Credit Card Fraud Detection

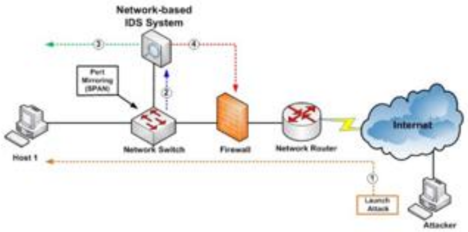
AM20216801
장철희

Index

- Examples of anomaly detection
- PreProcessing
- modeling
- Result
- Conclusion
- Reference

Examples of anomaly detection

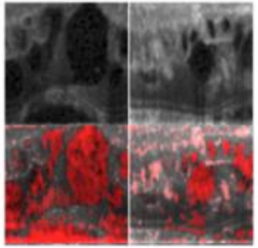
Cyber-intrusion



Fraud

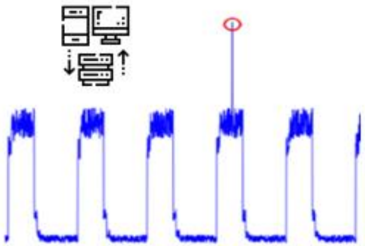
Malware

Medical

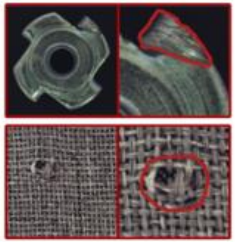


Social Network

Log file



IoT Big-Data



Industrial

Video Surveillance

Anomaly Detection

Supervised Anomaly Detection :

Supervised anomaly detection techniques require a data set that has been labeled as "normal" and "abnormal" and involves training a classifier. However, this approach is rarely used in anomaly detection due to the general unavailability of labelled data and the inherent unbalanced nature of the classes.

Semi-supervised (One-Class) Anomaly Detection

Semi-supervised anomaly detection techniques assume that some portion of the data is labelled. This may be any combination of the normal or anomalous data, but more often than not the techniques construct a model representing normal behavior from a given *normal* training data set, and then test the likelihood of a test instance to be generated by the model.

Unsupervised Anomaly Detection

Unsupervised anomaly detection techniques assume the data is unlabelled and are by far the most commonly used due to their wider and relevant application.

PreProcessing

[1]:

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

#for data preprocessing
from sklearn.decomposition import PCA

#for modeling
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest

#filter warnings
import warnings
warnings.filterwarnings("ignore")

import os
print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

```
['creditcard.csv']
```

PreProcessing

[2]:

```
df = pd.read_csv("../input/creditcard.csv")
df.head()
```

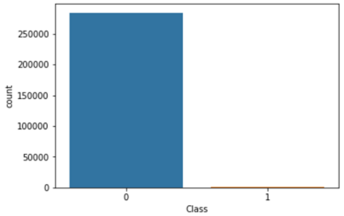
[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

[3]:

```
sns.countplot(df.Class)
plt.show()
print(df.Class.value_counts())
```




```
0    284315
1       492
Name: Class, dtype: int64
```

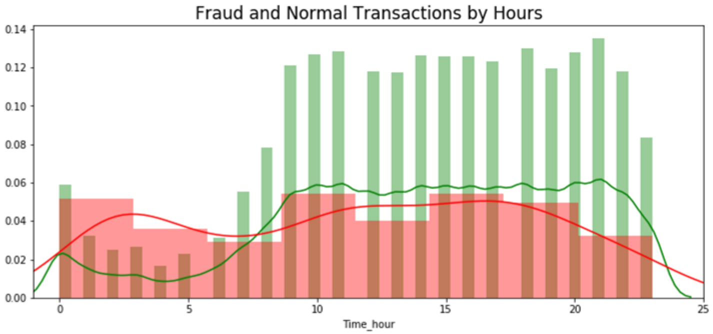
Data that has been converted to V1 to V28 by PCA was used due to confidentiality issues.

PreProcessing

[4]:

```
timedelta = pd.to_timedelta(df['Time'], unit='s')
df['Time_hour'] = (timedelta.dt.components.hours).astype(int)

plt.figure(figsize=(12,5))
sns.distplot(df[df['Class'] == 0]["Time_hour"], color='g')
sns.distplot(df[df['Class'] == 1]["Time_hour"], color='r')
plt.title('Fraud and Normal Transactions by Hours', fontsize=17)
plt.xlim([-1,25])
plt.show()
```



PreProcessing

```
[5]: cols= df[['Time', 'Amount']]

     pca = PCA()
     pca.fit(cols)
     X_PCA = pca.transform(cols)

     df['V29']=X_PCA[:,0]
     df['V30']=X_PCA[:,1]

     df.drop(['Time','Time_hour', 'Amount'], axis=1, inplace=True)

     df.columns
```

```
[5]: Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
            'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
            'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Class', 'V29', 'V30'],
           dtype='object')
```

PCA.fit() : found the principal

PCA.transform() : Transform data into new principal components

PreProcessing

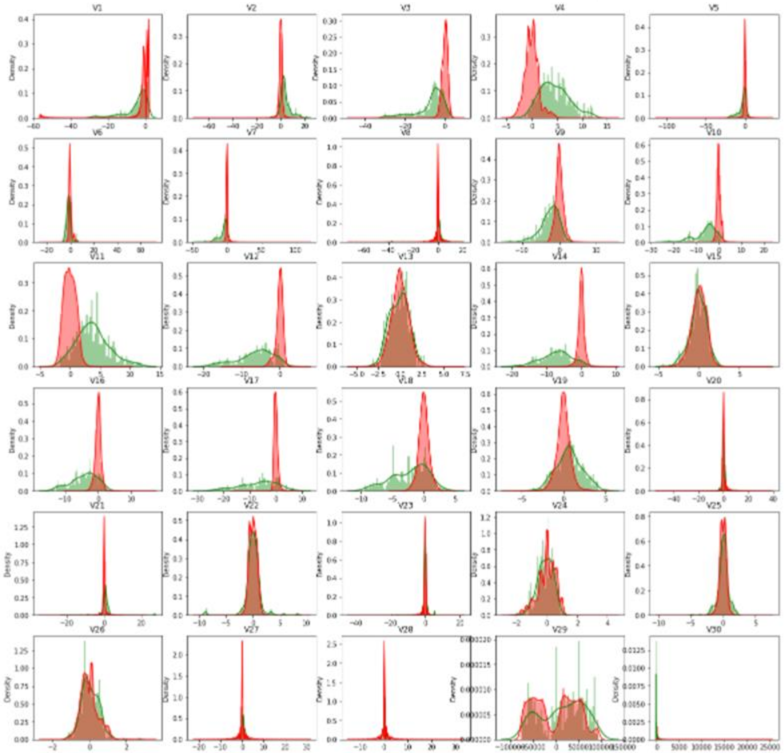
[6]:

```
columns = df.drop('Class', axis=1).columns
grid = gridspec.GridSpec(6, 5)

plt.figure(figsize=(20,10*2))

for n, col in enumerate(df[columns]):
    ax = plt.subplot(grid[n])
    sns.distplot(df[df.Class==1][col], bins = 50, color='g')
    sns.distplot(df[df.Class==0][col], bins = 50, color='r')
    ax.set_ylabel('Density')
    ax.set_title(str(col))
    ax.set_xlabel('')

plt.show()
```



PreProcessing

[7]:

```python
def ztest(feature):

    mean = normal[feature].mean()
    std = fraud[feature].std()
    zScore = (fraud[feature].mean() - mean) / (std/np.sqrt(sample_size))

    return zScore
```

[8]:

```python
columns= df.drop('Class', axis=1).columns
normal= df[df.Class==0]
fraud= df[df.Class==1]
sample_size=len(fraud)
significant_features=[]
critical_value=2.58

for i in columns:

    z_vavlue=ztest(i)

    if( abs(z_vavlue) >= critical_value):
        print(i," is statistically significant") #Reject Null hypothesis. i.e. H0
        significant_features.append(i)
```

```
V1  is statistically significant
V2  is statistically significant
V3  is statistically significant
V4  is statistically significant
V5  is statistically significant
V6  is statistically significant
V7  is statistically significant
V9  is statistically significant
V10  is statistically significant
V11  is statistically significant
V12  is statistically significant
V14  is statistically significant
V16  is statistically significant
V17  is statistically significant
V18  is statistically significant
V19  is statistically significant
V20  is statistically significant
V21  is statistically significant
V24  is statistically significant
V27  is statistically significant
V28  is statistically significant
V29  is statistically significant
V30  is statistically significant
```

z-test:

The z-test is an analytical technique that tests a hypothesis by comparing the means of two groups.

Valid transactions as our population
Fraud transactions as sample
Two tailed Z-test
Level of significance 0.01
Corresponding critical value is 2.58

Hypothesis:
H0: There is no difference (insignificant)
H1: There is a difference (significant)

Zscore=$(\bar{x}-\mu)/S.E$

PreProcessing

[9]:

```
significant_features.append('Class')
df= df[significant_features]

inliers = df[df.Class==0]
ins = inliers.drop(['Class'], axis=1)

outliers = df[df.Class==1]
outs = outliers.drop(['Class'], axis=1)

ins.shape, outs.shape
```

[9]:

```
((284315, 23), (492, 23))
```

[10]:

```
def normal_accuracy(values):

    tp=list(values).count(1)
    total=values.shape[0]
    accuracy=np.round(tp/total,4)

    return accuracy

def fraud_accuracy(values):

    tn=list(values).count(-1)
    total=values.shape[0]
    accuracy=np.round(tn/total,4)

    return accuracy
```

Modeling

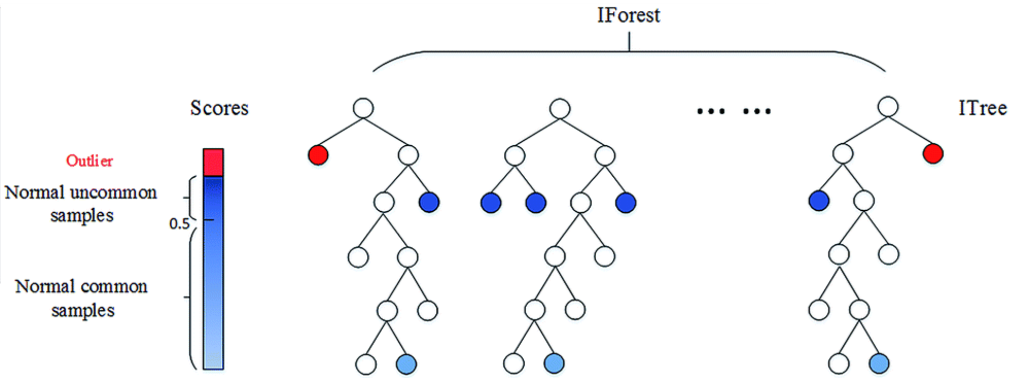
[11]:
```
state= 42

ISF = IsolationForest(random_state=state)
ISF.fit(ins)

normal_isf = ISF.predict(ins)
fraud_isf = ISF.predict(outs)

in_accuracy_isf=normal_accuracy(normal_isf)
out_accuracy_isf=fraud_accuracy(fraud_isf)
print("Accuracy in Detecting Normal Cases:", in_accuracy_isf)
print("Accuracy in Detecting Fraud Cases:", out_accuracy_isf)
```

```
Accuracy in Detecting Normal Cases: 0.9
Accuracy in Detecting Fraud Cases: 0.9004
```



IsolationForest

It is mainly used to detect outliers in the current data. As the name suggests, it is implemented based on a tree, and it is implemented by splitting the data at random and isolating all observations.

In particular, it has the advantage of being able to operate efficiently on data with many variables.

Modeling

[12]:
```
LOF = LocalOutlierFactor(novelty=True)
LOF.fit(ins)

normal_lof = LOF.predict(ins)
fraud_lof = LOF.predict(outs)

in_accuracy_lof=normal_accuracy(normal_lof)
out_accuracy_lof=fraud_accuracy(fraud_lof)
print("Accuracy in Detecting Normal Cases:", in_accuracy_lof)
print("Accuracy in Detecting Fraud Cases:", out_accuracy_lof)
```

```
Accuracy in Detecting Normal Cases: 0.9171
Accuracy in Detecting Fraud Cases: 0.5142
```

LocalOutlierFactor

LOF indicates how far each observation is within the data (outliers). The most important characteristic of LOF is not to consider all data as a whole, but rather to use the data surrounding that observation to identify the extent of an outlier from a local point of view.

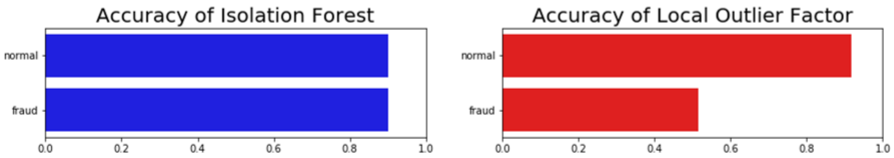
Result

[13]:

```
fig, (ax1,ax2)= plt.subplots(1,2, figsize=[15,2])

ax1.set_title("Accuracy of Isolation Forest",fontsize=20)
sns.barplot(x=[in_accuracy_isf,out_accuracy_isf],
            y=['normal', 'fraud'],
            label="classifiers",
            color="b",
            ax=ax1)
ax1.set(xlim=(0,1))

ax2.set_title("Accuracy of Local Outlier Factor",fontsize=20)
sns.barplot(x=[in_accuracy_lof,out_accuracy_lof],
            y=['normal', 'fraud'],
            label="classifiers",
            color="r",
            ax=ax2)
ax2.set(xlim=(0,1))
plt.show()
```



Conclusion

Both, Isolation Forest and Local Outlier Factor performed same in predicting Normal cases but Isolation Forest performed far better in detecting Fraud cases.

Reference

https://www.kaggle.com/code/sabanasimbutt/anomaly-detection-using-unsupervised-techniques

“Deep Learning for Anomaly Detection: A Survey,” 2019 arXiv