ISYE6740-HW4

pkubsad

March 2021

(a) (5 points) Select one image of "2" and one image of "6", and visualize the two images.

Answer: Based on the distribution plots we can conclude that:

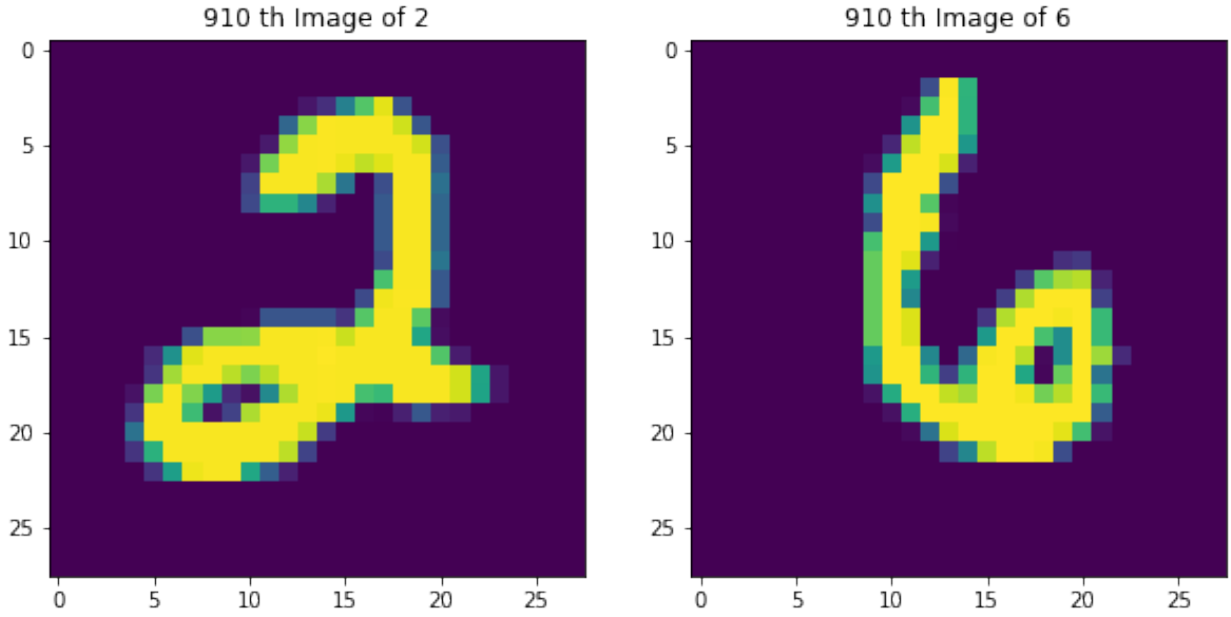


Figure 1: Image of 2 and 6

(b) (10 points) Write down detailed expression of the E-step and M-step in the EM algorithm

Answer:
Expectation step:

$$\tau_k^i = \frac{\pi_k N(x^i|\mu_k, \Sigma_k)}{\Sigma_{k'}^i \pi_{k'} N(x^i|\mu_{k'}, \Sigma_{k'})}$$

As per the homework walk through video,

$$N(x^i|\mu, \Sigma) = \frac{1}{\sqrt{2\pi^n}.\sqrt{|\Sigma|}} * exp(-\frac{1}{2}(X-\mu)^T.\Sigma^{-1}.(X-\mu))$$

$$\tau_k^i = \frac{\pi_k \frac{1}{\sqrt{2\pi^n}.\sqrt{|\Sigma_k|}} * exp(-\frac{1}{2}(X-\mu_k)^T.\Sigma_k^{-1}.(X-\mu_k))}{\Sigma_{k'}^i \pi_{k'} \frac{1}{\sqrt{2\pi^n}.\sqrt{|\Sigma_{k'}|}} * exp(-\frac{1}{2}(X-\mu_{k'})^T.\Sigma_{k'}^{-1}.(X-\mu_k'))}$$

$$\tau_k^i = \frac{\pi_k \frac{1}{\sqrt{|\Sigma_k|}} * exp(-\frac{1}{2}(X-\mu_k)^T.\Sigma_k^{-1}.(X-\mu_k))}{\Sigma_{k'}^i \pi_{k'} \frac{1}{\sqrt{|\Sigma_{k'}|}} * exp(-\frac{1}{2}(X-\mu_{k'})^T.\Sigma_{k'}^{-1}.(X-\mu_k'))}$$

Maximization step: update (μ_k, Σ_k, π_k)

$$\pi_k = \frac{\sum_i \tau_k^i}{m}$$

$$\mu_k = \frac{\sum_i \tau_k^i.x^i}{m}$$

$$\Sigma_k = \frac{\sum_i \tau_k^i (x^i - \mu_k)(x^i - \mu_k)^T}{\sum_i \tau_k^i}$$

where

$$(k = 1 \ldots K, i = 1 \ldots m)$$

(c) (15 points) Implement EM algorithm yourself. Plot the log-likelihood function versus the number of iterations to show your algorithm is converging.

Answer: I have used the demo code provided prof. X. Based on the output of the code, we can see there the log-likelihood varies until iteration 6 and then it settles down varying marginally.The algorithm finally converges after 22 iterations. Please find below the plot of log-likelihood vs iteration:

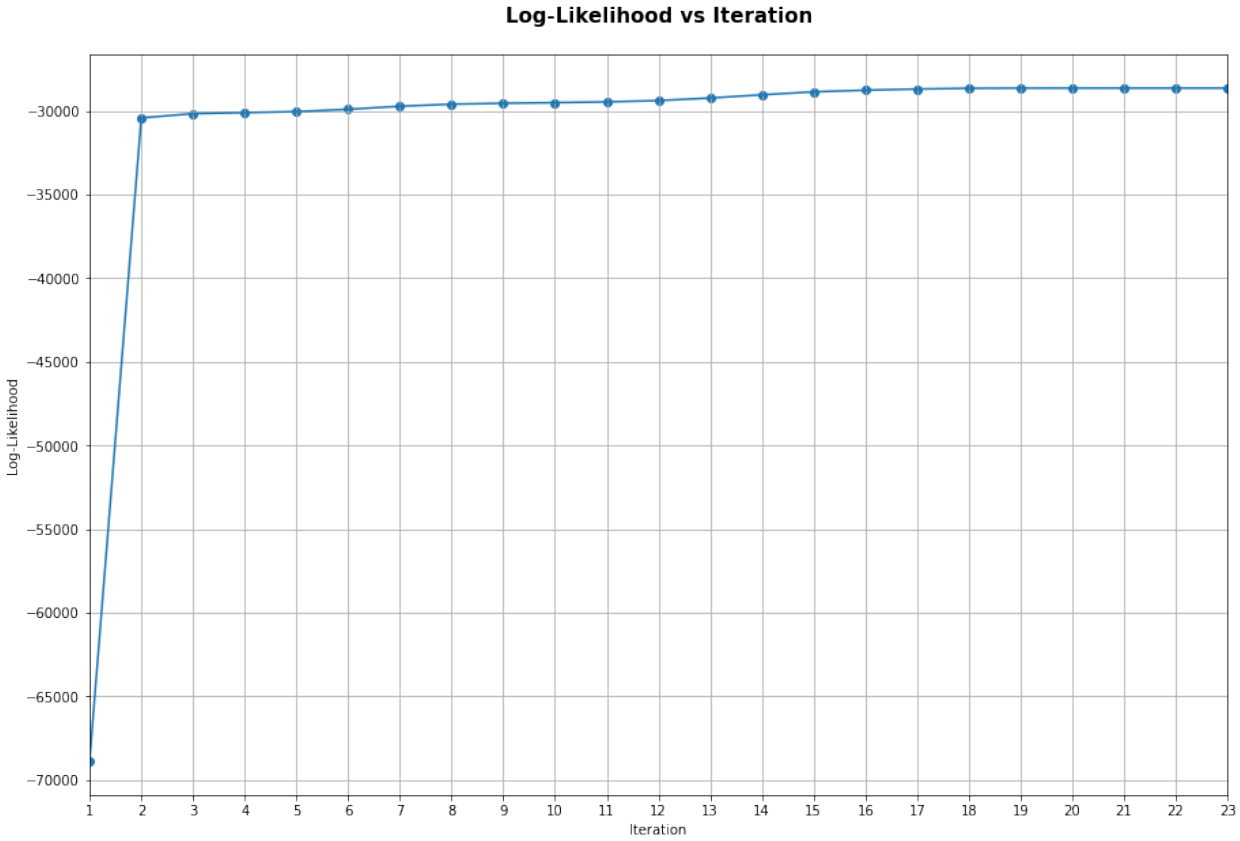


Figure 2: log-likelihood vs iterations

(d) (15 points) Report, the fitted GMM model when EM has terminated in your algorithms as follows. Make sure to report the weights for each component, and the mean of each component (you can reformat the vector to make them into 28-by-28 matrices and show images).

Answer: Mean values representation:

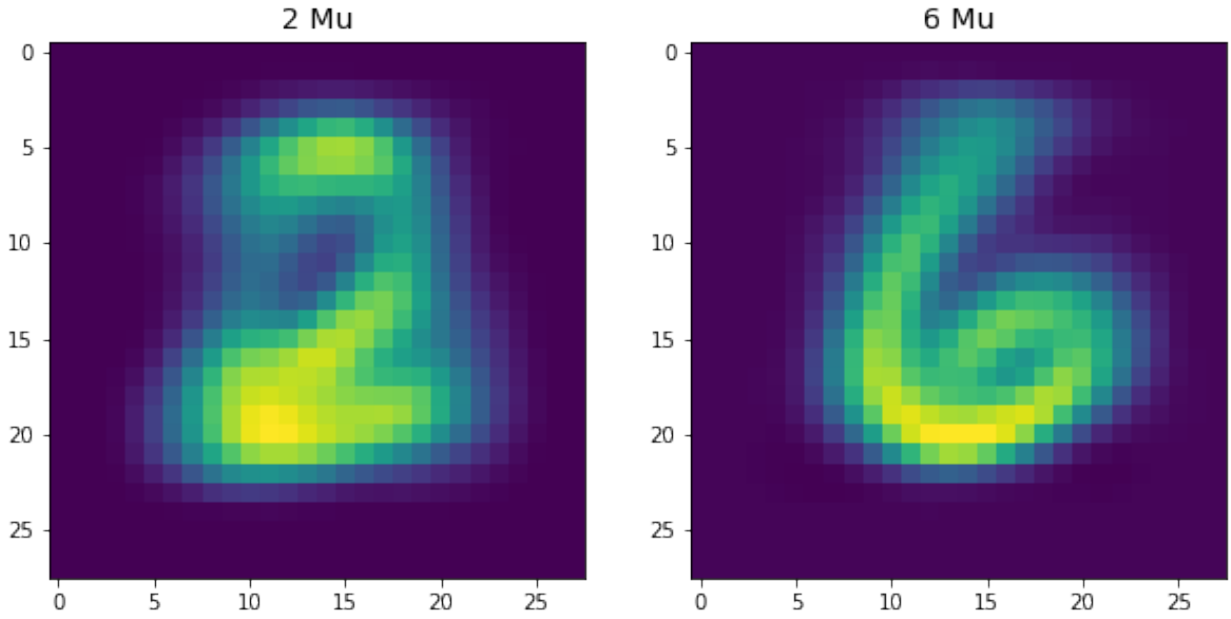


Figure 3: mean value representation of 2 components

Covariance values representation:

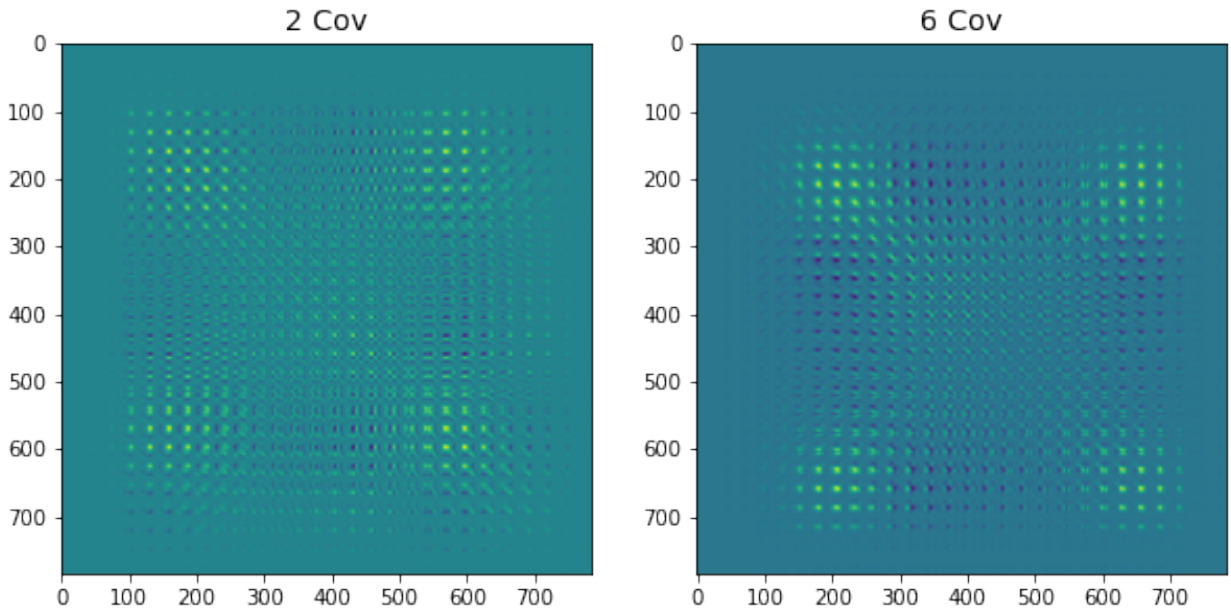


Figure 4: Covariance value representation of 2 components

(e) (15 points) Use the τ_k^i to infer the labels of the images, and compare with the true labels. Report the mis-classification rate for digits "2" and "6" respectively. Perform K-means clustering with $K = 2$ (you may call a package or use the code from your previous homework). Find out the mis-classification rate for digits

"2" and "6" respectively, and compare with GMM. Which one achieves the better performance?

Answer: Below is observations for number of miscalculations for 2 and 6:

C	EM	K-Means
2	51/1032= 4.94%	55/1032 = 5.33%
6	9/958 = 0.93%	68/958 = 7.10 %

Based on this observation, we can conclude EM algorithm performs better than K-Means clustering for this use case.

(2.1) (10 points) Show step-by-step mathematical derivation for the gradient of the cost function

Answer: As mentioned by prof in lecture, to calculate the gradient of the cost function, we will calculate partial derivative wrt theta.

$$l(\theta) = \sum_{i=1}^{n} \{-log(1 + exp{-}\theta x_i) + (y_i - 1)\theta x_i\}$$

$$\frac{\partial(l(\theta))}{\partial\theta} = \frac{\partial(\sum_{i=1}^{n}\{-log(1 + exp{-}\theta x_i) + (y_i - 1)\theta x_i\})}{\partial\theta}$$

$$\frac{\partial(l(\theta))}{\partial\theta} = \frac{\partial(\sum_{i=1}^{n}\{-log(1 + exp{-}\theta x_i)\}}{\partial\theta} + \frac{\partial\{(y_i - 1)\theta x_i\})}{\partial\theta}$$

Applying the chain rule to the derivative:

$$= \sum_{i=1}^{n} -\frac{(1 + exp\{-\theta x_i\})'}{(1 + exp\{-\theta x_i\})} + \sum_{i=1}^{n}(y_i - 1)x_i$$

$$= \sum_{i=1}^{n} \frac{x_i.exp\{-\theta x_i\}}{(1 + exp\{-\theta x_i\})} + \sum_{i=1}^{n}(y_i - 1)x_i$$

$$= \sum_{i=1}^{n} \frac{x_i.exp\{-\theta x_i\}}{(1 + exp\{-\theta x_i\})} + (y_i - 1)x_i$$

(2.2) (10 points) Write a pseudo-code for performing **gradient descent** to find the optimize

Answer: We cannot set the above function to 0 because that would not give closed formed solution. Instead we can perform gradient descent by following the below steps:

(a) Begin with randomly initialized point θ^0, in general, θ^i

(b) Select a step size γ and perform the below calculation:

$$\theta^{i+1} = \gamma.gradient\_function, f(\theta)$$

$$\theta^{i+1} = \gamma.\frac{\partial(l(\theta))}{\partial\theta}$$

$$\theta^{i+1} = \gamma.\sum_{i=1}^{n} \frac{x_i.exp\{-\theta x_i\}}{(1 + exp\{-\theta x_i\})} + (y_i - 1)x_i$$

(c) Define optimality condition, like a threshold value t. If θ^t - θ^{t+1} <= t, stop the process.

(d) Select decreasing step size at every iteration to avoid oscillation when the algorithm reaches optimal value.

(2.3) (10 points) Write the pseudo-code for performing the **stochastic gradient descent** algorithm to solve the training of logistic regression problem (**??**). Please explain the difference between gradient descent and stochastic gradient descent for training logistic regression.

Answer: Stochastic Gradient Descent (SGD) process:

(a) Split the entire data into K buckets.

(b) Perform the Gradient Descent algorithm mentioned above for each bucket.

(c) Update global parameters after ever bucket is executed.

(d) Iterate through all the buckets calculating the cost function for each bucket.

The main difference between Stochastic gradient descent (SGD) and gradient descent (GD) is, in GD we run the algorithm on the entire data set. If the data set is large, this is not an efficient way, wrt , compute and memory to iterate and calculate gradient descent. In SGD, we break the entire data into smaller buckets and perform GD on each bucket. This is lot faster compared to running GD on whole data set.

(2.4) (10 points) We will **show that the training problem in basic logistic regression problem is concave.** Derive the Hessian matrix of $\ell(\theta)$ and based on this, show the training problem (**??**) is concave (note that in this case, since we only have one feature, the Hessian matrix is just a scalar). Explain why the problem can be solved efficiently and gradient descent will achieve a unique global optimizer, as we discussed in class.

Answer: *Source* : *https* : *//math.wikia.org/wiki/Hessian_matrix* From the source mentioned above, definition of Hessian matrix , I am taking derivative of the gradient of the log likelihood function from Q2.1.

$$= \frac{\partial(\sum_{i=1}^{n} \frac{x_i.exp\{-\theta x_i\}}{(1+exp\{-\theta x_i\})} + (y_i - 1)x_i)}{\partial\theta}$$

$$= \frac{\partial(\sum_{i=1}^{n} \frac{x_i.exp\{-\theta x_i\}+(y_i-1)x_i(1+exp\{-\theta x_i\}))()}{(1+exp\{-\theta x_i\})}}{\partial\theta}$$

$$= \frac{\partial}{\partial}(\sum_{i=1}^{n} \frac{x_i[y_i exp\{-x_i\} + y_i - 1]}{1 + exp\{-\theta x_i\}})$$

$$= \sum_{i=1}^{n} \frac{\partial}{\partial\theta} \frac{(x_i.y_i.exp(-\theta x_i) + x_i y_i - x_i)}{1 + exp\{-\theta x_i\}}$$

Applying the chain rule and quotient rule to the derivative. (source: https://web.iit.edu/sites/web/files/departments/ac affairs/academic-resource-center/pdfs/product_quotient_rule.pdf)

$$= \sum_{i=1}^{n} \frac{(-x_i^2 * y_i.exp\{-\theta x_i\}).(1 + exp\{-\theta x_i\}) - (-x_i.exp\{-\theta x_i\}).(x_i.y_i.exp\{-\theta x_i\} + x_i.y_i - x_i)}{(1 + exp\{-\theta x_i\})^2}$$

$$= \sum_{i=1}^{n} \frac{-x_i^2.exp(-\theta x_i)}{1 + exp(-\theta x_i)^2}$$

This expression is a quadratic function which proves second derivative of the likelihood function is a concave optimization problem.

Q1

March 9, 2021

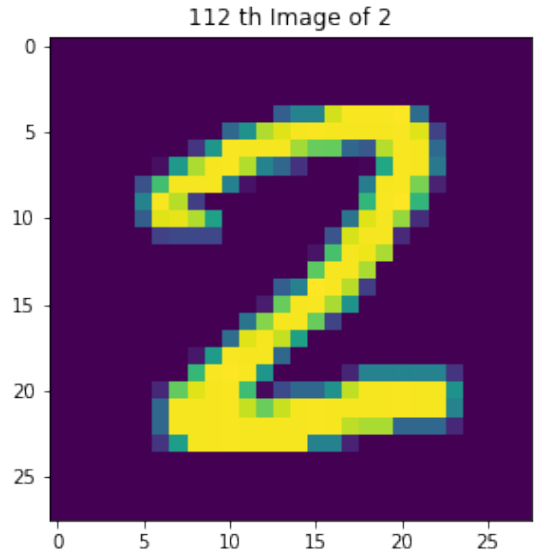
[1]:
```
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
import random
from scipy.stats import multivariate_normal as mvn
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

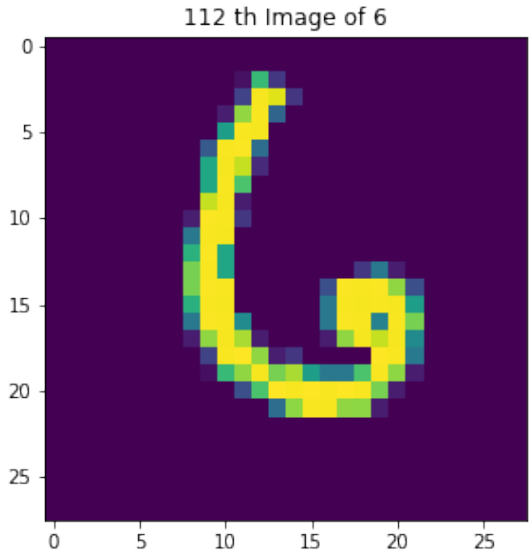
[2]:
```
data = scipy.io.loadmat('./data/data.mat')
data = data['data'].T
lbl = scipy.io.loadmat('./data/label.mat')
lbl = lbl['trueLabel']

print(data.shape)
print(lbl.shape)
display(lbl)
```

```
(1990, 784)
(1, 1990)
```

```
array([[2, 2, 2, ..., 6, 6, 6]], dtype=uint8)
```

[3]:
```
# Q1.a plotting random 2 and 6

fig, axes = plt.subplots(1,2,figsize=(10,5))
rand=random.randint(1,1032)
axes[0].imshow(data[np.argwhere(lbl==2)[:,1][rand],:].reshape((28,28)).T)
axes[0].set_title("{} th Image of 2".format(rand))
axes[1].imshow(data[np.argwhere(lbl==6)[:,1][rand],:].reshape((28,28)).T)
axes[1].set_title("{} th Image of 6".format(rand));
plt.show()
```

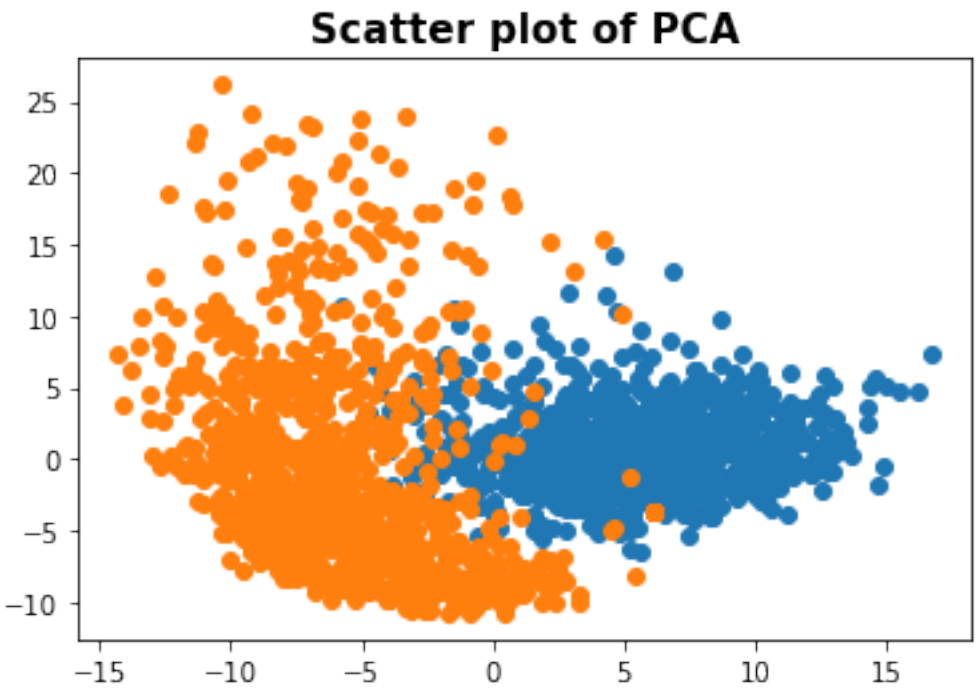




[4]:
```
# Q1c: Implementation of EM algorithm with initialization for mean and␣
 ↪initialization for covariance
data = scipy.io.loadmat('./data/data.mat')
data = data['data'].T
label = scipy.io.loadmat('./data/label.mat')
label = label['trueLabel']

scaler = preprocessing.StandardScaler().fit(data)
ndata = scaler.transform(data)


# # ndata = preprocessing.scale(data)
m, n = data.shape
# # (900,784)
# C = np.matmul(ndata.T, ndata)/m

# # pca the data
# d = 5  # reduced dimension
# V,ev,_ = np.linalg.svd(C)
# V = V[:, :d]

pca = PCA(n_components=5)
pdata= pca.fit_transform(ndata)


# project the data to the top 2 principal directions
# pdata = np.dot(ndata,V)
# display(pdata)
```

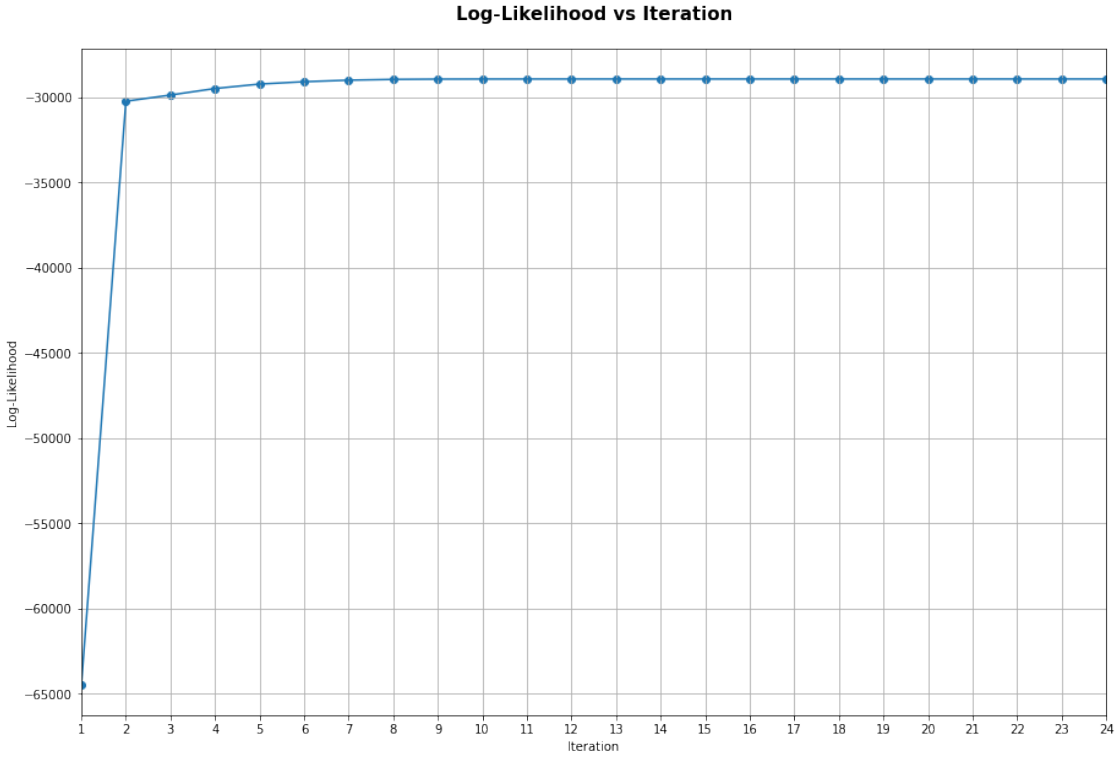
```
plt.scatter(pdata[np.where(label == 2),0],pdata[np.where(label == 2),1])
plt.scatter(pdata[np.where(label == 6),0],pdata[np.where(label == 6),1])
plt.title("Scatter plot of PCA", fontsize=15, weight='bold')
plt.show()

# EM-GMM
K = 2


# initialize prior
# np.random.seed(seed)
pi = np.random.random(K)
pi = pi/np.sum(pi)

# initial mean and covariance
# np.random.seed(seed)
mu = np.random.randn(K,5)
mu_old = mu.copy()

I=np.identity(5)
sigma = []
for ii in range(K):
    # to ensure the covariance psd
    # np.random.seed(seed)
    dummy = np.random.randn(5, 5)
    sigma.append(dummy@dummy.T+I)

# initialize the posterior
tau = np.full((m, K), fill_value=0.)

# # parameter for countour plot
# xrange = np.arange(-5, -5, 0.1)
# yrange = np.arange(-5, -5, 0.1)

# ####
maxIter= 100
tol = 1e-3

plt.ion()
logLH=[]
for ii in range(100):
    print("starting iteration,",ii)
    # E-step
    collection=[]
    for kk in range(K):
        tau[:, kk] = pi[kk] * mvn.pdf(pdata, mu[kk], sigma[kk])
```

```
    sum_tau = np.sum(tau, axis=1)
    sum_tau.shape = (m,1)
    l = np.sum(np.log(np.sum(tau, axis=1)))
    logLH.append(l)

    tau = np.divide(tau, np.tile(sum_tau, (1, K)))


    # M-step
    for kk in range(K):
        # update prior
        pi[kk] = np.sum(tau[:, kk])/m

        # update component mean
        mu[kk] = pdata.T @ tau[:,kk] / np.sum(tau[:,kk], axis = 0)

        # update cov matrix
        dummy = pdata - np.tile(mu[kk], (m,1)) # X-mu
        sigma[kk] = dummy.T @ np.diag(tau[:,kk]) @ dummy / np.sum(tau[:,kk],␣
↪axis = 0)



    if np.linalg.norm(mu-mu_old) < tol:
        print('training coverged')
        break
    mu_old = mu.copy()
    if ii==99:
        print('max iteration reached')
        break

print(logLH)

plt.figure(figsize=(15,10))
plt.title("Log-Likelihood vs Iteration\n", fontsize=15, weight='bold')
plt.plot(np.arange(1,ii+2),logLH)
plt.scatter(np.arange(1,ii+2),logLH)
plt.xlabel('Iteration')
plt.ylabel("Log-Likelihood")
plt.xticks(np.arange(1,ii+2))
plt.xlim(1,ii)
# plt.gca().invert_yaxis()
plt.grid('both');
```



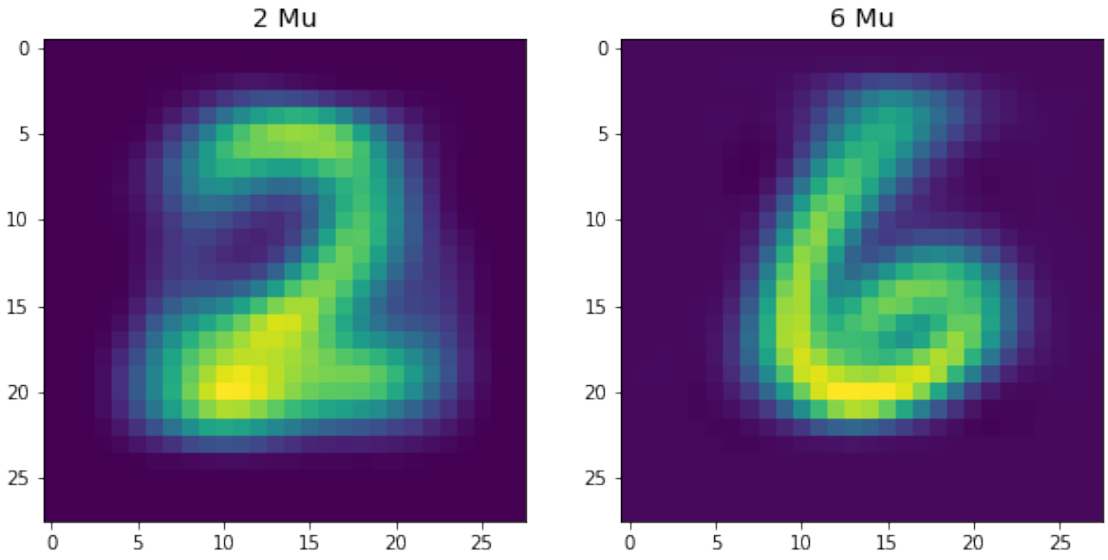

```
starting iteration, 0
starting iteration, 1
starting iteration, 2
starting iteration, 3
starting iteration, 4
starting iteration, 5
starting iteration, 6
starting iteration, 7
starting iteration, 8
starting iteration, 9
starting iteration, 10
starting iteration, 11
starting iteration, 12
starting iteration, 13
starting iteration, 14
starting iteration, 15
starting iteration, 16
starting iteration, 17
starting iteration, 18
starting iteration, 19
starting iteration, 20
starting iteration, 21
starting iteration, 22
starting iteration, 23
starting iteration, 24
```

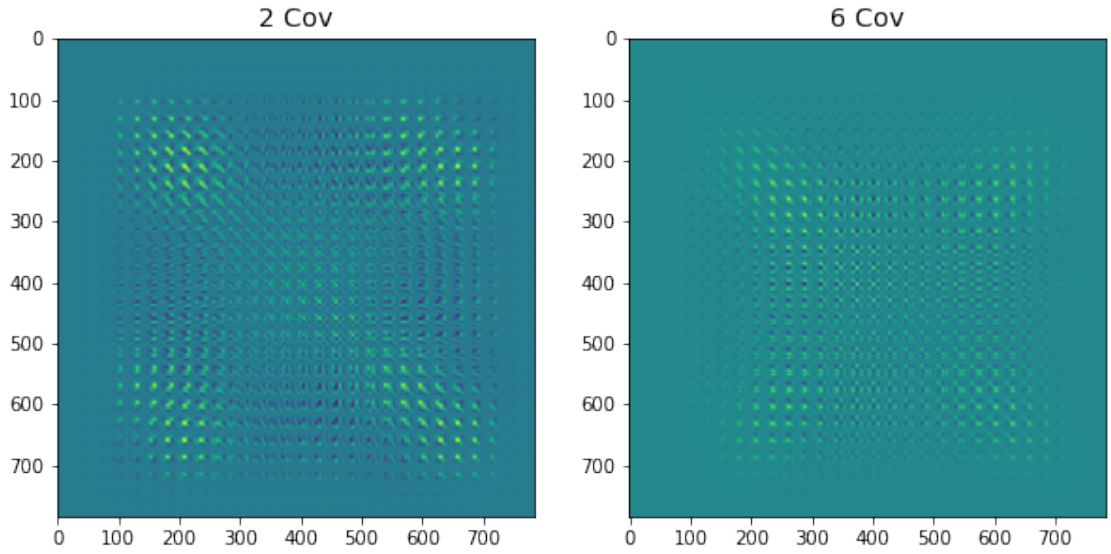
```
training coverged
[-64480.328679318394, -30228.982163320004, -29863.90706429426,
-29480.983264177536, -29219.693493032573, -29082.440045407835,
-28991.802078862318, -28945.498594707802, -28930.44505240937,
-28926.47377719405, -28925.344722226786, -28924.921658718646,
-28924.72116297928, -28924.615605336785, -28924.557718473377,
-28924.52536697856, -28924.507080187876, -28924.496660874534,
-28924.490688797057, -28924.48725020138, -28924.485263448863,
-28924.484112490078, -28924.48344436479, -28924.483055919012,
-28924.482829809807]
```



[5]:
```
# Q1d: Report, the fitted GMM model when EM has terminated

# D= np.diag(ev[:5])
res_mean_2= pca.inverse_transform(mu[0])
res_mean_2= scaler.inverse_transform(res_mean_2)

res_mean_6= pca.inverse_transform(mu[1])
res_mean_6= scaler.inverse_transform(res_mean_6)

fig, axes = plt.subplots(1,2,figsize=(10,5))
plt.title("Inverse transformed images of 2 and 6 for mu")
axes[0].imshow(res_mean_2.reshape(28,28).T)
```

```
axes[0].set_title("2 Mu", fontsize=14);
axes[1].imshow(res_mean_6.reshape(28,28).T)
axes[1].set_title("6 Mu", fontsize=14);
plt.show()

res_cov_2 = pca.inverse_transform(pca.inverse_transform(sigma[0]).T)
res_cov_6 = pca.inverse_transform(pca.inverse_transform(sigma[1]).T)
fig, axes = plt.subplots(1,2,figsize=(10,5))
plt.title("Inverse transformed images of 2 and 6 for covariance")
axes[0].imshow(res_cov_2)
axes[0].set_title("2 Cov", fontsize=14);
axes[1].imshow(res_cov_6)
axes[1].set_title("6 Cov", fontsize=14);
plt.show()
```





[6]:
```
# Q1e Find out the mis-classification rate for digits "2" and "6"

classification=[]
for t in tau:
    result = 2 if t[0]>=t[1] else 6
    classification.append(result)
c = np.array([classification])

miscalc_2=0
miscalc_6=0
total_2=np.count_nonzero(lbl == 2)
total_6=np.count_nonzero(lbl == 6)

for orig,calc in zip(lbl.T[:,0],c.T[:,0]):
#     print(orig,calc)
    if orig!=calc:
        if orig==2:
            miscalc_2+=1
        if orig==6:
            miscalc_6+=1

print("miscal_2",miscalc_2)
print("miscal_6",miscalc_6)
print("total 2s:",total_2)
print("total 6s:",total_6)

miscalc_rate_2=miscalc_2/total_2
miscalc_rate_6=miscalc_6/total_6
```

```
print(miscalc_rate_2*100)
print(miscalc_rate_6*100)


kmeans = KMeans(2)

np.random.seed(1)
kmeans.fit(data)

predictions = kmeans.labels_
predictions[predictions==0]=2
predictions[predictions==1]=6

miscalc_2=0
miscalc_6=0
print("predictions",predictions)
for orig,pred in zip(lbl.T[:,0],predictions):
    if orig!=pred:
            if orig==2:
                miscalc_2+=1
            if orig==6:
                miscalc_6+=1

print("miscal_2",miscalc_2)
print("miscal_6",miscalc_6)
print("total 2s:",total_2)
print("total 6s:",total_6)

miscalc_rate_2=miscalc_2/total_2
miscalc_rate_6=miscalc_6/total_6

print(miscalc_rate_2*100)
print(miscalc_rate_6*100)
```

```
miscal_2 25
miscal_6 152
total 2s: 1032
total 6s: 958
2.422480620155039
15.866388308977037
predictions [2 2 2 … 2 6 6]
miscal_2 55
miscal_6 68
total 2s: 1032
```

```
total 6s: 958
5.329457364341085
7.09812108559499
```