COMP540 STATISTICAL MACHINE LEARNING HW6

Chuanwenjie Wei, Yiqing Lu

April 14, 2017

1 EM for mixtures of Bernoullis

•MLE ESTIMATE

In M step, we already know the identity each x belongs to, so the complete likelihood function is supposed to be:

$$\begin{aligned} L(D;\pi,\mu) &= \prod_{i=1}^{m}\prod_{k=1}^{K}\left[p\left(z^{(i)}=k;\pi\right)p\left(x^{(i)}/z^{(i)}=k;\mu\right)\right]^{I\left(z^{(i)}=k\right)} \\ &= \prod_{i=1}^{m}\prod_{k=1}^{K}\left[\pi_k Bernoulli\left(x^{(i)},\mu_k\right)\right]^{I\left(z^{(i)}=k\right)} \end{aligned}$$

Take log on both sides, we derive:

$$LL(D;\pi,\mu) = \sum_{i=1}^{m}\sum_{k=1}^{K} I\left(z^{(i)}=k\right) log\left[\pi_k Bernoulli\left(x^{(i)},\mu_k\right)\right]$$

So the expected complete log likelihood function is:

$$\begin{aligned} E\left(LL_c(D;\pi,\mu)\right) &= E\left[\sum_{i=1}^{m}\sum_{k=1}^{K} I\left(z^{(i)}=k\right)\left(log\pi_k + logBern\left(x^{(i)},\mu_k\right)\right)\right] \\ &= \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)} log\pi_k + \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)} log\left[\mu_k^{x^{(i)}}\left(1-\mu_k\right)^{1-x^{(i)}}\right] \\ &= \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)} log\pi_k + \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)}\left[x^{(i)} log\mu_k + \left(1-x^{(i)}\right) log\left(1-\mu_k\right)\right] \end{aligned}$$

So we can get the MLE of $\mu_k$when:

$$\frac{\partial ELL\left(D;\pi,\mu\right)}{\partial\mu_k} = 0$$

$$\Longrightarrow\sum_{i=1}^{m}\gamma_k^{(i)}\left[\frac{x^{(i)}}{\mu_k}-\frac{1-x^{(i)}}{1-\mu_k}\right] = 0$$

$$\Longrightarrow\mu_k = \frac{\sum\limits_{i=1}^{m}\gamma_k^{(i)}x^{(i)}}{\sum\limits_{i=1}^{m}\gamma_k^{(i)}}$$

•MAP ESTIMATE

Since

$$\mu_k \sim Beta\left(\alpha,\beta\right)$$

So the likelihood function is:

$$L\left(D;\pi,\mu,\alpha,\beta\right) = \prod_{i=1}^{m}\prod_{k=1}^{K}\left[\pi_k Bernoulli\left(x^{(i)},\mu_k\right)\right]^{I\left(z^{(i)}=k\right)}\times\prod_{k=1}^{K}Beta\left(\mu_k;\alpha,\beta\right)$$

Therefore the expected complete log likelihood function is:

$$E\left(LL_c\left(D;\pi,\mu\right)\right) = E\left[\sum_{i=1}^{m}\sum_{k=1}^{K}I\left(z^{(i)}=k\right)\left(log\pi_k+logBern\left(x^{(i)},\mu_k\right)\right)+\sum_{k=1}^{K}logBeta\left(\mu_k;\alpha,\beta\right)\right]$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)}log\pi_k+\sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)}log\left[\mu_k^{x^{(i)}}\left(1-\mu_k\right)^{1-x^{(i)}}\right]+\sum_{k=1}^{K}log\left(\mu_k^{\alpha-1}\left(1-\mu_k\right)^{\beta-1}\right)$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)}log\pi_k+\sum_{i=1}^{m}\sum_{k=1}^{K}\gamma_k^{(i)}\left[x^{(i)}log\mu_k+\left(1-x^{(i)}\right)log\left(1-\mu_k\right)\right]+\sum_{k=1}^{K}\left[\left(\alpha-1\right)log\mu_k+\left(\beta-1\right)log\left(1-\mu_k\right)\right]$$

Set

$$\frac{\partial ELL\left(D;\pi,\mu\right)}{\partial\mu_k} = 0$$

we get:

$$\sum_{i=1}^{m}\gamma_k^{(i)}\left[\frac{x^{(i)}}{\mu_k}-\frac{1-x^{(i)}}{1-\mu_k}\right]+\frac{\alpha-1}{\mu_k}-\frac{\beta-1}{1-\mu_k} = 0$$

So

$$\mu_k = \frac{\sum\limits_{i=1}^{m}\gamma_k^{(i)}x^{(i)}+\alpha-1}{\sum\limits_{i=1}^{m}\gamma_k^{(i)}+\alpha+\beta-2}$$

2 Principal Components Analysis

- **I'm afraid there is something wrong with the notation: $f_u(x)$ can be written as the form of $f_u(x) = u^T xu$ when $u^T u = 1$ but you suppose uu^T to be 1 in this problem. Below, I will use the notation $u^T u = 1$.**

Since

$$f_u(x) = argmin_{v \in V} ||x - v||^2$$

where

$$V = \{au : \alpha \in \Re\}$$

So

$$\begin{aligned} f_u(x) &= \left(argmin_a ||x - au||^2\right) u \\ &= \left(argmin_\alpha \left(x^T x - 2au^T x + a^2 u^T u\right)\right) u \\ &= \left(-\frac{-2u^T x}{2u^T u}\right) u = u^T xu \end{aligned}$$

Then

$$\begin{aligned} argmin_{u:u^T u=1} \sum_{i=1}^{m} ||x^{(i)} - f_u\left(x^{(i)}\right)||^2 &= argmin_{u:u^T u=1} \sum_{i=1}^{m} ||x^{(i)} - u^T x^{(i)} u||^2 \\ &= argmin_{u:u^T u=1} \sum_{i=1}^{m} \left(x^{(i)} - u^T x^{(i)} u\right)^T \left(x^{(i)} - u^T x^{(i)} u\right) \\ &= argmin_{u:u^T u=1} \sum_{i=1}^{m} \left(x^{(i)T} x^{(i)} - 2\left(u^T x^{(i)}\right)^2 + u^T u \left(u^T x^{(i)}\right)^2\right) \\ &= argmin_{u:u^T u=1} \sum_{i=1}^{m} \left(-u^T x^{(i)}\right)^2 \\ &= argmax_{u:u^T u=1} u^T \left(\sum_{i=1}^{m} x^{(i)T} x^{(i)}\right) u \end{aligned}$$

We also know that x's have zero mean, and thus x corresponds to the first principal component for the data.

3. K-means clustering

3.1 Finding closest centroids

The function of finding closest centroids in utils_kmeans.py is shown in Fig.1.

```
######################### YOUR CODE HERE ########################################
# Instructions: Go over every example, find its closest centroid, and store     #
#               the index in the array idx at the appropriate location.         #
#               Concretely, idx[i] should contain the index of the centroid     #
#               closest to example i. Hence, it should be a value in the        #
#               range 0..K-1                                                    #
################################################################################

for i in range(X.shape[0]):
    min_dist = np.inf
    min_id = 0
    for j in range(K):
        temp = np.sum((X[i,:]-centroids[j,:])**2)
        if (min_dist > temp):
            min_dist = temp
            min_id = j
        idx[i]=min_id

################################################################################
#             END OF YOUR CODE                                                 #
################################################################################
```

Fig.1 The function of finding closest centroids

And the result of finding the closest centroids for the first 3 examples is shown in Fig.2.

```
Finding closest centroids.
Closest centroids for the first 3 examples: (should be [0 2 1]):  [0 2
1]
```

Fig.2 The closest centroids for the first 3 examples

3.2 Computing centroid means The function compute centroids in utils_kmeans.py is shown in Fig.3.

```
########################= YOUR CODE HERE ######################################
# Instructions: Go over every centroid and compute mean of all points that    #
#               belong to it. Concretely, the row vector centroids[i,:]       #
#               should contain the mean of the data points assigned to        #
#               centroid i.                                                   #
###############################################################################

for i in range(K):
    count = 0
    sum = np.zeros((1,X.shape[1]))
    for j in range(X.shape[0]):
        if (idx[j] == i):
            sum = sum + X[j,:]
            count = count + 1
    centroids[i,] = sum / count

################################################################################
#             END OF YOUR CODE                                                 #
################################################################################
```

Fig.3 The function of computing centroids

And the result of computing centroids after initial finding of closest centroids is shown in Fig.4.

```
Computing centroids means.
(3L, 2L)
(3L, 2L)
(3L, 2L)
Centroids computed after initial finding of closest centroids:
[[ 2.42830111  3.15792418]
 [ 5.81350331  2.63365645]
 [ 7.11938687  3.6166844 ]]
(the centroids should be
   [ 2.428301 3.157924 ],  [ 5.813503 2.633656 ], [ 7.119387 3.616684 ]
```

Fig.4 The result of centroids we computed

- **k-means on example dataset**

The result of runing the k-means algorithm on a toy 2D dataset is shown in Fig.5.

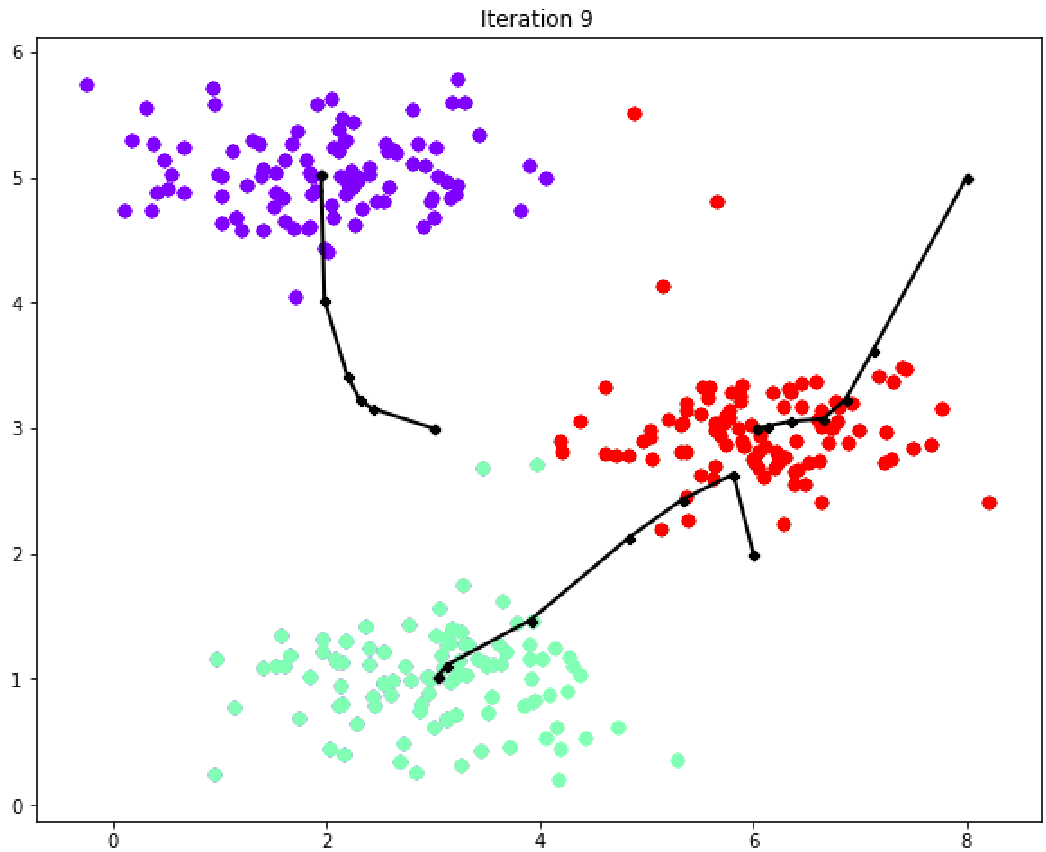

Fig.5 The result of k-means algorithm on the dataset

3.3 Random initialization The function kmeans init centroids in utils_kmeans.py is shown in Fig 6.

```
######################= YOUR CODE HERE ####################################
#  Construct a random permutation of the examples and pick the first K items  #
##############################################################################

temp_1 = np.random.permutation(X.shape[0])
temp_2 = X[temp_1,]
centroids = temp_2[:K , ]

###############################################################################
#              END OF YOUR CODE                                               #
###############################################################################
```

Fig.6 The function kmeans init centroids

• **Image compression with k-means** The result of compressing the image by using k-means algorithm is shown in Fig.7.

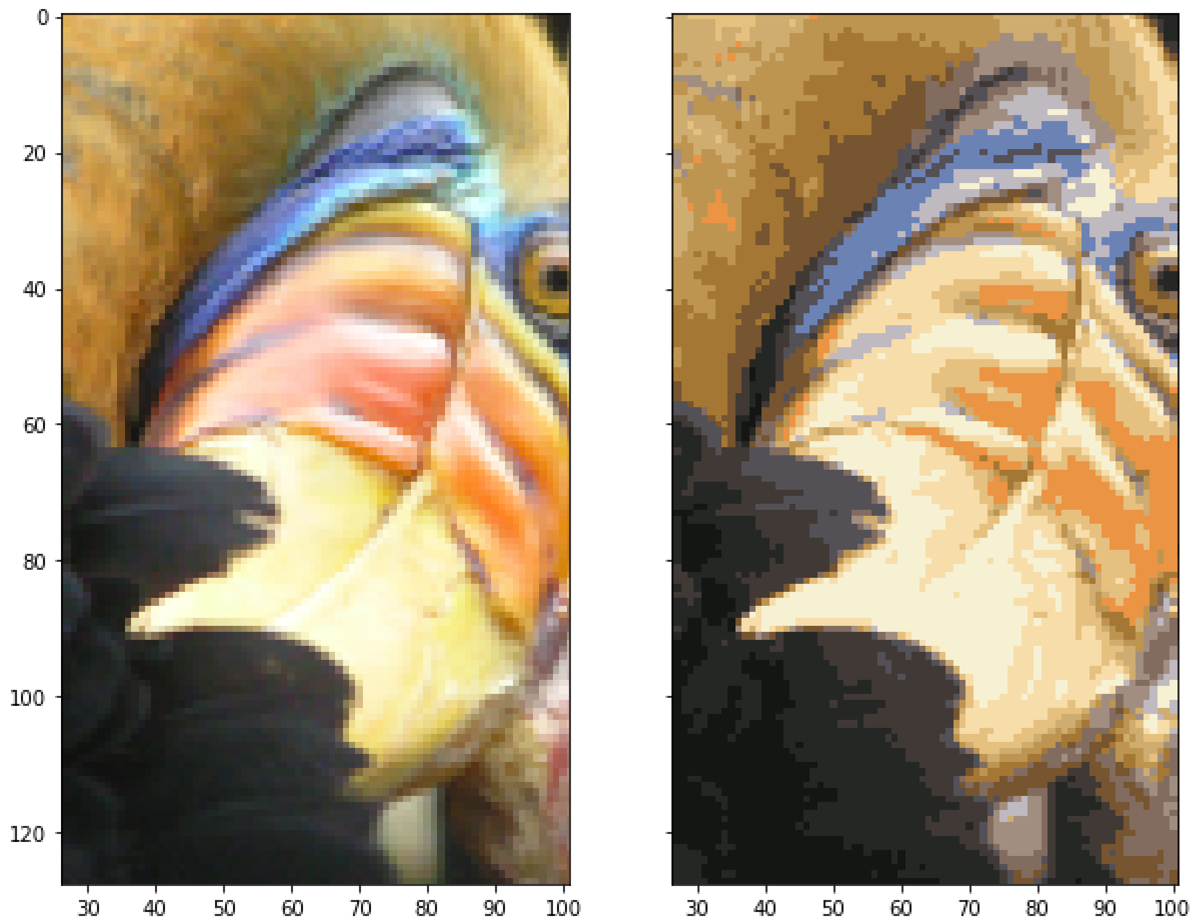

Fig.7 Original and reconstructed image

4. Principal Components Analysis

4.1 Implementing PCA

The function pca in utils_pca.py is shown in Fig.8.

```
##########################################################
#         YOUR CODE HERE                                 #
##########################################################

# compute the covariance of X and then use the
# svd function to compute the eigenvectors and
# eigenvalues of the covariance matrix

# When computing the covariance remember to divide by
# the number of rows in X

Sigma = np.dot(X.T ,X)/X.shape[0]
U,S,V=np.linalg.svd( Sigma , full_matrices=False)

#########################################################
#            END YOUR CODE                              #
#########################################################
```

Fig.8 The function pca in utils_pca.py

And the result of visualizing the eigenvectors is shown in Fig.9.

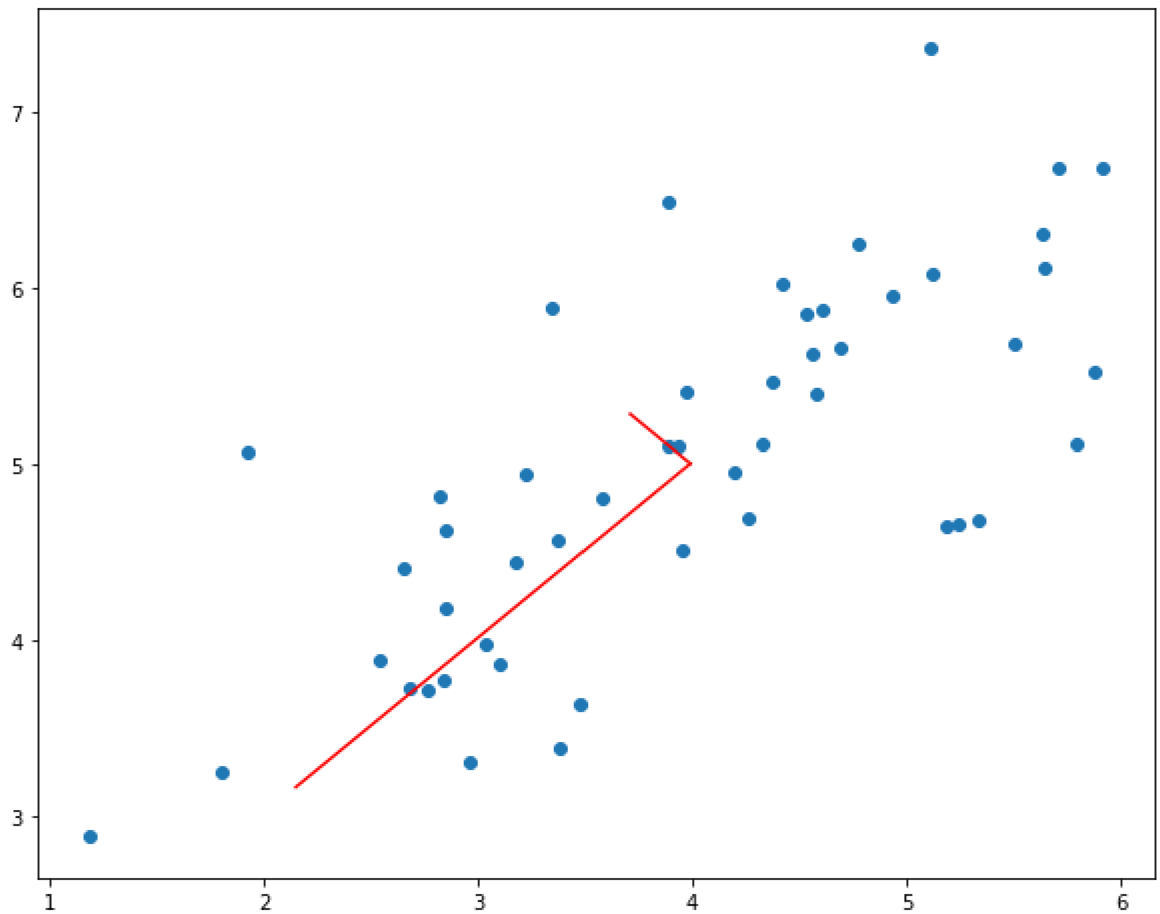

Fig.9 Computed eigenvectors of the dataset

4.2 Projecting the data onto the principal components

The function project data in utils_pca.py is shown in Fig.10.

```
##########################################################
#         YOUR CODE HERE                                 #
##########################################################

Z = np.zeros((X.shape[0], K))
U_temp = U[:,:K]
for i in range(X.shape[0]):
    Z[i,:] = np.dot(U_temp.T , X[i,:])

#########################################################
#           END YOUR CODE                               #
#########################################################
```

Fig.10 The function project data in utils_pca.py

4.3 Reconstructing an approximation of the data

The function recover data in utils_pca.py is shown in Fig.11.

```
##########################################################
#         YOUR CODE HERE                                 #
##########################################################

X_rec = np.zeros((Z.shape[0], U.shape[0]))
U_temp = U[:,:K]
for i in range(Z.shape[0]):
    X_rec[i,:] = np.dot(U_temp, Z[i,:])

#########################################################
#           END YOUR CODE                               #
#########################################################
```

Fig.11 The function recover data in utils_pca.py

The result of the projection and approximation of the first data is shown in Fig.12.

```
The projection of the first example (should be about 1.496)  [ 1.4963126
1]
Approximation of the first example (should be about [-1.058 -1.058])
[-1.05805279 -1.05805279]
```

Fig.12 The projection and approximation of the first data

• **Visualizing the projections** The result of projecting data by PCA is shown in Fig.13.

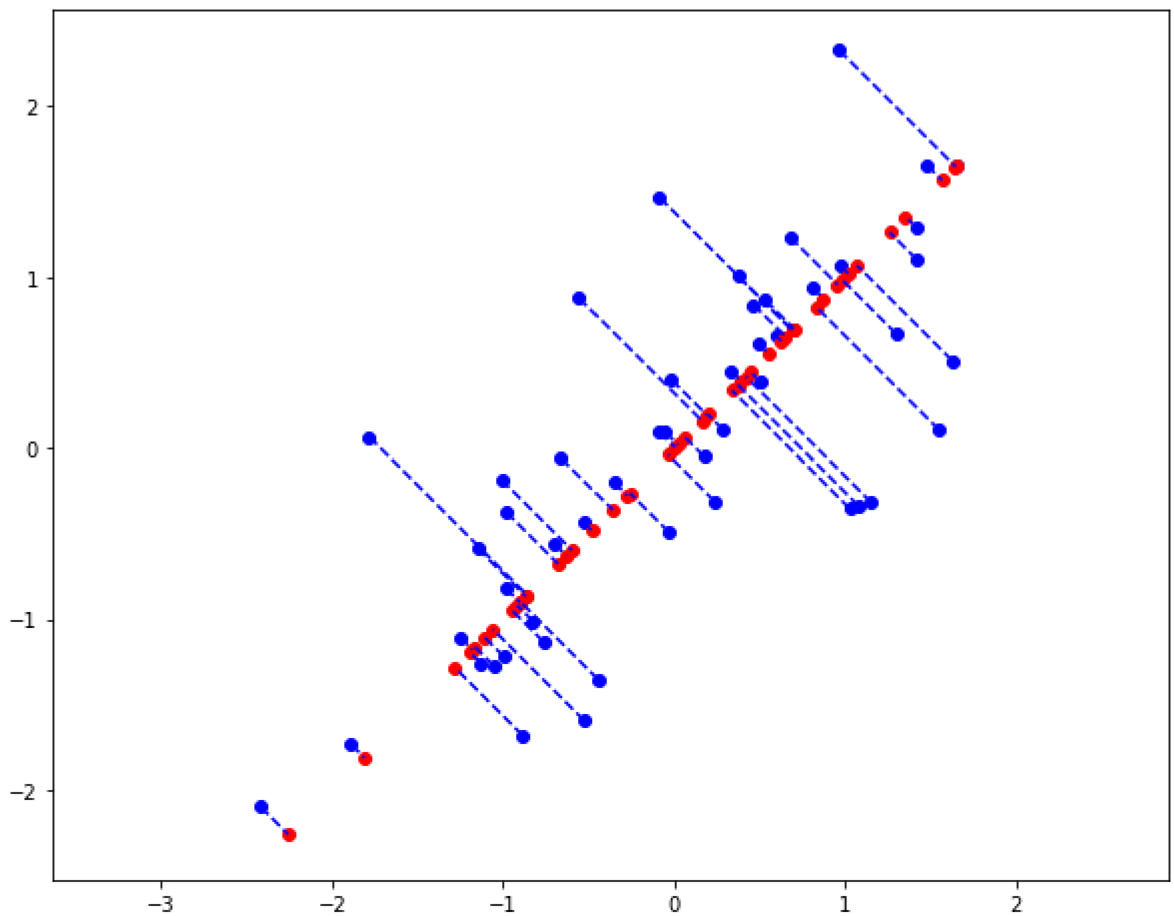

Fig.13 The normalized and projected data after PCA

• **Face image dataset** The result of eigenvalues of the face data set after running PCA is shown in Fig 14.

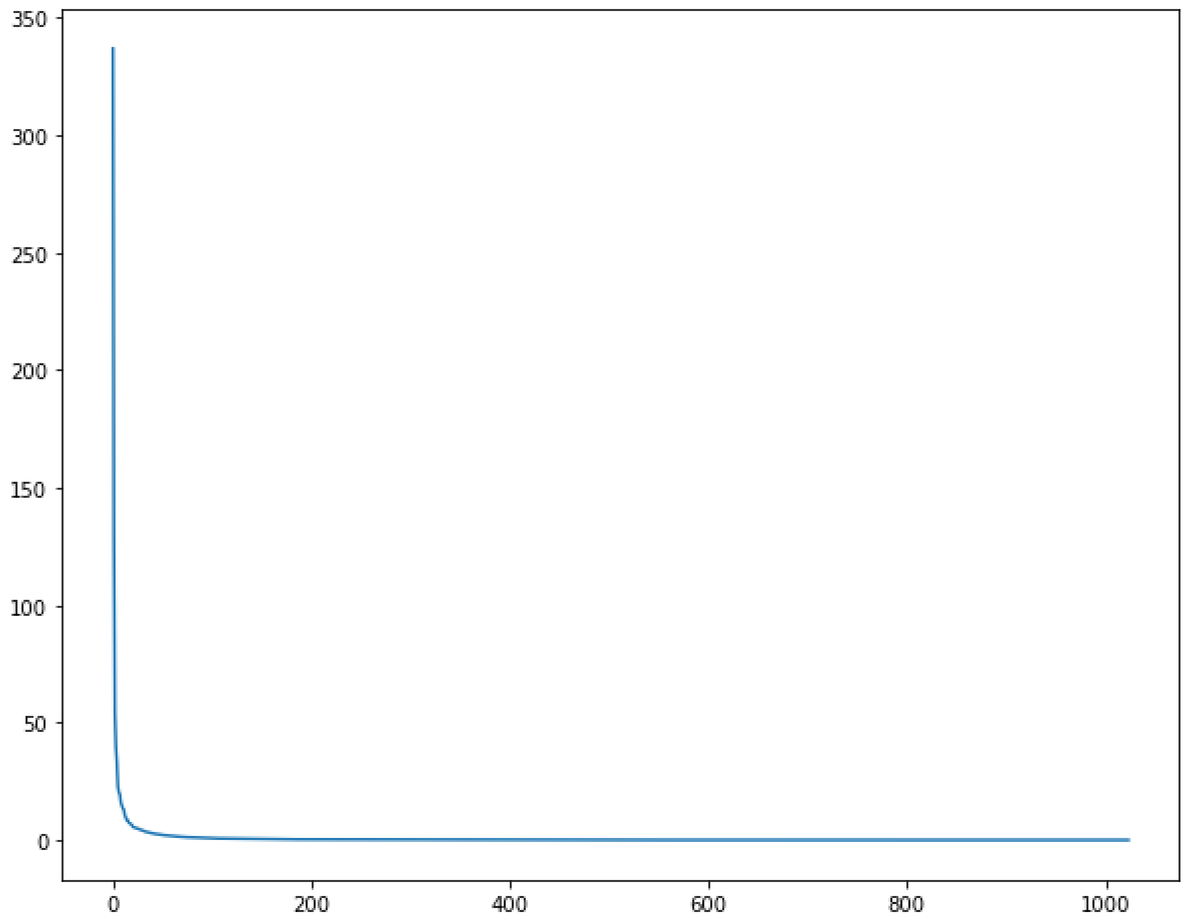

Fig.14 The eigenvalues of the face data set

The result of the top 25 eigenfaces is shown in Fig.15.

Fig.15 Principal components on the face dataset

• **Dimensionality reduction** The result of the recovered faces constructed out of top 100 principal components is shown in Fig.16.

Fig.16 The result of the recovered faces reconstructed from only the top 100 principal components

5. Anomaly detection

5.1 Estimating parameters of a Gaussian distribution

The function estimate_gaussian in utils_anomaly.py is shown in Fig.17.

```
################################################################
#                  YOUR CODE HERE                              #
################################################################

mu = np.average(X,axis=0)
var = np.var(X,axis=0,ddof=0)

################################################################
#                  END YOUR CODE                               #
################################################################
```

Fig.17 The function estimate_gaussian

• **Visualize the contours of the fitted Gaussian distribution** The result is shown in Fig.18.

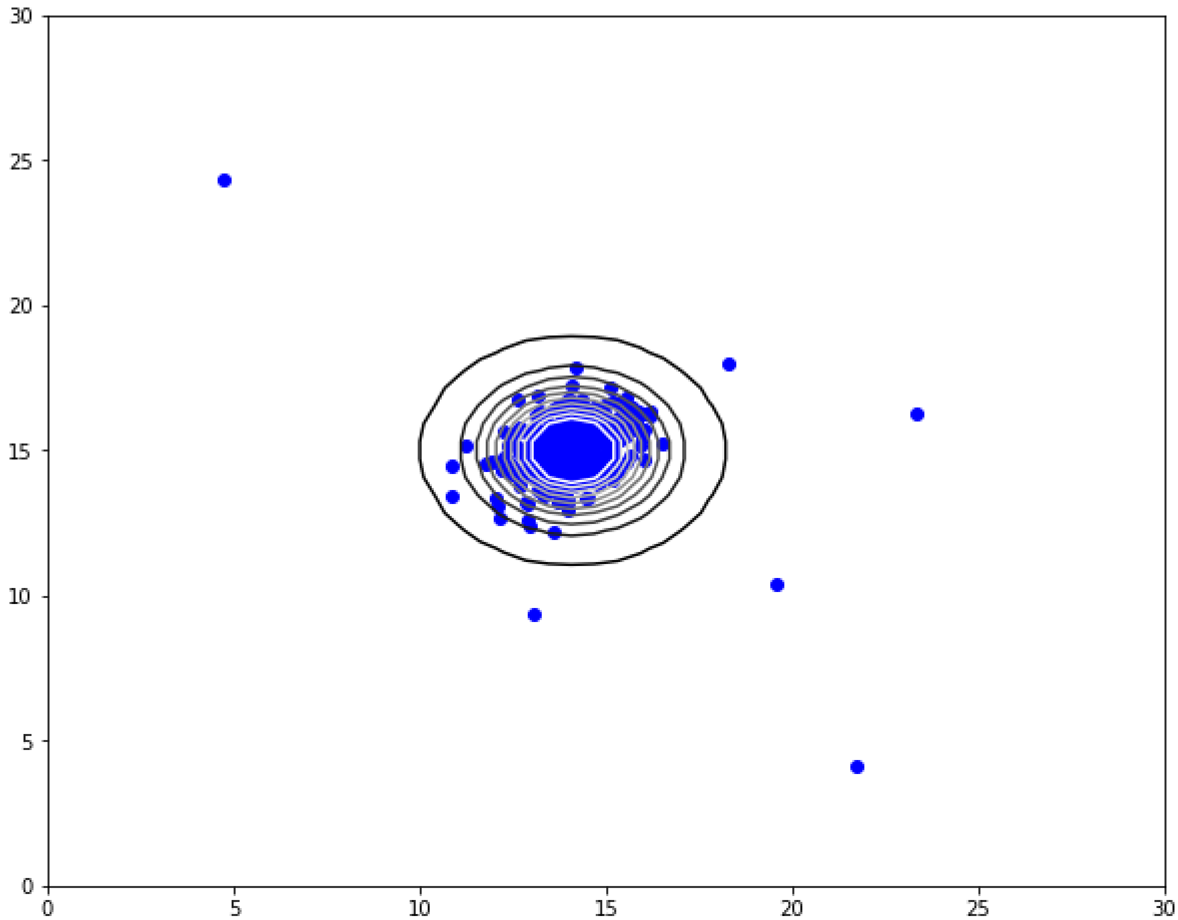

Fig.18 The Gaussian distribution contours of the distribution fit to the dataset

5.2 Selecting the threshold

The function select threshold in utils_anomaly.py is shown in Fig.19.

```
#####################################################################
#               YOUR CODE HERE                                      #
#####################################################################

mu = np.average(X,axis=0)
var = np.var(X,axis=0,ddof=0)

#####################################################################
#               END YOUR CODE                                       #
#####################################################################
```

Fig.19 The function select threshold

The result of epison = 8.99085277927e-05, the F1 score = 1.55555555556. The result of the classified anomalies is shown in Fig.20.

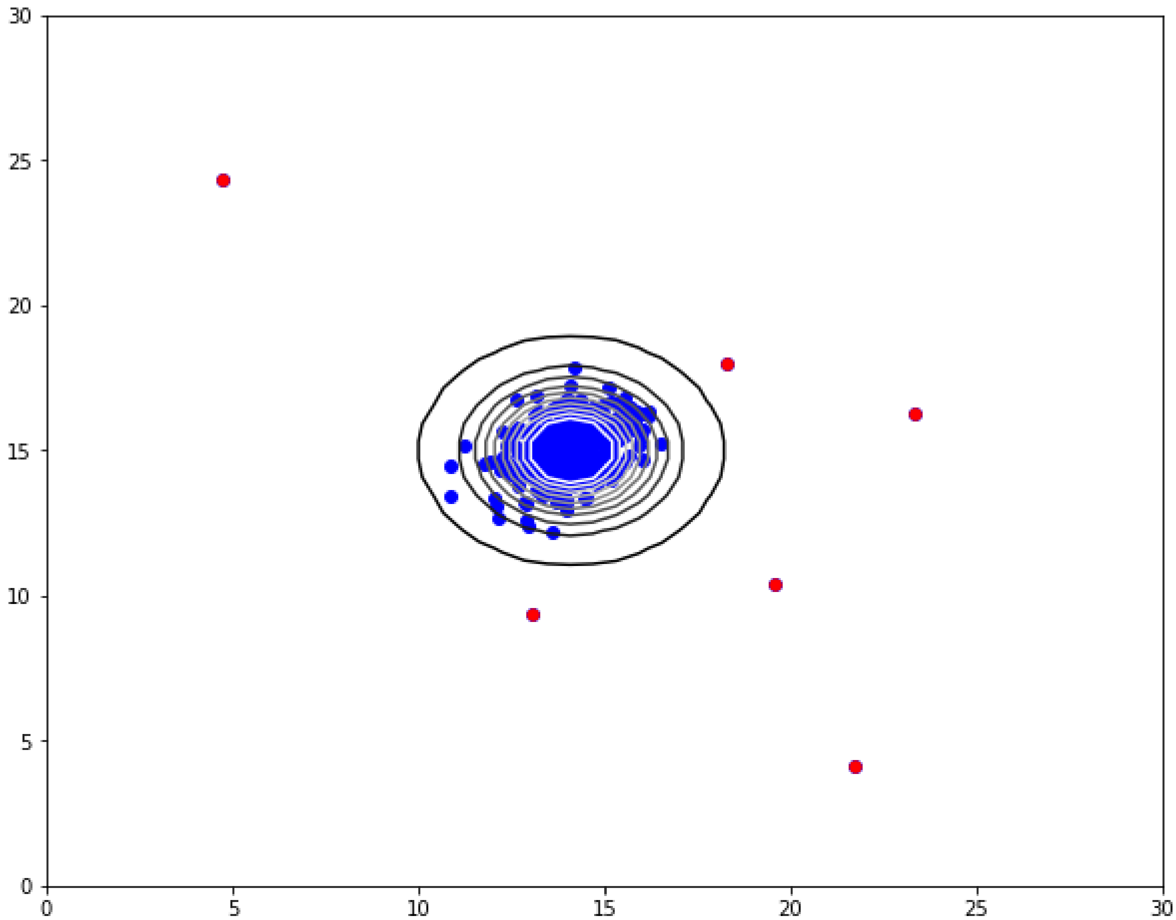

Fig.20 The classified anomalies

• **High dimensional dataset** By computing, we can get the best epison, best F1 score of the dataset 'anomalydata2.mat'.

best epison = 1.37722889076e-18, and best F1 score = 0.615384615385. And 117 anomalies are found.