HW5_178_Bryan_Oliande_13179240

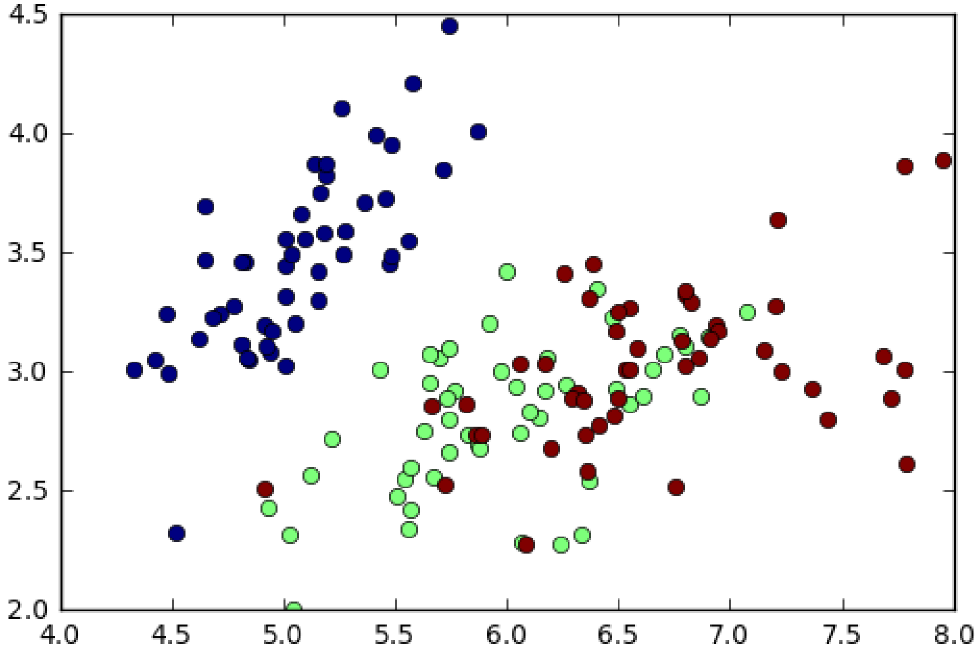
March 13, 2017

PROBLEM 1: BASICS OF CLUSTERING

(a) Load the usual Iris data restricted to the first two features, and ignore the class/target variable. Plot the data and see how clustered it looks

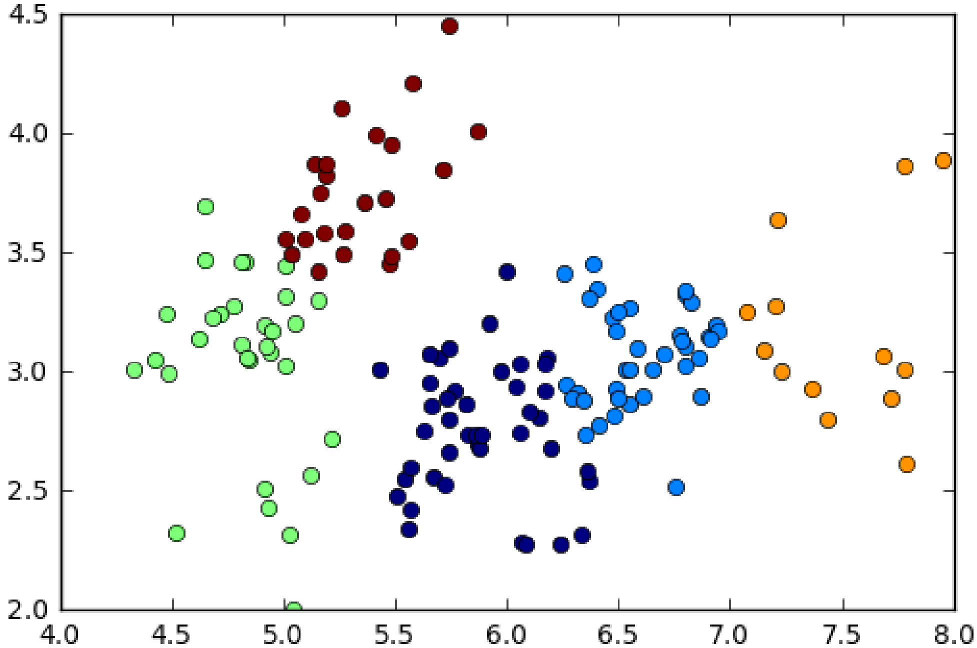
```
In [1]: import numpy as np
        import mltools as ml
        import matplotlib.pyplot as plt
        %pylab inline


        iris = np.genfromtxt("Desktop/178/HW5-code/data/iris.txt",delimiter=None) #

        Y = iris[:,-1] # target value is the last column

        X = iris[:,0:2] # features are the other columns

        ml.plotClassify2D(None,X,Y)
```

```
Populating the interactive namespace from numpy and matplotlib
```

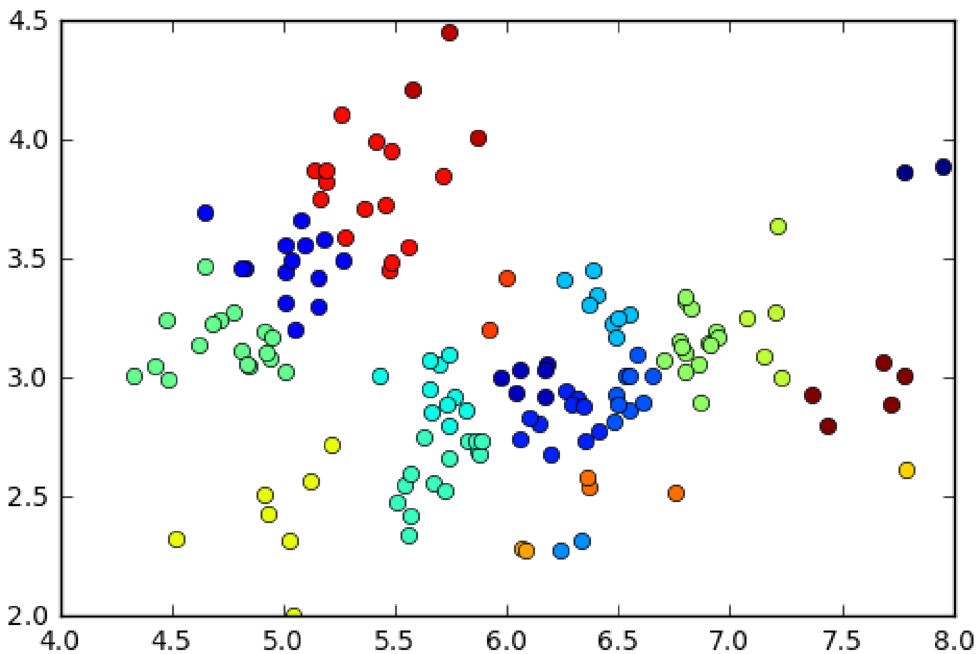


(b) Run k-means on the data, for k = 5 and k = 20. For each, turn in a plot with the data, colored by assignment, and the cluster centers using ml.plotClassify2D(None, X, z) #z is the cluster assignment of the data Try a few different initializations and see if they are the same.

In [2]:
```
#***NOTE: Different initializations were tried 'k++',
#'farthest', and different iteration values)
#and the default settings gave the best results
(z, c, sumd) = ml.cluster.kmeans(X, 5, init = 'random')
ml.plotClassify2D(None, X, z )
```

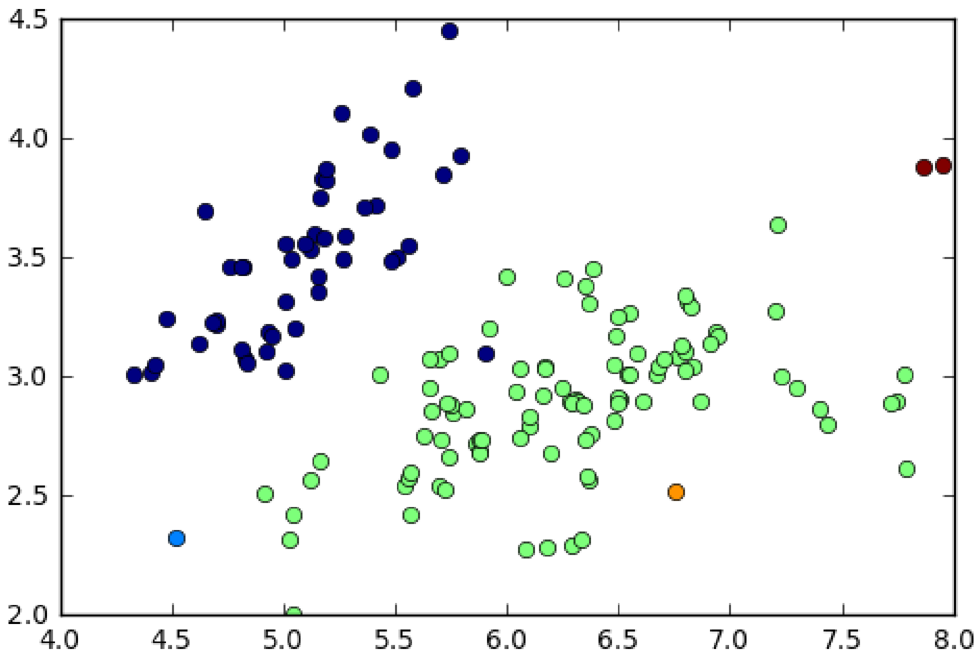



```
In [3]: #***NOTE: Different initializations were tried 'k++',
        #'farthest', and different iteration values)
        #and the default settings gave the best results
        (z, c, sumd) = ml.cluster.kmeans(X, 20)
        ml.plotClassify2D(None, X, z )
```

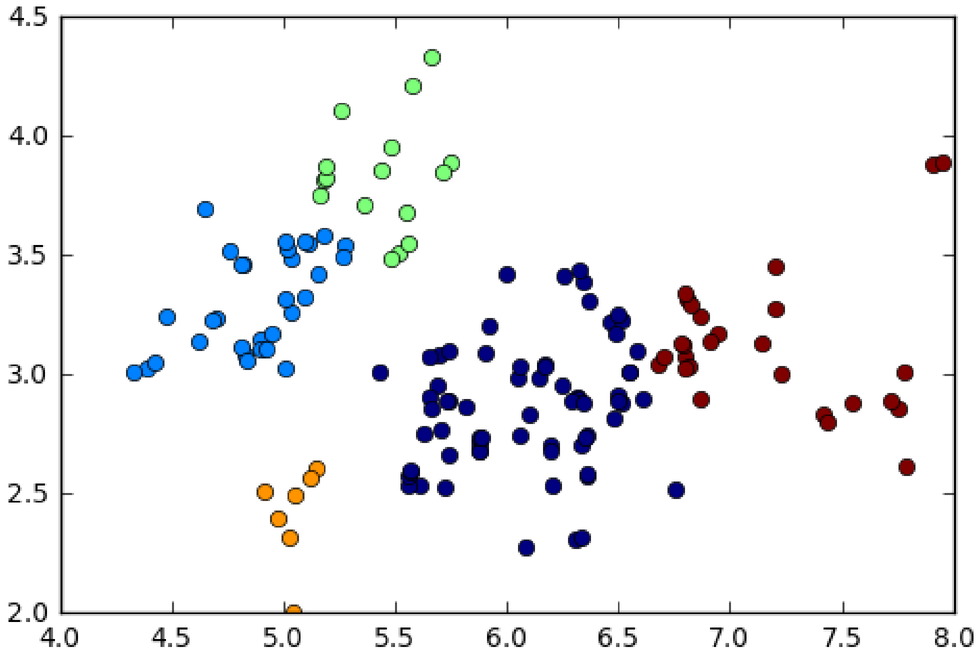


(c) Run agglomerative clustering on the data, using single linkage and then again using complete linkage, each with 5 and then 20 clusters. Again, plot with color the final assignment of the clusters, and describe their differences from each other and k-means

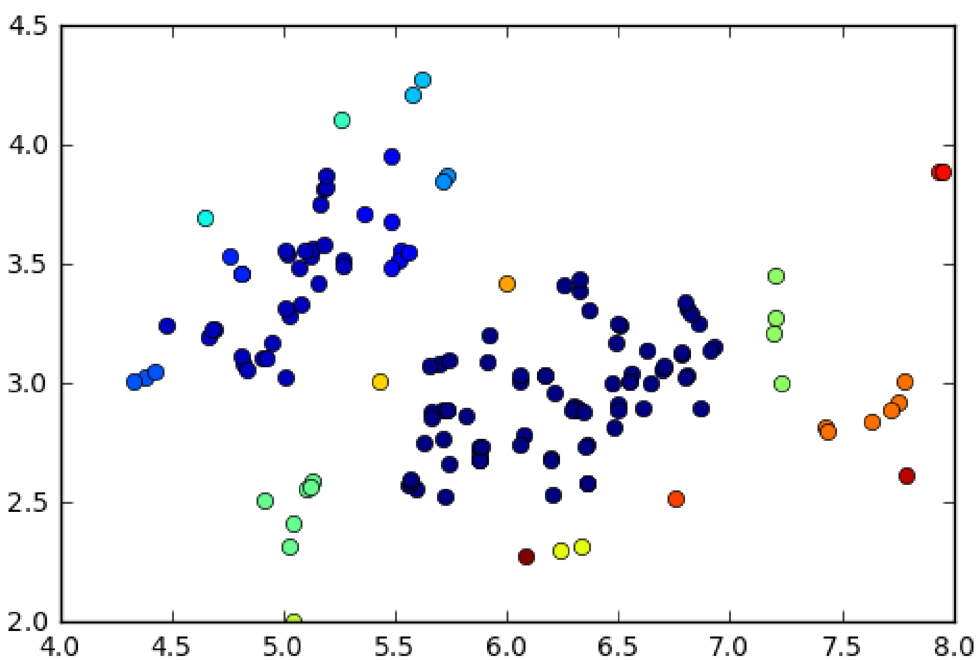
In [4]:
```
#***NOTE: Different initializations were tried including 'min', 'max',
#'means', or 'average'.
#and the default settings gave the best results
(z, c)= ml.cluster.agglomerative(X, 5, 'min') # min = single linkage
ml.plotClassify2D(None, X, z )
```



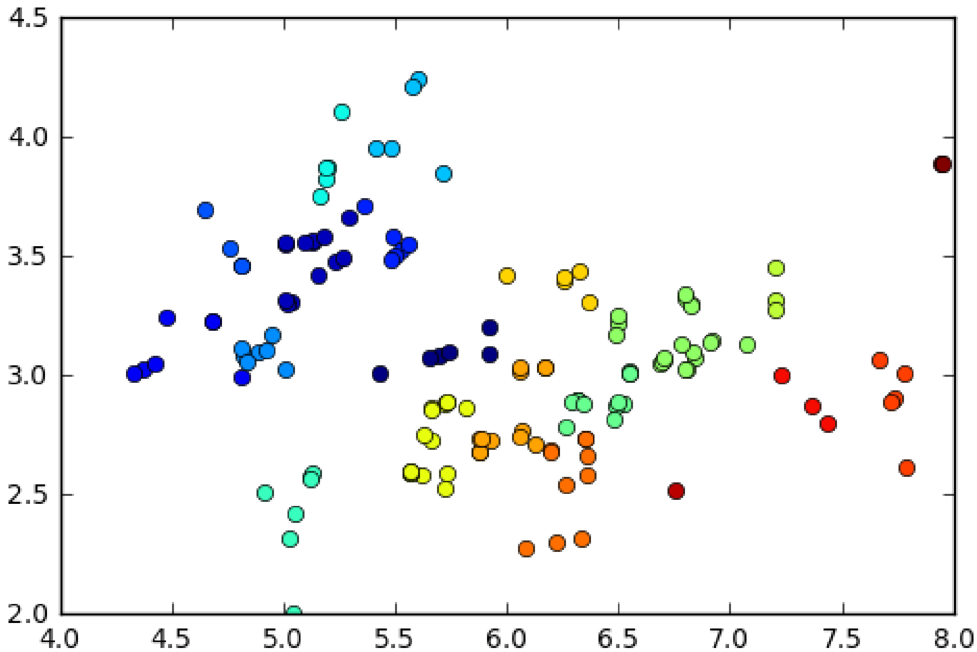
In [5]:
```
#***NOTE: Different initializations were tried including 'min',
#'max', 'means', or 'average'.
#and the default settings gave the best results
(z, c) = ml.cluster.agglomerative(X, 5, 'max') #max = complete linkage
ml.plotClassify2D(None, X, z )
```




```
In [6]: #***NOTE: Different initializations were tried including 'min',
        #'max', 'means', or 'average'.
        #and the default settings gave the best results
        (z, c)= ml.cluster.agglomerative(X, 20, 'min') # min = single linkage
        ml.plotClassify2D(None, X, z )
```



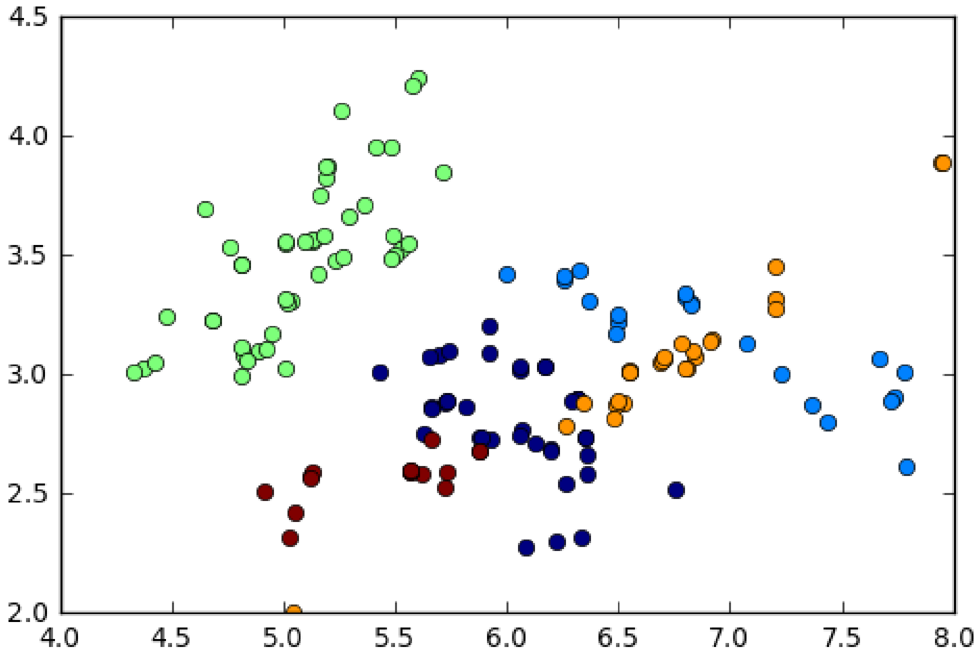
```
In [7]: #***NOTE: Different initializations were tried including 'min',
        #'max', 'means', or 'average'.
        #and the default settings gave the best results
        (z, c)= ml.cluster.agglomerative(X, 20, 'max') # max = complete linkage
        ml.plotClassify2D(None, X, z )
```



From the graphs, it seems like K = 5 is the best setting for the number of clusters. K = 20 is just far too many clusters to assign to only 2 outcome classeses. Single linkage gives somewhat scattered results, while complete linkage gives better looking results.

(d) EXTRA CREDIT: Run the Gaussian EM model with 5 components

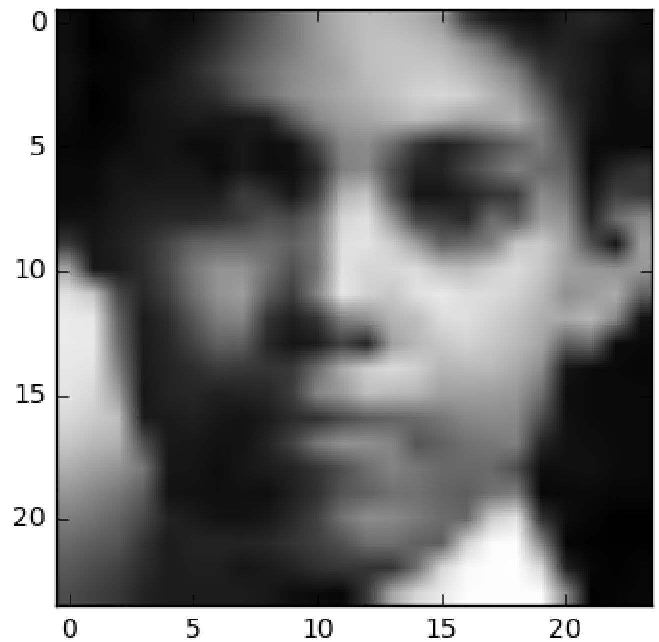
```
In [8]: (z, T, soft, ll)= ml.cluster.gmmEM(X, 5)
        ml.plotClassify2D(None, X, z )
```



Compare/Discuss differences with other clusterings. What do you think is most reasonable? It seems that the clustering with GMM are not separated cleanly, the clusterings for Kmeans with K = 5 are much nicer looking.

PROBLEM 2: EIGENFACES

```
In [9]: X = np.genfromtxt("Desktop/178/HW5-code/data/faces.txt",delimiter=None) # l
        plt.figure
        # pick a data point i = 5  for display
        img = np.reshape(X[5,:],(24,24)) # convert vectorized data point to 24x24 i
        plt.imshow( img.T , cmap="gray") # display image patch; you may have to squ

Out[9]: <matplotlib.image.AxesImage at 0x111100470>
```



(a) Subtract the mean of the face images to make your data zero-mean

```
In [10]: mu = X.mean(axis = 0)
         X_0 = X - mu
```

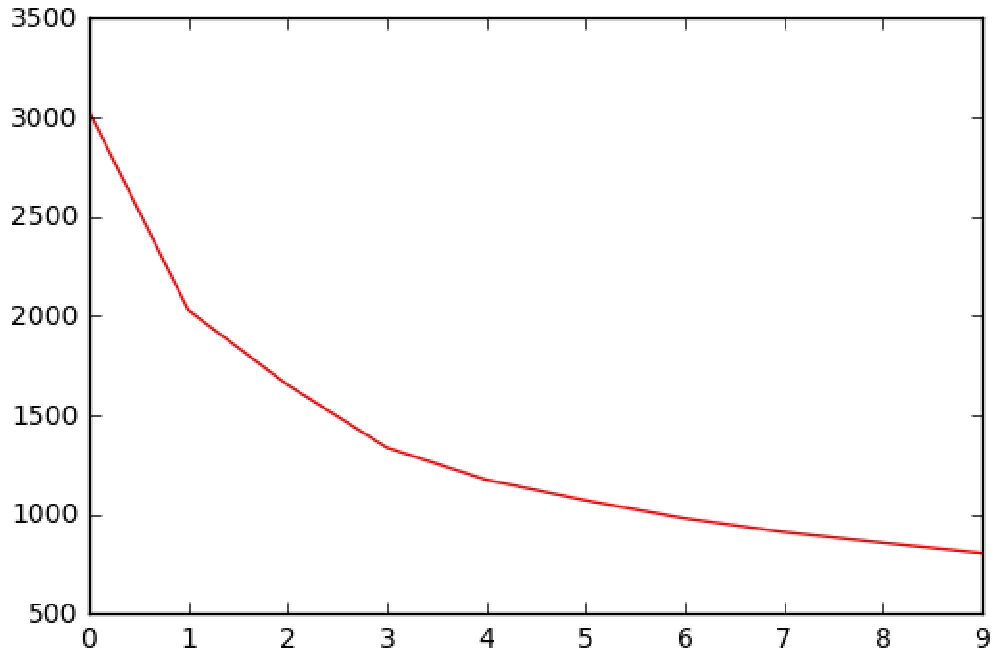
(b) Use scipy.linalg.svd to take the SVD of the data

```
In [11]: import scipy.linalg
         U, S, V = scipy.linalg.svd(X_0,  False)
         W = U.dot( np.diag(S) );
         print(U.shape, S.shape, V.shape)

(4916, 576) (576,) (576, 576)
```

(c) For K = 1...10, compute the approximation to X_0 given by the first k eigndirection.s Plot these MSE values as a function of K.

```
In [18]: err = [None] * 10
         numbers = [None] * 10

         for i in range(0, 10):
             numbers[i] = i
             Xhat_0 = W[:, :i].dot( V[:i, :])
             err[i] = ((X_0 - Xhat_0)**2).mean()

         plt.plot(numbers, err, 'r-')
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x110202438>]
```



(d) Display the first three principal directions of the data

```
In [13]: alpha = 2*np.median(np.abs(W[:,3]))
         im1 = np.reshape(mu + alpha*V[3, :], (24, 24))
         im2 = np.reshape(mu - alpha*V[3, :], (24, 24))
         _=plt.figure()
         f,[ax1,ax2] = plt.subplots(1,2);
         _=ax1.imshow(im1.T, cmap="gray"); _=ax1.axis('off')
         _=ax2.imshow(im2.T, cmap='gray'); _=ax2.axis('off')

<matplotlib.figure.Figure at 0x11126fcf8>
```

(e) Choose two faces and reconstruct them using only the first K principal directions, for K = 5, 10, 50, 100

```
In [19]: K = [5, 10, 50, 100]
         for k in K:
             alpha = 2*np.median(np.abs(W[:,k]))
             im1 = np.reshape(mu + alpha*V[k, :], (24, 24))
             im2 = np.reshape(mu - alpha*V[k, :], (24, 24))
             _=plt.figure()
             f,[ax1,ax2] = plt.subplots(1,2);
             _=ax1.imshow(im1.T, cmap="gray"); _=ax1.axis('off')
             _=ax2.imshow(im2.T, cmap='gray'); _=ax2.axis('off')
```

```
<matplotlib.figure.Figure at 0x11125f0b8>
```

<matplotlib.figure.Figure at 0x1101d5f60>

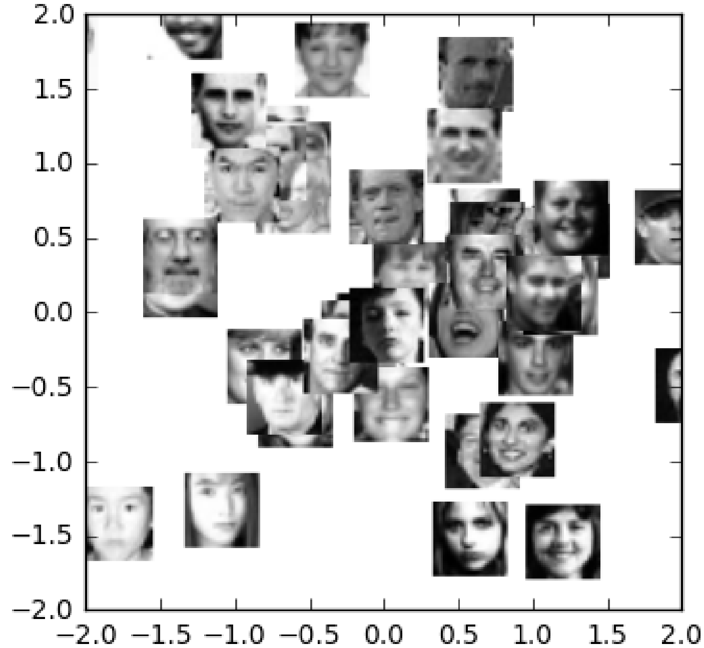
<matplotlib.figure.Figure at 0x11e15ff98>

<matplotlib.figure.Figure at 0x11e3409e8>

(f) Methods like PCA are often called "latent space" methods. Choose a few faces at random (about 15-25) and display them as images with the coordinates given by their coefficients on the first two principal components:

```
In [15]: idx = np.floor( 4916 * np.random.rand(50) ) # pick some data at random or
         idx = idx.astype('int')

         import mltools.transforms

         coord,params = ml.transforms.rescale( W[:,0:2] ) # normalize scale of "W"

         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( X[i,:], (24,24) ) # reshape to square
             plt.imshow( img.T , cmap="gray", extent=loc ) # draw each image
             plt.axis( (-2,2,-2,2) ) # set axis to reasonable visual scale
```



PROBLEM 3: WORK ON YOUR PROJECT
Okay.