

Cinnamon AI Bootcamp

Anomaly detection using Guassian Mixture Model

Name: Nguyen Truong Phat

Expectation

In one dimension, given an observation x_i from k^{th} cluster (parameterized by μ and σ). The probability of seeing x_i

$$\mathcal{N}(x_i | \mu_k, \sigma_k^2) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right)$$

for each $k = 1, 2, 3, 4, \dots, K$

- Without observing the data, the probability of x_i is from the k^{th} cluster
 $p(z_i = k) = \phi_k$
(the probability of a random point is drawn from k^{th} distribution)
This can be interpreted as the "weight" for k -th gaussian distribution in the mixture of models, also known as the prior.
- The probability of observing an x_i in the data is the weighted sum (corresponding with ϕ_k for each clusters) of all the K distribution function at the given point x
 $p(x_i) = \sum_{k=1}^K \mathcal{N}(x_i | \mu_k, \sigma_k^2) \phi_k$
- Therefore, given an observation x_i , the likelihood of x_i (posterior) belong to k^{th} cluster can be calculated:

$$b_{ik} = p(\mu_k, \sigma_k^2 | x_i) = \frac{\mathcal{N}(x_i | \mu_k, \sigma_k^2) p(z_i = k)}{p(x_i)} = \frac{\mathcal{N}(x_i | \mu_k, \sigma_k^2) \phi_k}{\sum_{k=1}^K \mathcal{N}(x_i | \mu_k, \sigma_k^2) \phi_k} \text{ (proven)}$$

For multivariate gaussian model, the normal can be calculated:

$$\mathcal{N}(x_i | \mu_k, \Sigma) = \frac{1}{(2\pi)^{d/2}} |\Sigma|^{-1/2} \exp\left[-\frac{1}{2} (x_i - \mu_k)^T \Sigma^{-1} (x_i - \mu_k)\right]$$

Whereas, Σ is a $d \times d$ covariance matrix that satisfies:

- positive semi-definite
- symmetric along diagonal

Maximization

For each step, we update the parameters (in a multivariate style) as follow:

$$m_k = \sum_i b_{ik}$$

(sum of all likelihood allocated to cluster k)

$$\hat{\phi}_k = \frac{m_k}{m}$$

(fraction of total likelihood)

$$\hat{\mu}_k = \frac{1}{m_k} \sum_i b_{ik} x_i$$

(Weighted mean of assigned data)

$$\hat{\Sigma}_k = \frac{1}{m_k} \sum_i b_{ik} (x_i - \hat{\mu}_k)^T (x_i - \hat{\mu}_k)$$

(Weighted covariance of assigned data)

Each step **strictly** (theoretically) increases the log-likelihood of our model

$$\log p(X) = \sum_i \log \left[\sum_c \pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c) \right]$$

Anomaly detection

For anomaly detection, we can then first calculate the probability of x_i to be drawn from our mixture of Gaussian (with trained params):

$$p(x_i) = \sum_{k=1}^K \mathcal{N}(x_i | \mu_k, \sigma_k^2) \phi_k$$

If $p(x_i) < \lambda$ (which is the threshold we specify in advance) then we will consider x_i to be abnormal

To perform anomaly detection, we can consider the anomaly detection as a classification problem, then we can use classification techniques and metrics to work with it. This time, we want to choose threshold that maximizes the Fscore of the classification task:

$$\lambda = \underset{\lambda}{\operatorname{argmax}}(Fscore)$$

My heuristics of finding a reasonable threshold goes like this:

```
# Init bestFScore and bestThresh to be 0
bestFScore ← 0
bestThresh ← 0

for every thresh in range from 0 to 1

    # Get prediction of model based with thresh
    y_predict ← model.predict(X, thresh)

    # calculate f score based on true score
    currentFScore ← calcFScore(y_true, y_predict)

    if FScore > bestFScore
        bestFScore ← currentFScore
        bestThresh ← thresh
```

The precision of this method will mostly based on how small we quantize this threshold range - the step of every thresh iteration.

References

- [1] [Machine Learning TV - Guassian Mixture Model for Clustering](#)
- [2] [Geeksforgeeks - Guassian Mixture Model](#)
- [3] [Alexander Ihler - Gaussian Mixture Models and EM](#)
- [4] [Siraj Raval - Gaussian Mixture Models](#)
- [5] <https://stats.stackexchange.com/questions/91045/can-a-multivariate-distribution-with-a-singular-covariance-matrix-have-a-density>
- [6] <https://math.stackexchange.com/questions/503476/multivariate-gaussian-density-from-singular-covariance>

Implementation

Import libraries

```
import numpy as np
from scipy.stats import multivariate_normal
%matplotlib notebook
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from GMM import GaussianMixtureModel
import time
```

Load data

```
from scipy.io import loadmat
data = loadmat('cardio.mat')
X = data['X']
Y = data['y'][:,0]
X_pos = X[Y==1]
X_neg = X[Y==0]
```

Fit the model

```
# If get an error...please kindly run again, it works like magic
max_iters = 25
model = GaussianMixtureModel(n_clusters=3)
model.fit(X_neg,max_iters=max_iters)
```

```
step  0 log likelihood:  -29648.09566980788
step  1 log likelihood:  -27017.829010462443
step  2 log likelihood:  -20763.011394036635
step  3 log likelihood:  -20411.81808227092
step  4 log likelihood:  -16254.972129542499
step  5 log likelihood:  -18808.83768816995
step  6 log likelihood:  -18222.167533118132
step  7 log likelihood:  -19019.31283192718
step  8 log likelihood:  -16693.097569541373
step  9 log likelihood:  -15754.080540565357
step 10 log likelihood:  -17165.63427113998
step 11 log likelihood:  -18144.603903942956
step 12 log likelihood:  -15576.866400905454
step 13 log likelihood:  -17999.78642934142
step 14 log likelihood:  -16478.429327816782
step 15 log likelihood:  -17680.948988829023
step 16 log likelihood:  -16259.88385697005
step 17 log likelihood:  -17685.400630068107
step 18 log likelihood:  -13612.120217283362
step 19 log likelihood:  -16744.341883208228
step 20 log likelihood:  -17184.802608799924
step 21 log likelihood:  -18089.517709427233
step 22 log likelihood:  -16251.41448306101
```

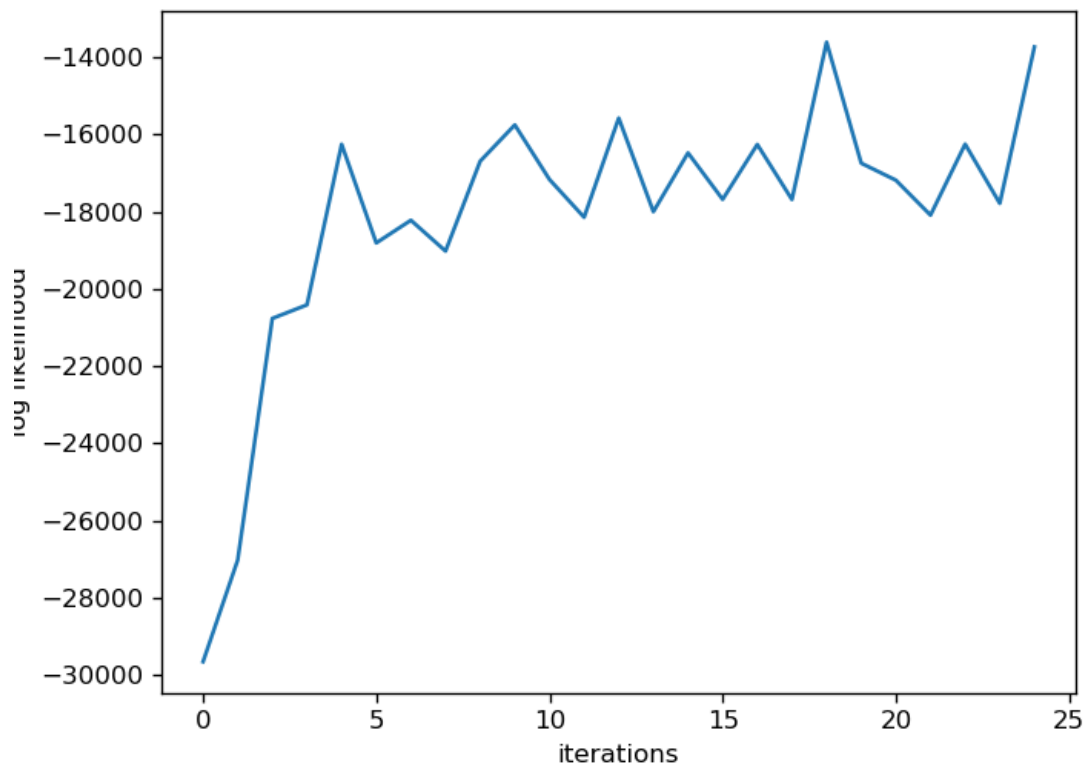
```
step 23 log likelihood: -17784.196754483743
step 24 log likelihood: -13736.053543139955
```

```
model.predict(X_neg)
```

```
array([2, 2, 2, ..., 1, 1, 1])
```

```
import matplotlib.pyplot as plt
plt.plot(range(0,max_iters),model.history['log_likelihood'])
plt.xlabel("iterations")
plt.ylabel("log likelihood")
```

<IPython.core.display.Javascript object>



```
Text(0, 0.5, 'log likelihood')
```

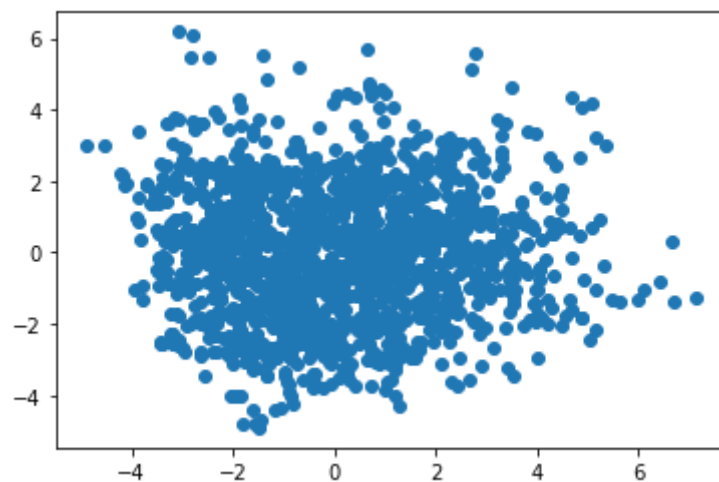
Theoretically speaking, the log likelihood should strictly increase every EM steps. But this fluctuates a lot. Proof for covariance exploding while compute GMM: <https://stackoverflow.com/questions/41216856/gmm-loglikelihood-isnt-monotonic>

Test on 2D Data

```
from sklearn.decomposition import PCA
# We use PCA to chop down component for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_neg)
```

```
%matplotlib inline
plt.scatter(X_pca[:,0],X_pca[:,1])
```

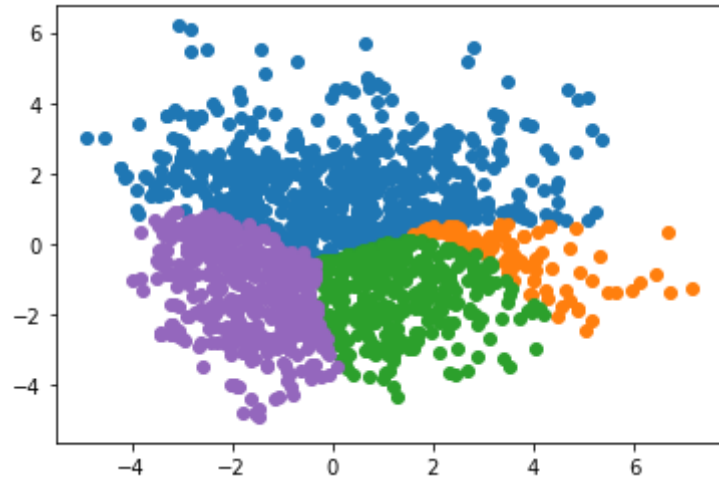
```
<matplotlib.collections.PathCollection at 0x7fa6964a8a90>
```



```
# color
n_clusters=5
GMM = GaussianMixtureModel(n_clusters=n_clusters)
GMM.fit(X_pca)
pred = GMM.predict(X_pca)
for i in range(n_clusters):
    plt.scatter(X_pca[pred==i][:,0],X_pca[pred==i][:,1])
```

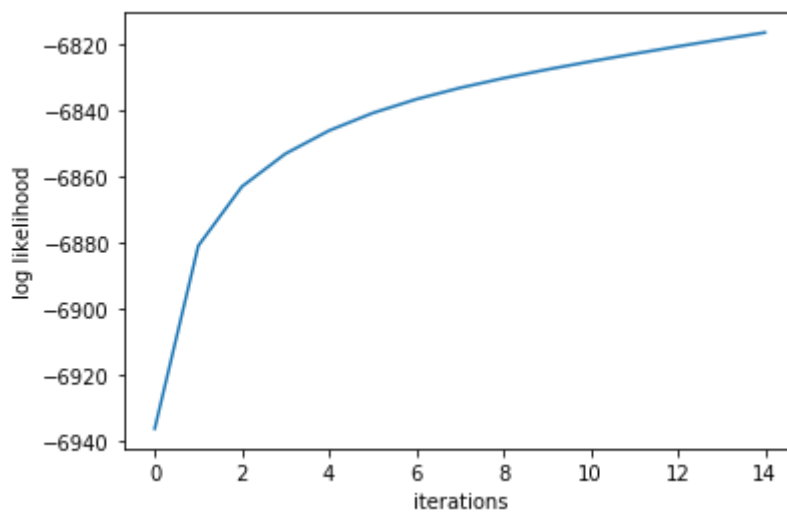
```
step 0 log likelihood: -6936.498042841801
step 1 log likelihood: -6881.033094855808
step 2 log likelihood: -6863.089359218
step 3 log likelihood: -6853.1139641404225
step 4 log likelihood: -6846.103005940182
step 5 log likelihood: -6840.786948358567
step 6 log likelihood: -6836.593528419803
step 7 log likelihood: -6833.16042820256
step 8 log likelihood: -6830.221247496595
```

```
step 9 log likelihood: -6827.586635106269
step 10 log likelihood: -6825.138137541355
step 11 log likelihood: -6822.813001653121
step 12 log likelihood: -6820.5832604964835
step 13 log likelihood: -6818.437720031083
step 14 log likelihood: -6816.3710552946595
```



```
import matplotlib.pyplot as plt
plt.plot(range(0, max_iters), GMM.history['log_likelihood'])
plt.xlabel("iterations")
plt.ylabel("log likelihood")
```

```
Text(0, 0.5, 'log likelihood')
```



This time, with low dimensional data, we can see the smooth curve without fluctuation.

Another 2D data

```
# We generate a dummy data
test_data = np.vstack([np.random.multivariate_normal([0,0],np.identity(2), size=
(500,)),
                        np.random.multivariate_normal([-1,3],np.array([[2, 1],
                                                                    [1, 1]]),
                                                                    size=(500,))])])
```

See how the algo converge..with animation

Animation

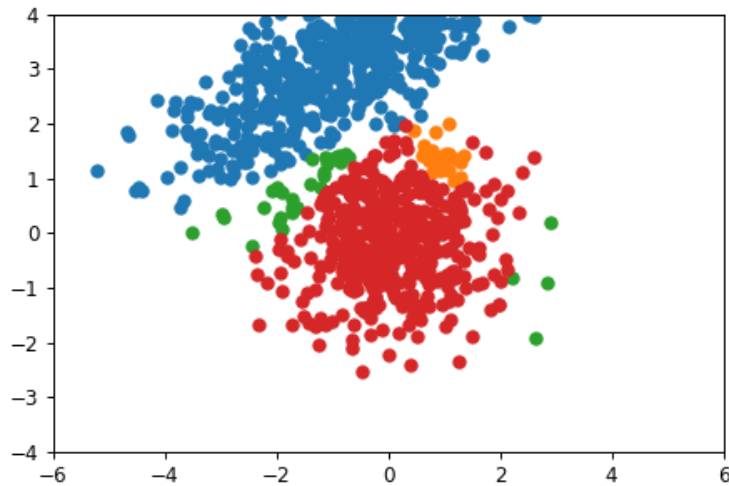
```
n_clusters=4
GMM = GaussianMixtureModel(n_clusters=n_clusters)
GMM.fit(test_data,max_iters=1)

pred = GMM.predict(test_data)
def update(curr):
    if curr==0:
        time.sleep(1)
    # check if animation is at the last frame, and if so, stop the animation a
    plt.cla()
    plt.axis([-6,6, -4,4])
    GMM.step(test_data)
    pred = GMM.predict(test_data)
    if curr == 50:
        a.event_source.stop()
    for i in range(n_clusters):
        plt.scatter(test_data[pred==i][:,0],test_data[pred==i][:,1])

fig = plt.figure()
a = animation.FuncAnimation(fig, update, interval=300)
```

```
step 0 log likelihood: -3597.0089883804017
```

```
<IPython.core.display.Javascript object>
```



Anomaly Detection

```
from scipy.io import loadmat
data = loadmat('cardio.mat')
X = data['X']
y = data['y'][:,0]
X_neg = X[y==0]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
X_train_pos = X_train[y_train==1]
X_train_neg = X_train[y_train==0]
```

```
max_iters = 15
n_clusters = 50
model = GaussianMixtureModel(n_clusters=n_clusters)
model.fit(X_neg,max_iters=max_iters)
model.fit_anomaly(X_train,y_train)
y_pred = model.predict_anomaly(X_test)
print("f_score",f1_score(y_test,y_pred))
```

```
step 0 log likelihood: -29868.031893025778
step 1 log likelihood: -25992.91774279897
step 2 log likelihood: -12006.193659331431
step 3 log likelihood: -11539.055401766866
step 4 log likelihood: 1868.388206985157
step 5 log likelihood: -2675.832581853343
step 6 log likelihood: -1883.259437267283
step 7 log likelihood: -1321.4469390036493
```



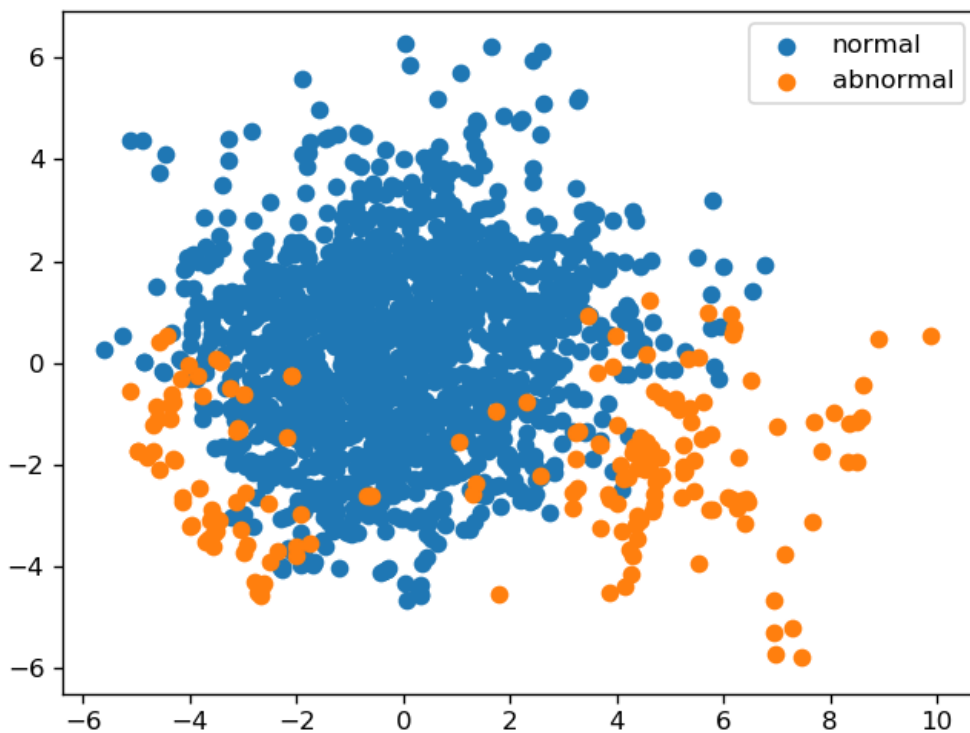
```
step 8 log likelihood: -1095.7346028413508
step 9 log likelihood: 1419.0057023779998
step 10 log likelihood: 390.158272769224
step 11 log likelihood: 2504.339117670591
step 12 log likelihood: 607.6431981096216
step 13 log likelihood: -1108.3172664638541
step 14 log likelihood: 4360.656247917903
f_score 0.6073298429319371
```

Visualization

For the sake of visualization, we'll transfer our data to a lower dimensional space using PCA with `n_components = 2` (2D)

```
from sklearn.decomposition import PCA
# We use PCA to chop down component for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.figure()
plt.scatter(X_pca[y==0][:,0],X_pca[y==0][:,1])
plt.scatter(X_pca[y==1][:,0],X_pca[y==1][:,1])
plt.gca().legend(['normal', 'abnormal'])
```

<IPython.core.display.Javascript object>



```
<matplotlib.legend.Legend at 0x7f95ba308dd0>
```

Looks like a lot of abnormal points are very close to normal one, which makes this a hard problem

```
X_pca_train, X_pca_test, y_pca_train, y_pca_test = train_test_split(X_pca, y,
test_size=0.3, random_state=42)
X_pca_neg = X_pca[y==0]
max_iters = 20
n_clusters = 10
model = GaussianMixtureModel(n_clusters=n_clusters)
model.fit(X_pca_neg, max_iters=max_iters)
model.fit_anomaly(X_train, y_train)
print("f_score", f1_score(model.predict_anomaly(X_test), y_test))
```

```
step 0 log likelihood: -6799.681463511612
step 1 log likelihood: -6780.82062910768
step 2 log likelihood: -6772.3958123271295
step 3 log likelihood: -6767.136885233344
step 4 log likelihood: -6763.18008116076
step 5 log likelihood: -6759.795269089522
step 6 log likelihood: -6756.624989256733
step 7 log likelihood: -6753.475115096862
step 8 log likelihood: -6750.240268912708
step 9 log likelihood: -6746.875841882809
step 10 log likelihood: -6743.387674776432
step 11 log likelihood: -6739.827672285434
step 12 log likelihood: -6736.287476629035
step 13 log likelihood: -6732.882884812425
step 14 log likelihood: -6729.727817547063
step 15 log likelihood: -6726.906743456097
step 16 log likelihood: -6724.4586743077125
step 17 log likelihood: -6722.378712136084
step 18 log likelihood: -6720.631871325933
step 19 log likelihood: -6719.169276134922
f_score 0.5471698113207547
```

```
/home/phat-ngu/.conda/envs/tf2/lib/python3.7/site-
packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: F-score
is ill-defined and being set to 0.0 due to no predicted samples.
'precision', 'predicted', average, warn_for)
```

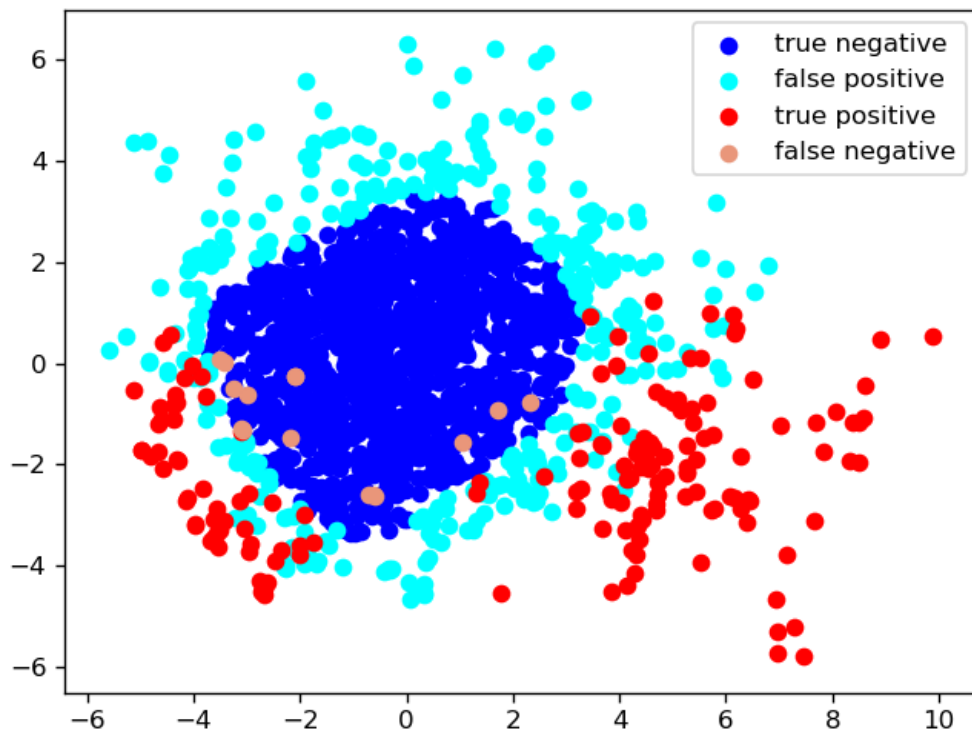
```
y_pred = model.predict_anomaly(X_pca)
X_true_pos = X_pca[np.logical_and((y_pred==1), (y==1))]
X_true_neg = X_pca[np.logical_and((y_pred==0), (y==0))]
X_false_pos = X_pca[np.logical_and((y_pred==1), (y!=1))]
X_false_neg = X_pca[np.logical_and((y_pred==0), (y!=0))]
```

Visualize the classification result

```
plt.figure()

plt.scatter(X_true_neg[:,0],X_true_neg[:,1],color="blue")
plt.scatter(X_false_pos[:,0],X_false_pos[:,1],color="cyan")
plt.scatter(X_true_pos[:,0],X_true_pos[:,1],color="red")
plt.scatter(X_false_neg[:,0],X_false_neg[:,1],color='darksalmon')
plt.gca().legend(['true negative','false positive','true positive','false
negative'])
```

<IPython.core.display.Javascript object>



<matplotlib.legend.Legend at 0x7f95ba28d950>