ESE 542: Statistics for Data Science **Spring 2019**

HW 6

Name: *Tyler Olivieri*

1. PDF of first 3 problems:

Tyler Olivieri HW 6 #1 Consider a classification problem with N different classes.

let prior probability of class $n \in N$ be π_n

denote $f_n(x) = Pr(X=x \mid Y=n)$ observations in class n are drawn from $N(\mu_n, \Sigma_n)$ where $\Sigma_n = \Sigma$ $\forall n$.

a) Use bayes theorem to find $Pr(Y=n \mid X=x)$

$$Pr(X, Y) = Pr(X, Y)$$

$$Pr(X \mid y=n)\, p(y=n) = Pr(y \mid X=x)\, Pr(X=x)$$

$$Pr(y \mid X=x) = \frac{p(X \mid y=n)\, p(y=n)}{pr(X=x)}$$

$$\Rightarrow p(y=n \mid X=x) = \frac{p(X=x \mid y=n)\, p(y=n)}{p(X=x)}$$

$$= \frac{f_n(x)\, \pi_n}{p(X=x)}$$

$$= \frac{f_n(x)\, \pi_n}{\sum_{i=1}^{k} p(X=x, y=i)}$$

$$= \frac{f_n(x)\, \pi_n}{\sum_{i=1}^{k} p(X=x \mid y=i)\, p(y=i)}$$

$$p(y=n \mid X=x) = \frac{f_n(x)\, \pi_n}{\sum^{k} p(X=x \mid y=i)\, \pi_i}$$

$$\Pr(y=n \mid X=x) = \frac{f_n(x)\,\pi_n}{\sum_{i=1}^{K} f_i(x)\,\pi_i}$$

b) Derive the linear descriminant function, $\delta_n(x)$ and write the classification rule for the predicted class, $\hat{y}$ for an LDA in terms of $\delta_n(x)$

$$f_n(x) = \frac{1}{\sqrt{(2\pi)^p|\Sigma|}} \exp\left(-\tfrac{1}{2}(x-\mu_n)\Sigma^{-1}(x-\mu_n)\right)$$

$$\Pr(y=n \mid X=x) = \frac{\frac{1}{\sqrt{(2\pi)^p|\Sigma|}} \exp\left(-\tfrac{1}{2}(x-\mu_n)\Sigma^{-1}(x-\mu_n)\right)\pi_n}{p(X=x)}$$

$$\log \Pr(y=n \mid X=x) = \log \pi_n + \log\left[\frac{1}{\sqrt{(2\pi)^p|\Sigma|}} \exp\left(-\tfrac{1}{2}(x-\mu_n)^T\Sigma^{-1}(x-\mu_n)\right)\right]$$

$$-\log p(X=x)$$

$$= \log \pi_n + \log\left(\frac{1}{\sqrt{(2\pi)^p|\Sigma|}}\right) + \log\left(\exp\left(-\tfrac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu_n - \tfrac{1}{2}\mu_n^T\Sigma^{-1}\mu_n\right)\right)$$

$$-\log p(X=x)$$

$$\delta_n(x) = \log \Pr(y=n \mid X=x) = \log \pi_n - \tfrac{1}{2}x^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu_n - \tfrac{1}{2}\mu_n^T\Sigma^{-1}\mu_n$$

$$+ \log\left(\frac{1}{\sqrt{(2\pi)^p|\Sigma|}}\right) - \log p(X=x)$$

will drop out

the classification rule should be

$$\hat{y} = \underset{y \in \{1, \dots, n\}}{\operatorname{argmax}} \; f_n(x)$$

taking the max wrt y removes constants in $f_n(x)$.

$$\hat{y} = \underset{y \in \{1, \dots, n\}}{\operatorname{argmax}} \left(\log \pi_n + x^T \Sigma^{-1} \mu_n - \tfrac{1}{2} \mu_n^T \Sigma^{-1} \mu_n \right)$$

where we estimate π_n, μ_n with $\hat{\pi}_n, \hat{\mu}_n$

c) Derive the decision boundary for the LDA in the case of two classes, a and b.

The decision boundary for two classes a, b is the set of points where

$$f_a(x) = f_b(x)$$

$$\log \pi_a + x^T \Sigma^{-1} \mu_a - \tfrac{1}{2} \mu_a^T \Sigma^{-1} \mu_a = \log \pi_b + x^T \Sigma^{-1} \mu_b - \tfrac{1}{2} \mu_b^T \Sigma^{-1} \mu_b$$

This is an equation of a line in x.

(substitute estimates for $\pi_a, \pi_b, \mu_a, \mu_b$)

$\hat{\pi}_a, \hat{\pi}_b, \hat{\mu}_a, \hat{\mu}_b$

d) Consider two classes, $a=1$ and $b=2$. You are given $\hat{\pi}_a = .6$, $\hat{\pi}_b = .4$ $\hat{\mu}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\hat{\mu}_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$ and

$$\hat{\Sigma} = \begin{bmatrix} 1.5 & .1 \\ .1 & 1 \end{bmatrix}$$

Find the decision boundary and classification rule of the corresponding LDA. How would the observation $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ be classified?

The decision boundary are the points x s.t

$$\log(.6) + x^T \begin{bmatrix} 1.5 & .1 \\ .1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \tfrac{1}{2} \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1.5 & .1 \\ .1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \log(.4) + x^T \begin{bmatrix} 1.5 & .1 \\ .1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -2 \\ 1 \end{bmatrix} - \tfrac{1}{2} \begin{bmatrix} -2 & 1 \end{bmatrix} \begin{bmatrix} 1.5 & .1 \\ .1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

$$-.51 + x^T \begin{bmatrix} .71 \\ -.07 \end{bmatrix} - .36 = -.92 + x^T \begin{bmatrix} -2.14 \\ 1.21 \end{bmatrix} - 2.75$$

$$-.87 + x^T \begin{bmatrix} .71 \\ -.07 \end{bmatrix} = -3.67 + x^T \begin{bmatrix} -2.14 \\ 1.21 \end{bmatrix}$$

classify x as class a if

$$-.87 + x^T \begin{bmatrix} .71 \\ -.07 \end{bmatrix} > -3.67 + x^T \begin{bmatrix} -2.14 \\ 1.21 \end{bmatrix}$$

and b otherwise.

$$-.87 + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} .71 \\ -.07 \end{bmatrix} = -.23$$

$$-3.27 + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} -2.14 \\ 1.21 \end{bmatrix} = -4.6$$

since $-.23 > -4.6 \Rightarrow$ classify x as a.

e) This relates to logistic regression under the binary case in the following:

with two classes the decision boundary for gaussian bayes is when $\log p(y=1|x) = \log p(y=0|x)$

so choose class 1 when $\log p(y=1|x) - \log p(y=0|x) > 0$

$$\Rightarrow \log \frac{p(y=1|x)}{p(y=0|x)} > 0$$

the quantity $\log \frac{p(y=1|x)}{p(y=0|x)}$ is the log odds

which is the basis of logistic regression

c) The naive Bayes classifier is similar to LDA, except it assumes each predictor is conditionally independent of every other predictor given class n. Derive the classification rule for $\hat{y}$ under this classifier. How does this relate to logistic regression in the binary case (i.e for two classes?)

$$p(X|y) = \prod_{i=1}^{p} p(X_i|y)$$ under naive bayes conditionally independent assumption.

$$p(y=n|X=x) = \frac{f_n(x)\,\pi_n}{p(X=x)} = \frac{\prod_{i=1}^{p} p(X_i|y=n)\,\pi_n}{p(X=x)}$$

$$\log p(y=n|X=x) = \log\left(\prod_{i=1}^{p} p(X_i|y=n)\right) + \log \pi_n - \log p(X=x)$$

$$= \sum_{i=1}^{p} \log p(X_i|y=n) + \log \pi_n + \log p(X=x)$$

$$= \sum_{i=1}^{p} \log \frac{1}{\sqrt{(2\pi)^p \sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2}(X_i - \mu_n)^2\right) + \log \pi_n + \underbrace{\log p(X=x)}_{\text{constant wrt } y.}$$

$$\hat{y} = \underset{y \in \{1,2,\dots,n\}}{\operatorname{argmax}} \log p(y=n|X=x)$$ is classification rule

Substitute above expression except for $\log p(X=x)$ as it does not effect maximization wrt y. Substitute estimates for necessary parameters.

Tyler Olivieri HW6 #2 k-means clustering

a) Show that setting the objective function the sum of the squared Euclidean distances of points from the center of their cluster

$$obj = \sum_{k=1}^{K} \sum_{x \in C_k} \sum_{i=1}^{p} (c_{ki} - x_i)^2$$

results in an update rule where the optimal centroid is the mean of the points in the cluster.

min obj wrt C_k

$$\frac{d\,obj}{dc_k} = \sum_{x \in C_k} \sum_{i=1}^{p} 2(c_{ki} - x_i) = 0$$

$$2 \sum_{x \in C_k} \sum_{i=1}^{p} (c_{ki} - x_i) = 0$$

$$2 \sum_{x \in C_k} \sum_{i=1}^{p} (c_{ki} - x_i) = 0$$

$$\sum_{x \in C_k} \sum_{i=1}^{p} c_{ki} - \sum_{x \in C_k} \sum_{i=1}^{p} x_i = 0$$

$$\sum_{x \in C_k} \sum_{i=1}^{p} c_{ki} = \sum_{x \in C_k} \sum_{i=1}^{p} x_i$$

let n_k be # points assigned to cluster k. $x \in C_k$.

$$n_k \sum_{i=1}^{p} c_{ki} = \sum_{x \in C_k} \sum_{i=1}^{p} x_i$$

$$\sum_{i=1}^{p} C_{ki} = \frac{\sum_{x \in C_k} \left(\sum_{i=1}^{p} x_i \right)}{n_k} = \frac{\sum_{i=1}^{p} \sum_{x \in C_k} x_i}{n_k}$$

n_k ← total # of points assigned to cluster k.

So set C_{ki} to be the mean of all data points assigned to cluster k.

$$C_{ki} = \frac{\sum_{x \in C_k} x_i}{n_k}$$

for each feature, set cluster center to be mean of data points assigned to the cluster center.

b) Show that setting the objective function to the sum of the manhattan distances of points from the center of their clusters,

$$obj = \sum_{k=1}^{k} \sum_{x \in C_k} \sum_{i=1}^{p} |C_{ki} - x_i|$$

results in an update rule where the optimal centroid is the median of the cluster.

$$\frac{d\,obj}{d\,C_k} = \sum_{x \in C_k} \sum_{i=1}^{p} \frac{C_{ki} - x_i}{|C_{ki} - x_i|} = 0$$

when $C_{ki} - x_i > 0$ then $\frac{C_{ki} - x_i}{|C_{ki} - x_i|} = \frac{C_{ki} - x_i}{C_{ki} - x_i} = 1$

when $C_{ki} - x_i < 0$ then $\frac{C_{ki} - x_i}{|C_{ki} - x_i|} = \frac{C_{ki} - x_i}{x_i - C_{ki}} = -\frac{C_{ki} - x_i}{-1(C_{ki} - x_i)}$

$= \frac{1}{-1} = -1$

$\Rightarrow \frac{C_{ki} - x_i}{|C_{ki} - x_i|} = \text{sign}(C_{ki} - x_i)$ where $\text{sign}(x) = 1$ when $x > 0$

$= -1$ when $x < 0$

$$\sum_{i=1}^{p} \left(\sum_{x \in C_k} \text{sign}(C_{ki} - x_i) \right) = 0$$

This can only equal zero when the number of the positive elements equals the number of negative elements. So for C_{ki}

$C_{ki} = \text{median}(x_i \in C_{ki})$ then

Tyler Olivieri HW6 P3 PCA

Consider the dataset

X	y	
0	1	1
1	1	2
2	1	3
2	3	4
3	2	5
3	3	6
4	5	7

a) Normalize the data and derive the two principal components in sorted order.

$$\mu_x = \frac{0+1+2+2+3+3+4}{7} = 2.14$$

$$\mu_y = \frac{1+1+1+3+2+3+5}{7} = 2.29$$

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_x)^2 = \frac{1}{6}\sum_{i=1}^{7}(x_i-2.14)^2 = 1.55$$

$$\sigma_x = \sqrt{\sigma_x^2} = \sqrt{1.55} = 1.24$$

$$\sigma_y^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i-\mu_y)^2 = \frac{1}{6}\sum_{i=1}^{7}(y_i-2.29)^2 = 1.92$$

$$\sigma_y = \sqrt{\sigma_y^2} = \sqrt{1.92} = 1.38$$

Normalized data $x_i := \frac{x_i - \mu_x}{\sigma_x}$ $\quad y_i := \frac{y_i - \mu_y}{\sigma_y}$

0 mean
std_dev = 1

x_{norm}	y_{norm}
-1.72	-.93
-.92	-.93
-.11	-.93
-.11	.52
.69	-.21
.69	.52
1.40	1.94

X-norm ~ (0,1)
Y-norm " "

let $A = \begin{bmatrix} -1.72 & -.93 \\ -.92 & -.93 \\ -.11 & -.93 \\ -.11 & .52 \\ .69 & .21 \\ .69 & .52 \\ 1.49 & 1.96 \end{bmatrix}$ be the data matrix.

Find principal components

$T = AW$ where the columns of W are the eigenvectors of A^TA.

(by SVD) $= U\Sigma W^TW$ since W is chosen to be orthonormal $W^TW = I$

$= U\Sigma$

SVD of A: (Done in python)

$$A = \underbrace{\begin{bmatrix} -.52 & -.54 \\ -.36 & .06 \\ -.2 & .56 \\ .08 & -.42 \\ .18 & .33 \\ .24 & .116 \\ .68 & -.32 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 3.6 & 0 \\ 0 & 1.04 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .71 & .71 \\ .71 & -.71 \end{bmatrix}}_{W^T}$$

Now we have $T = U\Sigma = \begin{bmatrix} -.52 & -.54 \\ -.36 & .06 \\ -.2 & .56 \\ -.08 & -.42 \\ .18 & .33 \\ .24 & .11 \\ .68 & -.32 \end{bmatrix} \begin{bmatrix} 3.6 & 0 \\ 0 & 1.04 \end{bmatrix}$

$$T = U\Sigma = \begin{bmatrix} -1.88 & -.56 \\ -1.31 & .06 \\ -.74 & .58 \\ .29 & -.44 \\ .64 & .34 \\ .86 & .12 \\ 2.44 & -.33 \end{bmatrix}$$

The new transformed dataset using the first principal component is

$$\tilde{X} = \begin{bmatrix} -1.88 \\ -1.31 \\ -.74 \\ .29 \\ .64 \\ .86 \\ 2.44 \end{bmatrix}$$

Which is the first column of T.

b) repeat the previous analysis but do not normalize the data.

Is pca scale-invariant

SVD of A_unnormalized: (when you multiply $U\Sigma$, they are irrelevant. removing ~~n-r~~ columns of U, and 0's rows and columns of Σ) because python libs give SVD this way except numpy, which I did not use

$$A_{\text{unnormalized}} = \underbrace{\begin{bmatrix} .08 & .44 \\ .15 & -.04 \\ .22 & -.51 \\ .37 & .37 \\ .37 & -.55 \\ .45 & -.11 \\ .67 & .30 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 9.52 & 0 \\ 0 & 1.54 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} .68 & .73 \\ -.73 & .68 \end{bmatrix}}_{V^T}$$

$$T_{\text{unnormalized}} = U\Sigma = \begin{bmatrix} .73 & .68 \\ 1.41 & -.06 \\ 2.09 & -.79 \\ 3.60 & .56 \\ 3.50 & -.85 \\ 4.24 & -.17 \\ 6.39 & .46 \end{bmatrix}$$

$$\tilde{X}_{\text{unnormalized}} = \begin{bmatrix} .73 \\ 1.41 \\ 2.09 \\ 3.60 \\ 3.50 \\ 4.24 \\ 6.39 \end{bmatrix}$$

$\Rightarrow$ PCA is <u>not</u> scale-invariant $\tilde{X} \neq \tilde{X}_{\text{unnormalized}}$

$$T = U\Sigma = \begin{bmatrix} -1.88 & -.56 \\ -1.31 & .06 \\ -.74 & .58 \\ .29 & -.44 \\ .64 & .34 \\ .86 & .12 \\ 2.44 & -.33 \end{bmatrix}$$

The new transformed dataset using the first principal component is

$$\tilde{X} = \begin{bmatrix} -1.88 \\ -1.31 \\ -.74 \\ .29 \\ .64 \\ .86 \\ 2.44 \end{bmatrix}$$

Which is the first column of T.

2. Sparse Principal Component Analysis

   (a) The top five components are:

      i. First component:
      [0.48788336 -0.26512898 0.47333547 0.13915442 0.19742679 -0.04588071 0.00406675 0.37030119 -0.43272085 0.25453535 -0.07317678 0.11248878]
      Expained Variance (%): 26.009731

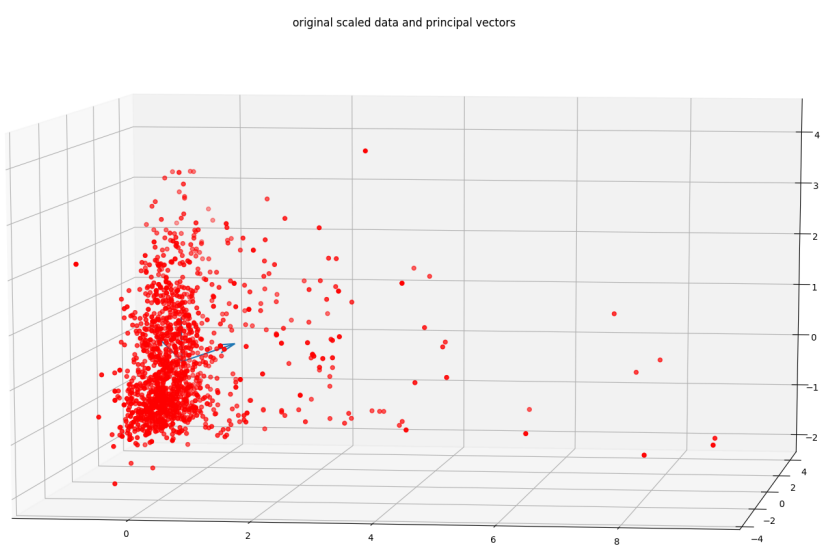
      ii. Second component:
      [-0.00417321 0.33896786 -0.1373581 0.16773634 0.18978819 0.25948314 0.36397137 0.33078079 -0.06544015 -0.10933362 -0.50270865 -0.47316621]
      Expained Variance (%): 18.68235

      iii. Third component:
      [-0.16482854 -0.22708884 0.10022856 0.24362014 -0.02660785 0.61611132 0.54073214 -0.16872267 0.06977056 0.21291324 0.22497138 0.22336929]
      Expained Variance (%): 14.024331

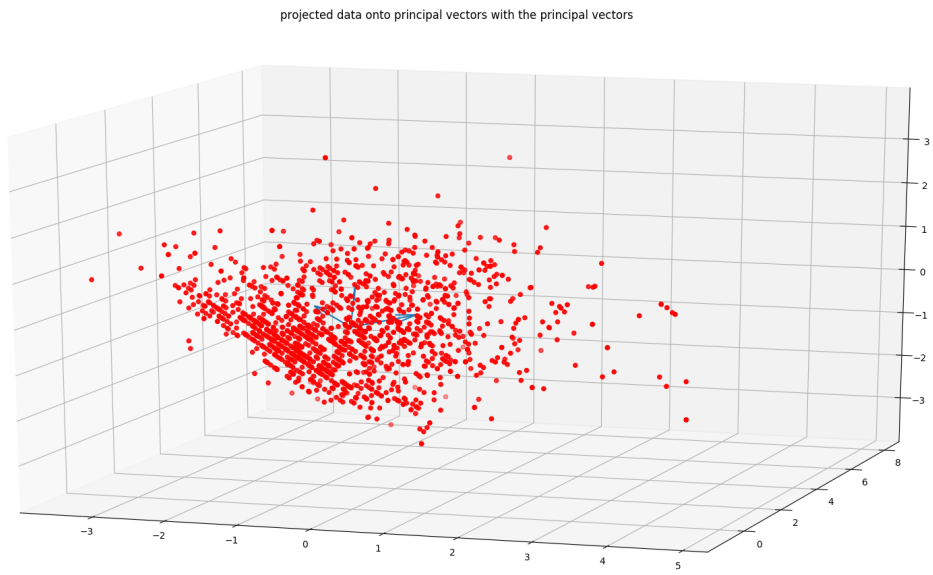
      iv. Fourth component:
      [-0.23109808 0.04185824 -0.0567358 -0.38303758 0.65477782 -0.03371148 -0.02845973 -0.20069341 -0.00546618 0.56050237 -0.09170143 -0.03666923]
      Expained Variance (%): 10.125174

      v. Fifth component:
      [-0.07877938 0.29937933 -0.12014871 0.70936319 0.26623723 -0.15941286 -0.21845284 0.20879298 0.25764682 0.21483493 0.25972635 0.13758414]]
      Expained Variance (%): 08.11053

   (b) The scatter plot of data with principal components overlayed:



The scatter plot of projected data onto the principal components:



(c) The R^2 value is 0.321.

The model coefficients are:

x1: 0.0627

x2: -0.2787

x3: 0.3206

Accordingt to "An Introduction to statistical learning theory in R", the loadings define the direction in the the feature space along which the data vary the most. These are normalized eigenvectors. We can obtrain these from sklearn pca components attribute.

[0.48931422 -0.23858436 0.46363166 0.14610715 0.21224658 -0.03615752 0.02357485 0.39535301 -0.43851962 0.24292133 -0.11323206]

[-0.11050274 0.27493048 -0.15179136 0.27208024 0.14805156 0.51356681 0.56948696 0.23357549 0.00671079 -0.03755392 -0.38618096]

[-0.12330157 -0.44996253 0.23824707 0.10128338 -0.09261383 0.42879287 0.3224145 -0.33887135 0.05769735 0.27978615 0.47167322]

(d) After using sparse pca, the R^2 value is 0.323.

The model coefficients are:

x1: -4.6641

x2: -2.9691

x3: -21.9510

The loadings are:

[-33.87131683 15.72735184 -31.68687504 -8.83849947 -13.74979027 2.44079956 0. -26.9239972 30.00435285 -15.78203424 6.83363042]

[-6.80249127 0. 0. 14.92471163 1.00922725 34.92211952 34.79148067 0. 0.35007081 3.41756928 -1.92414012]

[0. 26.78666359 -15.35603701 0. 7.4091256 -3.59452513 0. 18.65177905 0. -11.59880185 -30.60466288]

The results are similar, as expected. Both PCA and sparse PCA projected datasets produced a model that could explain about the same amount of variance in the data (we can observe this

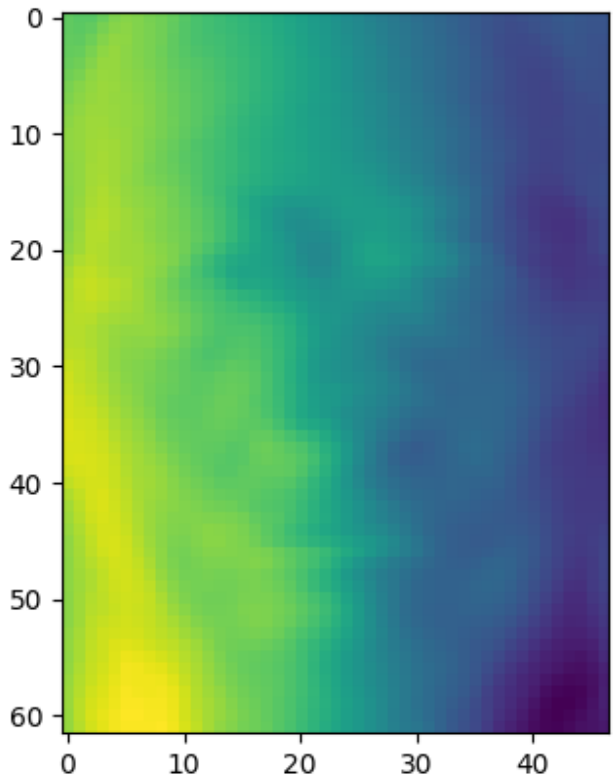
from the R^2 value). This more or less states that PCA and sparse PCA preserved about the same amount of variance in the original dataset through both transformations. The different coefficients from the regression suggest that even though about the same amount of variance is preserved, the projected data is different (with respect to the coordinates). Actually observing the projected data from sparse PCA shows many components are 0 (which is the goal), to project into a sparse representation. This can make analyzing high dimensional datasets easier.

```
import csv
from sklearn.decomposition import *
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import statsmodels.api as sm

file_str = "/home/snazyman/stat_ds/data_science/hw6/winequality-red.csv"

# read in data
with open(file_str) as csvfile:
    data = []
    data_reader = csv.reader(csvfile)

    for row in data_reader:
        data.append(row)

    categories = data[0]
    data = data[1:]

    X = np.array(data)
    X = X.astype(np.float)

    # perform PCA on entire dataset
    # normalize data
    X_scaled = preprocessing.scale(X)

    # perform PCA
    pca_all = PCA(n_components = len(categories))
    pca_5 = PCA(n_components = 5)

    X_pca_all = pca_all.fit_transform(X_scaled)
    X_pca_5 = pca_5.fit_transform(X_scaled)

    # report top 5 components
    print(pca_5.components_)

    # report explained variance
    print(pca_5.explained_variance_ratio_)

    # Consider the variables residual sugar, pH, and alcohol from the dataset
    # Compute the three corresponding principal components. Create a 3-D scatterplot
    # of this data with the principal components overlayed
    A = X_scaled[:,[3,8,10]]
```

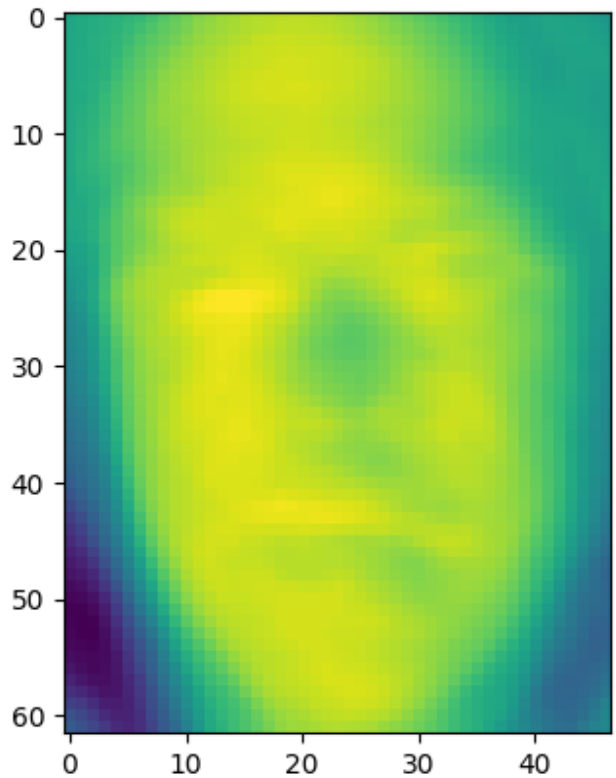
```
pca_3 = PCA(n_components = 3)
A_pca_3 = pca_3.fit_transform(A)

# first plot, original scaled data and principal vectors
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(A[:,0],A[:,1],A[:,2],color='red')
for length, vector in zip(pca_3.explained_variance_, pca_3.components_):
    v = vector * np.sqrt(length)
    ax.quiver(pca_3.mean_[0],pca_3.mean_[1],pca_3.mean_[2], pca_3.mean_[0] + v[0],

plt.title("original scaled data and principal vectors")
plt.show()

# second plot, projected data onto principal vectors with the principal vectors
fig2 = plt.figure()
ax2 = Axes3D(fig2)
ax2.scatter(A_pca_3[:,0],A_pca_3[:,1],A_pca_3[:,2],color='red')
for length, vector in zip(pca_3.explained_variance_, pca_3.components_):
    v = vector * np.sqrt(length)
    ax2.quiver(pca_3.mean_[0], pca_3.mean_[1], pca_3.mean_[2], pca_3.mean_[0] + v[

plt.title("projected data onto principal vectors with the principal vectors")
plt.show()

# using entire dataset except for the variable quality, obtrain theh top three pri

# seperate data into quality and the rest
B = X_scaled[:,:-1]
B_label = X_scaled[:,11]

# perform pca
pca_3 = PCA(n_components = 3)
B_pca_3 = pca_3.fit_transform(B)

# Fit a multiple regression model over the entire reduced dataset to predict quali
#B_pca_3 = sm.add_constant(B_pca_3)
model = sm.OLS(B_label, B_pca_3).fit()

print(model.summary())

# the loadings of a component define a vector that is the direction in the feature
print(pca_3.components_)

# repeat the above analysis on quality using sparse PCA
Spca_3 = SparsePCA(n_components = 3)
B_Spca_3 = Spca_3.fit_transform(B)

# Fit a multiple regression model over the entire reduced dataset to predict quali
#B_Spca_3 = sm.add_constant(B_Spca_3)
```

```
model = sm.OLS(B_label, B_Spca_3).fit()

print(model.summary())
print(Spca_3.components_)
```

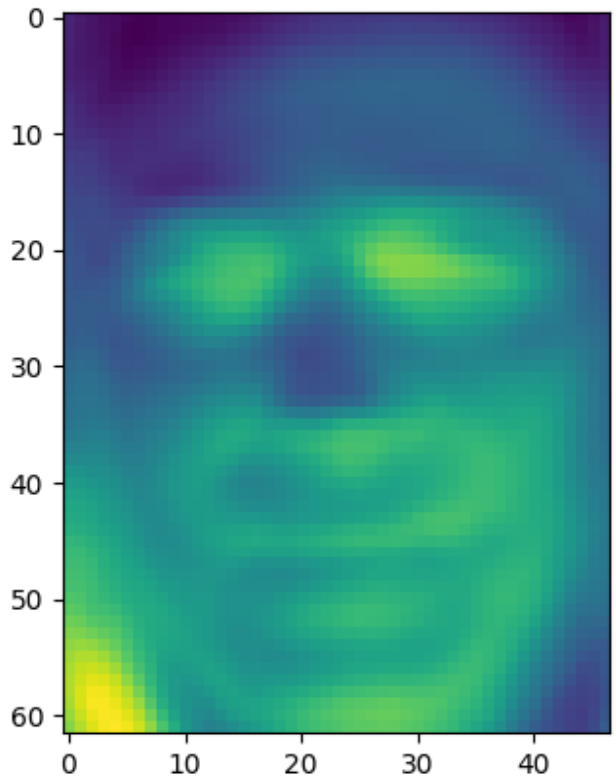
3. The eigenfaces for the first 150 components can be seen below. (The png's didnt fit on the page nicely, so this is a lot of pages, I apologize)



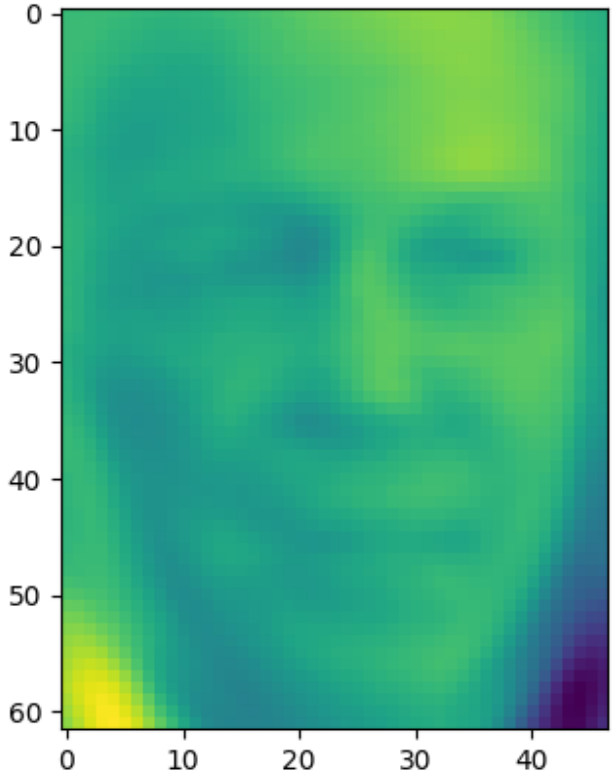
First eigenface:



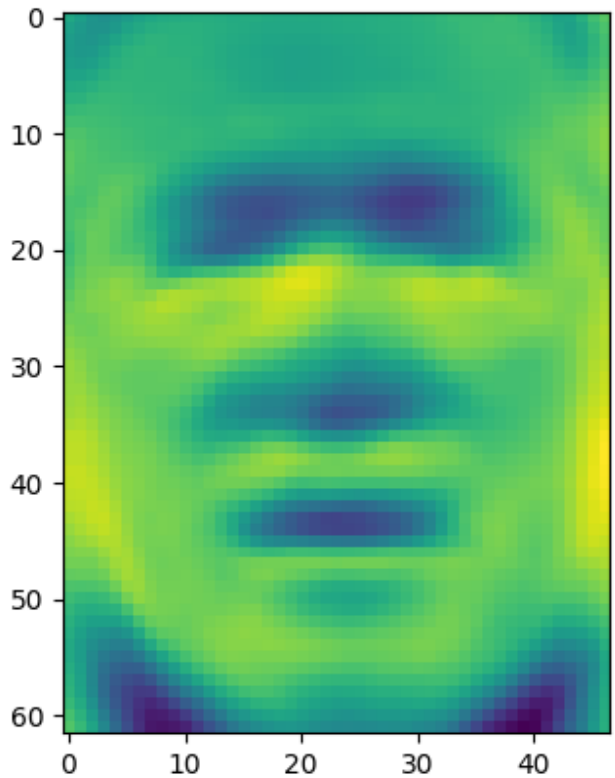
Second:



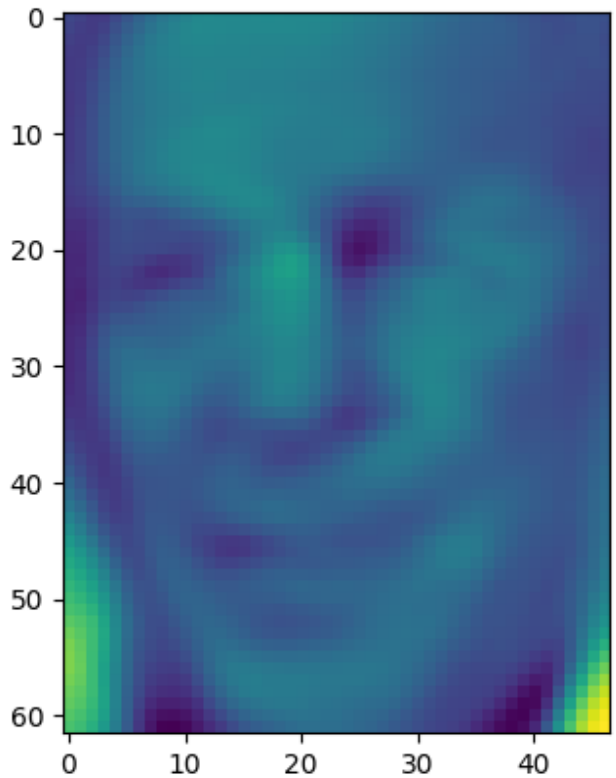
Third:



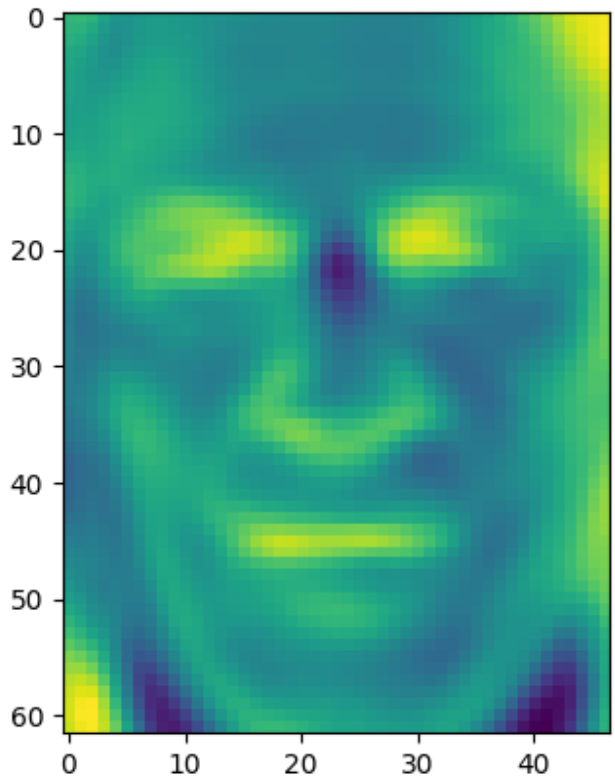
Fourth:



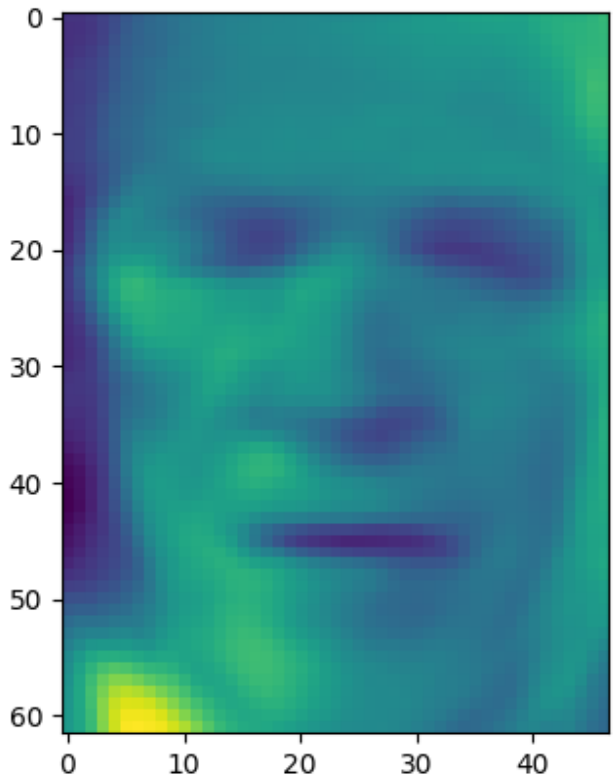
Fifth:



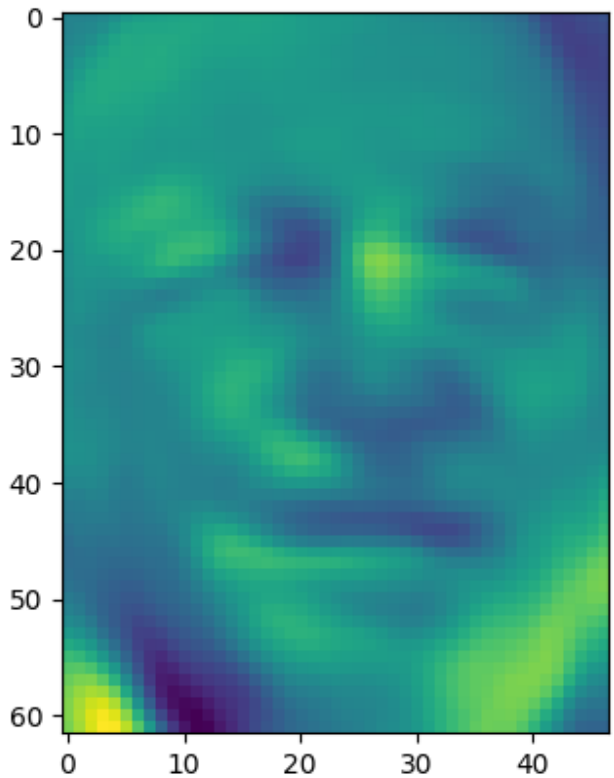
Sixth:



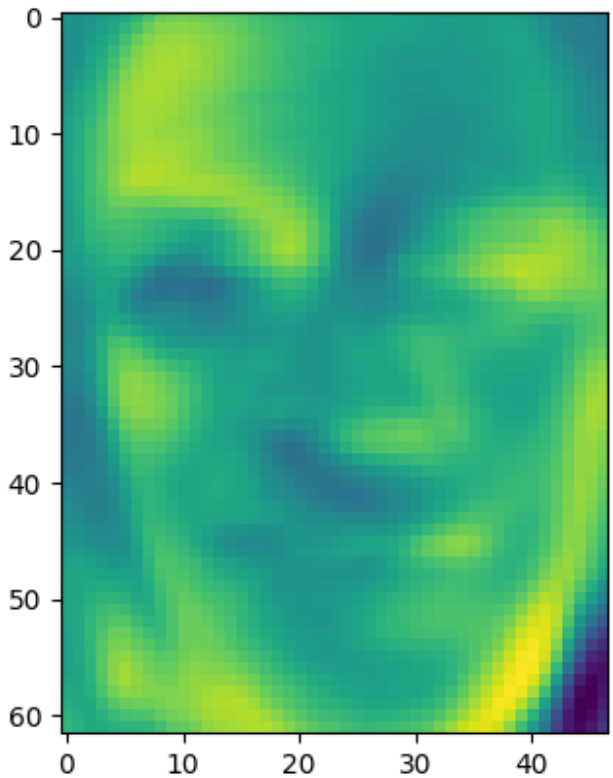
Seventh:



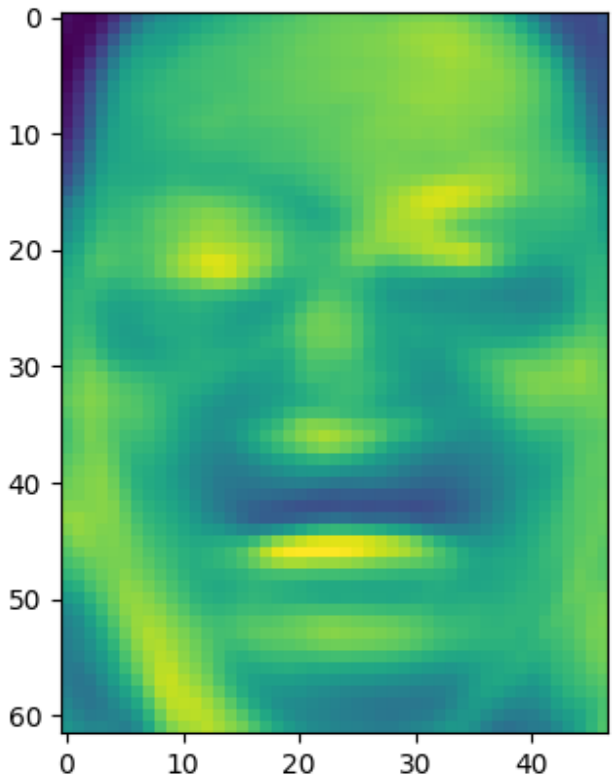
8:



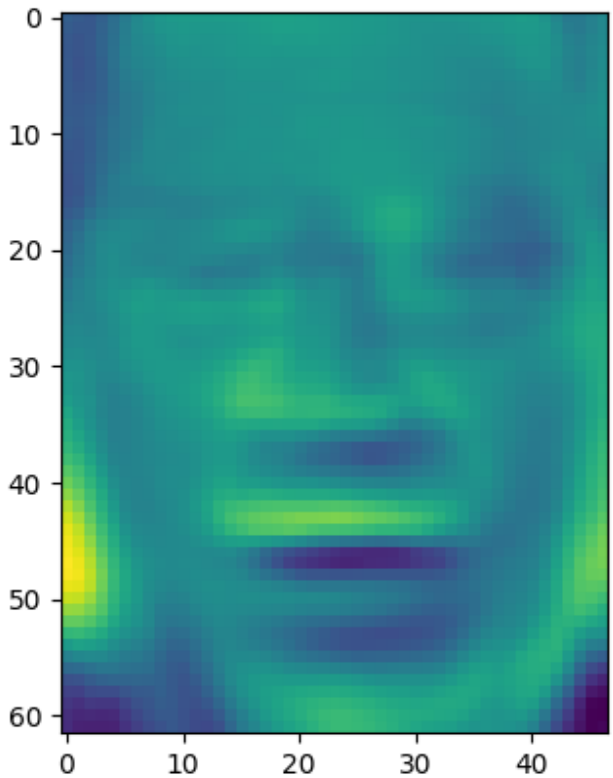
9:



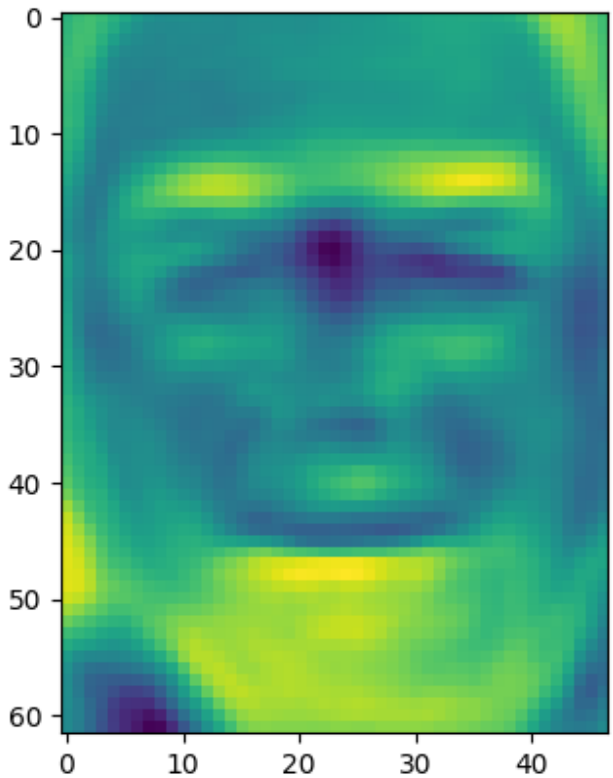
10:



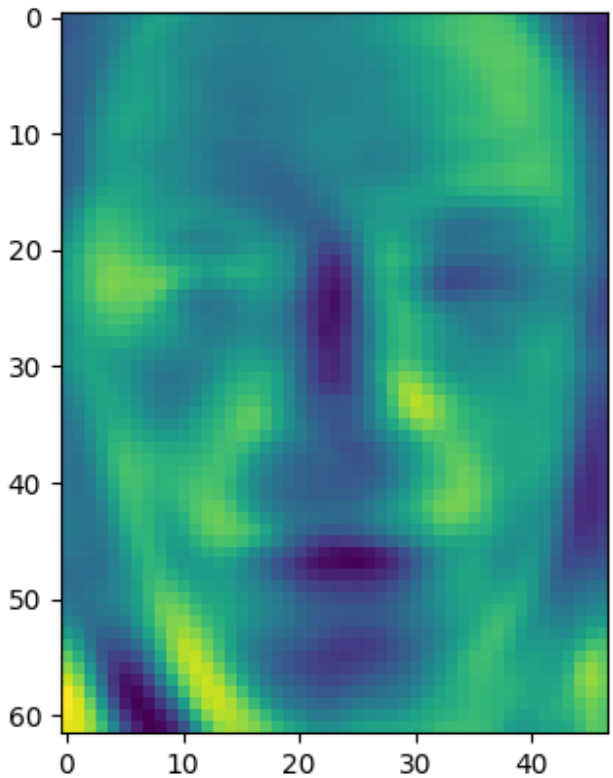
11:



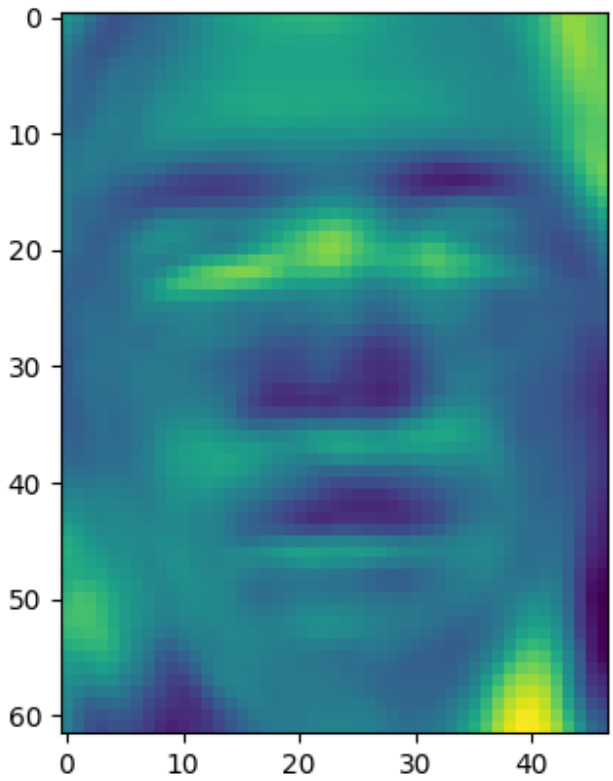
12:



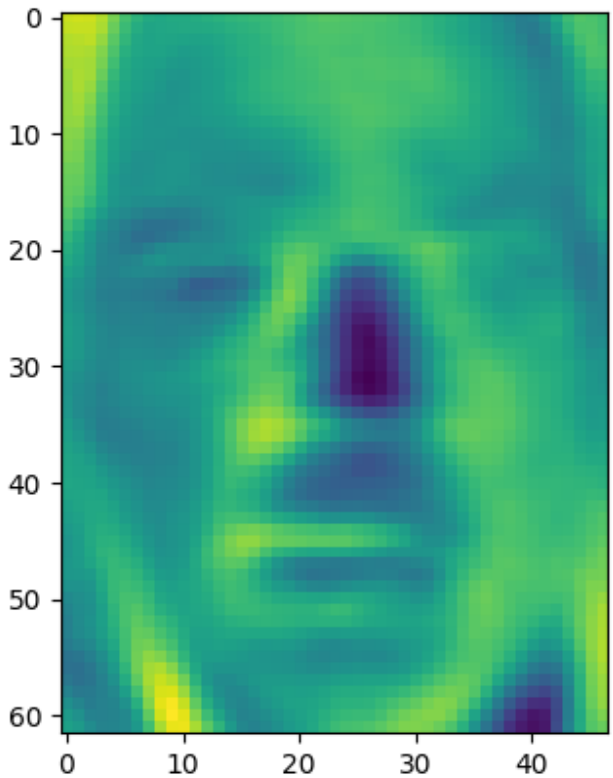
13:



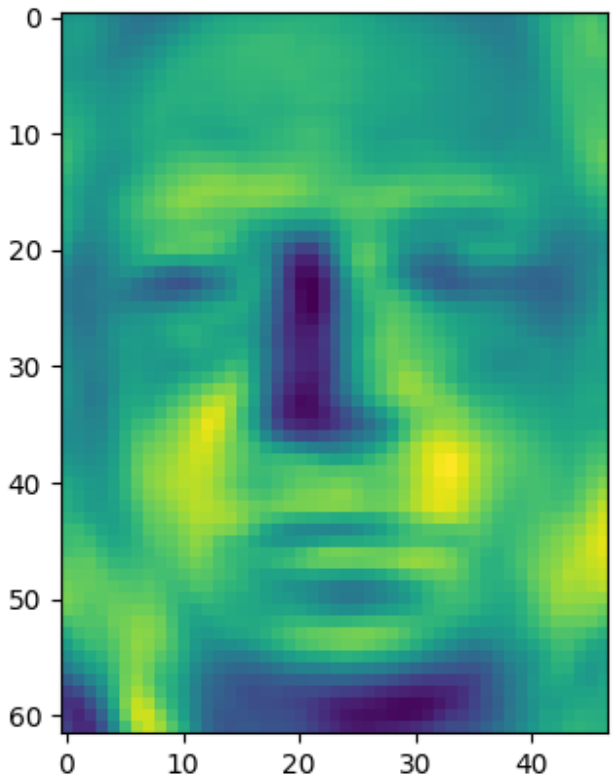
14:



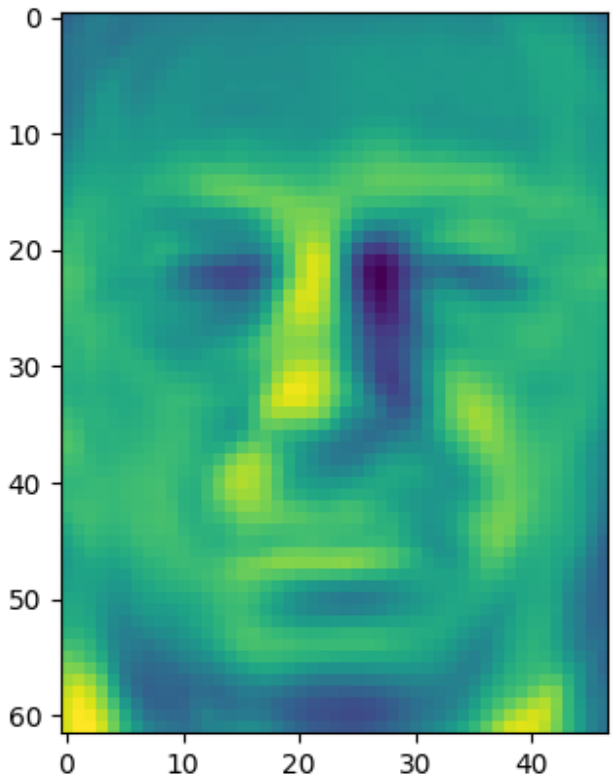
15:



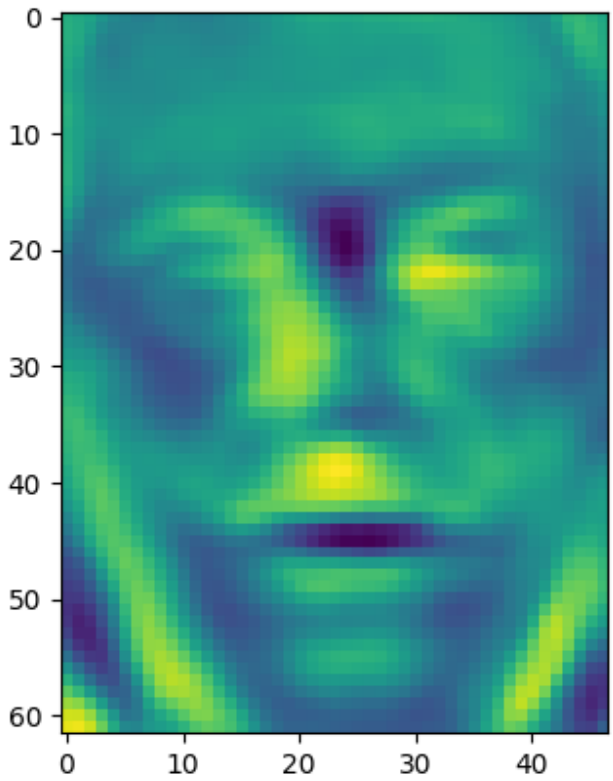
16:



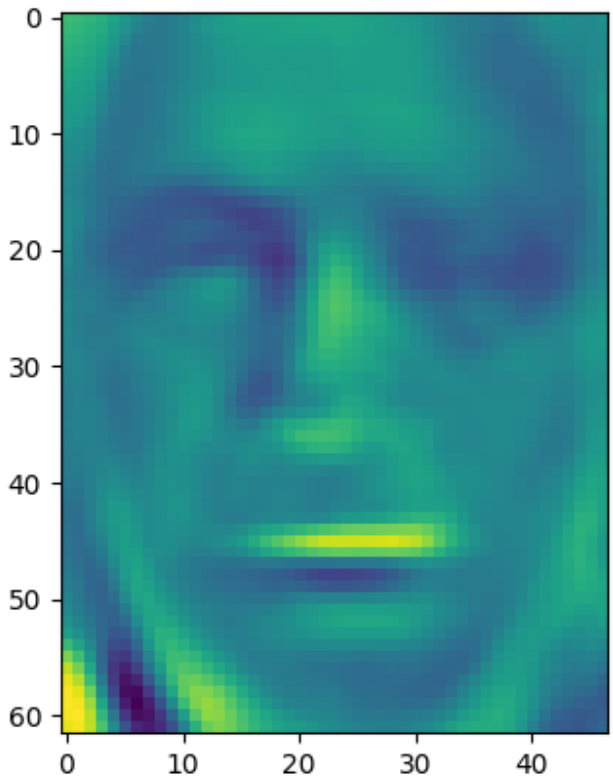
17:



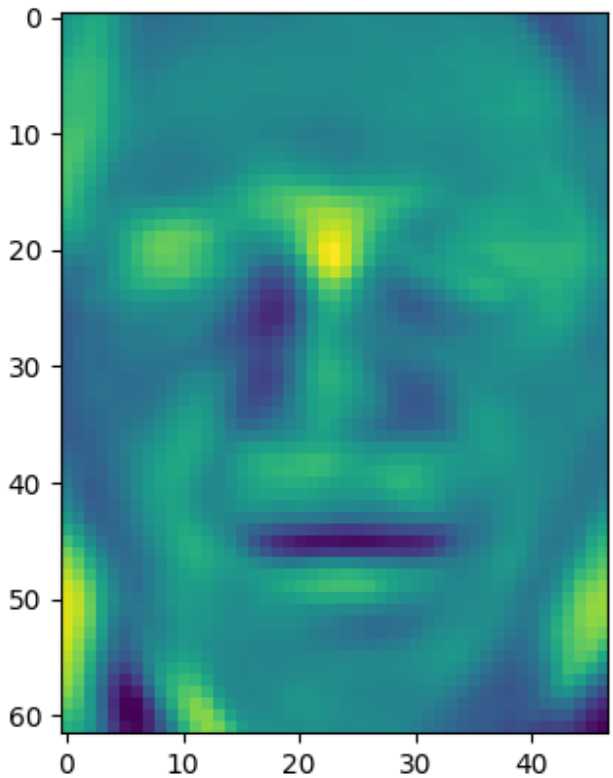
18:



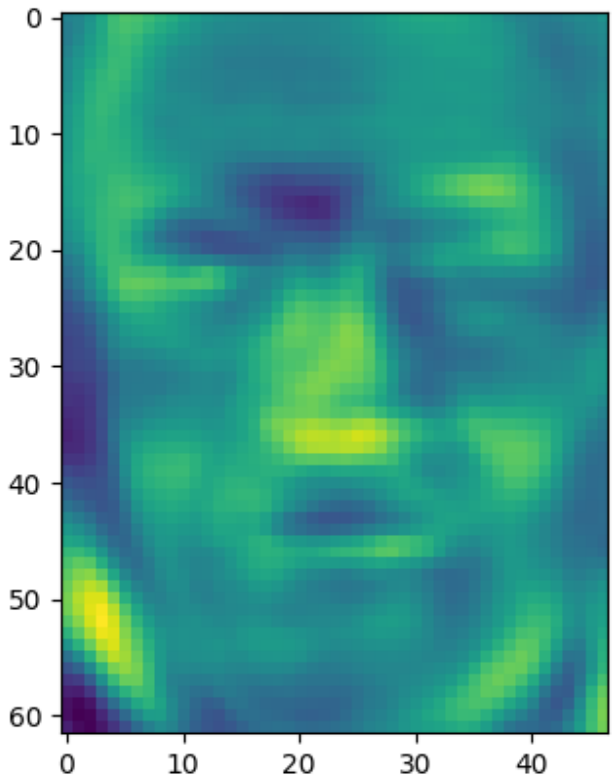
19:



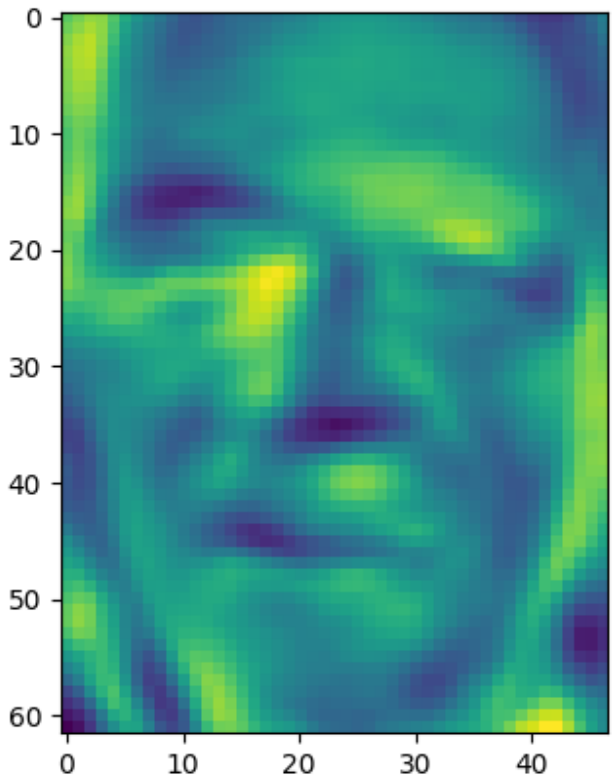
20:



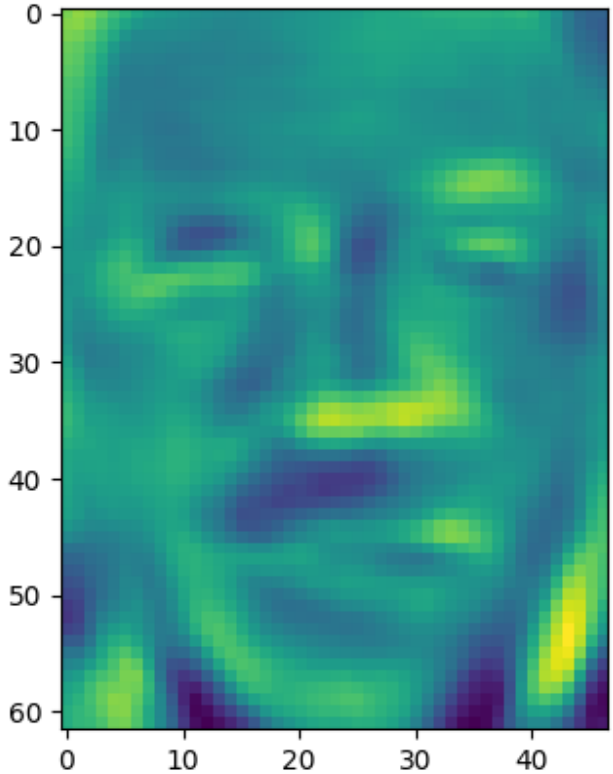
21:



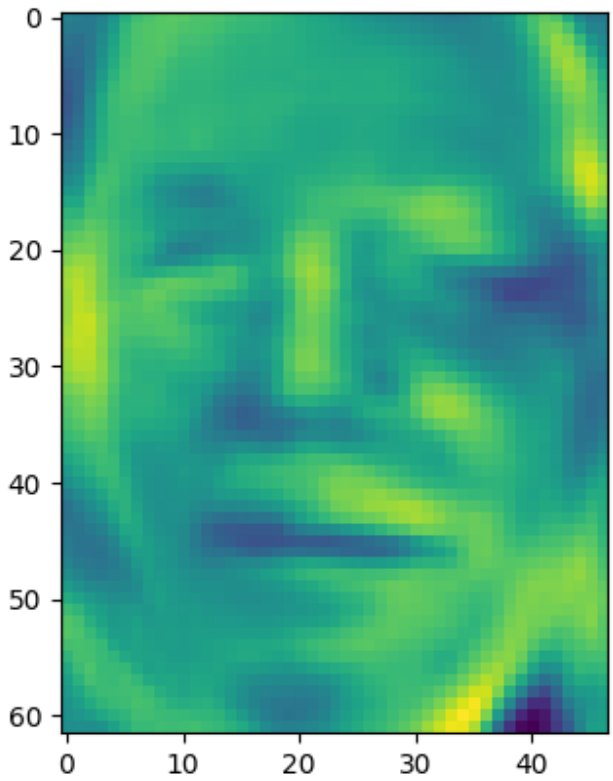
22:



23:

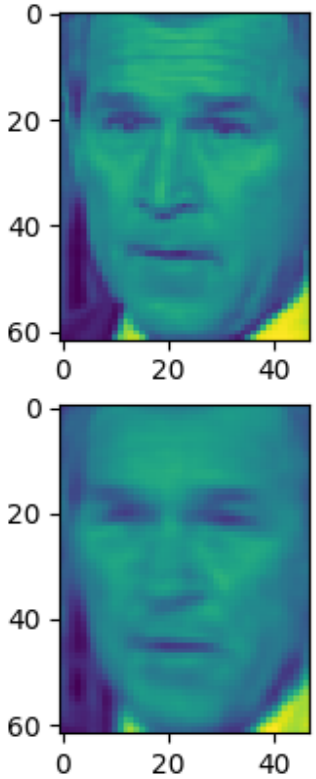


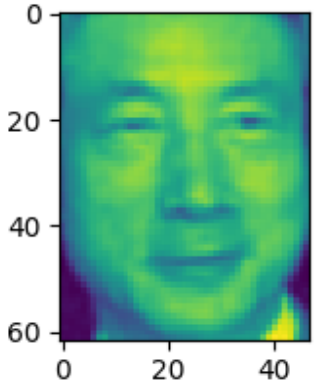
24:

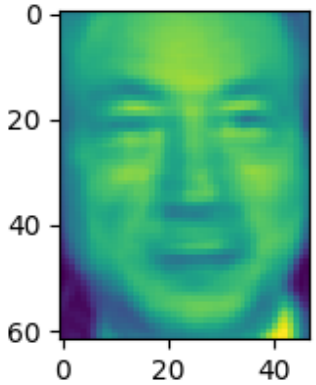


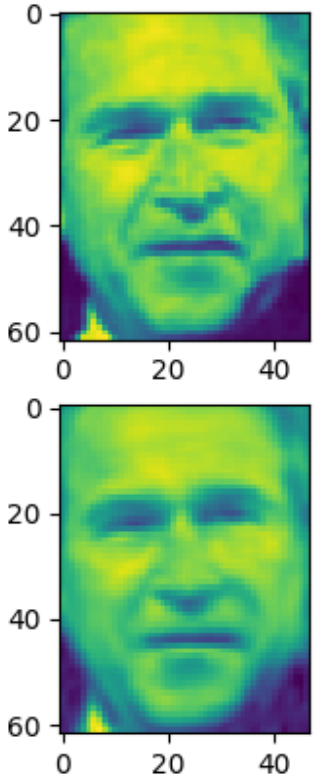
25:

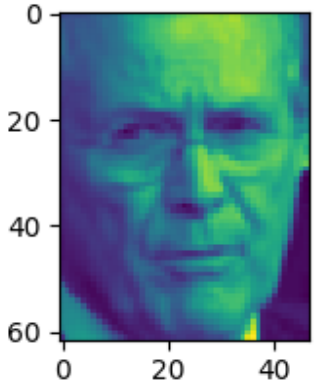
A reconstruction of 5 selcted faces:

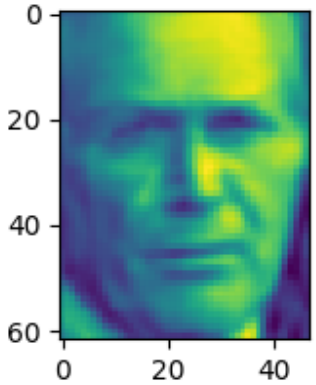

0
20
40
60
0
20
40
0
20
40
60
0
20
40

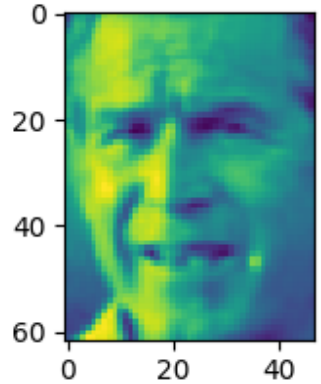

0
20
40
60
0
20
40

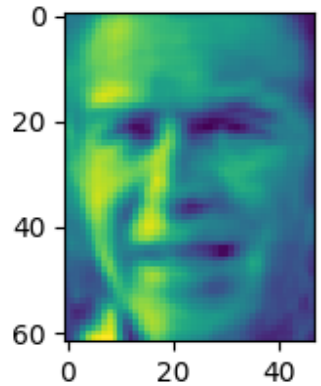

0
20
40
60
0
20
40


0
20
40
60
0
20
40
0
20
40
60
0
20
40


0
20
40
60
0
20
40


0
20
40
60
0
20
40





Python script used to generate the above images:

```
from sklearn.datasets import fetch_lfw_people
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# download face dataset
faces = fetch_lfw_people(min_faces_per_person=60)
n_samples, h, w = faces.images.shape
data = faces.data

# perform RandomPCA on the first 150 components
n_components = 150
pca = PCA(n_components=n_components,svd_solver='randomized', whiten=True)
pca_faces = pca.fit(data)

# show eigenface of first 25 prinicpal components
eigenfaces = pca_faces.components_.reshape((n_components, h, w))

# used this to plot and save eigenfaces
#for i in range(1,26):
#     plt.imshow(eigenfaces[i,:,:])
```

```
#     f = plt.figure(i+1)
    #f.savefig(f"eigenface_{i}.png")
#plt.show()

# selct some images from original dataset
Im1 = faces.images[1,:,:]
Im2 = faces.images[100,:,:]
Im3 = faces.images[500,:,:]
Im4 = faces.images[1000,:,:]
Im5 = faces.images[1300,:,:]

# project them onto the first 150 principal components
pca_Im = pca_faces.transform(faces.data)
# reconstruct from lower dimensional space to original spce
pca_reconstructed = pca_faces.inverse_transform(pca_Im).reshape((n_samples,h,w))

# grab those images
pca_Im1 = pca_reconstructed[1,:,:]
pca_Im2 = pca_reconstructed[100,:,:]
pca_Im3 = pca_reconstructed[500,:,:]
pca_Im4 = pca_reconstructed[1000,:,:]
pca_Im5 = pca_reconstructed[1300,:,:]

# plot first image and reconstruction
plt.figure(1)
plt.subplot(211)
plt.imshow(Im1)
plt.subplot(212)
plt.imshow(pca_Im1)
plt.show()

# plot image and reconstruction
plt.figure(2)
plt.subplot(211)
plt.imshow(Im2)
plt.subplot(212)
plt.imshow(pca_Im2)
plt.show()

# plot image and reconstruction
plt.figure(3)
plt.subplot(211)
plt.imshow(Im3)
plt.subplot(212)
plt.imshow(pca_Im3)
plt.show()

# plot image and reconstruction
plt.figure(4)
plt.subplot(211)
plt.imshow(Im4)
```

```
plt.subplot(212)
plt.imshow(pca_Im4)
plt.show()

# plot image and reconstruction
plt.figure(5)
plt.subplot(211)
plt.imshow(Im5)
plt.subplot(212)
plt.imshow(pca_Im5)
plt.show()
```