In [4]:

```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import scipy as sp
import cv2
import os
import sys
#suppress warnings for cleanliness
import warnings
warnings.filterwarnings('ignore')
```

executed in 111ms, finished 19:20:07 2019-04-20

In [5]:

```

dir='att_faces'
Skip = ''
path = os.getcwd()+'/'+dir+'/'
alpha = []
for i in range (1, 41):
    s = "s" + str(i)
    alpha.append(s)

targets = []
pictures = []
for p,letter in enumerate(alpha):
    dirs = os.listdir( path+letter+'/' )
    for item in dirs:
        if item == '.DS_STORE':
            continue
        if item[0:2] == Skip:
            continue
        local_path =path+letter+'/'+item
        img = cv2.imread(local_path,0)
        pictures.append(img)
        targets.append(letter)
print("Sample size = ", len(pictures))
pictures = np.asarray(pictures)
```

executed in 77ms, finished 19:20:07 2019-04-20

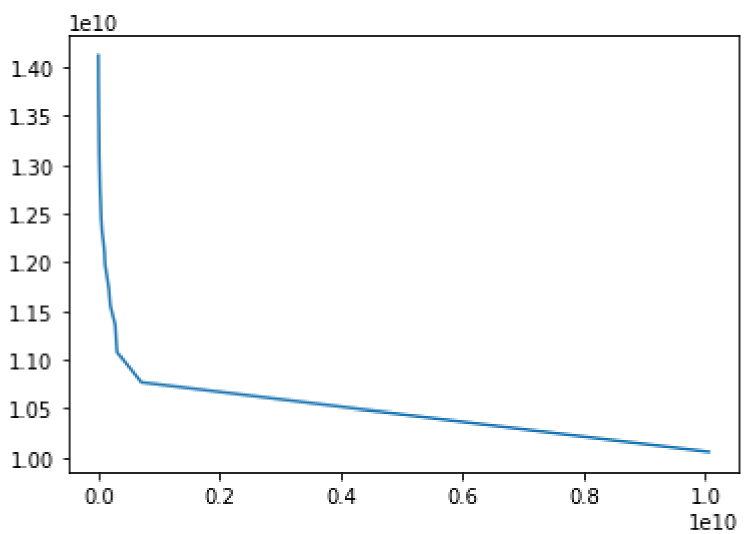
```
Sample size =  400
```

In [21]:
```
from sklearn.model_selection import train_test_split
#stack
print(pictures.shape)
A = np.zeros((400, 112*92))
counter = 0
for img in pictures:
    flatt = img.ravel()
    for i in range(len(flatt)):
        A[counter][i] = flatt[i]
    counter = counter +1
X_train, X_test, y_train, y_test = train_test_split(A, targets, test_size=0
#subtract mean from each row
u = []
A = A.T
for i in range(X_train.shape[1]):
    colmean = np.mean(X_train,axis=1)
    X_train[:,i] = X_train[:,i]  - colmean
    u.append(colmean)
#eignvalues
w, v= np.linalg.eig(np.matmul(X_train, X_train.T))
print(w, v)
summed = []
cum = 0
for i in w:
    cum = cum + i
    summed.append(cum)
plt.plot(w, summed)
#plot the eigvenvalues vs cum sum for highest variance
#plt.plot(w, sum(w[,;5]))

#values to threshold
```

executed in 14.0s, finished 21:10:16 2019-04-20

```
 [-0.06298718 -0.08152625 -0.00721302 ...   0.02616307   0.03564895
  -0.01080187]]
```

Out[21]: [<matplotlib.lines.Line2D at 0x125034c88>]



In [25]:
```
#we will take the top 10% of eigenvalues it looks like
len(w) * .1
```

executed in 4ms, finished 21:12:21 2019-04-20

Out[25]: 30.0

In [62]:
```
m = .9
e = []
for i in range(int(m*len(w))):
    e.append(v[i])
r = []

#build atoms
for img in X_train:
    r.append(np.matmul(np.asarray(img)[0:int(m*len(w))].T, np.asarray(e)))

#PCA from sklearn
from sklearn.decomposition import PCA
pca = PCA(n_components=int(m*len(w)), svd_solver='randomized', whiten=True)
eigenfaces = pca.components_.reshape(int(m*len(w)), pictures[0].shape[0], p
x_train_pca = pca.transform(X_train)
x_test_pca = pca.transform(X_test)
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(x_train_pca, y_train)

predictions = neigh.predict(x_test_pca)
print(classification_report(y_test, predictions))


#my pca
#reconstruct testing
X_test_pca = []
for img in X_test:
    X_test_pca.append(np.matmul(np.asarray(img)[0:int(m*len(w))].T, np.asar

#reconstruct training
X_train_pca = []
for img in r:
    X_train_pca.append(img ** np.asarray(e))

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(r, y_train)

predictions = neigh.predict(X_test_pca)
print(classification_report(y_test, predictions))
```

executed in 988ms, finished 21:51:57 2019-04-20

	precision	recall	f1-score	support
s1	0.00	0.00	0.00	4
s10	1.00	0.33	0.50	3
s11	1.00	0.33	0.50	3
s12	0.00	0.00	0.00	4
s13	1.00	0.50	0.67	2

s14	1.00	1.00	1.00	1
s15	0.00	0.00	0.00	3
s16	1.00	0.50	0.67	2
s18	0.00	0.00	0.00	3
s19	0.15	1.00	0.27	2
s2	0.00	0.00	0.00	2
s20	0.00	0.00	0.00	1
s21	0.00	0.00	0.00	2
s22	0.00	0.00	0.00	1
s23	1.00	0.25	0.40	4
s24	0.00	0.00	0.00	4
s25	0.00	0.00	0.00	2
s26	0.00	0.00	0.00	2
s27	0.00	0.00	0.00	4
s28	0.00	0.00	0.00	3
s29	0.00	0.00	0.00	2
s3	0.00	0.00	0.00	2
s30	1.00	1.00	1.00	1
s31	0.00	0.00	0.00	1
s32	0.00	0.00	0.00	1
s33	1.00	0.33	0.50	3
s34	0.00	0.00	0.00	2
s35	0.00	0.00	0.00	1
s36	0.00	0.00	0.00	1
s37	0.00	0.00	0.00	2
s38	1.00	0.67	0.80	3
s39	0.00	0.00	0.00	5
s4	0.00	0.00	0.00	4
s40	0.00	0.00	0.00	4
s5	0.04	1.00	0.08	3
s6	1.00	1.00	1.00	3
s7	0.00	0.00	0.00	1
s8	1.00	0.43	0.60	7
s9	1.00	0.50	0.67	2

micro avg	0.22	0.22	0.22	100
macro avg	0.31	0.23	0.22	100
weighted avg	0.34	0.22	0.22	100

	precision	recall	f1-score	support
s1	0.00	0.00	0.00	4
s10	1.00	1.00	1.00	3
s11	0.00	0.00	0.00	3
s12	0.00	0.00	0.00	4
s13	0.33	1.00	0.50	2
s14	0.00	0.00	0.00	1
s15	0.00	0.00	0.00	3
s16	0.00	0.00	0.00	2
s18	0.00	0.00	0.00	3
s19	0.00	0.00	0.00	2
s2	0.00	0.00	0.00	2
s20	0.00	0.00	0.00	1
s21	0.00	0.00	0.00	2
s22	0.00	0.00	0.00	1
s23	0.00	0.00	0.00	4
s24	0.00	0.00	0.00	4

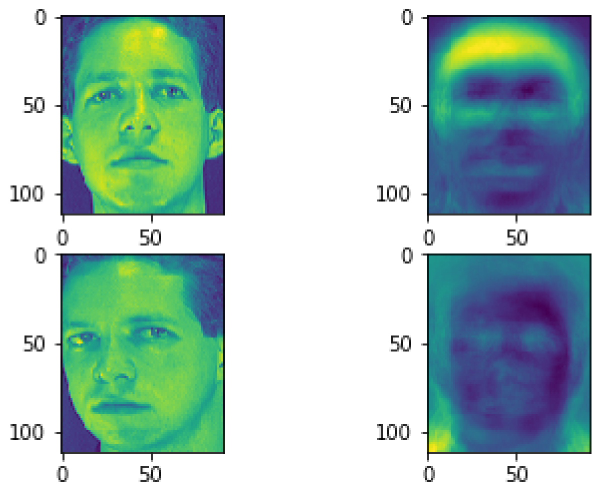
```
         s25       0.00      0.00      0.00         2
         s26       0.00      0.00      0.00         2
         s27       0.00      0.00      0.00         4
         s28       0.00      0.00      0.00         3
         s29       0.00      0.00      0.00         2
          s3       0.00      0.00      0.00         2
         s30       0.00      0.00      0.00         1
         s31       0.00      0.00      0.00         1
         s32       0.00      0.00      0.00         1
         s33       0.00      0.00      0.00         3
         s34       0.00      0.00      0.00         2
         s35       0.00      0.00      0.00         1
         s36       0.00      0.00      0.00         1
         s37       0.00      0.00      0.00         2
         s38       0.00      0.00      0.00         3
         s39       0.00      0.00      0.00         5
          s4       0.00      0.00      0.00         4
         s40       0.05      1.00      0.09         4
          s5       0.00      0.00      0.00         3
          s6       0.00      0.00      0.00         3
          s7       0.00      0.00      0.00         1
          s8       0.00      0.00      0.00         7
          s9       0.00      0.00      0.00         2

   micro avg       0.09      0.09      0.09       100
   macro avg       0.04      0.08      0.04       100
weighted avg       0.04      0.09      0.04       100
```

Analysis

When higher thresholds were used, the resulting classification scores went up. In the below, you can see the reconstructed images of the faces. Above, you can see the accuracy score with thresholding of 90%, this went up from the accuracy of 10% thresholding which was nearly half. The optimal thresholding seems to be 90% in terms of classification accuracy, but, the gains are marginal.

In [61]:
```
#example test image and what we found
plt.subplot(221)
plt.imshow(pictures[0])
plt.subplot(222)
plt.imshow(eigenfaces[0])
plt.subplot(223)
plt.imshow(pictures[1])
plt.subplot(224)
plt.imshow(eigenfaces[1])
```
executed in 279ms, finished 21:51:51 2019-04-20

Out[61]: <matplotlib.image.AxesImage at 0x12544d128>



In [ ]: