Programming Exercises

Eigenface for Face Recognition

1. The Faces dataset was downloaded and separated into training and test data (along with the associated labels) based on the indices supplied in the accompanying text files.

2. The data separation into the respective variables is performed as shown in the code, while flattening the 50x50 dimension images into 2500x1 dimensional vectors. The shape of the training set is (540, 2500) while that of the test set is (100, 2500)

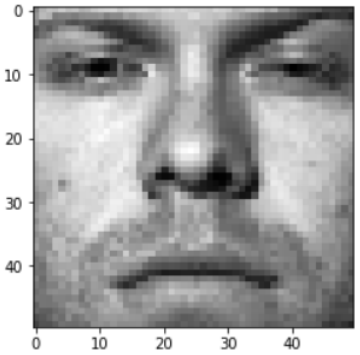


Figure 1: Image with index 10 in training set

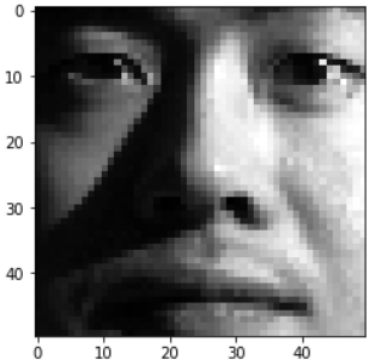


Figure 2: Image with index 11 in test set

3. The average face is computed as directed and the grayscale output is shown in Fig.3.

4. After subtracting the average face from each image, a sample of the result obtained is displayed in grayscale in Fg.4.

5. V^T is obtained after performing eigen decomposition, and the top 10 faces are as shown in Fig. 5.

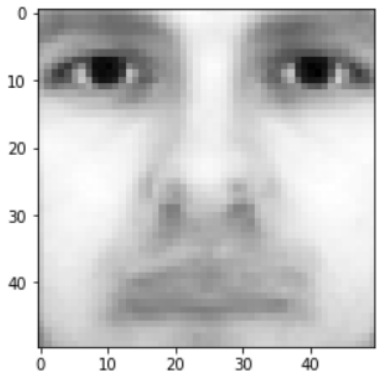


Figure 3: Average face as computed

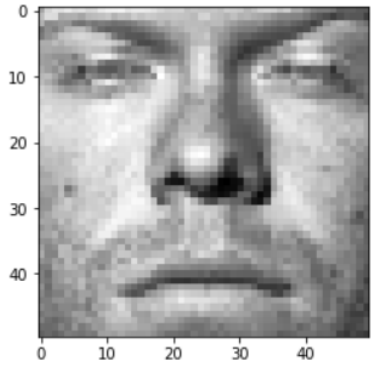


Figure 4: Face from Fig.1 after subtracting average face

6. The feature matrices are generated as directed for both X and X_{test}.

7. The classification accuracy for the training set (r=10) turns out to be 0.7685 while that for the test set (r=10) turns out to be 0.8. The accuracy values for r values ranging from 1 to 200 are as shown in Fig.6. The accuracy seems to reach its maximum around $r = 50$ but is satisfactorily high for any value $r = 25$ onwards.

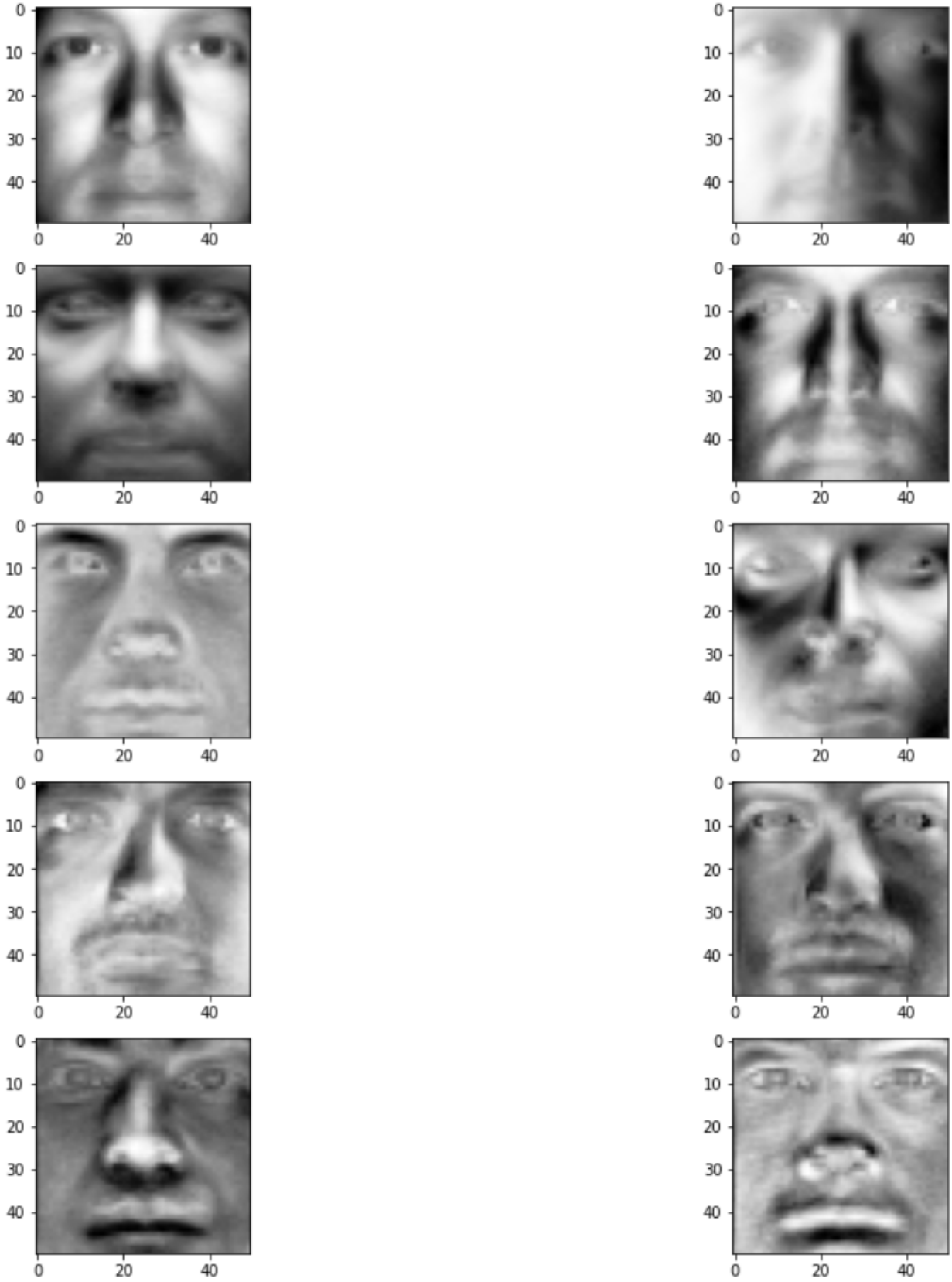


Figure 5: First ten eigen faces

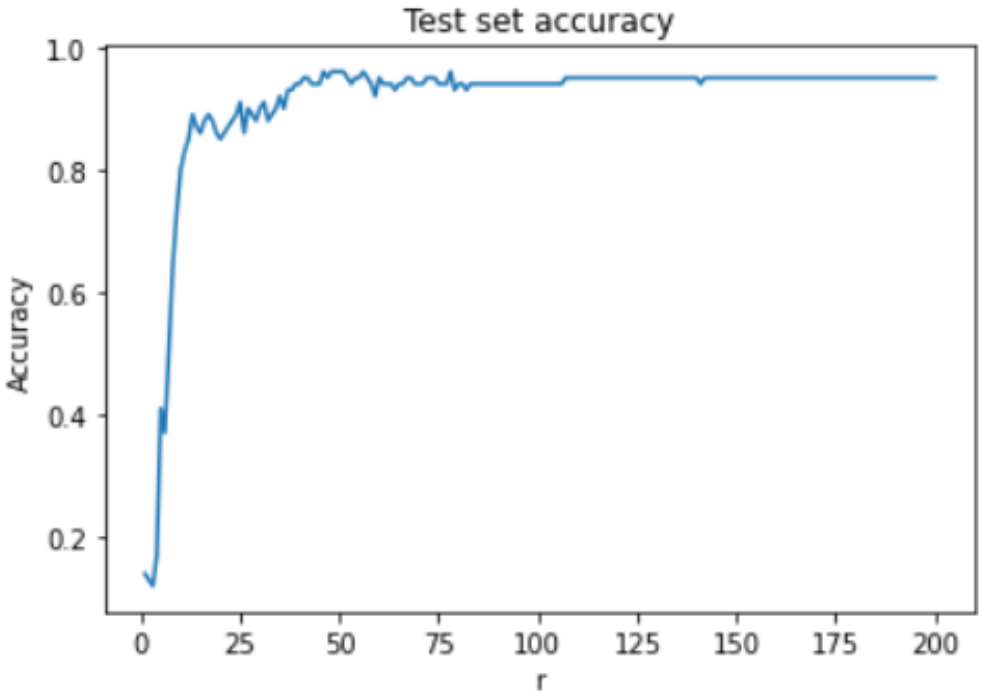


Figure 6: Accuracy v. r

Implement EM Algorithm

1. Upon using a scatterplot to visualize the old faithful geyser dataset, we obtain Fig.7.

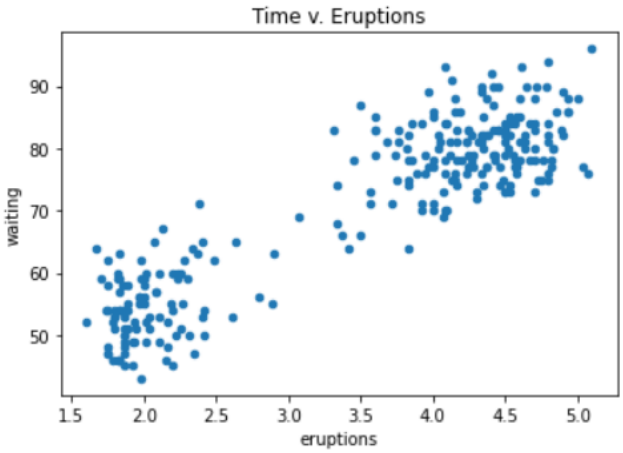


Figure 7: Scatterplot of dataset

2. The formula for the E-step is as given below:

$$P_\theta(z = k|x) = \frac{P_\theta(z = k, x)}{P_\theta(x)} = \frac{P_\theta(z = k)P_\theta(z = k)}{\Sigma_{l=1}^{K} P_\theta(x|z = l)P_\theta(z = l)}$$

3. In the M-step, we calculate the values of μ_k, Σ_k and ϕ_k. The eformulae for the same are given below:

$$\mu_k = \frac{\Sigma_{i=1}^{n} P(z = k|x^{(i)})x^{(i)}}{n_k}$$

$$\Sigma_k = \frac{\Sigma_{(i=1)}^{n} P(z = k|x^{(i)})(x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T}{n_k}$$

$$\phi_k = \frac{n_k}{n}$$

4. We implement the EM Algorithm as required.
For the termination criteria, it is important to be able to understand when the algorithm has converged in order to stop the computation. A good measure of this would be the change in the log likelihood, given the fact that that is the key computation that is performed by the algorithm, and is updated in each iteration. Keeping these factors in mind and having experimented with the code, we decide to set the computation to stop when the difference between the log likelihood pre and post update is less than 0.1, which is a good estimate of when the algorithm has, essentially, converged.
As for the variation in the values of μ_1 and μ_2 across the iterations, we generate the plot shown in Fig.8 to visualize the same.

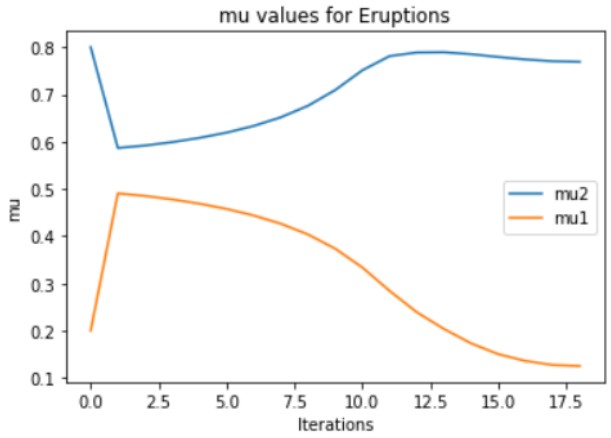


Figure 8: Values of μ_1 and μ_2

5. After experimenting with K-meeans on the same dataset, we find that both the codes assign practically the same clusters (except one point that changes clusters). The chief point of difference between the two approaches is that EM uses likelihood to predict clustering, whereas K-means uses Euclidean distance. However, we did expect the algorithms to perform similarly given how the data is rather distributed.

Written Exercises

SVD and Eigen Decomposition

SVD factorizes an $m * n$ matrix, X, into three matrices.

- U, which is an $m * m$ orthonormal matrix
- D, an $m * n$ diagonal matrix with non-negative real numbers
- V, an $n * n$ orthonormal matrix

We know that the eigendecomposition of X can be described as,

$$X = Q\Lambda Q^{-1} \tag{1}$$

where λ is the corresponding eigenvalue.
An SVD of X would be of the form

$$X = uwv^T$$

Given that u and v are orthonormal, we can compute X^TX as follows by using the SVD above,

$$\begin{aligned} X^TX &= (uwv^T)^Tuwv^T \\ &= vw^Tu^Tuwv^T \\ &= vw^Twv^T \\ &= vDv^T \\ &= vDv^{-1} \end{aligned}$$

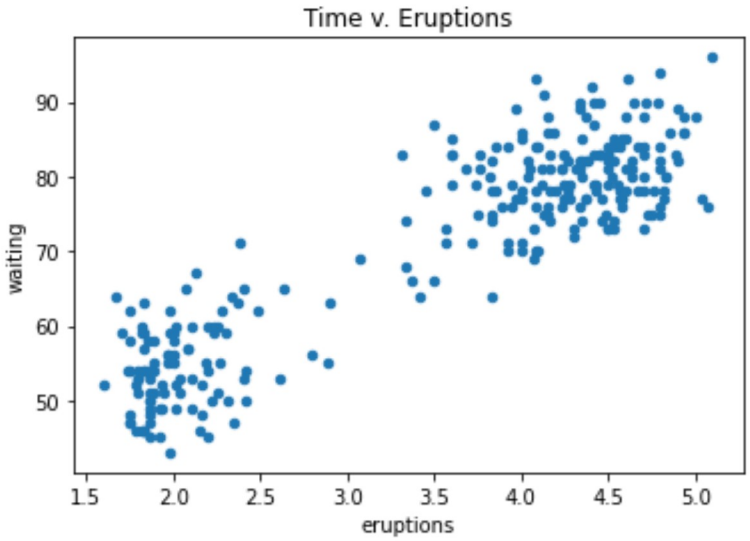
where $D = ww^{-1}$.
This is the same form as the one in (1), i.e. the eigendecomposition.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import seaborn as sns
from sklearn.cluster import KMeans

%matplotlib inline
```

In [2]:
```
geyser = pd.read_csv("C:\\Users\\sidar\\Documents\\Study Materials\\Assignment
geyser = geyser.drop("Unnamed: 0", axis=1)
```

In [3]:
```
geyser.plot(kind="scatter", x="eruptions", y="waiting")
plt.title("Time v. Eruptions")
```

Out[3]:
```
Text(0.5, 1.0, 'Time v. Eruptions')
```



In [4]:
```
geyser = geyser.to_numpy()
```

In [5]:
```
#Implementing deprecated mlab function

def bivariate_normal(X, Y, x_sigma=1.0, y_sigma=1.0, x_mu=0.0, y_mu=0.0, xy_s

    X_mu = X-x_mu
    Y_mu = Y-y_mu

    rho = xy_sigma/(x_sigma*y_sigma)
    z = X_mu**2/x_sigma**2 + Y_mu**2/y_sigma**2 - 2*rho*X_mu*Y_mu/(x_sigma*y_
    den = 2*np.pi*x_sigma*y_sigma*np.sqrt(1-rho**2)
    return np.exp(-z/(2*(1-rho**2))) / den
```

In [6]:
```
def gmm_loglik(X, mean, cov, mix_coef):

    sum2 = 0
    for i in range(X.shape[0]):
        sum1 = 0
        for k in range(mix_coef.shape[0]):
            sum1 += mix_coef[k] * multivariate_normal.pdf(X[i], mean=mean[k],
        sum2 += np.log(sum1)
    loglik = sum2

    return loglik
```

In [7]:
```
def e_step(X, mean, cov, mix_coef):

    resp = np.zeros((X.shape[0],mean.shape[0]))
    num = np.zeros(mix_coef.shape[0])
    denomin=0
    for n in range(X.shape[0]):
        den = 0
        for k in range(mix_coef.shape[0]):
            num[k] = mix_coef[k] * multivariate_normal.pdf(X[n,:], mean[k,:],
            den += mix_coef[k] * multivariate_normal.pdf(X[n,:], mean[k,:],co
        for k in range(mix_coef.shape[0]):
            resp[n,k] = num[k] / den

    return resp
```

In [8]:
```
def m_step(X, resp):

    N = np.zeros(resp.shape[1])
    #iterate over clusters
    for k in range(resp.shape[1]):
        #init
        N[k] = resp[:,k].sum(axis=0)
        sum_mean = 0
        sum_cov = 0
        #new means
        for i in range(X.shape[0]):
            sum_mean += resp[i,k] * X[i, :]
        sum_mean /= N[k]
        #new covs
        for i in range(X.shape[0]):
            A = X[i, :] - sum_mean
            sum_cov += resp[i,k] *np.outer(A, A.T)
        mean[k,:] = sum_mean
        cov[k,:,:] = sum_cov / N[k]
    NN = N.sum(axis=0)
    mix_coef = N / NN

    return mean, cov, mix_coef
```

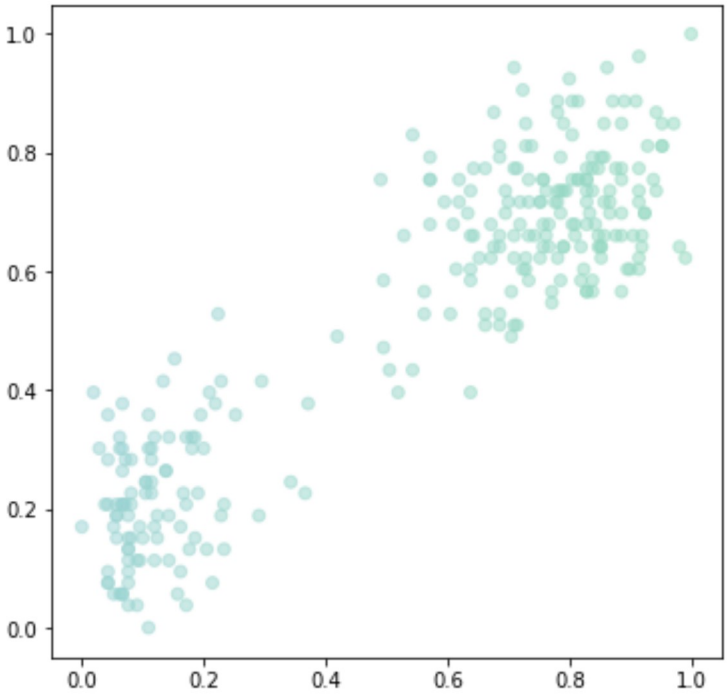
In [9]:
```
def plot_gmm_2d(norm, resp, mean, cov, mix_coef):
    plt.figure(figsize=[6, 6])
    palette = np.array(sns.color_palette('pastel', n_colors=3))[[0, 2]]
    colors = resp.dot(palette)

    plt.scatter(norm[:, 0], norm[:, 1], c=colors, alpha=0.5)

    for index, m in enumerate(mean):
        mus.append([m[0],m[1]])
```

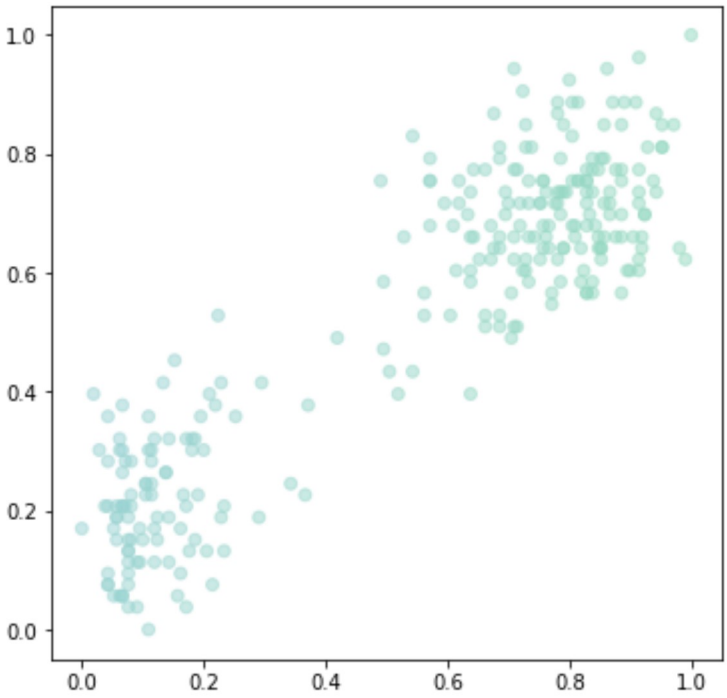
In [10]:
```
norm = np.zeros(geyser.shape)
norm[:,0] = (geyser[:,0] - np.amin(geyser[:,0]))/(np.amax(geyser[:,0]) - np.ar
norm[:,1] = (geyser[:,1] - np.amin(geyser[:,1]))/(np.amax(geyser[:,1]) - np.ar
```

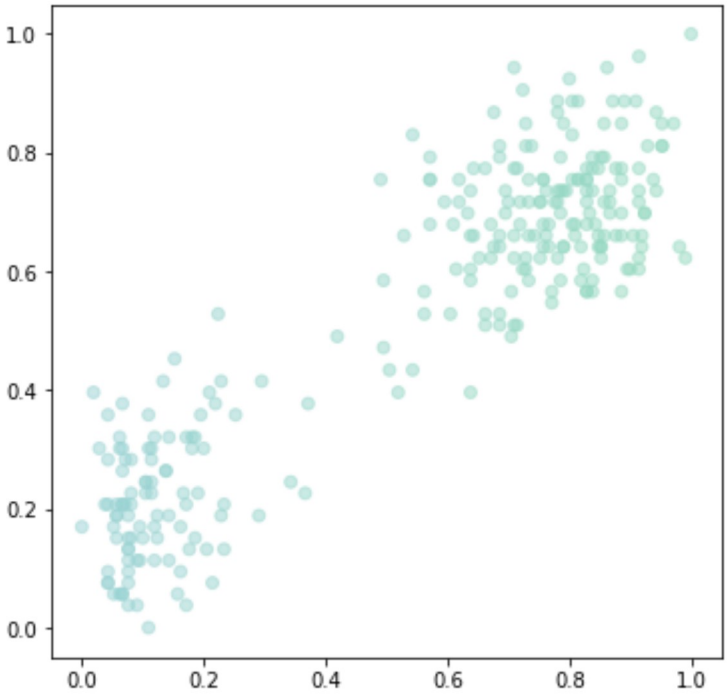
In [50]:
```
max_iters = 100
mean = np.array([[0.2, 0.6], [0.8, 0.4]])
cov = np.array([0.5 * np.eye(2), 0.5 * np.eye(2)])
mix_coef = np.array([0.5, 0.5])
mus = []
```

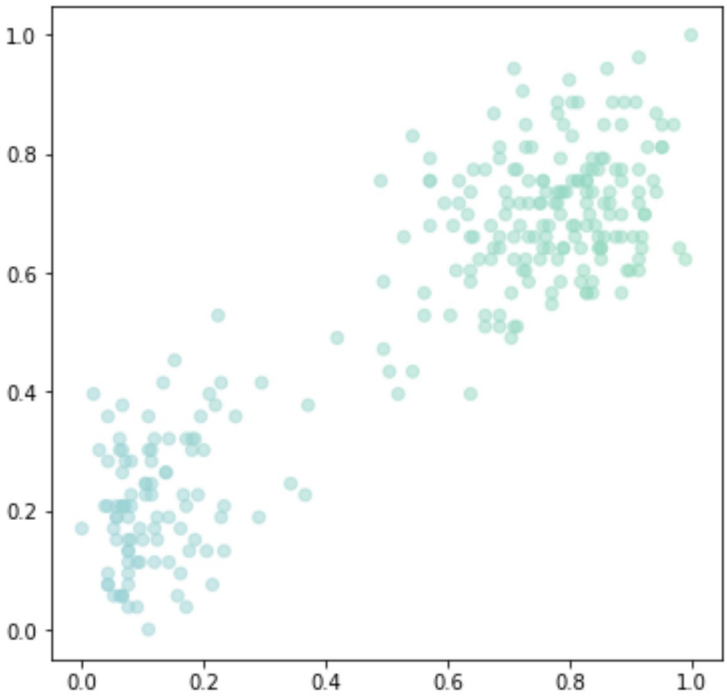
In [51]:
```
old_loglik = gmm_loglik(norm, mean, cov, mix_coef)
resp = e_step(norm, mean, cov, mix_coef)
print('At initialization: log-likelihood = ' + str(old_loglik))
plot_gmm_2d(norm, resp, mean, cov, mix_coef)

# Perform the EM iteration
for i in range(max_iters):
    resp = e_step(norm, mean, cov, mix_coef)
    mean, cov, mix_coef = m_step(norm, resp)
    new_loglik = gmm_loglik(norm, mean, cov, mix_coef)
    if (new_loglik-old_loglik) < 0.1:
        break
    old_loglik = new_loglik
    plot_gmm_2d(norm, resp, mean, cov, mix_coef)
```

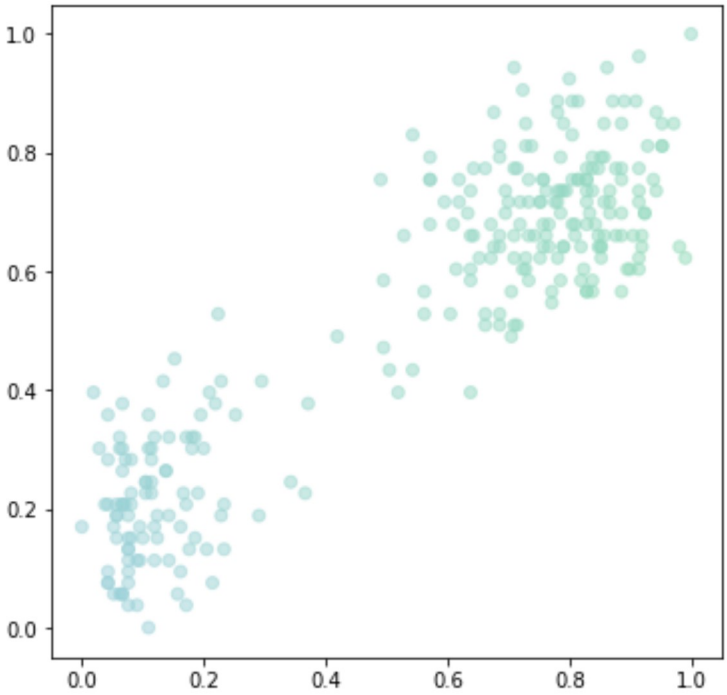
```
At initialization: log-likelihood = -382.70551524206564
```

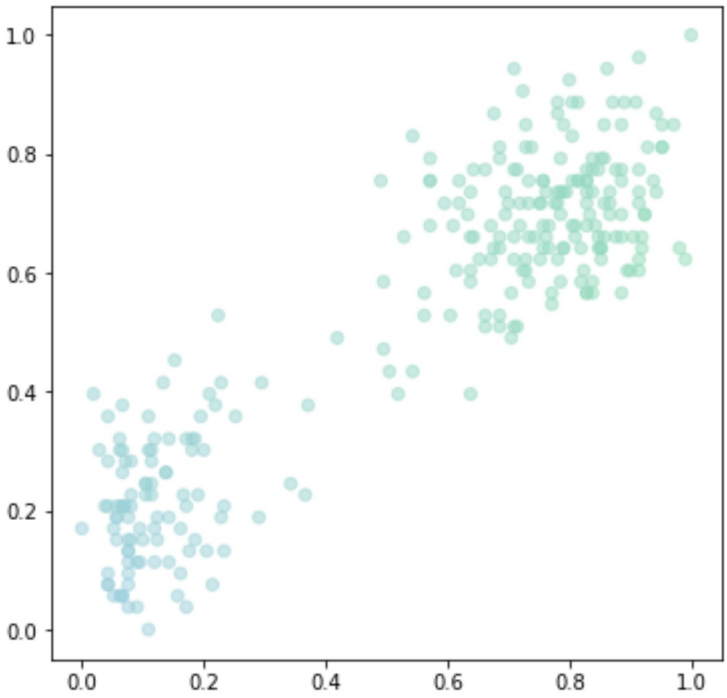

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

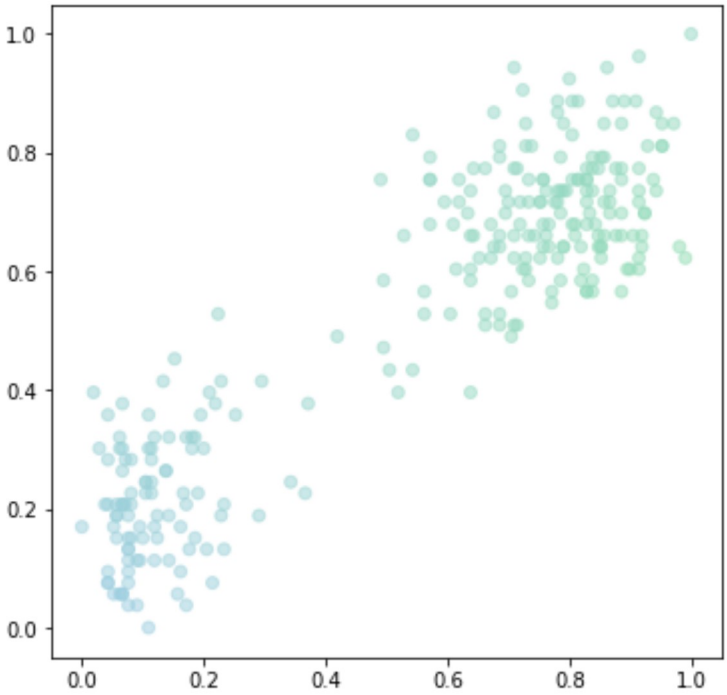

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

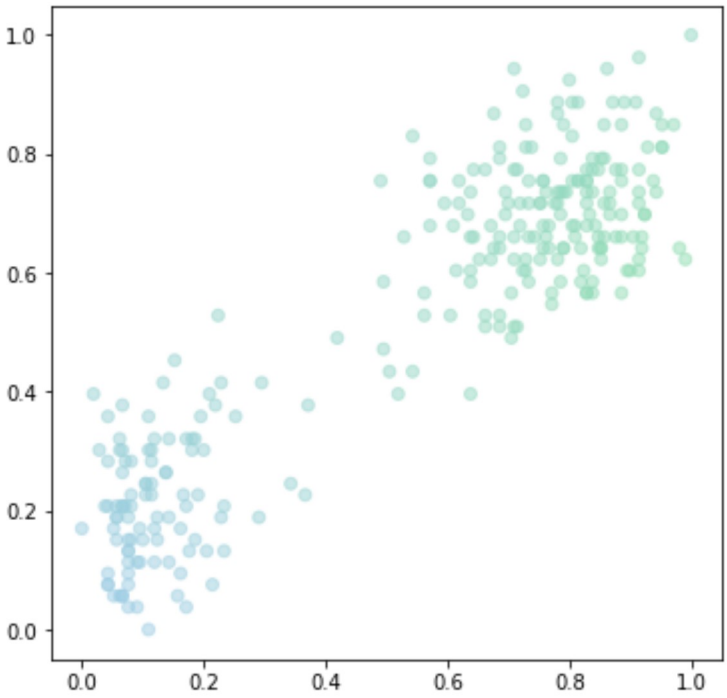

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

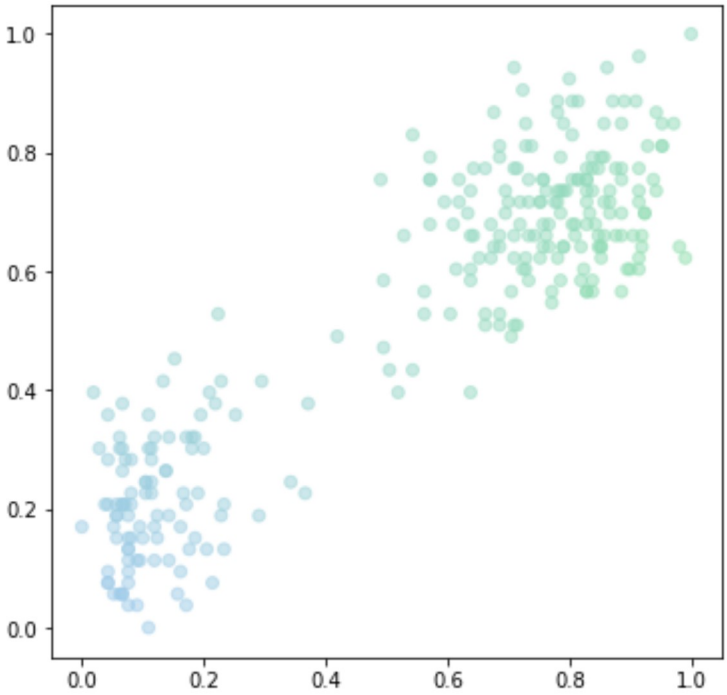

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

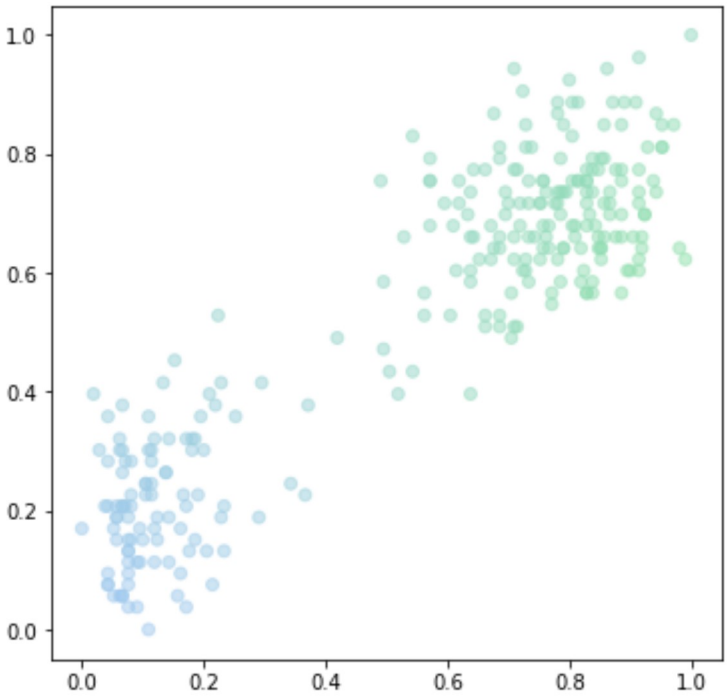

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

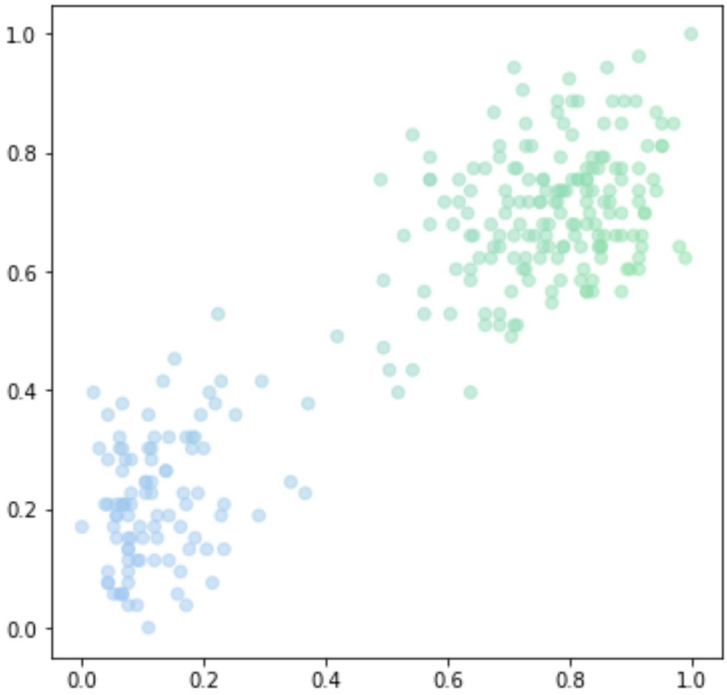

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

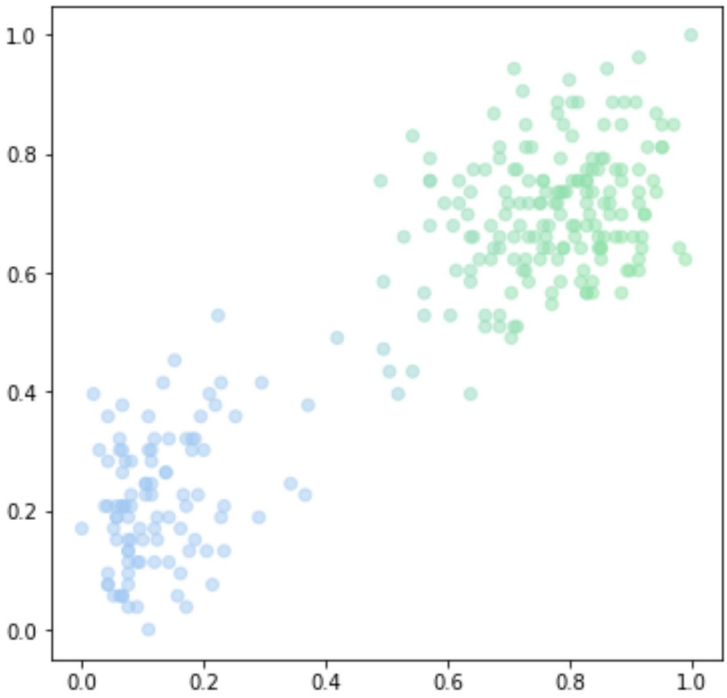

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

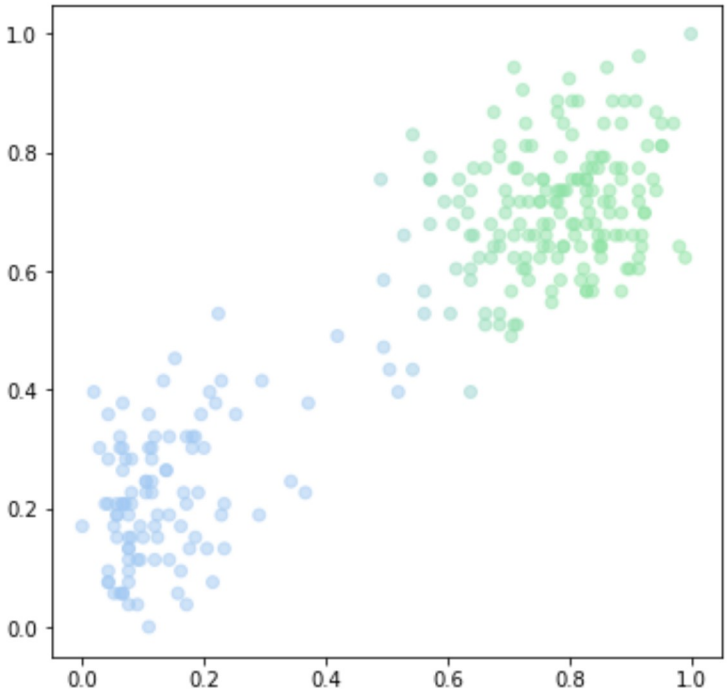

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

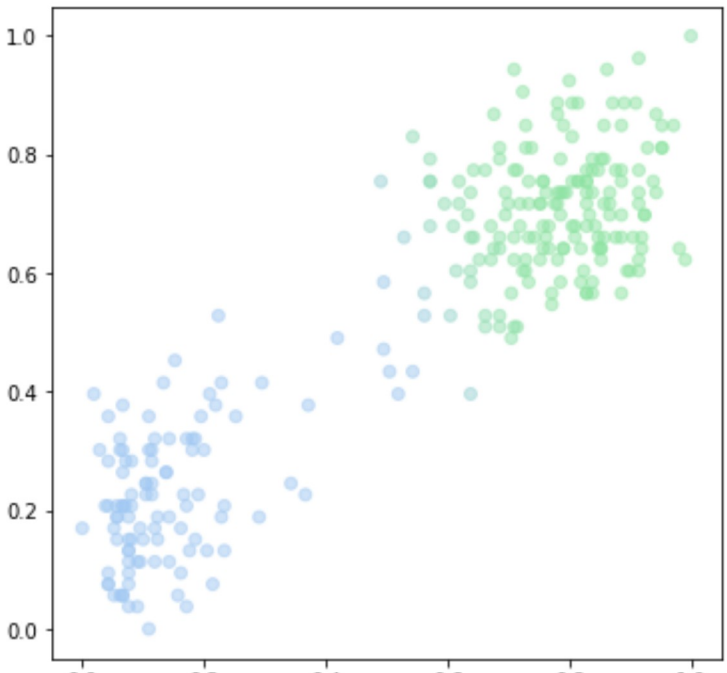

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0





In [52]:
```
mus
```

Out[52]:
```
[[0.2, 0.6],
 [0.8, 0.4],
 [0.4904413329960295, 0.4941470877574137],
 [0.5865309448489864, 0.5574127221230718],
 [0.4850398683627134, 0.48994502972152376],
 [0.5918063830401906, 0.5615093760919231],
```

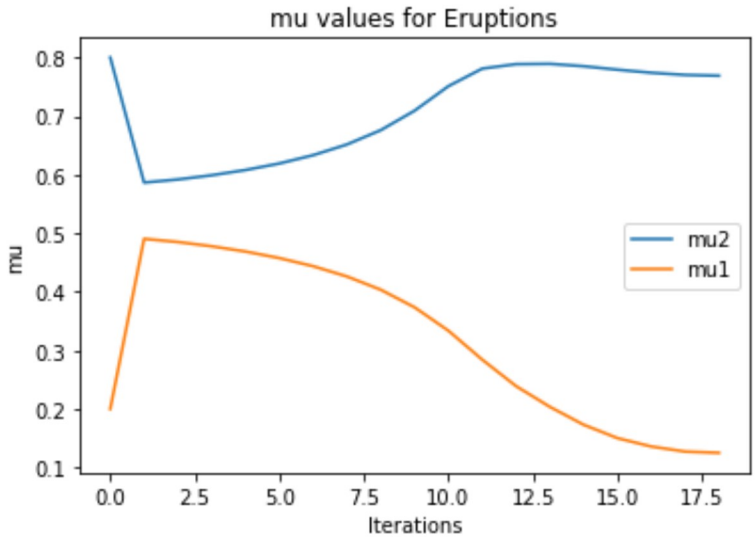
```
[0.47775657312399794, 0.4843264236688593],
[0.5989652373848385, 0.5670205198711853],
[0.4687307002862276, 0.47740469743800734],
[0.6079109638518467, 0.5738649485998433],
[0.4575950304175625, 0.46889816143101104],
[0.6190802398726641, 0.5823749228948586],
[0.44373276684658886, 0.45834773212327345],
[0.633224720566115, 0.5931086408124018],
[0.4261982648051199, 0.4450784855714789],
[0.6515506817333153, 0.6069337978338625],
[0.40352330827356786, 0.42810487495106664],
[0.6759910994931407, 0.6251793742077565],
[0.37349620891068014, 0.40608959201048855],
[0.7092277004946084, 0.6495232143542067],
[0.33359698699862855, 0.3777675031933352],
[0.7510803502604053, 0.6792441334053066],
[0.28422672663126564, 0.34252398951419194],
[0.7810380555915526, 0.7004908340281232],
[0.23901914978954336, 0.30792659580768184],
[0.7886466351987348, 0.7076529271885541],
[0.20361088199186497, 0.2797970250292006],
[0.7891781338491926, 0.7098089753793674],
[0.1730578448148051, 0.25456390473926055],
[0.7850900071315116, 0.7086827498999902],
[0.15020621566316741, 0.23503492197029693],
[0.7791857057817129, 0.705887703196373],
[0.1359027563879881, 0.22421770172879502],
[0.7739425496991081, 0.7020264552643697],
[0.12728460101619776, 0.21868289727388696],
[0.7702066863708011, 0.6987139341026809],
[0.1251530318106843, 0.21691476882508398],
```

In [53]:

```
mu1 = []
mu2 = []

for i in range(0, len(mus)):
    if i%2==0:
        mu1.append(mus[i])
    else:
        mu2.append(mus[i])
```

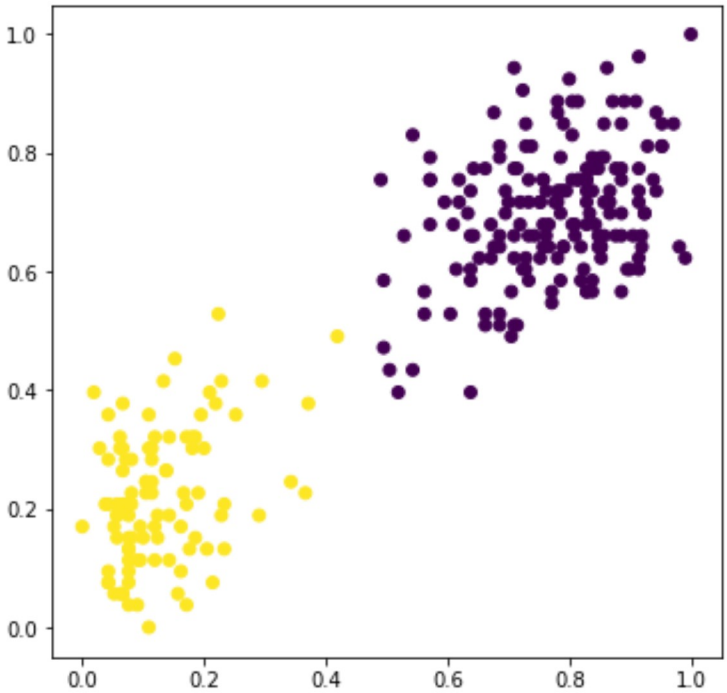
In [56]:

```
plt.plot([x[0] for x in mu2])
plt.plot([x[0] for x in mu1])
plt.xlabel('Iterations')
plt.ylabel('mu')
plt.legend(['mu2', 'mu1'])
plt.title('mu values for Eruptions')
plt.show()
```



In [43]:
```
KM = KMeans(n_clusters = 2).fit(norm)
```

In [44]:
```
plt.figure(figsize=[6, 6])
plt.scatter(norm[:, 0], norm[:, 1], c=KM.labels_)
plt.show()
```



In [37]:
```
import numpy as np
import imageio
from matplotlib import pylab as plt
import matplotlib.cm as cm
from sklearn.linear_model import LogisticRegression

%matplotlib inline
```

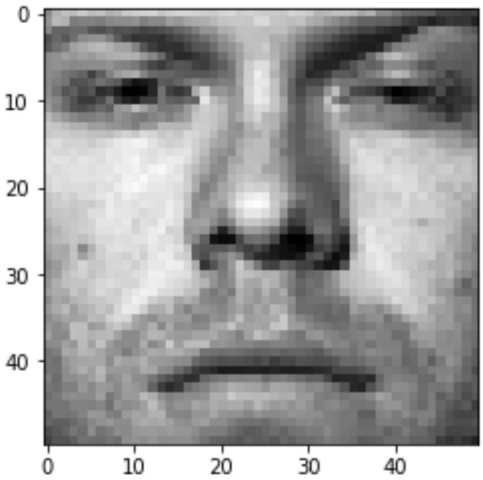
In [38]:
```
train_labels, train_data = [], []

for line in open("C:\\Users\\sidar\\Documents\\Study Materials\\Assignments\\I
    im = imageio.imread("C:\\Users\\sidar\\Documents\\Study Materials\\Assignm
    train_data.append(im.reshape(2500,))
    train_labels.append(line.strip().split()[1])

train_data, train_labels = np.array(train_data, dtype=float), np.array(train_l
print(train_data.shape, train_labels.shape)
```

```
(540, 2500) (540,)
```

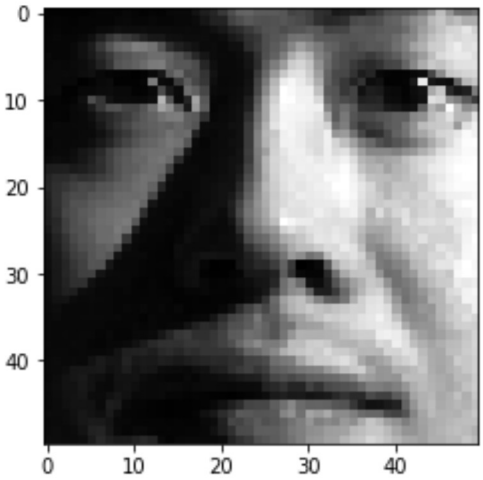
In [39]:
```
test_labels, test_data = [], []

for line in open("C:\\Users\\sidar\\Documents\\Study Materials\\Assignments\\I
    im = imageio.imread("C:\\Users\\sidar\\Documents\\Study Materials\\Assignm
    test_data.append(im.reshape(2500,))
    test_labels.append(line.strip().split()[1])

test_data, test_labels = np.array(test_data, dtype=float), np.array(test_label
print(test_data.shape, test_labels.shape)
```

```
(100, 2500) (100,)
```

In [40]:
```
plt.imshow(train_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



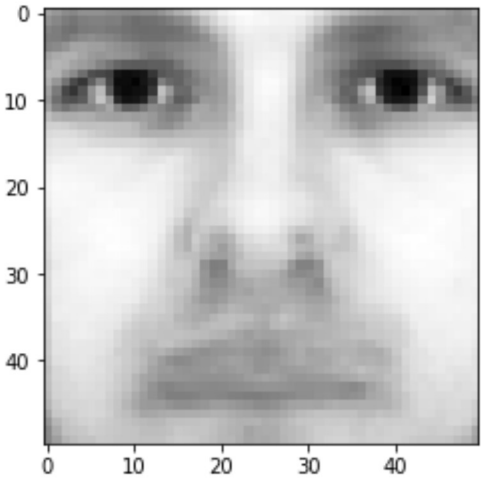
In [41]:
```
plt.imshow(test_data[11, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



In [42]:
```
avg_face = []
i = 0
while (i < train_data.shape[1]):
    avg_face.append(np.sum(train_data[:,i]))
    i+=1

avg_face = np.array(avg_face, dtype=float)
avg_face = avg_face / train_data.shape[0]

len(avg_face)
```

Out[42]:
```
2500
```

In [43]:
```
plt.imshow(avg_face.reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

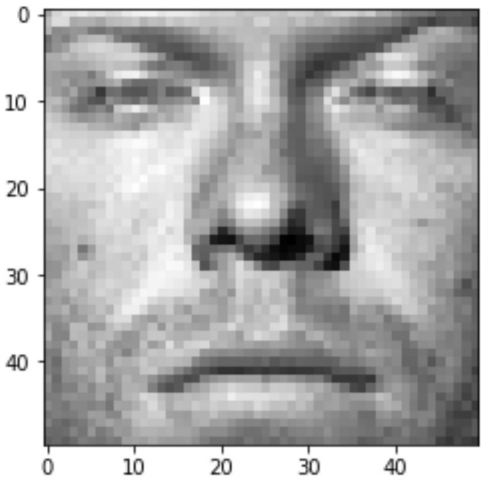


In [44]:
```
X = []
X_test = []
i = 0
j = 0

while i<train_data.shape[0]:
    X.append(train_data[i,:] - avg_face)
    i += 1

while j<test_data.shape[0]:
    X_test.append(test_data[j,:] - avg_face)
    j += 1
```

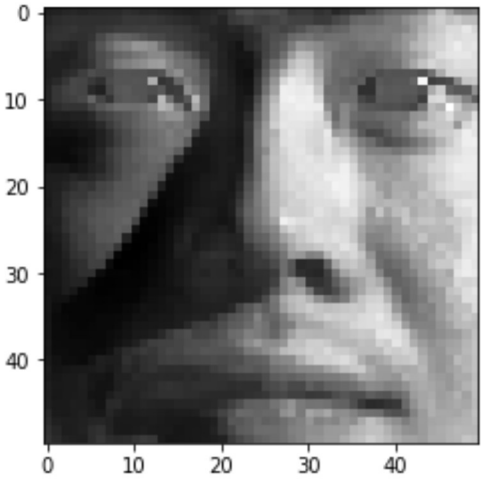
In [45]:
```
X = np.array(X, dtype=float)
X_test = np.array(X_test, dtype=float)

X.shape
```

Out[45]:
```
(540, 2500)
```

In [46]:
```
plt.imshow(X[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



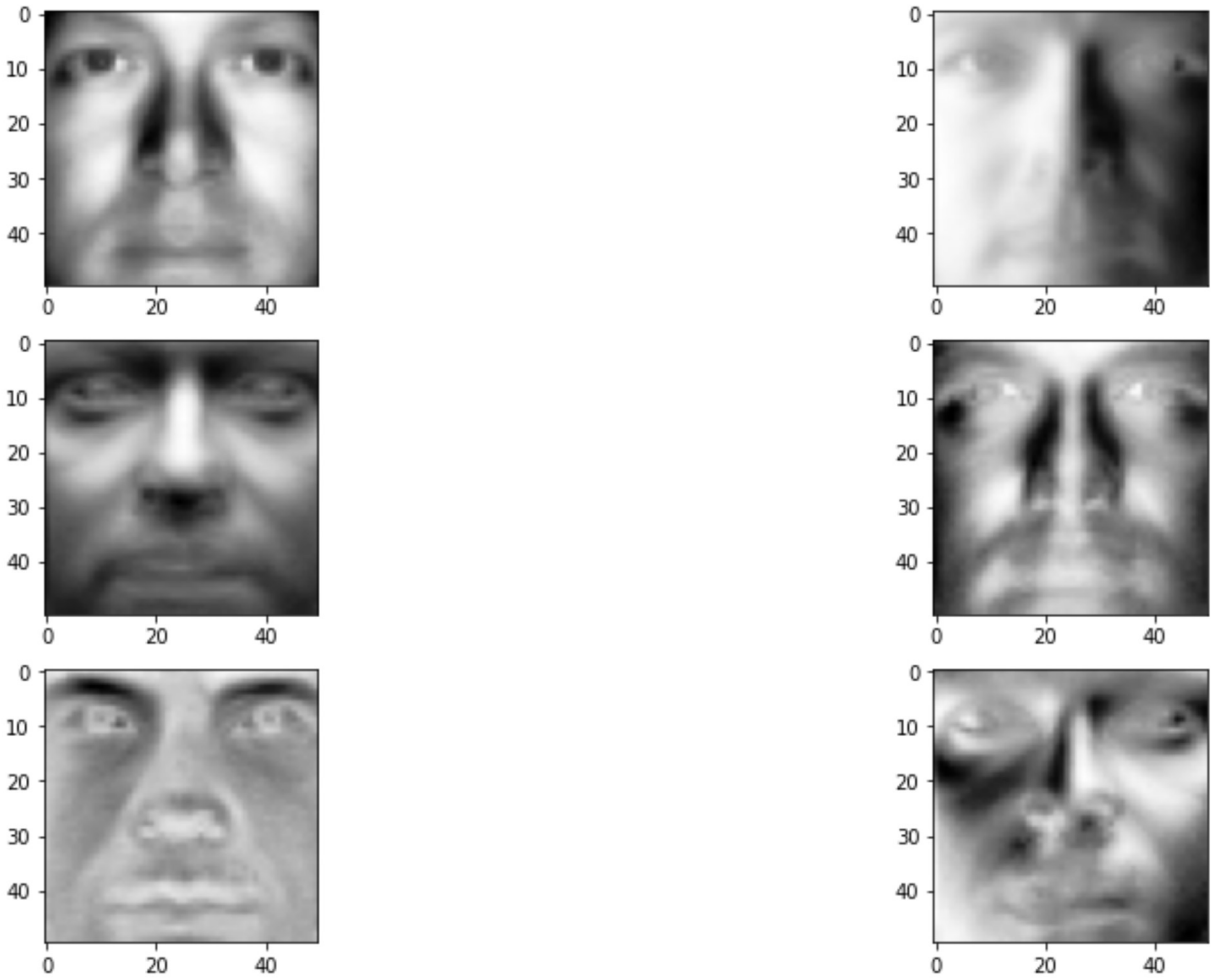
In [47]:
```
plt.imshow(X_test[11, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```



In [48]:
```
u, s, vT = np.linalg.svd(X)
```

In [49]:
```
plt.figure()

f, ax = plt.subplots(5,2, figsize=(15,15))

ax[0,0].imshow(vT[0].reshape(50,50), cmap = cm.Greys_r)
ax[0,1].imshow(vT[1].reshape(50,50), cmap = cm.Greys_r)
ax[1,0].imshow(vT[2].reshape(50,50), cmap = cm.Greys_r)
ax[1,1].imshow(vT[3].reshape(50,50), cmap = cm.Greys_r)
ax[2,0].imshow(vT[4].reshape(50,50), cmap = cm.Greys_r)
ax[2,1].imshow(vT[5].reshape(50,50), cmap = cm.Greys_r)
ax[3,0].imshow(vT[6].reshape(50,50), cmap = cm.Greys_r)
ax[3,1].imshow(vT[7].reshape(50,50), cmap = cm.Greys_r)
ax[4,0].imshow(vT[8].reshape(50,50), cmap = cm.Greys_r)
ax[4,1].imshow(vT[9].reshape(50,50), cmap = cm.Greys_r)
```

Out[49]:
```
<matplotlib.image.AxesImage at 0x1d7a832ebb0>

<Figure size 432x288 with 0 Axes>
```



In [50]:
```
r = 10

F = X.dot(vT[:r,:].T)
F_test = X_test.dot(vT[:r,:].T)
```

In [51]:
```
F.shape
```

Out[51]:
```
(540, 10)
```

In [52]:
```
logreg = LogisticRegression(multi_class = "ovr", max_iter = 25000)
logreg.fit(F, train_labels)
```

Out[52]:
```
LogisticRegression(max_iter=25000, multi_class='ovr')
```

In [53]:
```
score_train = logreg.score(X = F, y = train_labels)
print("Accuracy on the training set for r=10: " + str(score_train))
```

```
Accuracy on the training set for r=10: 0.7685185185185185
```

In [54]:
```
score_test = logreg.score(X = F_test, y = test_labels)
print("Accuracy on the test set for r=10: " + str(score_test))
```

```
Accuracy on the test set for r=10: 0.8
```

In [35]:

```
r = 1
scores = []
while r<=200:
    F = X.dot(vT[:r,:].T)
    F_test = X_test.dot(vT[:r,:].T)
    logreg.fit(F, train_labels)
    score_test = logreg.score(X = F_test, y = test_labels)
    scores.append(score_test)
    r+=1
```

In [56]:

```
count = []
i = 1
while i<=200:
    count.append(i)
    i+=1
```

In [65]:

```
plt.plot(count, scores)
plt.xlabel("r")
plt.ylabel("Accuracy")
plt.title("Test set accuracy")
plt.show()
```