Unsupervised Learning

Fraud Dectection on Bank Transactions

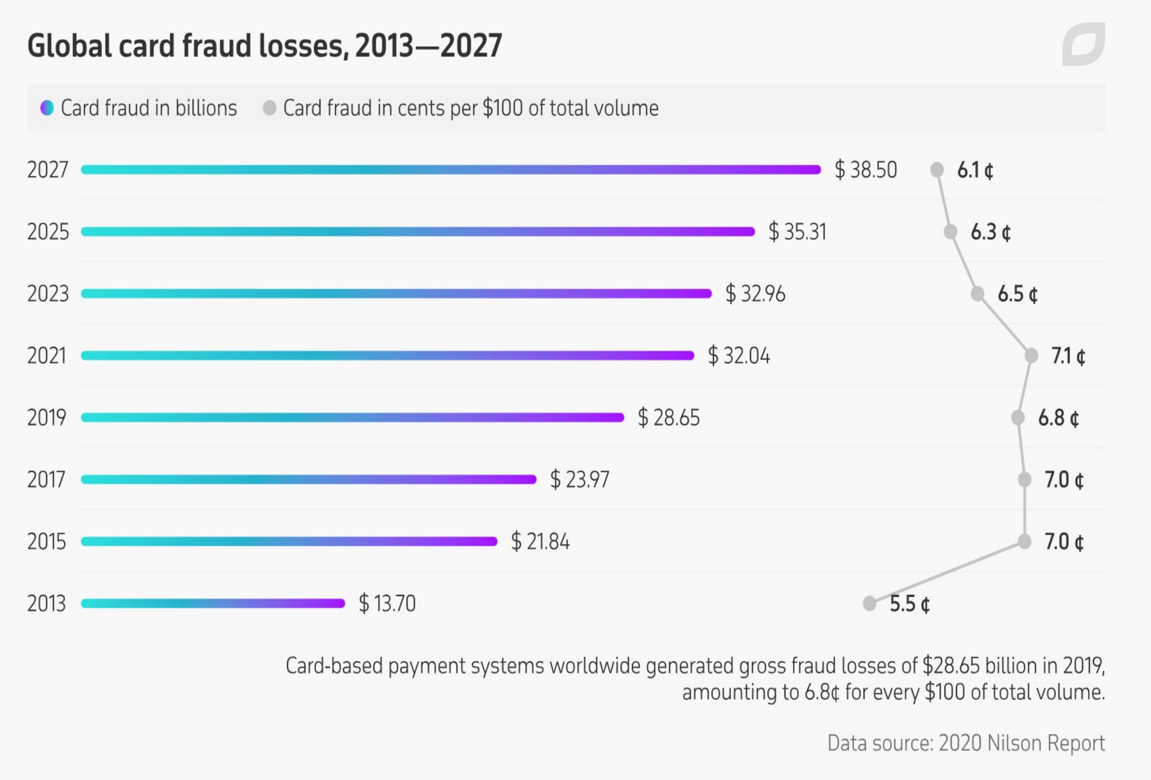
Nguyen Thien Toan

Outline:

1. Introduction
2. Models
3. Idea to improve models

Part 1: Introduction

Problem



Card fraud is one of the biggest threats to organizations today. Card fraud is simply defined as unauthorized, deliberate deception to secure unfair or unlawful access to a victim‟s transaction card in order to defraud him (Salem, 2012).

A fraud detection system usually comes to play when the fraudsters outwit the fraud prevention mechanism and initiate fraudulent transactions. In the business world, the application of data mining technique to fraud detection is of special interest as a result of the great losses companies suffer due to such fraudulent activities. This work describes data mining technique and its application to card fraud detection.

Why unsupervised learning?

Unsupervised learning are models that don't take the y(target) as an argument. It does not fit the data so you don't need the ground truth to make predictions. Banks are in a similar situation where they don't know if a transaction is fraud when the transaction is happening.

The unsupervised models will take a feature matrix X and make labels based on unseen structures and hidden correlations within the feature space.

What we should consider when using unsupervised learning models?

- You cannot get precise information regarding data sorting, and the output as data used in unsupervised learning is labeled and not known
- Less accuracy of the results is because the input data is not known and not labeled by people in advance. This means that the machine requires to do this itself.
- The spectral classes do not always correspond to informational classes.
- The user needs to spend time interpreting and label the classes which follow that classification.
- Spectral properties of classes can also change over time so you can’t have the same class information while moving from one image to another.

Describe data

- **Leisure and travel related transactions have high rates of fraud**

```
df = pd.read_csv('data/banksim.csv')
print(df.groupby('category').mean())
```

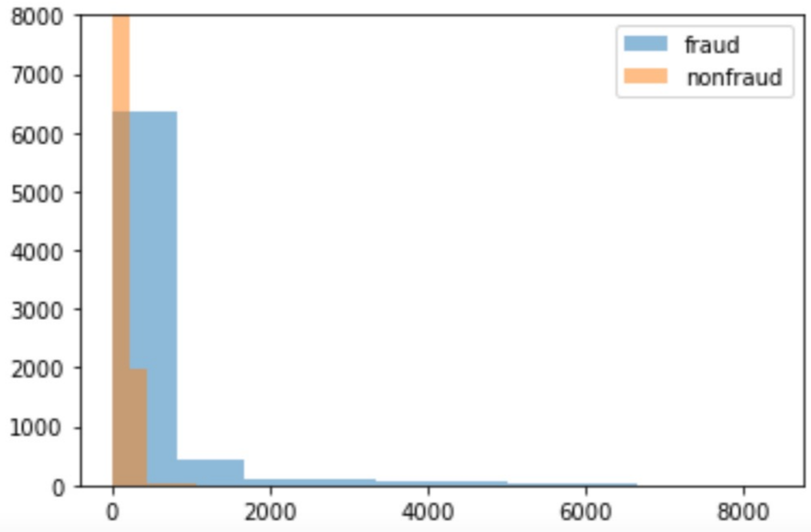
```
                              step       amount     fraud
category
'es_barsandrestaurants'   75.210576    43.461014  0.018829
'es_contents'             99.633898    44.547571  0.000000
'es_fashion'              95.426092    65.666642  0.017973
'es_food'                107.100861    37.070405  0.000000
'es_health'              100.636211   135.621367  0.105126
'es_home'                 89.760322   165.670846  0.152064
'es_hotelservices'        92.966170   205.614249  0.314220
'es_hyper'                77.837652    45.970421  0.045917
'es_leisure'              84.667335   288.911303  0.949900
'es_otherservices'        70.445175   135.881524  0.250000
'es_sportsandtoys'        81.332834   215.715280  0.495252
'es_tech'                 95.034177   120.947937  0.066667
'es_transportation'       94.953059    26.958187  0.000000
'es_travel'               85.104396  2250.409190  0.793956
'es_wellnessandbeauty'    90.658094    65.511221  0.047594
```

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0

Fraud transactions tend to be for large amounts

```
df_fraud = df.loc[df['fraud'] == 1]
df_non_fraud = df.loc[df['fraud'] == 0]

# Plot histograms of the amounts in fraud and non-fraud data
plt.hist(df_fraud['amount'], alpha=0.5, label='fraud')
plt.hist(df_non_fraud['amount'], alpha=0.5, label='nonfraud')
plt.legend()
plt.ylim(0,8000)
plt.show()
```



Part 2: Models

Preprocessing data

- One-Hot Encoding
- MinMaxScaler

```
y = df['fraud']
X = df.drop('fraud', axis=1)
X = np.array(df).astype(np.float)
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

```
<ipython-input-23-44d19454be5a>:3: DeprecationWarning: `np.float` is a
o silence this warning, use `float` by itself. Doing this will not modi
cally wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.
cations
  X = np.array(df).astype(np.float)
```

```
X_scaled
```

```
array([[0.00000000e+00, 5.00000000e-01, 2.06810025e-01, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       [1.38908182e-04, 6.66666667e-01, 1.62478579e-01, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [2.77816363e-04, 5.00000000e-01, 7.51345685e-02, ...,
        0.00000000e+00, 1.00000000e+00, 0.00000000e+00],
       ...,
       [9.99722184e-01, 1.66666667e-01, 1.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 1.00000000e+00],
       [9.99861092e-01, 1.66666667e-01, 1.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 6.66666667e-01, 1.00000000e+00, ...,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

	Unnamed: 0	age	amount	fraud	M	es_barsandrestaurants	es_contents	es_fashion	es_food	es_health	es_home	es_hotelservices	es_hyper
0	0	3	49.71	0	0	0	0	0	0	0	0	0	0
1	1	4	39.29	0	0	0	0	0	0	1	0	0	0
2	2	3	18.76	0	0	0	0	0	0	0	0	0	0
3	3	4	13.95	0	1	0	0	0	0	0	0	0	0
4	4	2	49.87	0	1	0	0	0	0	0	0	0	0

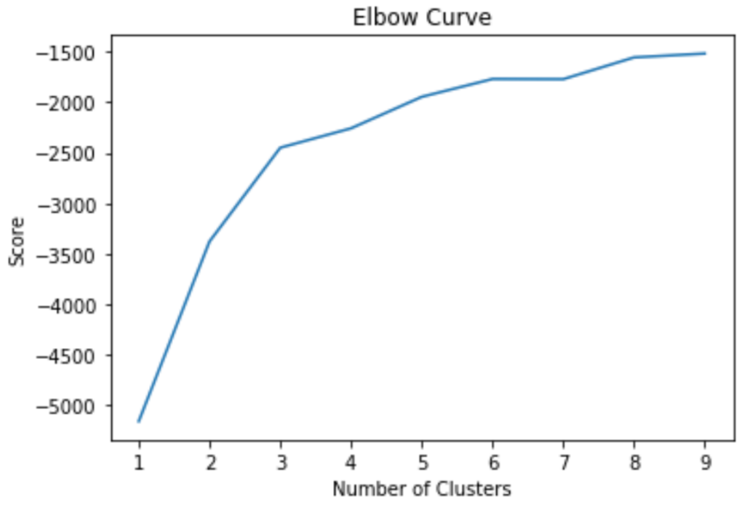
Kmean Method

Elbow method points out that 4 cluster is the best

```
clustno = range(1, 10)

kmeans = [MiniBatchKMeans(n_clusters=i) for i in clustno]
score = [kmeans[i].fit(X_scaled).score(X_scaled) for i in range(len(kmeans))]

plt.plot(clustno, score)
plt.xlabel('Number of Clusters')
plt.ylabel('Score')
plt.title('Elbow Curve')
plt.show()
```



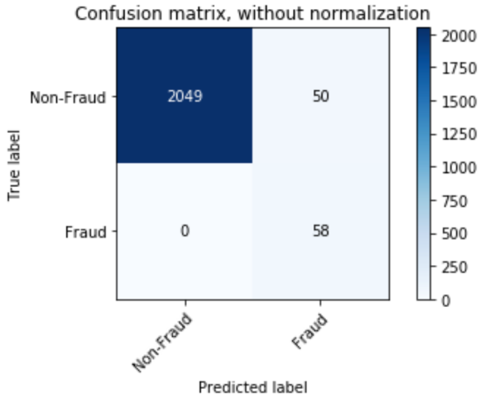
```
R2: 0.114
Roc_Auc: 0.988

Classifcation report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2099
           1       0.54      1.00      0.70        58

   micro avg       0.98      0.98      0.98      2157
   macro avg       0.77      0.99      0.84      2157
weighted avg       0.99      0.98      0.98      2157

Confusion matrix, without normalization
```




```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=0)

kmeans = MiniBatchKMeans(n_clusters=4, random_state=42).fit(X_train)

# Obtain predictions and calculate distance from cluster centroid
X_test_clusters = kmeans.predict(X_test)
X_test_clusters_centers = kmeans.cluster_centers_
dist = [np.linalg.norm(x-y) for x, y in zip(X_test, X_test_clusters_centers[X_test_clusters])]

# Create fraud predictions based on outliers on clusters
km_y_pred = np.array(dist)
km_y_pred[dist >= np.percentile(dist, 95)] = 1
km_y_pred[dist < np.percentile(dist, 95)] = 0

print_model_result(y_test, km_y_pred)
```

DBSCAN METHOD

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=0)

db = DBSCAN(eps=0.9, min_samples=10, n_jobs=-1).fit(X_train)

pred_labels = db.labels_
n_clusters = len(set(pred_labels)) - (1 if -1 in y else 0)

print('Estimated number of clusters: %d' % n_clusters)
print("Homogeneity: %0.3f" % homogeneity_score(y_train, pred_labels))
print("Silhouette Coefficient: %0.3f" % silhouette_score(X_train, pred_labels))
```

```
Estimated number of clusters: 20
Homogeneity: 0.862
Silhouette Coefficient: 0.556
```

```
def dbscan_predict(model, X):
    nr_samples = X.shape[0]

    y_new = np.ones(shape=nr_samples, dtype=int) * -1

    for i in range(nr_samples):
        diff = model.components_ - X[i, :]  # NumPy broadcasting

        dist = np.linalg.norm(diff, axis=1)  # Euclidean distance

        shortest_dist_idx = np.argmin(dist)

        if dist[shortest_dist_idx] < model.eps:
            y_new[i] = model.labels_[model.core_sample_indices_[shortest_dist_idx]]

    return y_new
```

Predict new labels on test dataset

```
pred_labels = dbscan_predict(db, X_test)
n_clusters = len(set(pred_labels)) - (1 if -1 in y else 0)

print('Estimated number of clusters: %d' % n_clusters)
print("Homogeneity: %0.3f" % homogeneity_score(y_test, pred_labels))
print("Silhouette Coefficient: %0.3f" % silhouette_score(X_test, pred_labels))
```

```
Estimated number of clusters: 20
Homogeneity: 0.868
Silhouette Coefficient: 0.559
```

```python
db_df = pd.DataFrame({'clusternr':pred_labels, 'fraud':y_test})

db_df['predicted_fraud'] = 0
for cluster in smallest_clusters:
    db_df['predicted_fraud'].loc[db_df['clusternr']==cluster] = 1

#db_df['predicted_fraud'] = np.where((db_df['clusternr']==21) | (db_df['clusternr']==9),1 , 0)

print_model_result(db_df['fraud'], db_df['predicted_fraud'])
```

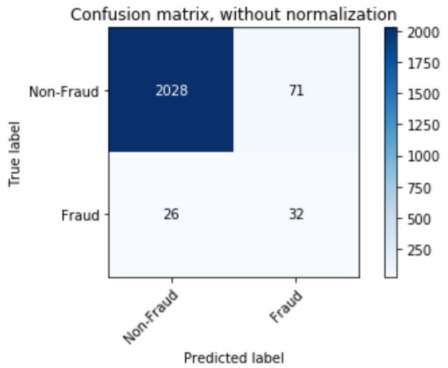
```
R2: -0.719
Roc_Auc: 0.759

Classifcation report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98      2099
           1       0.31      0.55      0.40        58

   micro avg       0.96      0.96      0.96      2157
   macro avg       0.65      0.76      0.69      2157
weighted avg       0.97      0.96      0.96      2157

Confusion matrix, without normalization
```



Part 3: Idea to improve models

No models are perfect. We need to improve it everyday

- Since the sihoullet score in both KMean and DBSCAN were not good, just around 0.5 meaning they were not seperated perfectly, it is good to clean data deeply(exclude some reduntdant points).
- I found a paper (link below), their technique can improve DBSCAN model. It improve the accuracy of the cluster and decrease the execute time. https://www.ijraset.com/research-paper/techniques-to-enhance-the-performance-of-dbscan-clustering-algorithm