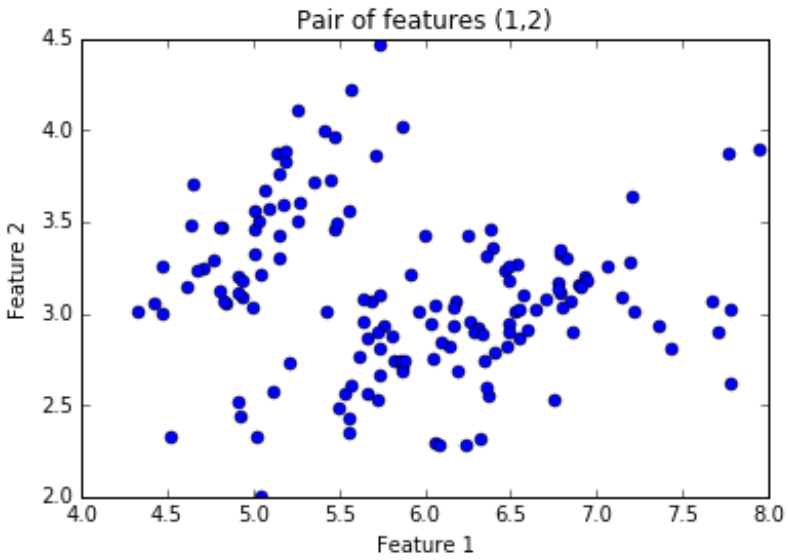
In []:

```
"""
Author: Tim Nguyen - 37486306
"""
```

Problem 1:

In [3]:

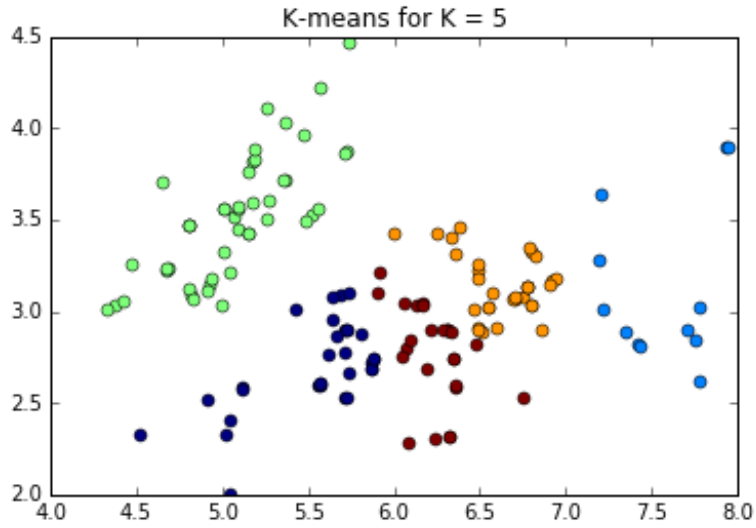
```
# Problem 1a:
import numpy as np
import mltools as ml
import mltools.cluster as cluster
import matplotlib.pyplot as plt

%matplotlib inline

iris = np.genfromtxt("data/iris.txt", delimiter=None) # load the text file
X = iris[:,0:2] # load the first two features

# plot the data
plt.title("Pair of features (1,2)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.plot(X[:,0], X[:,1], 'o');
```



In [42]:

```
# Problem 1b:
''' k = 5 '''
z, c, sumd = cluster.kmeans(X,5);

# plot the data
plt.title("K-means for K = 5")
ml.plotClassify2D(None, X, z);

print "Score =", sumd
```

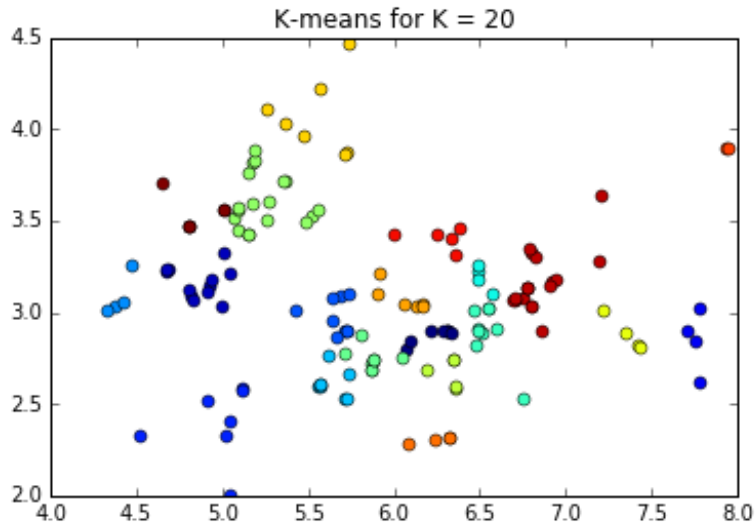
```
Score = 24.2654280204
```



In [41]:

```
''' k = 20 '''
z, c, sumd = cluster.kmeans(X,20);

# plot the data
plt.title("K-means for K = 20")
ml.plotClassify2D(None, X, z);

print "Score =", sumd
```

```
Score = 4.14226786301
```



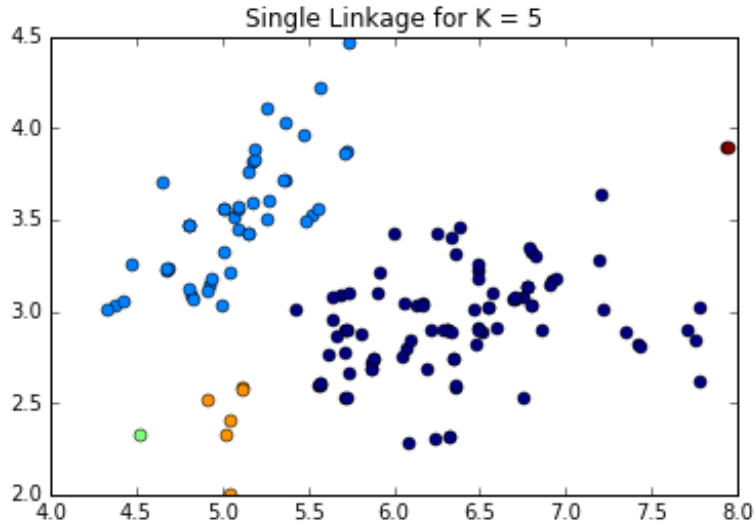
In []:

```
"""
According to the sumd scores above, we have the best score when k = 20
"""
```

In [40]:

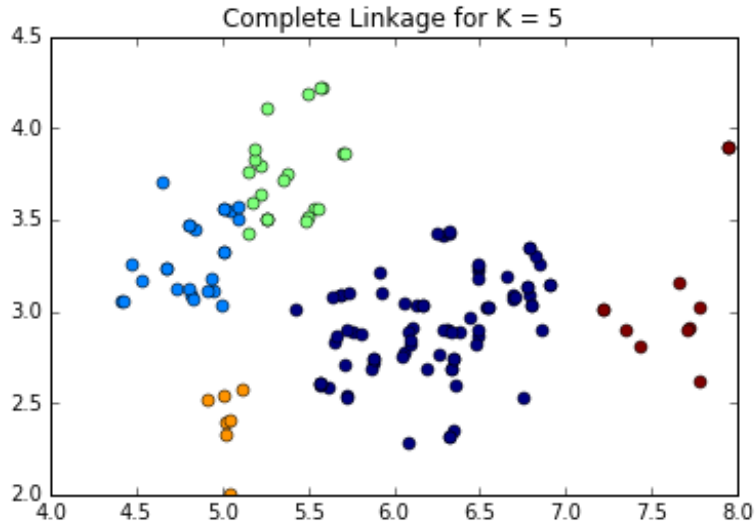
```
# Problem 1c
''' k = 5 single linkage '''
z, join = cluster.agglomerative(X, 5, method='min')

# plot the data
```

```
plt.title("Single Linkage for K = 5");
ml.plotClassify2D(None, X, z);
```



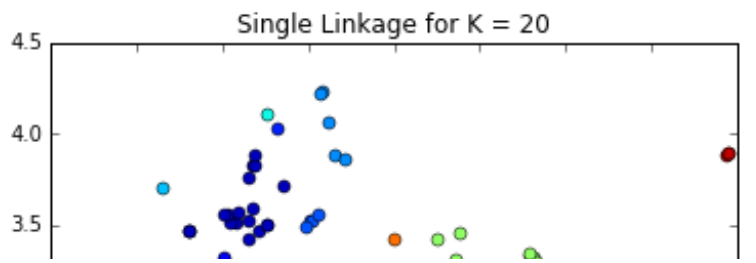
In [48]:

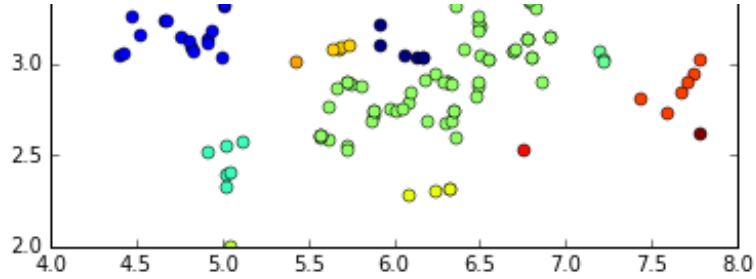
```
''' k = 5 complete linkage '''
z, join = cluster.agglomerative(X, 5, method='max')

# plot the data
plt.title("Complete Linkage for K = 5");
ml.plotClassify2D(None, X, z);
```



In [46]:

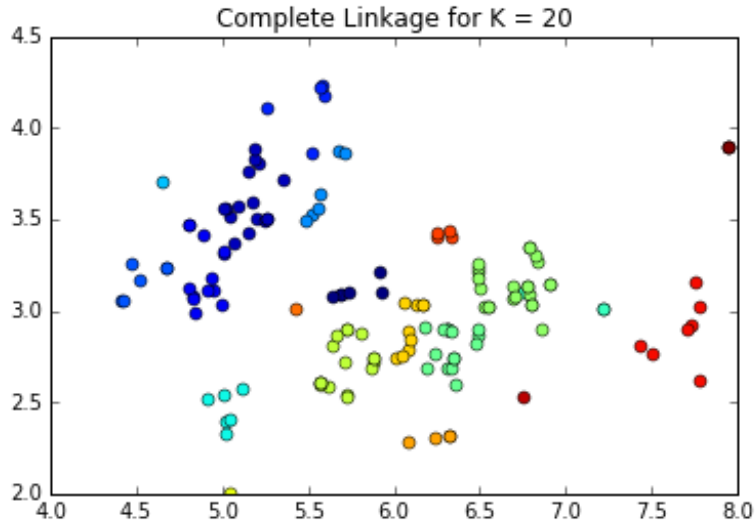
```
''' k = 20 single linkage '''
z, join = cluster.agglomerative(X, 20, method='min')

# plot the data
plt.title("Single Linkage for K = 20");
ml.plotClassify2D(None, X, z);
```





In [47]:

```
''' k = 20 complete linkage '''
z, join = cluster.agglomerative(X, 20, method='max')

# plot the data
plt.title("Complete Linkage for K = 20");
ml.plotClassify2D(None, X, z);
```



In []:

```
"""
According to the graphs, we can see that single linkage makes the model simpler
and more accurate than complete linkage. For k-mean,
it produces a single partitioning. However, agglomerative can give different pa
rtitionings depending on the level-of-resolution we
are looking at.
"""
```

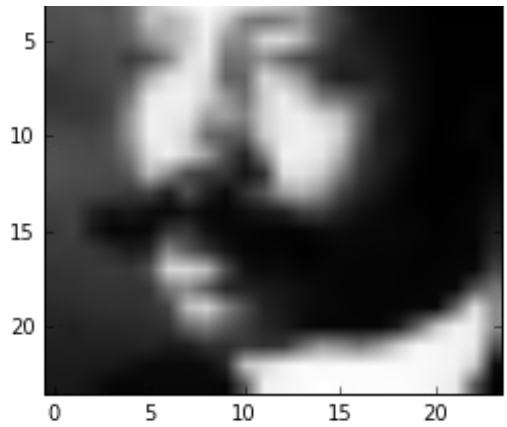
Problem 2:

In [61]:

```
X = np.genfromtxt("data/faces.txt", delimiter=None) # load face dataset
plt.figure()
img = np.reshape(X[0,:], (24,24)) # convert vectorized data point to 24x24
image patch
plt.imshow(img.T, cmap="gray") # display image patch; you may have to squint
```

Out[61]:

```
<matplotlib.image.AxesImage at 0x7f02d2043890>
```



In [62]:

```
# Problem 2a:
mean = np.mean(X, axis=0)
X0 = X - mean

print "X0 =", X0
```

```
X0 = [[   9.95240033   15.13161107   10.03254679 ...,  -18.58136697
   -84.98494711  -94.25386493]
 [ -64.04759967  -65.86838893  -61.96745321 ...,  -19.58136697
   -57.98494711  -97.25386493]
 [ -80.04759967  -76.86838893  -74.96745321 ...,  -35.58136697
   -38.98494711  -40.25386493]
 ...,
 [ -66.04759967  -64.86838893  -64.96745321 ...,  -72.58136697
   -71.98494711  -68.25386493]
 [ -21.04759967  -17.86838893  -15.96745321 ...,  -69.58136697
   -70.98494711  -72.25386493]
 [ -29.04759967  -30.86838893  -28.96745321 ...,  152.41863303
   150.01505289  149.74613507]]
```

In [63]:

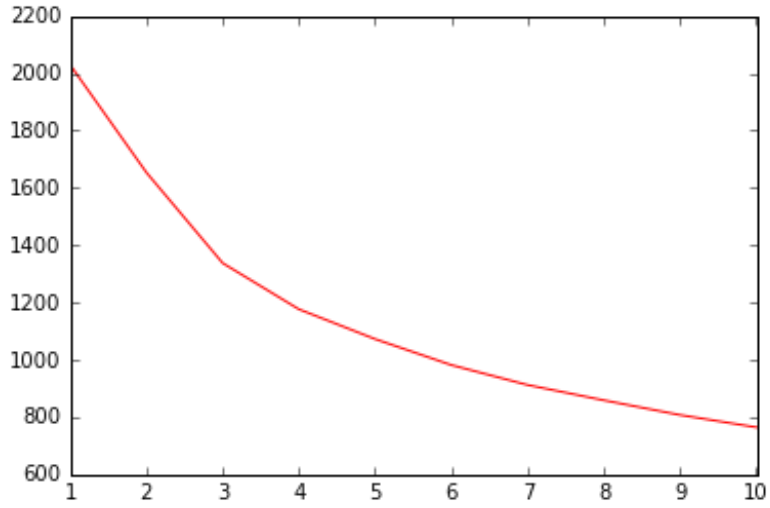
```
# Problem 2b
import scipy.linalg

U, S, V = scipy.linalg.svd(X0, full_matrices=False)
W = U.dot(np.diag(S))

print U.shape, S.shape, V.shape
```

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

In [64]:

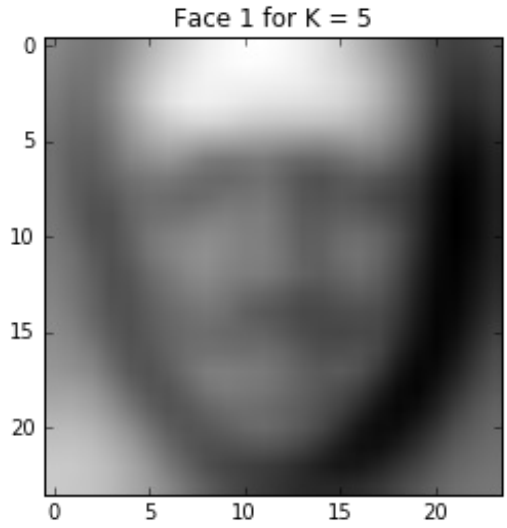
```
# Problem 2c
mse = []

for k in range(1, 11):
    X0hat = W[:, :k].dot(V[:k,:])
    mse.append(np.mean((X0 - X0hat)**2))

# plot the data
_, axis = plt.subplots()
axis.plot(range(1,11), mse, c='red')
```

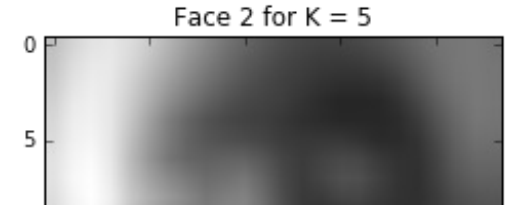
```
axis.set_xticks(range(1,11))
plt.show()
```

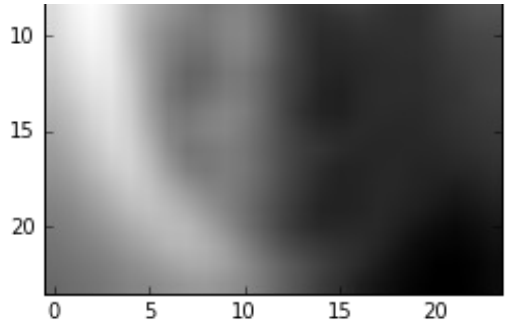


In [72]:

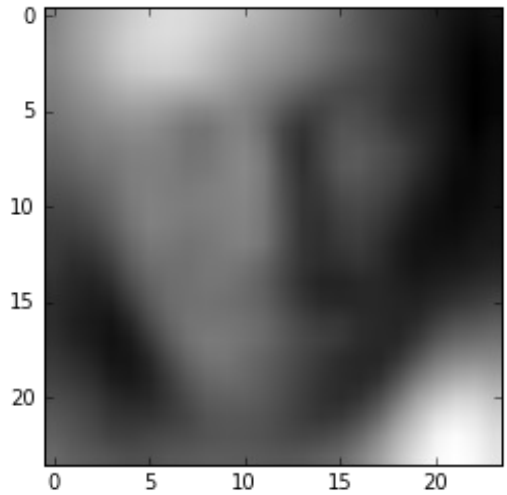
```
# Problem 2d and 2e
K = [5, 10, 50]

for k in K:
    X0hat = W[:, :k].dot(V[:k,:])
    f1 = X0hat[0,:]
    f2 = X0hat[1,:]

    img = np.reshape(f1,(24,24))
    plt.imshow(img.T, cmap="gray")
    plt.title("Face 1 for K = " + str(k))
    plt.show()

    img = np.reshape(f2, (24,24))
    plt.imshow(img.T, cmap="gray")
    plt.title("Face 2 for K = " + str(k))
    plt.show()
```



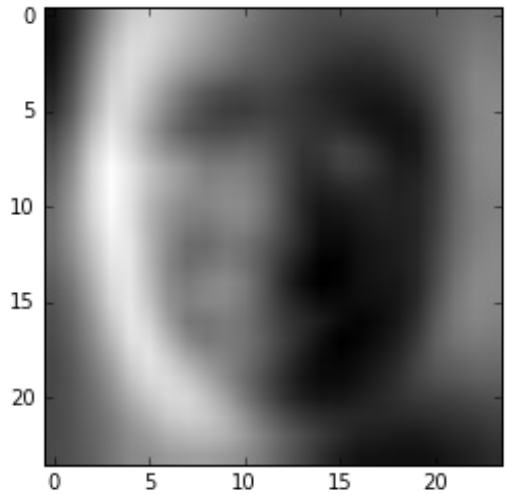




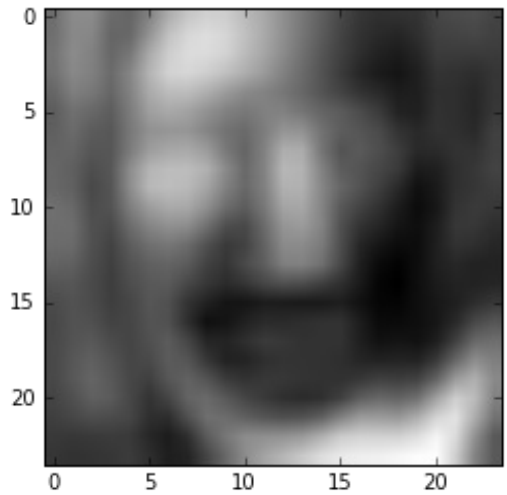
Face 1 for K = 10



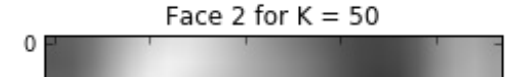
Face 2 for K = 10

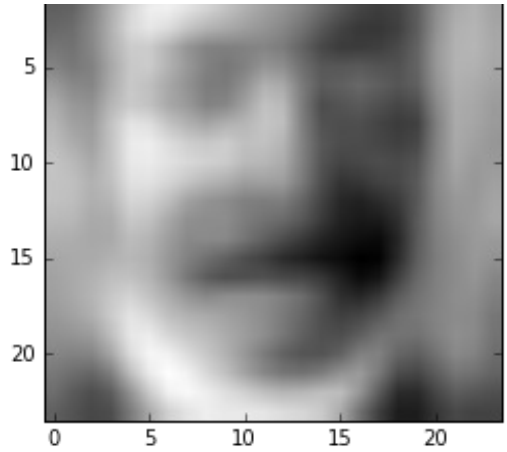


Face 1 for K = 50



Face 2 for K = 50





In [69]:

```
# Problem 2f
idx = [i for i in range(15)]

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

for i in idx:
    loc = (coord[i,0], coord[i,0] + 0.5, coord[i,1], coord[i, 1] + 0.5) # where t
o place the image & size
    img = np.reshape(X[i,:], (24,24))
    plt.imshow(img.T, cmap="gray", extent=loc) # draw each image
    plt.axis((-2,2,-2,2))

plt.show()
```