MACHINE LEARNING HOMEWORK 3

		Q1	Q2	Q3	Total
Grade	Max	1	2	2	5 points
	Expected	1	2	2	5

STUDENT

NAME SURNAME : ALPEREN KANTARCI

STUDENT NUMBER : 504191504

DEADLINE : 15/12/2019

COURSE NAME : MACHINE LEARNING

INSTRUCTOR : YUSUF YASLAN

QUESTION 1:

PCA is a unsupervised dimensionality reduction methods which tries to increase class variance on the projected space. It is easy to implement and it is powerful method. We are trying to project k dimensional data to d dimensional space. Since we need to maximize the variance first we calculate covariance matrix. Then we calculate the eigen values and their corresponding eigen vectors for better projection that ensures highest variance. Then eigenvectors give us the new projection space. In this question we map the data into 2 dimensional space. You can see the code and algorithm in the Figure 1, 2 and 3. Also the result of the PCA in the opdigit dataset can be seen in Figure 4.

```
dataset   = np.loadtxt('data.txt', delimiter=',', usecols=range(0,64))
labels    = np.loadtxt('data.txt', delimiter=',', dtype=np.str , usecols=[64])

def mean(mat):
    mean_vec = np.zeros(mat.shape[1])
    for ind,i in enumerate(mat):
        mean_vec += i

    mean_vec = mean_vec/mat.shape[0]
    return mean_vec

def covariance(mat,mean):
    num_feature = mat.shape[1]
    cov_matrix = np.zeros((num_feature,num_feature))
    for i in range(num_feature):
        for j in range(num_feature):
            cov_matrix[i][j] = np.sum( (mat[:,i]-mean[i])*(mat[:,j]-mean[j]) )/(mat.shape[0]-1)
    return cov_matrix
cov_mat = covariance(dataset,mean(dataset))
```

Figure 1 : Data loading and covariance calculation

```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
reduced = []
for vec in eig_vecs:
    reduced.append(vec[0:2])

reduced = np.array(reduced).T

x = []
y = []
for i in range(len(dataset)):
    temp = np.dot(reduced, dataset[i].T)    #Find the x and y values
    x.append(temp[0])
    y.append(temp[1])


plt.scatter(x,y,s=1,c="r")
samples = random.sample(range(0,len(x)),200)   #Take random 200 points
for i in samples:
    plt.annotate(labels[i],(x[i],y[i])) #Draw classes of the points
```

Figure 2 : Eigen value calculation and highest two eigenvectors

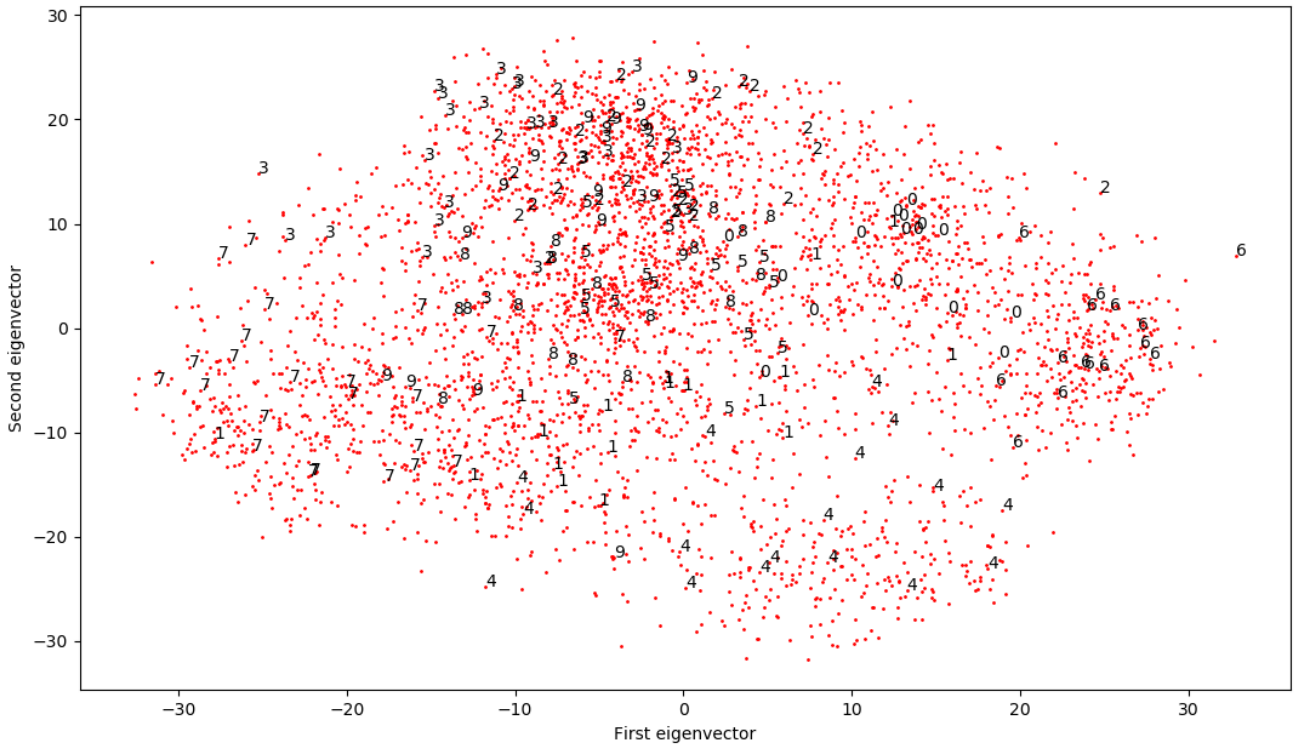

Figure 3 : Dataset after projected into two features with PCA

QUESTION 2:

For the second question we implement the Gaussian Mixture algorithm for clustering. Expectation Maximization algorithm is employed for the Gaussian Mixture. We are basically trying to estimate the probabilities to belonging a Gaussian of the data. Also sigma and mu values are estimated for each Gaussian. There are two parameters in the algorithm; number of iteration and number of clusters. In Fig 6, 7 and 8 you can see the results of the algorithm for k = 2, 3 and 4 and among them best number of clusters is 3 before after that number it overcluster the data. Number of iterations can be estimated when the coefficients and gamma values do not change much but for easier implementation I chose 100 iterations of expectation and maximization steps. Expectation step calculates the gamma values and coefficients. Maximization step reassign the data points by using new gamma values and coefficients and updates the sigma and mu values for each Gaussian. You can see the code in Figure 4 and 5.

```python
# Expectation step calculates the coefficients
def e_step(data,k,probabilities,gamma_values,mu,sigma):
    likelihood = np.zeros( (data.shape[0], k) )
    for i in range(k):
        distribution = multivariate_normal(mean=mu[i],cov=sigma[i])
        likelihood[:,i] = distribution.pdf(data)

    coefficients = (likelihood * gamma_values) / (likelihood * gamma_values).sum(axis=1)[:, np.newaxis]
    gamma_values = coefficients.mean(axis=0)
    return coefficients, gamma_values

# Maximization step updates the new mu and sigma values by using the coefficients from expectation
def m_step(data,k,probabilities,mu,sigma):
    for i in range(k):
        weight = probabilities[:, [i]]
        total_weight = weight.sum()
        mu[i] = (data * weight).sum(axis=0) / total_weight
        sigma[i] = np.cov(data.T,aweights=(weight/total_weight).flatten(),bias=True)

    return mu, sigma
```

Figure 4 : E and M steps

```python
def cluster(data,k=3):
    num_data, num_feature = data.shape[0], data.shape[1]

    gamma_values = np.array([1/k for i in range(k)])
    probabilities = np.full( shape=(num_data,num_feature), fill_value=1/k)

    random_row = np.random.randint(low=0, high=num_data, size=k)
    mu = [data[i,:] for i in random_row]
    sigma = [np.cov(data.T) for i in range(k)]

    n_iter = 100
    for i in range(n_iter):
        #Until it converge continously perform e and m steps
        probabilities, gamma_values = e_step(data,k,probabilities, gamma_values,mu,sigma)
        mu, sigma = m_step(data,k,probabilities,mu,sigma)

    likelihood = np.zeros( (data.shape[0], k) )
    for i in range(k):
        distribution = multivariate_normal(mean=mu[i],cov=sigma[i])
        likelihood[:,i] = distribution.pdf(data)

    coefficients = (likelihood * gamma_values) / (likelihood * gamma_values).sum(axis=1)[:, np.newaxis]
    return np.argmax(coefficients, axis=1)
```

Figure 5 : Clustering function that creates k different gaussians and create predictions for each data value

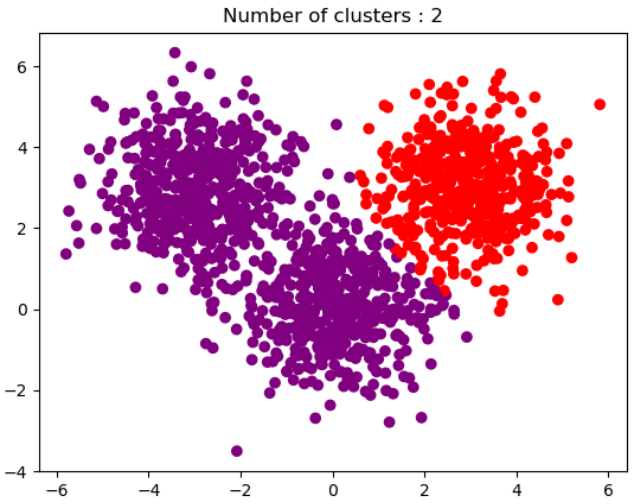

Figure 6 : GMM with 2 clusters

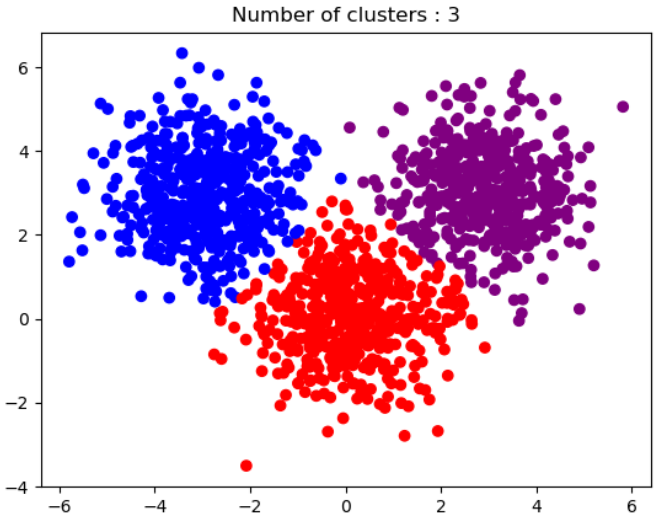

Figure 7 : GMM with 3 clusters and best cluster choice

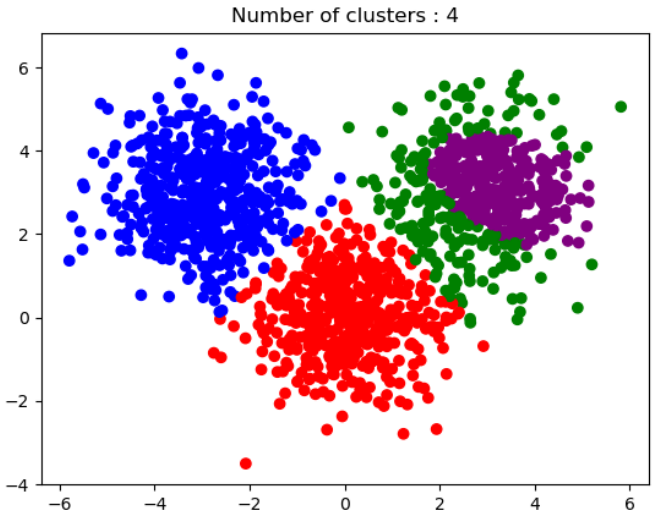

Figure 8 : GMM with 4 clusters

QUESTION 3:
A)

Q3-)

a-) $\pi_A = \pi_B = \frac{1}{2}$ probability of selecting coin A or coin B

X_i = Observed ith series Z_i = Unobserved $Z_i = 1$ if it belongs coin A

H_i = number of heads in ith sequence

$$P(X_i|H_i) = P(X_i|Z_i=1,H_i)P(Z_i=1) + P(X_i|Z_i=0,H_i)P(Z_i=0)$$

* $P(X_i|H_i) = \binom{10}{H_i}\theta_A^{H_i}(1-\theta_A)^{10-H_i}\pi_A + \binom{10}{H_i}\theta_B^{H_i}(1-\theta_B)^{10-H_i}\pi_B$ We use bromial

* $P(X|H_i) = \prod_{i=1}^{5} P(X_i|H_i)$

$$P(X_i, Z_i|H_i) = \left(\binom{10}{H_i}\theta_A^{H_i}(1-\theta_A)^{10-H_i}\pi_A\right)^{Z_i} \cdot \left(\binom{10}{H_i}\theta_A^{H_i}(1-\theta_A)^{10-H_i}\pi_B\right)^{1-Z_i}$$

$$\ln(P(X,Z|H)) = \sum_{i=1}^{n} \ln(P(X_i,Z_i|H_i))$$

$$P(Z_i|X_i) = \frac{P(X_i,Z_i|H_i)}{P(X_i|H_i)}$$

E step = $$\frac{\pi_A\left(\binom{10}{H_i}\theta_A^{H_i}(1-\theta_A^{H_i})^{10-H_i}\right)}{\pi_A\left(\binom{10}{H_i}\theta_A^{H_i}(1-\theta_A^{H_i})^{10-H_i} + \pi_B\left(\binom{10}{H_i}\theta_B^{H_i}(1-\theta_B^{H_i})^{10-H_i}\right)\right.} = \frac{\theta_A^{H_i}(1-\theta_A)^{10-H_i}}{\left(\theta_A^{H_i}(1-\theta_A)^{10-H_i} + \theta_B^{H_i}(1-\theta_B)^{10-H_i}\right)}$$

We can remove $\binom{10}{H_i}$ because constant at everywhere

M step = $$\frac{\partial E_{Z|X}(z_i^{(7)})}{\partial \theta_A} = \frac{\sum_{i=1}^{n} E_{Z|X}(z_i^{(7)})H_i}{\sum_{i=1}^{n} E_{Z|X}(z_i)}$$

B)

For these experiments we can use EM algorithm to find parameters (coefficients for each coin). There are two coins in this example and binomial distribution is used because only outcomes are 1 or 0. By using the formulas in the part a, I wrote the program that you can see in Figure 9 and 10. Program takes sequence of H and T list as input and calculates parameters for each coin. You can see that we iterate E and M steps until probabilities do not change more than 1e-7 which is convergence threshold. For faster convergence and better initialization my initial parameter estimation is between 0.3 and 0.9 but it doesn't make significant difference. Also you can see the final predicted parameters in Figure 11. Program calculated head probability of the coin 1 as **0.796** and head probability of the coin 2 as **0.519**

```
def experiment(data):
    # Randomly make guess but closer guesses are better
    prob_c1 = np.random.uniform(0.3,0.9)
    prob_c2 = np.random.uniform(0.3,0.9)

    it = 1
    # While parameters didn't converge iterate the E and M steps
    while True:
        print("Iteration {}, Prob. Coin 1 :{}, Prob. Coin 2 :{}".format(it, prob_c1, prob_c2))
        latest = [prob_c1,prob_c2] # Set latest parameters
        coin1_e, coin2_e = e_step(data, prob_c1, prob_c2) # GGet the expectation probabilities
        prob_c1, prob_c2 = m_step(coin1_e, coin2_e) # Get the new maximization parameters
        it+=1
        if np.abs(prob_c1-latest[0]) < 1e-7 and np.abs(prob_c2-latest[1]) < 1e-7:
            break
    return (prob_c1,prob_c2)
```

Figure 9 : Experiment code that shows iterations of the E and M steps

```python
def e_step(data, prob_c1, prob_c2):
    heads_A = 0
    tails_A = 0
    heads_B = 0
    tails_B = 0
    for t in data:
        num_heads = t.count("H")
        num_tails = t.count("T")

        bin_likelihood_c1 = pow(prob_c1, num_heads) * pow(1-prob_c1, num_tails)
        bin_likelihood_c2 = pow(prob_c2, num_heads) * pow(1-prob_c2, num_tails)
        total_bin_likelihood = (bin_likelihood_c1 + bin_likelihood_c2)
        prob_A = bin_likelihood_c1 / total_bin_likelihood
        prob_B = bin_likelihood_c2 / total_bin_likelihood
        heads_A += prob_A * num_heads
        tails_A += prob_A * num_tails
        heads_B += prob_B * num_heads
        tails_B += prob_B * num_tails
    return (heads_A, tails_A), (heads_B, tails_B)

def m_step(coin1_e, coin2_e):
    #Calculate new probabilities that best suits with the experiment data

    theta_A = coin1_e[0] / (np.sum(coin1_e))
    theta_B = coin2_e[0] / (np.sum(coin2_e))
    return theta_A, theta_B
```

Figure 10 : E and M steps

```
alperen@simit-Lab193: ~/Desktop/ML/ML_HW3
File  Edit  View  Search  Terminal  Help
(Keras) alperen@simit-Lab193:~/Desktop/ML/ML_HW3$ python q3.py
Iteration 1, Prob. Coin 1 :0.4294375606648432, Prob. Coin 2 :0.38444390005669443
Iteration 2, Prob. Coin 1 :0.6846157133949031, Prob. Coin 2 :0.6210317613085575
Iteration 3, Prob. Coin 1 :0.7063377990000862, Prob. Coin 2 :0.6127528263697322
Iteration 4, Prob. Coin 1 :0.7269591309994741, Prob. Coin 2 :0.59282747161042
Iteration 5, Prob. Coin 1 :0.7500695250432686, Prob. Coin 2 :0.569533933654563
Iteration 6, Prob. Coin 1 :0.7707349162265056, Prob. Coin 2 :0.5482864781384532
Iteration 7, Prob. Coin 1 :0.7845507698181662, Prob. Coin 2 :0.5336506204224831
Iteration 8, Prob. Coin 1 :0.7916811927187656, Prob. Coin 2 :0.5257780040797995
Iteration 9, Prob. Coin 1 :0.794787939919992, Prob. Coin 2 :0.5221673004486035
Iteration 10, Prob. Coin 1 :0.7960278440763584, Prob. Coin 2 :0.5206371281174434
Iteration 11, Prob. Coin 1 :0.7965033230939208, Prob. Coin 2 :0.5200095594323049
Iteration 12, Prob. Coin 1 :0.796682493141708, Prob. Coin 2 :0.5197552482240895
Iteration 13, Prob. Coin 1 :0.7967494655405097, Prob. Coin 2 :0.5196525830321385
Iteration 14, Prob. Coin 1 :0.7967743920074001, Prob. Coin 2 :0.5196111672269618
Iteration 15, Prob. Coin 1 :0.7967836426028045, Prob. Coin 2 :0.5195944535923215
Iteration 16, Prob. Coin 1 :0.7967870671697651, Prob. Coin 2 :0.5195877036840383
Iteration 17, Prob. Coin 1 :0.796788331812703, Prob. Coin 2 :0.5195849753864028
Iteration 18, Prob. Coin 1 :0.7967887975786914, Prob. Coin 2 :0.5195838716750731
Iteration 19, Prob. Coin 1 :0.7967889686053622, Prob. Coin 2 :0.519583424809125
Iteration 20, Prob. Coin 1 :0.7967890311906006, Prob. Coin 2 :0.5195832437417779

Final parameters determined by E-M Algorithm are Coin1:0.79679, Coin2:0.51958
(Keras) alperen@simit-Lab193:~/Desktop/ML/ML_HW3$
```

Figure 11 : Iterations of the program and final parameters