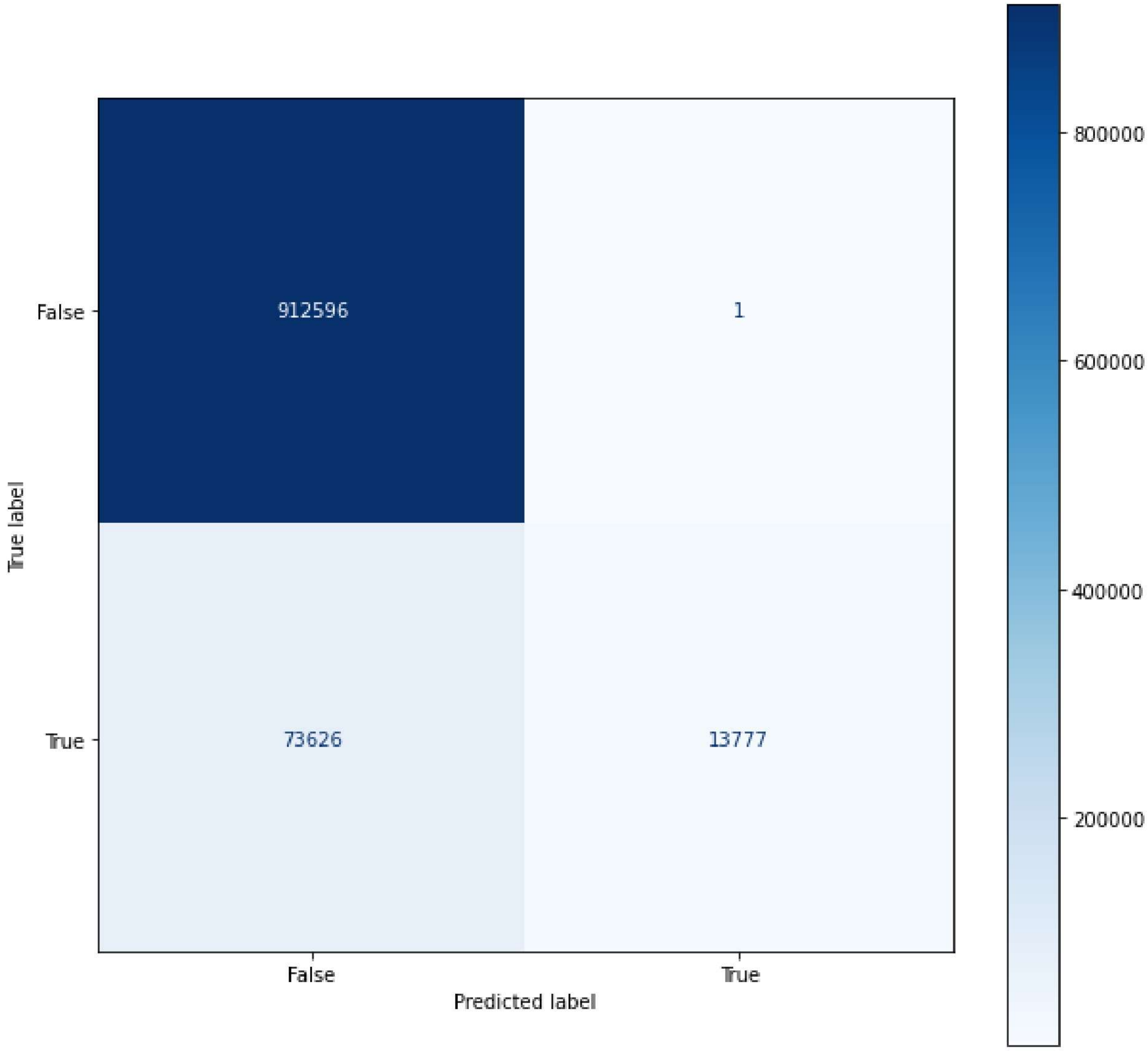
```
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.96    912597
         1.0       1.00      0.16      0.27     87403

    accuracy                           0.93   1000000
   macro avg       0.96      0.58      0.62   1000000
weighted avg       0.93      0.93      0.90   1000000
```

In [ ]:

```
cm = confusion_matrix(check_df['fraud'], check_df['pred'])
cmp = ConfusionMatrixDisplay(cm, display_labels=[False, True])
fig, ax = plt.subplots(figsize=(10,10))
cmp.plot(ax=ax, cmap=plt.cm.Blues)
plt.show()
```



PCA Implementation with train_test_split

I do not talk about this part in the report, but it just further shows the implementation of PCA for fraud detection on the dataset

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5)
```

In [ ]:
```
pca = PCA(n_components=3)
X_pca_train = pd.DataFrame(pca.fit_transform(X_train), index = X_train.index)
X_restored_train = pd.DataFrame(pca.inverse_transform(X_pca_train), index = X_pca_train
pca.explained_variance_ratio_
```

Out[ ]:
```
array([0.28648441, 0.28093876, 0.24848135])
```

In [ ]:
```
reconstruction_errors = get_anomaly_scores(X_train, X_restored_train)
check_df_train = pd.concat([reconstruction_errors, y_train], axis = 1).reset_index(drop
check_df_train.columns = ['score', 'fraud']
check_df_train.sort_values('score').head()
```

Out[ ]:

	score	fraud
443238	0.25516	1.0
103044	0.25521	1.0
492164	0.25524	1.0
332598	0.25525	1.0
240035	0.25525	1.0

In [ ]:

```
check_df_train[check_df_train['score'] < .25847].groupby('fraud').count()
```

Out[ ]:

	score
fraud	
1.0	6883

In [ ]:

```
X_pca_test = pd.DataFrame(pca.transform(X_test), index = X_test.index)
X_restored_test = pd.DataFrame(pca.inverse_transform(X_pca_test), index = X_pca_test.in
```

In [ ]:

```
reconstruction_errors = get_anomaly_scores(X_test, X_restored_test)
check_df_test = pd.concat([reconstruction_errors, y_test], axis = 1).reset_index(drop=T
check_df_test.columns = ['score', 'fraud']
check_df_test.sort_values('score').head()
```

Out[ ]:

	score	fraud
358645	0.25504	1.0
38050	0.25527	1.0
202068	0.25530	1.0
13145	0.25532	1.0
185183	0.25532	1.0

In [ ]:

```
check_df_test['pred'] = np.where(check_df_test['score'] < .25847, 1.0, 0.0)
check_df_test.sort_values('score').head()
```

Out[ ]:

	score	fraud	pred
358645	0.25504	1.0	1.0
38050	0.25527	1.0	1.0
202068	0.25530	1.0	1.0
13145	0.25532	1.0	1.0
185183	0.25532	1.0	1.0

In [ ]:

```
print(classification_report(check_df_test['fraud'], check_df_test['pred']))
```

```
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.96    456401
         1.0       1.00      0.15      0.26     43599

    accuracy                           0.93    500000
   macro avg       0.96      0.58      0.61    500000
weighted avg       0.93      0.93      0.90    500000
```

In [ ]:

```
cm_test = confusion_matrix(check_df_test['fraud'], check_df_test['pred'])
cmp_test = ConfusionMatrixDisplay(cm_test, display_labels=[False, True])
fig, ax = plt.subplots(figsize=(10,10))
cmp_test.plot(ax=ax, cmap=plt.cm.Blues)
plt.show()
```