# PD_7

June 12, 2021

[1]:
```
import itertools
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

from sklearn.metrics import (f1_score,
                             precision_score,
                             recall_score,
                             silhouette_score)
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

[2]:
```
data_path_prefix = '../../'
```

[3]:
```
train = pd.read_csv(data_path_prefix + 'train.csv')
test = pd.read_csv(data_path_prefix + 'test.csv')
val = pd.read_csv(data_path_prefix + 'val.csv')
```

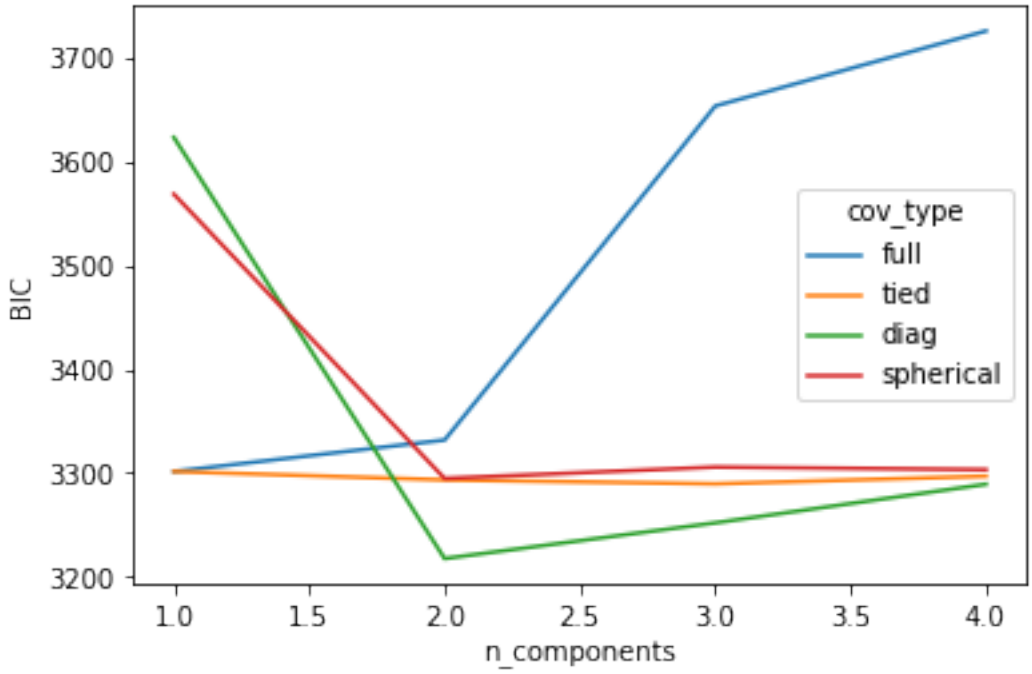
## 0.1 Choose the right component number

[4]:
```
X = StandardScaler().fit_transform(train)
n_components = range(1,5)
cov_types = ['full', 'tied', 'diag', 'spherical']
results = []
for cov_type in cov_types:
    for i in n_components:
        gmm = GaussianMixture(n_components=i, covariance_type=cov_type,␣
 ↪random_state=123)
        gmm.fit(X)
        results.append((cov_type, i, gmm.bic(X)))

results = pd.DataFrame(results, columns=['cov_type', 'n_components', 'BIC'])
```

[5]:
```
sns.lineplot(data=results, x='n_components', y='BIC', hue='cov_type')
```

[5]: <AxesSubplot:xlabel='n_components', ylabel='BIC'>




The best (lowest) Bayesian information criterion is achived for diagonal covariance type and for 2 components.

### 0.2 Choose the threshold

[6]:
```
estimator = Pipeline([
    ('scaler', StandardScaler()),
    ('gmm', GaussianMixture(
        n_components=2, covariance_type='diag', random_state=123))])

estimator.fit(train)
```

[6]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('gmm',
                 GaussianMixture(covariance_type='diag', n_components=2,
                                 random_state=123))])
```

[7]:
```
# get probabilities on train
train_prob = estimator.score_samples(train)
train_result = pd.DataFrame(train_prob, columns=['log_prob'])
train_result['class'] = 'train'
```

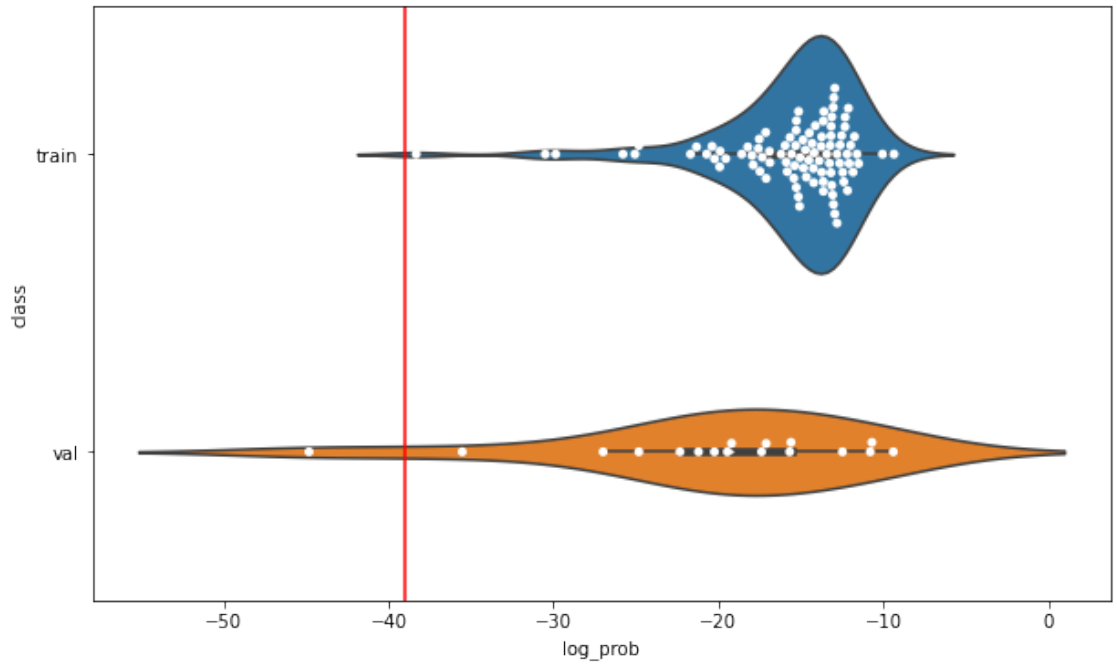
[8]:
```
# get probabilities on val
val_prob = estimator.score_samples(val)
val_result = pd.DataFrame(val_prob, columns=['log_prob'])
val_result['class'] = 'val'
```

[9]:
```
train_val = pd.concat([train_result, val_result])
```

[10]:
```
threshold = np.floor(np.min(train_prob))
```

The threshold was set as minimal logarithm of probability of points in training dataset.

[11]:
```
plt.figure(figsize=(10,6))
sns.violinplot(data=train_val, x='log_prob', y='class')
sns.swarmplot(data=train_val, x='log_prob', y='class', color='white')
plt.axvline(x=threshold, color='red')
plt.show()
```




We can see 4 observations from validation set would be classified as outliers.

### 0.3 Evaluate model on test dataset

[12]:
```
def predict(X, estimator, threshold):
    probs = estimator.score_samples(X)
    return np.array([1 if x<threshold else 0 for x in probs])
```

[13]:
```
X_test = test.drop(['class'], axis=1)
y_test = test['class']

predicted = predict(X_test, estimator, threshold)
metrics = [f1_score, precision_score, recall_score]
scores = [(m.__name__, m(y_test, predicted)) for m in metrics]
pd.DataFrame(scores, columns=['metric', 'score'])
```

[13]:

| | metric | score |
|---|---|---|
| 0 | f1_score | 0.571429 |
| 1 | precision_score | 1.000000 |
| 2 | recall_score | 0.400000 |

Precision score is high and racall is low. It means model classifies outliers as inliers. Hence precision = 1, all inliers were classified as inliers. F1 score is also quite low. Probably the threshold could be higher, but the choice depends on specific situation in which outliers detection is applied.