Outlier detection algorithm

장 보 민 , 손 수 민

2021.06.03

INDEX

- Stock Market Prediction
- Credit Card Fraud Detection

Credit Card Fraud Detection

Random Forest model을 기준으로

6가지 알고리즘을 사용하여 outlier를 제거 하고 성능을 비교함

6가지 알고리즘 : One-Class SVM, Isolation Forest, IQR, Elliptic curve, MCD, LOF

Credit Card Fraud Detection

Anonymized credit card transactions labeled as fraudulent or genuine

https://www.kaggle.com/mlg-ulb/creditcardfraud

▪ Credit Card Fraud Detection

1) 데이터 분포, 전처리

데이터 확인 및 분포

In [2]:
```
data=pd.read_csv('creditcard.csv',error_bad_lines=False)
```

In [3]:
```
df=pd.DataFrame(data)
df=df.dropna(axis=0)
df=df.drop(['Time'],axis=1)
```

In [4]:
```
normal=df[df['Class']==0]
Fraud=df[df['Class']==1]
```

In [5]:
```
if __name__=="__main__":
    print("전체 데이터의 분포 ₩n 전체거래 : {}₩n 정상거래 : {}₩n 이상거래 : {}₩n".format(len(df),len(normal),len(Fraud)))
    print("정상거래의 비율 : {:.2f}₩n이상거래의 비율 : {:.2f}₩n".format(len(normal)/len(df)*100,len(Fraud)/len(df)*100))
```

```
전체 데이터의 분포
 전체거래 : 247302
 정상거래 : 246853
 이상거래 : 449

정상거래의 비율 : 99.82
이상거래의 비율 : 0.18
```

데이터의 비율이 불균형하다는 것을 알 수 있음.

Time 속성 제외

데이터 속성 : V1~V28, Amount, class

247302 rows × 30 columns

Credit Card Fraud Detection

1) 데이터 분포, 전처리

5	V26	V27	V28	Amount	Class
9	-0.189115	0.133558	-0.021053	149.62	0.0
0	0.125895	-0.008983	0.014724	2.69	0.0
2	-0.139097	-0.055353	-0.059752	378.66	0.0
6	-0.221929	0.062723	0.061458	123.50	0.0
0	0.502292	0.219422	0.215153	69.99	0.0
.	...	...	...	...	...
3	-0.265578	0.520126	0.260147	1.00	0.0
3	-0.469358	-0.054827	0.030833	400.00	0.0
9	-0.275729	0.531661	0.275495	9.32	0.0
6	-0.148970	-0.097911	-0.042843	85.25	0.0
3	0.155941	0.232338	0.078931	0.99	0.0

V26	V27	V28	Class	Log Ammount
-0.189115	0.133558	-0.021053	0.0	5.008105
0.125895	-0.008983	0.014724	0.0	0.989913
-0.139097	-0.055353	-0.059752	0.0	5.936641
-0.221929	0.062723	0.061458	0.0	4.816249
0.502292	0.219422	0.215153	0.0	4.248367
...	...	...	...	...
-0.265578	0.520126	0.260147	0.0	0.001000
-0.469358	-0.054827	0.030833	0.0	5.991467
-0.275729	0.531661	0.275495	0.0	2.232270
-0.148970	-0.097911	-0.042843	0.0	4.445600

```
eps=0.001
df['Log Ammount'] = np.log(df.pop('Amount')+eps)
```

Amount의 값이 너무 편차가 커서, log space로 변경해주었음.

Credit Card Fraud Detection

2) 모델 – RandomForest

```
# from sklearn.model_selection import GridSearchCV

# params = { 'n_estimators' : [10, 100],
#            'max_depth' : [6, 8, 10, 12],
#            'min_samples_leaf' : [8, 12, 18],
#            'min_samples_split' : [8, 16, 20]
#            }

# # RandomForestClassifier 객체 생성 후 GridSearchCV 수행
# rf_clf = RandomForestClassifier(random_state = 0, n_jobs = -1)
# grid_cv = GridSearchCV(rf_clf, param_grid = params, cv = 3, n_jobs = -1)
# grid_cv.fit(X_data, Y_data)

# print('최적 하이퍼 파라미터: ', grid_cv.best_params_)
# print('최고 예측 정확도: {:.4f}'.format(grid_cv.best_score_))
```

```
rf= RandomForestClassifier(n_estimators = 100,
                           max_depth = 10,
                           min_samples_leaf = 8,
                           min_samples_split = 8,
                           random_state = 0,
                           n_jobs = -1)
```

GridSearchCV를 이용해 최적의 파라미터를 구해서 적용함.

Credit Card Fraud Detection

2) 모델 – RandomForest

```
X_train_val, X_test, Y_train_val, Y_test = train_test_split(X_data,Y_data, test_size=0.2, random_state=777, stratify=Y_data)
```

```
X_train_val = X_train_val.to_numpy ()
Y_train_val = Y_train_val.to_numpy ()
```

```
skf = StratifiedKFold(n_splits=5)
k=5
```

```
result_Y_val=[]
result_Y_val_pred=[]
model =  rf

i=0
for train_index,val_index in skf.split(X_train_val,Y_train_val):
    i=i+1
    print(str(i),"번째 cross validation")
    X_train = X_train_val[train_index]
    Y_train = Y_train_val[train_index]
    X_val = X_train_val[val_index]
    Y_val = Y_train_val[val_index]

    sm = SMOTE()
    X_train_oversampled, Y_train_oversampled = sm.fit_sample(X_train, Y_train)
    model.fit(X_train_oversampled, Y_train_oversampled)

    Y_val_pred = rf.predict(X_val)
    result_Y_val.append(Y_val)
    result_Y_val_pred.append(Y_val_pred)
```

```
1 번째 cross validation
2 번째 cross validation
3 번째 cross validation
4 번째 cross validation
5 번째 cross validation
```

- Train, val, test 부분으로 나눔.
- 5번의 교차검증
- 매 교차검증마다 smote로 데이터 불균형 해결

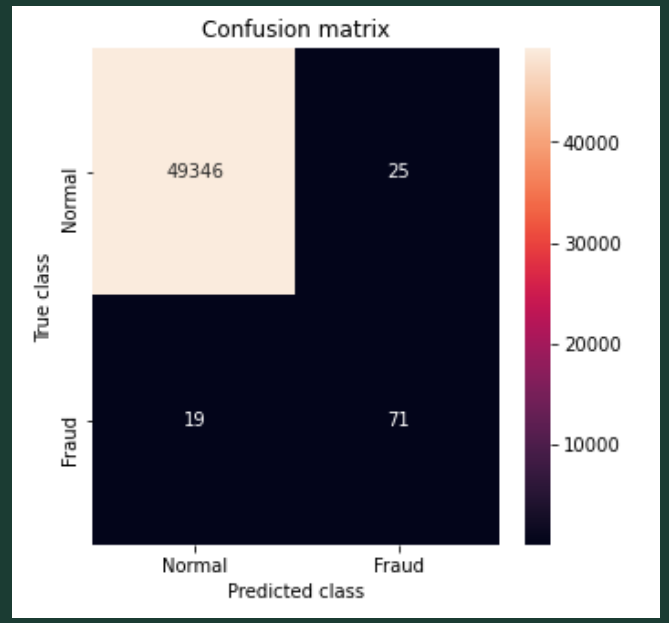
- 데이터의 비율이 불균형해서 test_data의 결과가 제대로 나오지 않았음.
=> 10%에서 20%로 비율을 늘려줌

Credit Card Fraud Detection

2) 모델 – RandomForest

```
evaluation(Y_test,Y_pred)

accuracy : 0.9991104102221953
precision : 0.7395833333333334
recall: 0.7888888888888889
F1-Score : 0.7634408602150538
auc_score: 0.8941912593762870
```



Test 결과

- RandomForest & 이상치 제거 X
- F1-score : 0.76
- Auc-score : 0.84

Credit Card Fraud Detection

3) 이상치 제거 – Isolation Forest

```
iforest = IsolationForest(n_estimators=100, max_samples=len(normal),
                          contamination=0.0005, max_features=1.0,
                          bootstrap=False, n_jobs=-1, random_state=1)
```

```
iforest_f = IsolationForest(n_estimators=100, max_samples=len(normal),
                          contamination=0.0005, max_features=1.0,
                          bootstrap=False, n_jobs=-1, random_state=1)
```

```
normal['anomaly']=normal_pred
normal_outliers=normal.loc[normal['anomaly']==-1]
normal_outlier_index=list(normal_outliers.index)
print(normal['anomaly'].value_counts())
normal=normal.drop(['anomaly'],axis=1)
```

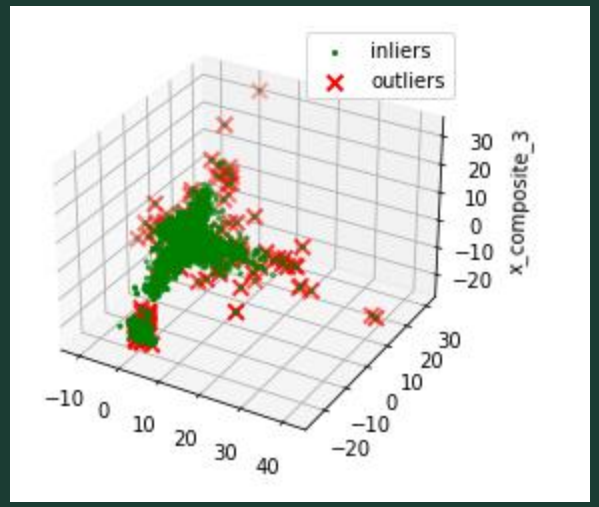
```
 1    246729
-1       124
Name: anomaly, dtype: int64
```

```
fraud['anomaly']=fraud_pred
fraud_outliers=fraud.loc[fraud['anomaly']==-1]
fraud_outlier_index=list(fraud_outliers.index)
print(fraud['anomaly'].value_counts())
fraud=fraud.drop(['anomaly'],axis=1)
```

```
 1    448
-1      1
Name: anomaly, dtype: int64
```

normal, Fraud data로 구분 후 각각의 데이터에 대한 이상치를 추출해 제거

Normal : 124개 Fraud : 1개 제거



PCA를 이용해 차원을 줄인 후

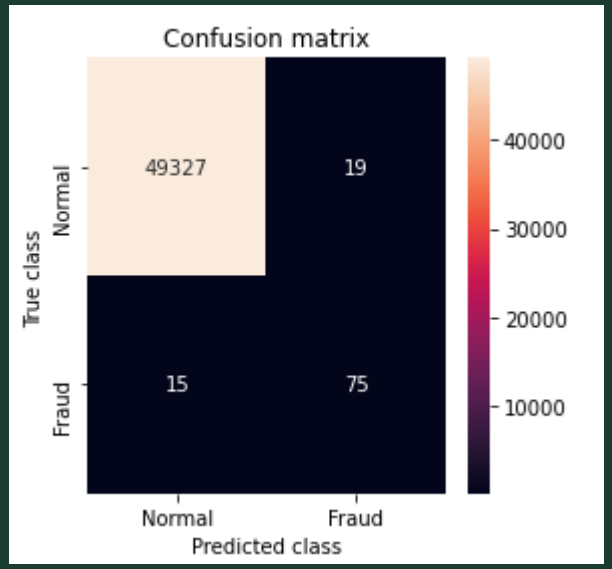
Outlier를 시각화

Credit Card Fraud Detection

3) 이상치 제거 – Isolation Forest

```
evaluation(Y_test,Y_pred)
```

```
accuracy : 0.9993122420907841
precision : 0.7978723404255319
recall: 0.8333333333333334
F1-Score : 0.8152173913043479
auc_score: 0.9164741485294318
```



이상치 제거 후 Random Forest에서 예측한 결과

- F1-score : 0.76 -> 0.81
- Auc-score : 0.84 -> 0.91

Credit Card Fraud Detection

3) 이상치 제거 – Elliptic curve

```
elpenv = EllipticEnvelope(contamination=0.0005,support_fraction = 0.9)
elpenv_f = EllipticEnvelope(contamination=0.0005,support_fraction = 0.9)
```

```
elpenv.fit(normal)
normal_pred =  elpenv.predict(normal)
elpenv_f.fit(fraud)
fraud_pred = elpenv_f.predict(fraud)
```

```
normal['anomaly']=normal_pred
normal_outliers=normal.loc[normal['anomaly']==-1]
normal_outlier_index=list(normal_outliers.index)
print(normal['anomaly'].value_counts())
normal=normal.drop(['anomaly'],axis=1)
```

```
 1    246729
-1       124
Name: anomaly, dtype: int64
```

```
fraud['anomaly']=fraud_pred
fraud_outliers=fraud.loc[fraud['anomaly']==-1]
fraud_outlier_index=list(fraud_outliers.index)
print(fraud['anomaly'].value_counts())
fraud=fraud.drop(['anomaly'],axis=1)
```

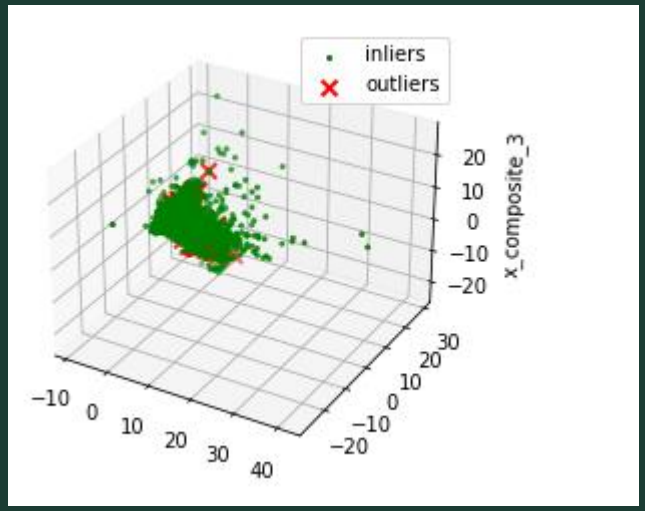
```
 1    448
-1      1
Name: anomaly, dtype: int64
```

normal, Fraud data로 구분 후 각각의 데이터에 대한 이상치를 추출해 제거

Normal : 124개 Fraud : 1개 제거

PCA를 이용해 차원을 줄인 후

Outlier를 시각화

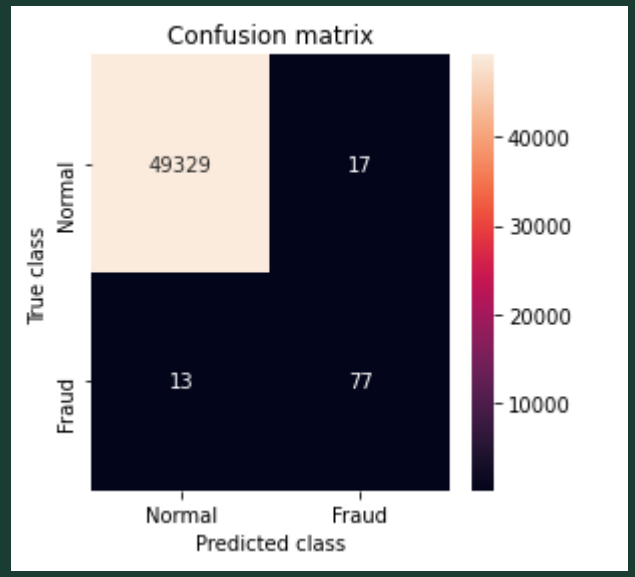


Credit Card Fraud Detection

3) 이상치 제거 – Isolation Forest

```
evaluation(Y_test,Y_pred)
```

```
accuracy : 0.9993931547859859
precision : 0.8191489361702128
recall: 0.8555555555555555
F1-Score : 0.8369565217391304
auc_score: 0.9276055247076201
```



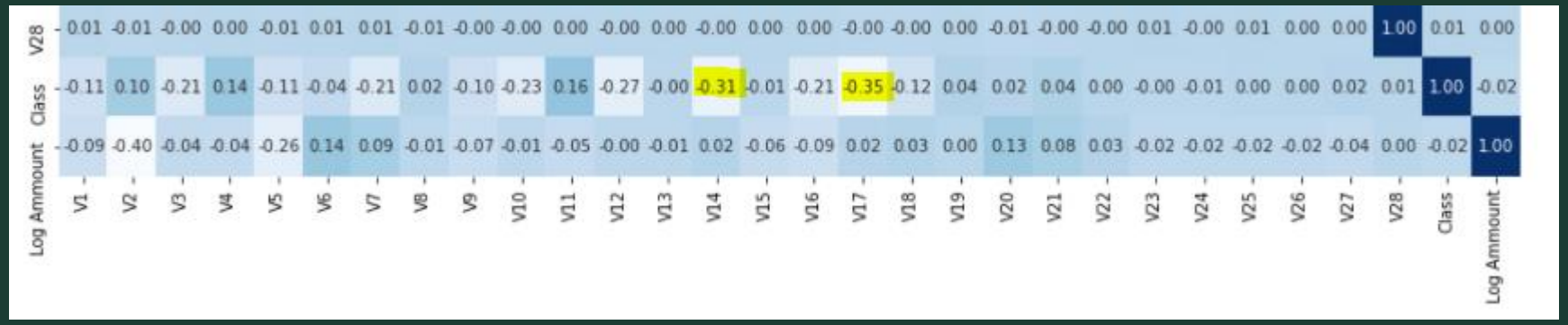
이상치 제거 후 Random Forest에서 예측한 결과

- F1-score : 0.76 -> 0.83
- Auc-score : 0.84 -> 0.92

Credit Card Fraud Detection

3) 이상치 제거 – IQR

```
corr=data.corr(method='pearson')
plt.figure(figsize=(20,20))
df_heatmap = sns.heatmap(corr, cbar = True, annot = True, annot_kws={'size' : 10}, fmt = '.2f', square = False, cmap = 'Blues')
```



Pearson을 이용해서 class와 연관 있는 속성이 V14, V17인 것을 알아 냄.

V17과 V14 중 V14를 기준으로 사분위값으로 최소, 최대 제한선 밖에 존재하는 이상치를 제외함.

Credit Card Fraud Detection

3) 이상치 제거 – IQR

```
def outlier_treatment(datacolumn):
    sorted(datacolumn)
    Q1,Q3 = np.percentile(datacolumn , [25,75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range,upper_range
```

```
normal=data[data['Class']==0]
fraud=data[data['Class']==1]
```

```
l,u=outlier_treatment(normal.V14)
```

```
normal[ (normal.V14 > u) | (normal.V14 < l) ]
```

```
normal_outlier_index=normal[ (normal.V14 > u) | (normal.V14 < l) ].index
```

```
len(normal_outlier_index)
```

```
12482
```

데이터의 속성을 넣으면 최대 제한선, 최소제한선을 반환하는 함수를 만들고, normal data의 V14 속성으로 normal 데이터의 이상치를 제외함.

Credit Card Fraud Detection

3) 이상치 제거 – IQR

```
l_f,u_f=outlier_treatment(fraud.V14)
```

```
fraud[ (fraud.V14 > u_f) | (fraud.V14 < l_f) ]
```

	V1	V2	V3	V4	V5	V6	V7	
8296	-2.125490	5.973556	-11.034727	9.007147	-1.689451	-2.854415	-7.810441	2
8615	-3.891192	7.098916	-11.426467	8.607557	-2.065706	-2.985288	-8.138589	2
9035	-2.589617	7.016714	-13.705407	10.343228	-2.954461	-3.055116	-9.301289	3
9252	-5.454362	8.287421	-12.752811	8.594342	-3.106002	-3.179949	-9.252794	4
9487	-4.153014	8.204797	-15.031714	10.330100	-3.994426	-3.250013	-10.415698	4
41943	-2.140511	4.104871	-8.996859	4.028391	-5.131359	-4.153568	-9.360095	1

6 rows × 30 columns

```
fraud_outlier_index=fraud[ (fraud.V14 > u_f) | (fraud.V14 < l_f) ].index
```

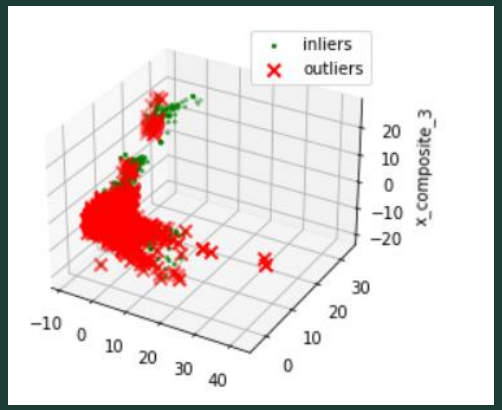
```
print(len(normal_outlier_index),len(fraud_outlier_index))
```

```
12482 6
```

```
outlier_index=np.concatenate([normal_outlier_index, fraud_outlier_index])
```

Fraud data도 normal과 마찬가지로 이상치를 구해 제외시킴.

Normal data에서는 12482, Fraud data에서는 6개의 이상치가 나옴.

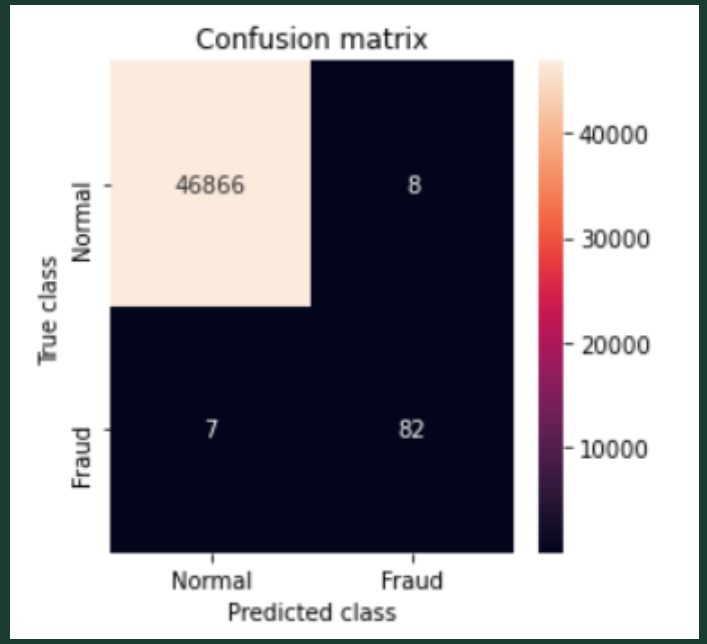


Credit Card Fraud Detection

3) 이상치 제거 – IQR

```
evaluation(Y_test,Y_pred)
```

```
accuracy : 0.9996805996209782
precision : 0.9111111111111111
recall: 0.9213483146067416
F1-Score : 0.9162011173184358
auc_score: 0.9605888221495542
```



이상치 제거 후 Random Forest에서 예측한 결과

- F1-score : 0.76 -> 0.91
- Auc-score : 0.84 -> 0.96

Credit Card Fraud Detection

4) 비교 평가

	Accuracy	Precision	Recall	F1-score	Auc-score
RF	0.999	0.739	0.788	0.763	0.894
RF_IsolationForest	0.999	0.797	0.833	0.815	0.916
RF_EllipticEnvelope	0.999	0.819	0.855	0.836	0.927
RF_IQR	0.999	0.911	0.921	0.916	0.960

현재 RandomForest 모델에서는 IQR 이상체 제거가 높은 성능을 보임.

Credit Card Fraud Detection

5) 차후 개선점

- LOF와 SVM의 경우, 이상치를 제거하지 않았을 때 보다 낮은 성능을 보이고 있음.
- 파라미터의 문제인지, 알고리즘 자체의 문제인지 알아보기 위해 GridSearchCV를 이용해 파라미터를 구하고 있음. 오류 발생으로 계속 찾아보고 있음.

```
accuracy : 0.9989683631361761
precision : 0.6695652173913044
recall: 0.8555555555555555
F1-Score : 0.751219512195122
auc_score: 0.9273927415033077
```

```
accuracy : 0.999295424403183
precision : 0.5714285714285714
recall: 0.837209302325581
F1-Score : 0.679245283018868
auc_score: 0.918324643695925
```

```
nus = [0.001, 0.01, 0.1, 1]
gammas = [0.001, 0.01, 0.1, 1]
scores = ['f1']
tuned_parameters = {'kernel' : ['rbf'], 'gamma' : gammas, 'nu': nus}
tuned_ocsvm = OneClassSVM()
grid = GridSearchCV(tuned_ocsvm, tuned_parameters, scoring = scores, refit = False, verbose=3, n_jobs=
grid.fit(X_data,Y_data)
print('final params', grid.best_params_)   # 최적의 파라미터 값 출력
print('best score', grid.best_score_)      # 최고의 점수
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
```

```
/home/sumin816/anaconda3/envs/test/lib/python3.7/site-packages/joblib/externals/loky/process_executor.p
1: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a to
rt worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
/home/sumin816/anaconda3/envs/test/lib/python3.7/site-packages/sklearn/model_selection/_search.py:921:
arning: One or more of the test scores are non-finite: [nan nan nan nan nan nan nan nan nan nan nan nan
nan nan nan]
  category=UserWarning
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-abca8cea89e2> in <module>
      6 grid = GridSearchCV(tuned_ocsvm, tuned_parameters, scoring = scores, refit = False, verbose=3,
bs=2)
      7 grid.fit(X_data,Y_data)
----> 8 print('final params', grid.best_params_)   # 최적의 파라미터 값 출력
      9 print('best score', grid.best_score_)      # 최고의 점수

AttributeError: 'GridSearchCV' object has no attribute 'best_params_'
```

감사합니다.