CS178_HW5

Sergio Elias

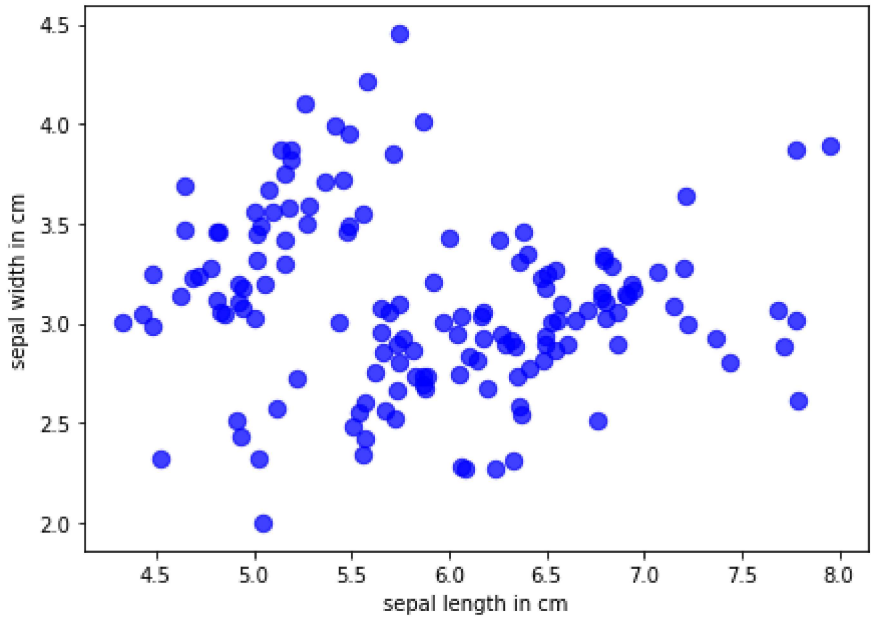
In [1]:
```
#from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
import mltools.transforms
import scipy.linalg
#import mltools.cluster as cl

np.random.seed(0)
%matplotlib inline
```

In [2]:
```
iris = np.genfromtxt("data/iris.txt" ,delimiter = None)
#print(iris.shape) # (148, 5)
```

In [3]:
```
X = iris[:, 0:2]     # first two feautes - columns
Y = iris[:, -1]   # tager values the last column
# print(X.shape , Y.shape) # (148, 2) (148)
```

1.1

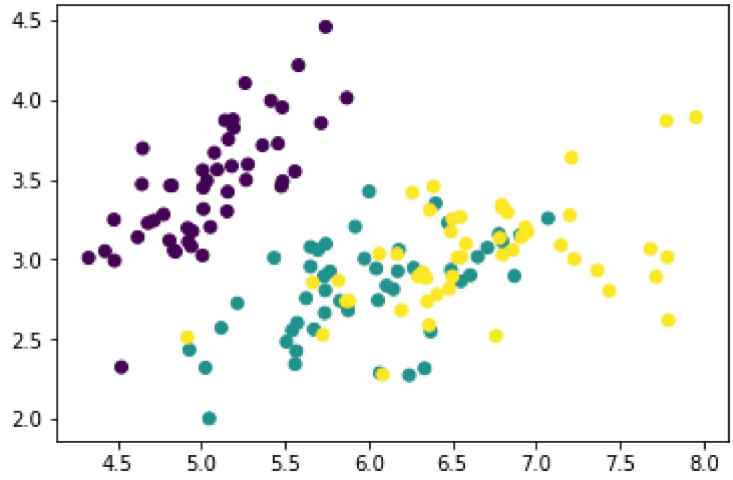
In [4]:
```
f, ax = plt.subplots(1, 1, figsize=(7, 5))

ax.scatter(X[:,0], X[:,1], s=70, color='blue', alpha=0.75)
plt.xlabel("sepal length in cm")
plt.ylabel("sepal width in cm")

plt.show()
```



It think in the plot above their should exist from (2 - 4) clusters.

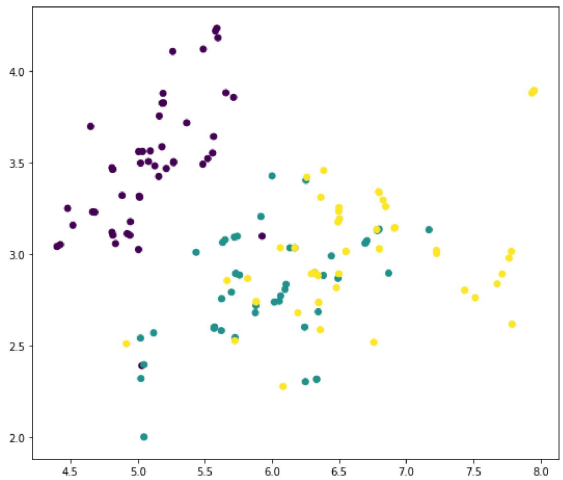
1.2

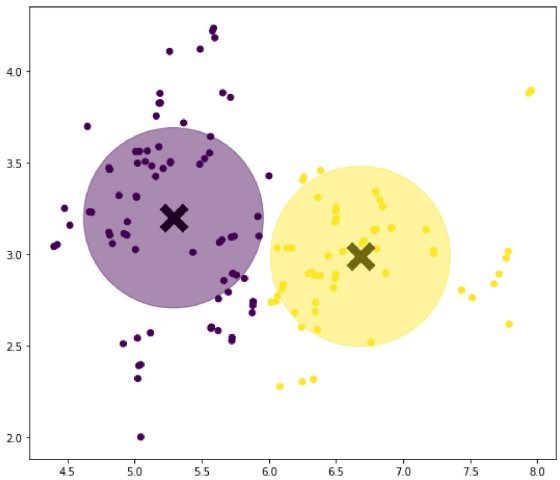
In [5]:
```
#X,Y = ml.datagen.data_GMM(500, 3, get_Z=True) # Random data distribution

plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.show()
```



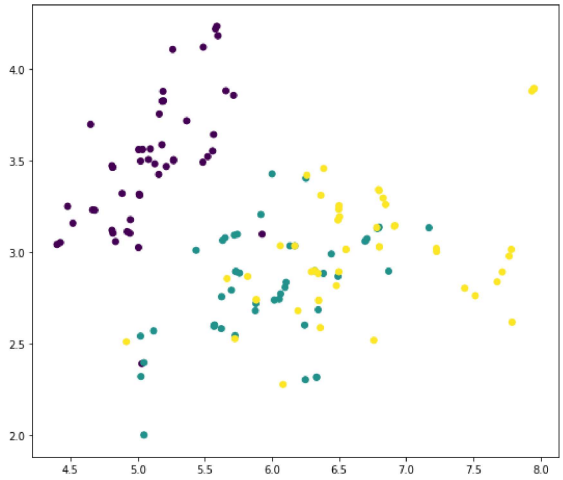
In [26]:
```
import warnings
warnings.filterwarnings('ignore')
# best = 99999

# ks = [2,5,20]
# for i in list(ks):
#     z, c, sumd = ml.cluster.kmeans(X, K= i, init = 'k++') #Try a few differe
nt initializations and check
#     #wheter they find the same solution

#     ml.plotClassify2D(None, X, z)
#     plt.plot([c[:,0]], [c[:,1]], 'ro')

#     plt.title('Random + distance (“k-means++”)')
#     print('\033[1m' +'K = '+ '\033[0m' , i, )
#     plt.show()
#     best = min(best, sumd)
#     print ('best score:', best)
# warnings.filterwarnings('default')

ks = [2, 5, 20]

for k in ks:
    d_base    = []  # list of tuples
    for i in range(5):
        np.random.seed(i)
        Z, mu, sumd = ml.cluster.kmeans(X, K= k, init = 'k++')
        d_base.append( (i, Z, mu, sumd) )
        #sumd_list.append( sumd )

    k_means_i = min(d_base, key=lambda x: x[3]) # get best sumd
    i, Z, mu, sumd = k_means_i
    print("k =", k, "\nseed_i =", i, "\nbest score =", sumd)

    fig, ax = plt.subplots(1, 2, figsize=(20, 8))
    # Plotting the original data
    ax[0].scatter(X[:, 0], X[:, 1], c=Y)

    # Plotting the clustered data
    ax[1].scatter(X[:, 0], X[:, 1], c=Z) # Plotting the data
    ax[1].scatter(mu[:, 0], mu[:, 1], s=500, marker='x', facecolor='black', lw
=8) # Plotting the centroids
    ax[1].scatter(mu[:, 0], mu[:, 1], s=30000, alpha=.45, c=np.unique(Z)) # La
zy way of plotting the clusters area :)

    plt.show()

print('\033[1m' + 'The one with the best score is when we run K-means 20 Best
 Score :'  + '\033[0m', sumd)
```

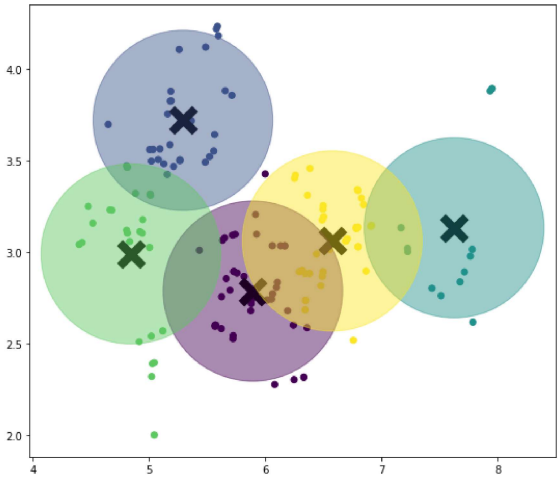
```
k = 2
seed_i = 0
best score = 58.7968398311
```



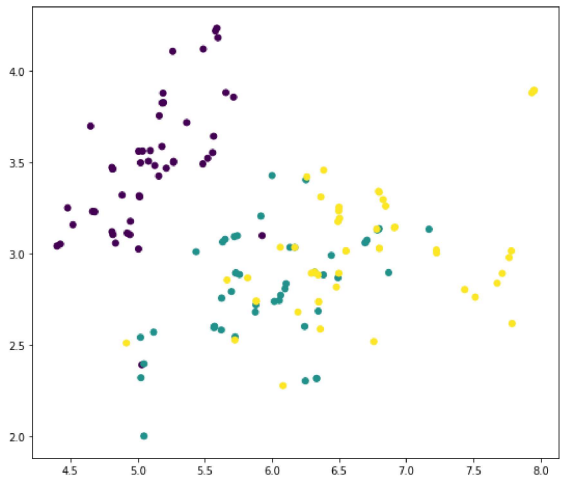


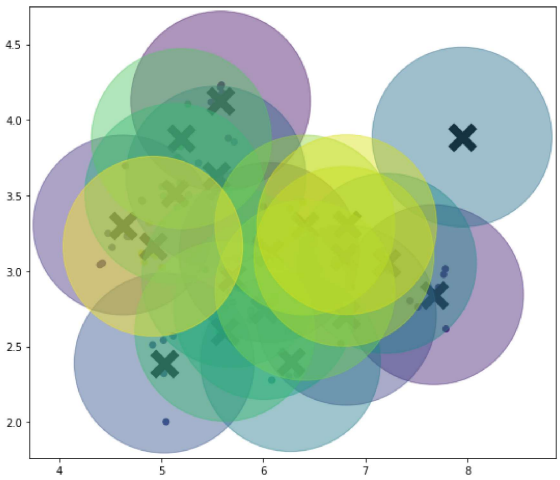
```
k = 5
seed_i = 2
best score = 20.7226648286
```





```
k = 20
seed_i = 2
best score = 3.10086262025
```



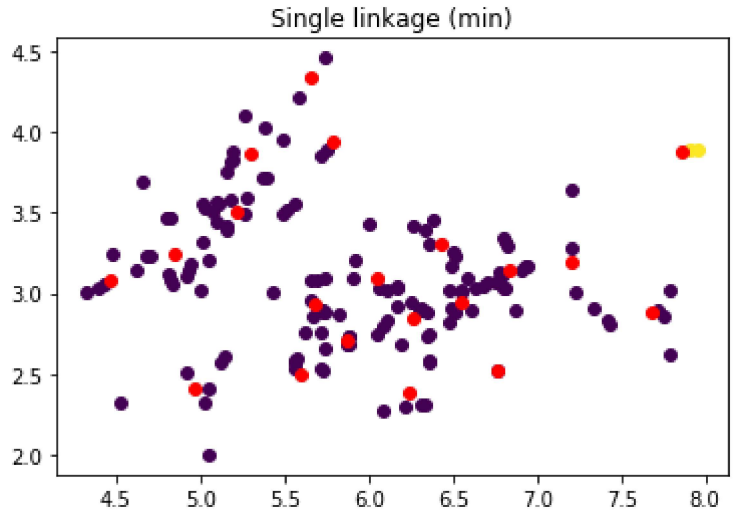


```
The one with the best score is when we run K-means 20 Best Score : 3.10086262
025
```

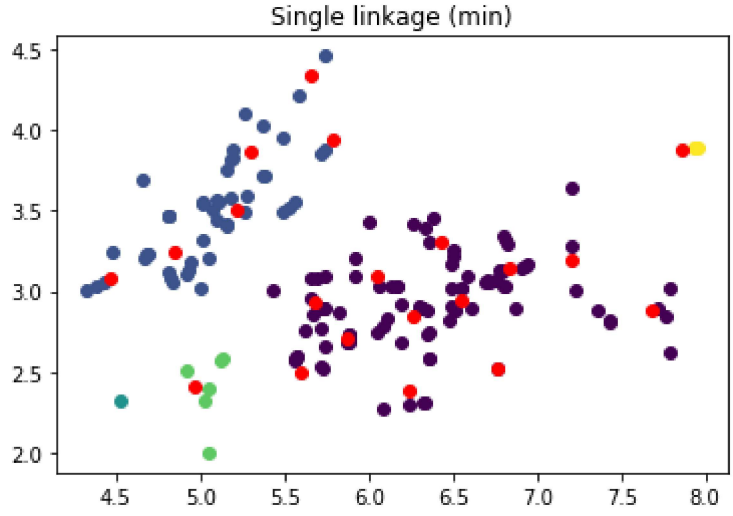
1.3

In [10]:
```
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = (6,4)

ks = [2,5,20]
for i in list(ks):
    z, join = ml.cluster.agglomerative(X, K=i, method ='min')
    ml.plotClassify2D(None, X, z)
    plt.title('Single linkage (min)')
    print('\033[1m' +'K = '+ '\033[0m' , i, )
    plt.plot([mu[:,0]], [mu[:,1]], 'ro');
    plt.show()


warnings.filterwarnings('default')
```

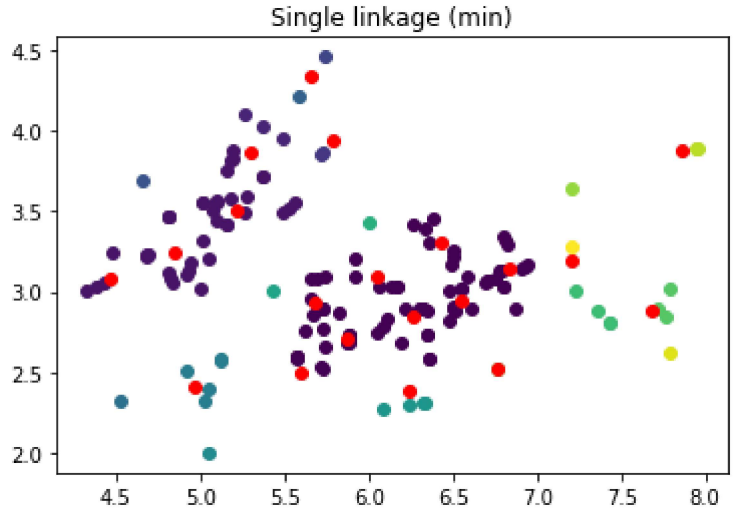
```
K =  2
```


Single linkage (min)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

```
K =  5
```

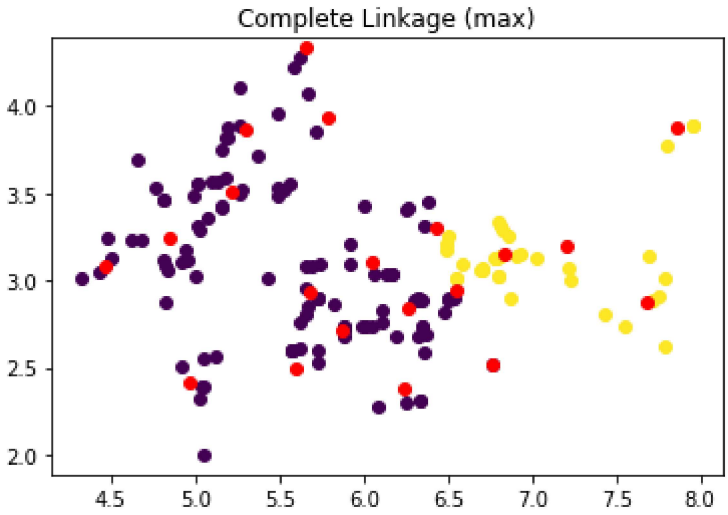

Single linkage (min)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

```
K =  20
```

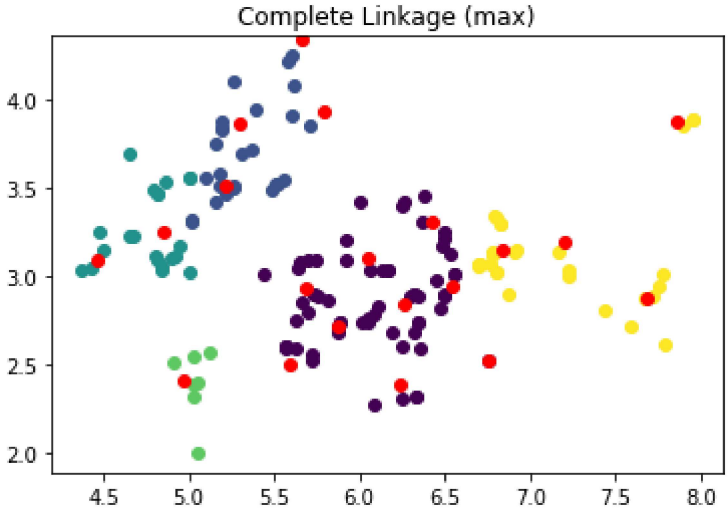

Single linkage (min)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

In [12]:
```
import warnings
warnings.filterwarnings('ignore')
#1c (k=5, single linkage)

plt.rcParams['figure.figsize'] = (6,4)


ks = [2,5,20]
for i in list(ks):
    z, join = ml.cluster.agglomerative(X, K=i, method ='max')
    ml.plotClassify2D(None, X, z)
    plt.title('Complete Linkage (max)')
    print('\033[1m' +'K = '+ '\033[0m' , i, )
    plt.plot([mu[:,0]], [mu[:,1]], 'ro');
    plt.show()


warnings.filterwarnings('default')
```

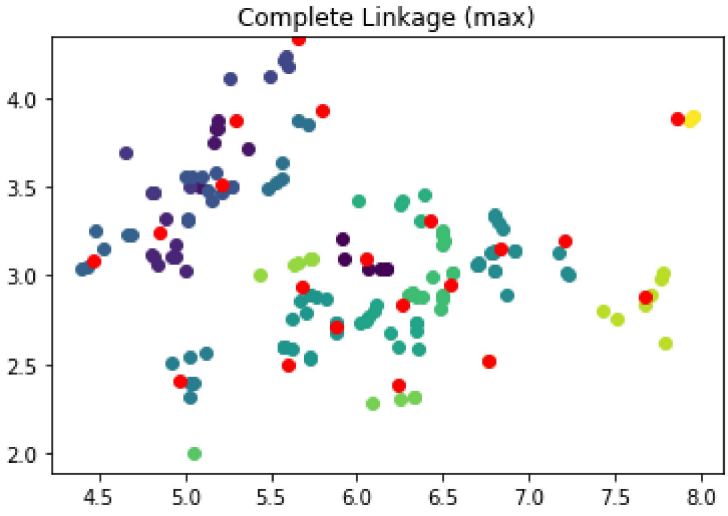
K = 2


Complete Linkage (max)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

K = 5


Complete Linkage (max)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

K = 20


Complete Linkage (max)
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

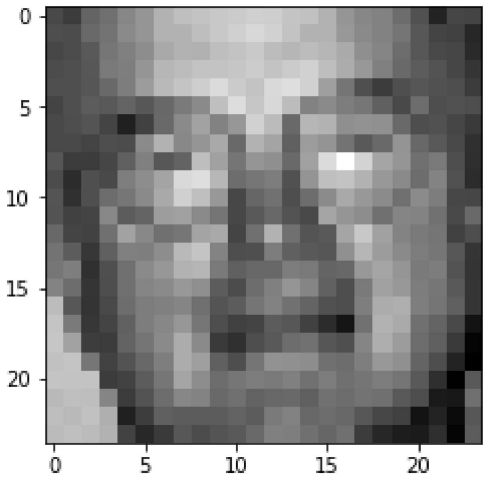
1.4 Both agglomerative and k-means have similarities for k = 2, but as k increases then you start seeing differences. They both have different partioning. With k-means the partitions seem to be independent to each other as in agglomerative it can give different partionings depending on the level of K we are looking at.

2.1

In [13]:
```
faces = np.genfromtxt("data/faces.txt" ,delimiter = None)
#print(faces.shape) # (4916, 576)
```

In [14]:
```
#plt.figure()
# pick a data point i for display
from random import *

i = randint(0, 4915)
print('Image i = ', i)
img = np.reshape(faces[i,:],(24,24)) # convert vectorized data to 24x24 image
 patches
plt.imshow( img.T , cmap="gray") # display image patch; you may have to squint

print('Here we are understanding the data format using PCA.')
```

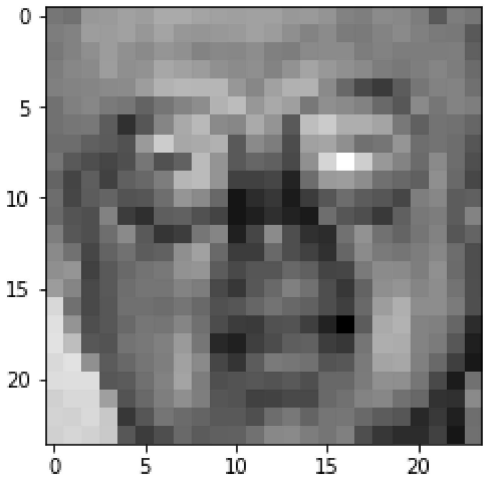
```
Image i =  3149
Here we are understanding the data format using PCA.
```



In [15]:
```
print('Subtract the mean of the face images (X0 = X - U)')
print('Ploting the mean face for image i = ', i)

mu = np.mean( faces, axis=0, keepdims=True )
X0 = faces - mu

img_mean = np.reshape(X0[i,:],(24,24)) # convert vectorized data to 24x24 imag
e patches
plt.imshow( img_mean.T , cmap="gray") # display image patch; you may have to s
quint
```

```
Subtract the mean of the face images (X0 = X - U)
Ploting the mean face for image i =  3149
```

Out[15]: <matplotlib.image.AxesImage at 0x13827b8d160>



2.2

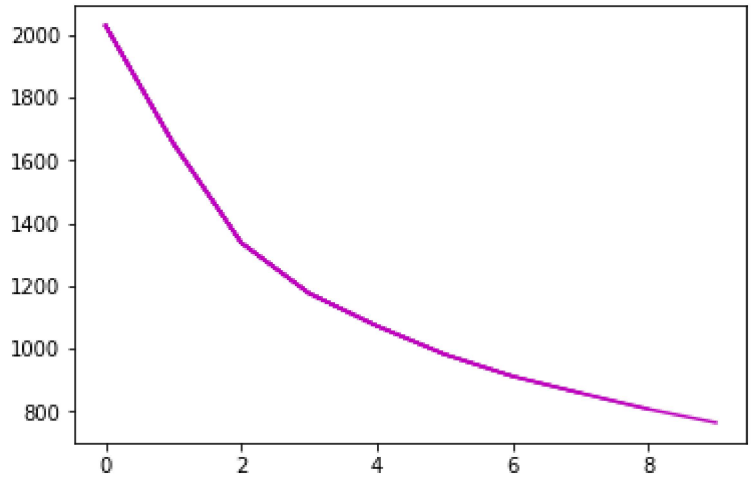
In [16]:
```
U, S, Vh = scipy.linalg.svd(X0, full_matrices = False)
W = U.dot( np.diag(S) )
print('Vh Shape: ', Vh.shape)
print('W Shape : ', W.shape)
```

```
Vh Shape:  (576, 576)
W Shape :  (4916, 576)
```

2.3

In [17]:
```
print('Printing and ploting MSE values for K= 1...10.')
mseList = [None] * 10

for K in range(1,11):
    Xhat = W[:, :K].dot( Vh[:K,:] )
    mseList[K-1] = np.mean(((X0-Xhat)**2))
    plt.plot(range(10), mseList , 'm-')
    print('K =', K, ':', mseList[K-1])
```

```
Printing and ploting MSE values for K= 1...10.
K = 1 : 2028.5457074
K = 2 : 1653.32626374
K = 3 : 1337.38893397
K = 4 : 1176.27288642
K = 5 : 1072.28739065
K = 6 : 981.659208432
K = 7 : 912.146369559
K = 8 : 858.677138779
K = 9 : 807.432078317
K = 10 : 764.486897211
```



2.4

In [18]:
```
print('Here I present the first three images by computing: ')
print('μ + α V[j,:] \n\n')

for K in range(3):
    alpha = 2*(np.median(np.abs( W[:,K])))
    image = np.reshape( mu + alpha*Vh[K, :], (24,24))
    plt.figure()
    plt.title('μ + α V[j,:]')
    plt.imshow(image.T, cmap='gray')
    print('Image: ', K + 1)
    plt.show()
```

Here I present the first three images by computing:
μ + α V[j,:]

Image: 1

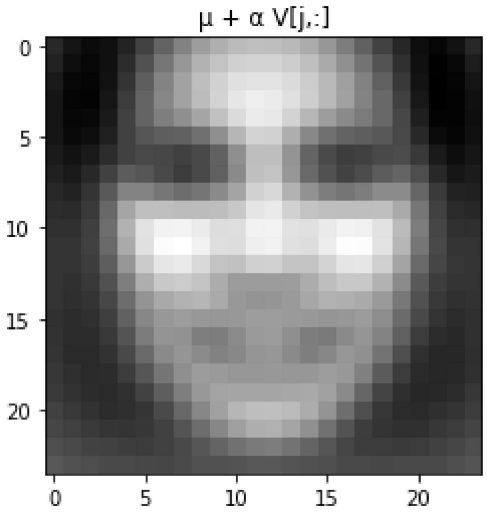


Image: 2

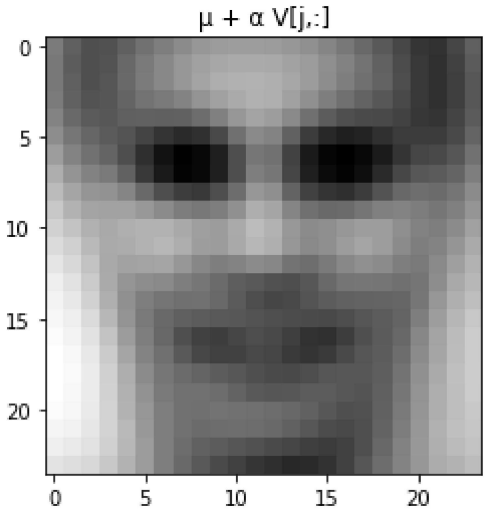
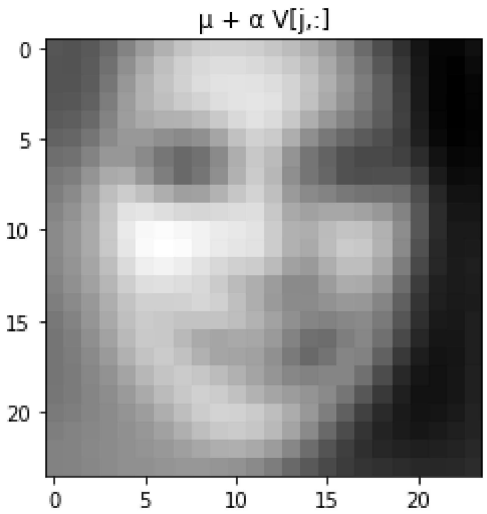


Image: 3



In [19]:
```
print('Here I present the first three images by computing: ')
print('μ - α V[j,:] \n\n')

for K in range(3):
    alpha = 2*(np.median(np.abs( W[:,K])))
    image1 = np.reshape( mu - alpha*Vh[K, :], (24,24))
    plt.figure()
    plt.title('μ - α V[j,:]')
    plt.imshow(image1.T, cmap='gray')
    print('Image: ', K + 1)
    plt.show()
```

Here I present the first three images by computing:
μ - α V[j,:]

Image: 1

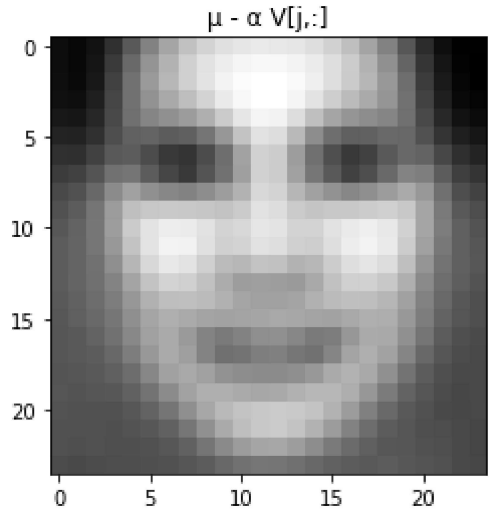


Image: 2

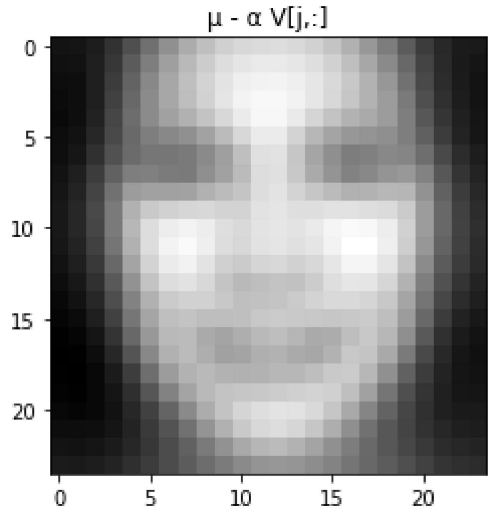
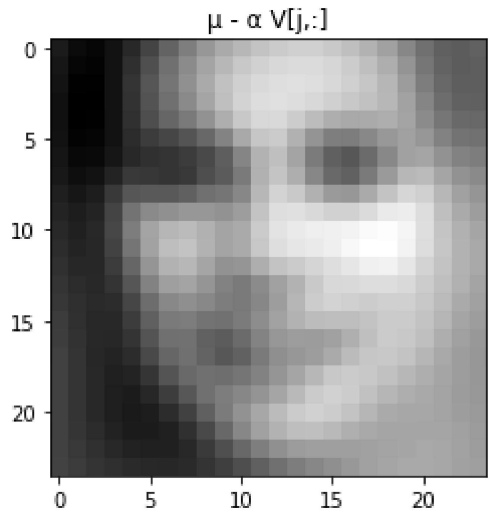


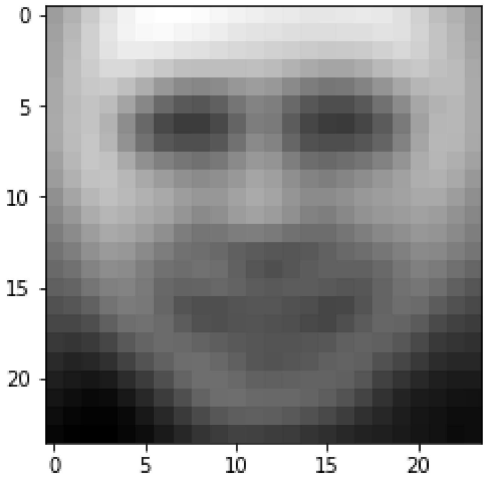
Image: 3



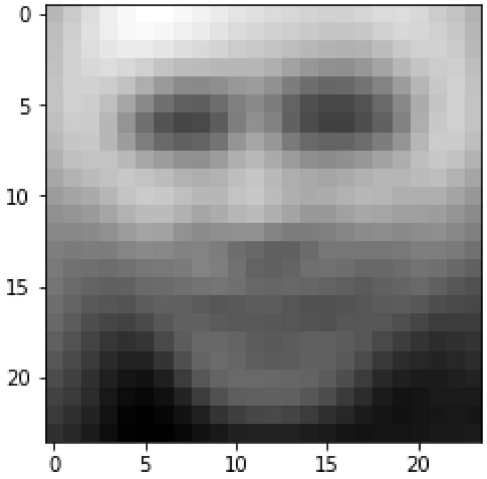
2.5

In [20]:
```
#W Shape :  (4916, 576)
from random import *

x1 = randint(0, 4915)    # Pick a random number.
listK = [5,10,50,100]


for j,k in enumerate(listK):
    print('First Random Image # : ', x1)
    print('K = ', k)
    image = mu + W[x1, :k].dot(Vh[:k])
    image = np.reshape(image, (24,24))
    plt.figure()
    plt.imshow(image.T, cmap='gray')
    plt.show()

print('================================================================================
===========')

x2 = randint(0, 4915)    # Pick a random number.

for j,k in enumerate(listK):
    print('Second Random Image # : ', x2)
    print('K = ', k)
    image = mu + W[x2, :k].dot(Vh[:k])
    image = np.reshape(image, (24,24))
    plt.figure()
    plt.imshow(image.T, cmap='gray')
    plt.show()
```

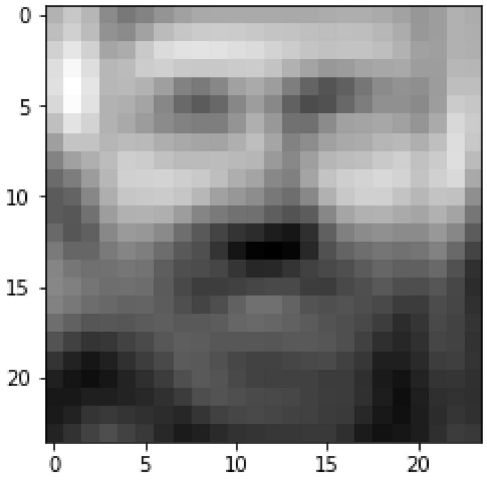
```
First Random Image # :  4077
K =  5
```



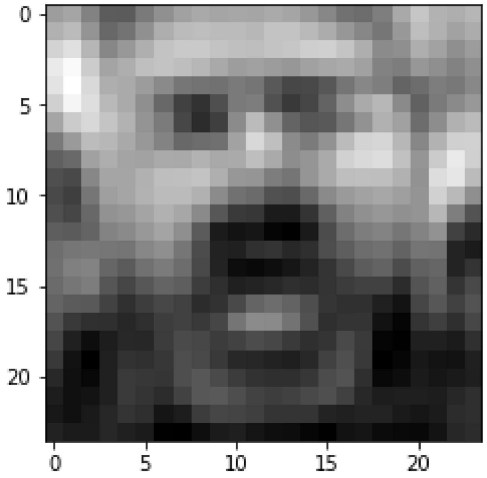
```
First Random Image # :  4077
K =  10
```



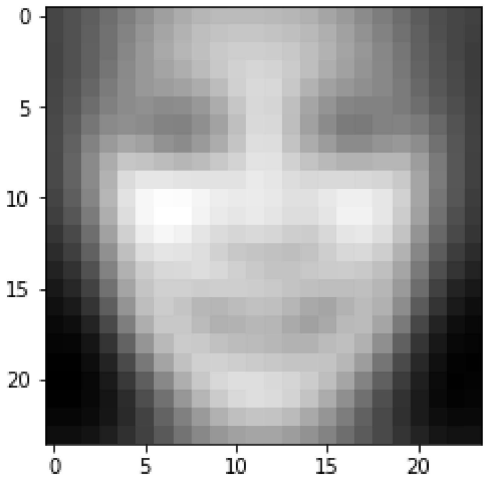
```
First Random Image # :  4077
K =  50
```



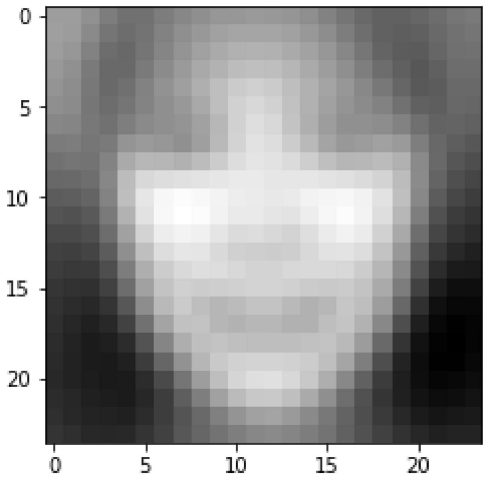
```
First Random Image # :  4077
K =  100
```



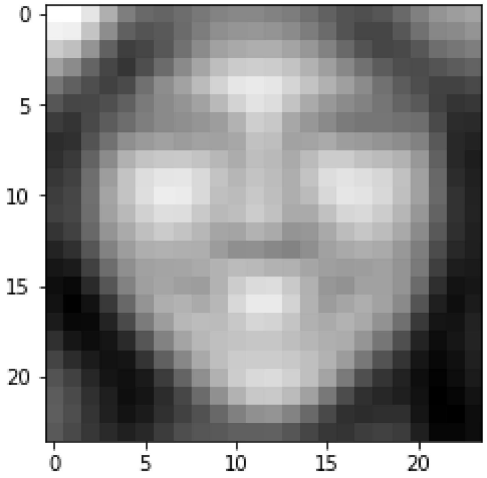
```
================================================================================
=====
Second Random Image # :  2261
K =  5
```



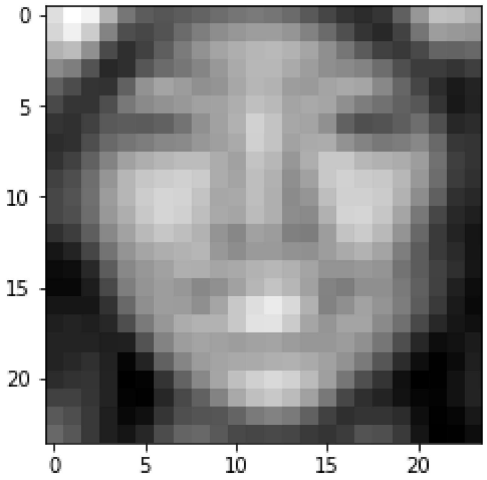
```
Second Random Image # :  2261
K =  10
```



```
Second Random Image # :  2261
K =  50
```



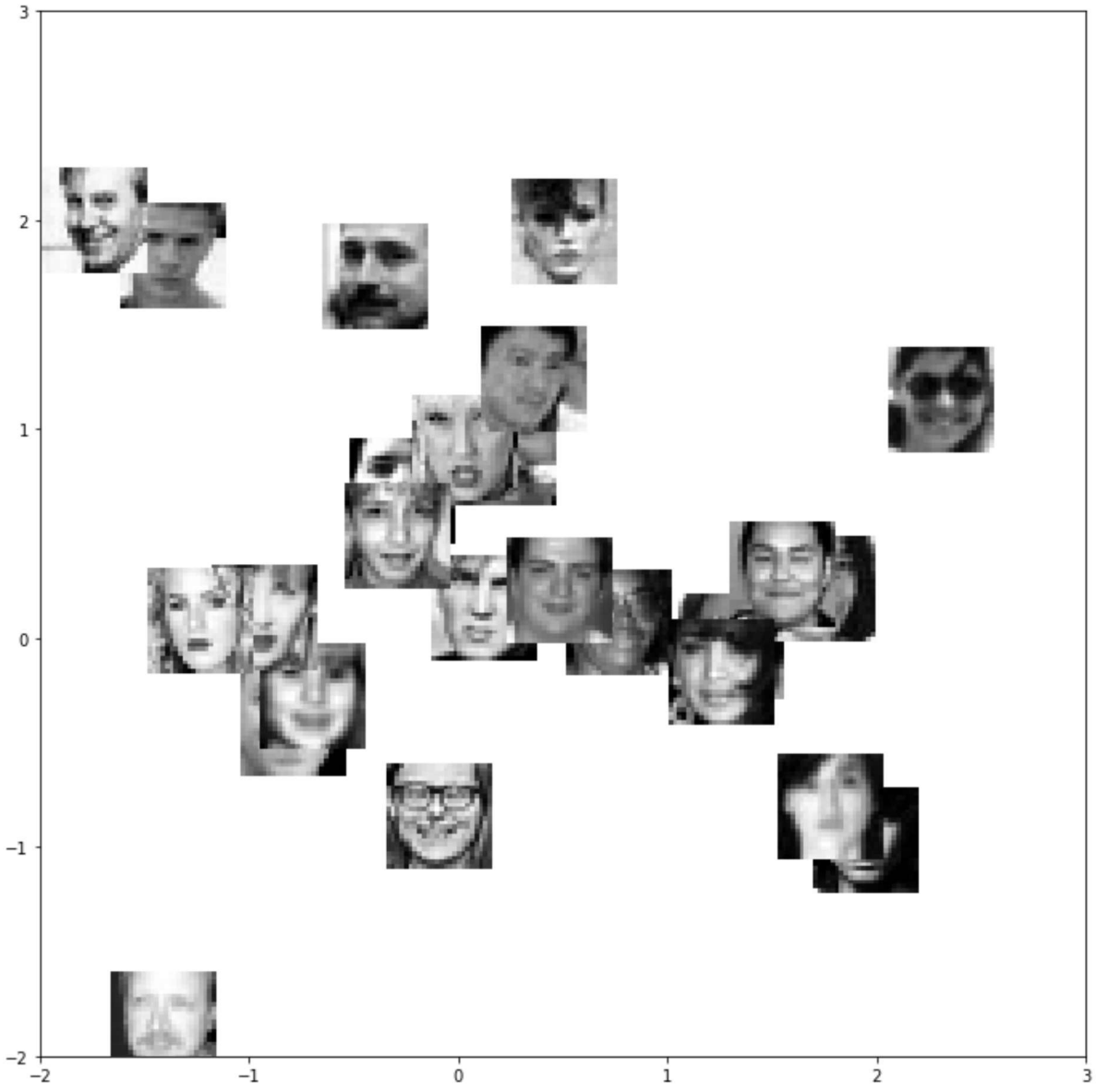

```
Second Random Image # :  2261
K =  100
```



2.6

In [21]:
```
import warnings
warnings.filterwarnings('ignore')


idx = [[]]*25
for i in range(25):
    randInt = randint(0, 4915)
    idx[i] = randInt

print('Random image array: ',idx)

# pick some data (randomly or otherwise); an array of integer indices

plt.rcParams['figure.figsize'] = (12, 12)
coord,params = ml.transforms.rescale( W[:,0:2] )
# normalize scale of "W" locations
plt.figure(); plt.hold(True); # you may need this for pyplot


for i in idx:

    # compute where to place image (scaled W values) & size
    loc = (coord[i,0],coord[i,0]+0.5, coord[i,1],coord[i,1]+0.5)
    img = np.reshape( faces[i,:], (24,24) ) # reshape to square
    #plt.figure(figsize=(20,10))
    plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
    plt.axis( (-2,3,-2,3) )

plt.show()




warnings.filterwarnings('default')
```

```
Random image array:  [641, 83, 2503, 2651, 4703, 3661, 772, 3927, 4397, 2162,
4273, 4678, 2050, 4624, 3531, 1839, 1788, 1646, 1158, 159, 3488, 2537, 1479,
3326, 4444]
```



Statement of Collaboration

Troughout the code snippet I provided as much information needed. With the code snippet provided by the Proffesor/TA on github I was able to implement the information needed to accomplish the homework problems. Going over lecture notes, piazza, stack overflow and looking at the ml tools file I was able to work on the implemented code needed. I also put detailed comment in how I was able to calculate and plot of the homework question. Although most of the steps where straight forward, the most challenging part for me was to be able to implement 2.4. Like I mentioned above, with the help of piazza and the discussion lecture notes, I was able to accomplish it.