CS 613 - Machine Learning

John Obuch

Assignment 3 - Dimensionality Reduction & Clustering

Introduction

In this assignment you'll work on visualizing data, reducing its dimensionality and clustering it.

You may not use any functions from machine learning library in your code, however you may use statistical functions. For example, if available you **MAY NOT** use functions like

- pca
- k-nearest neighbors functions

Unless explicitly told to do so. But you **MAY** use basic statistical functions like:

- std
- mean
- cov
- eig

Grading

Part 1 (Theory)	23pts
Part 2 (PCA)	25pts
Part 3 (Eigenfaces)	20pts
Part 3 (Clustering)	25pts
Report	7pts
TOTAL	100pts

Table 1: Grading Rubric

DataSets

Labeled Faces in the Wild Datasaet This dataset consists of celebrities download from the Internet from the early 2000s. We use the grayscale version from sklearn.datasets.

we will download the images in a specific way as shown below. You will have 3,023 images, each 87x65 pixels large, belonging to 62 different people.

```
from sklearn.datasets import fetch_lfw_people
import matplotlib.pyplot as plt
import matplotlib.cm as cm

people = fetch_lfw_people(min_faces_per_person=20, resize=0.7)
image_shape = people.images[0].shape

fig, axes = plt.subplots(2, 5, figsize=(15, 8),
                         subplot_kw={'xticks': (), 'yticks': ()})
for target, image, ax in zip(people.target, people.images, axes.ravel()):
    ax.imshow(image, cmap=cm.gray)
    ax.set_title(people.target_names[target])
```

1 Theory Questions

1. Consider the following data:

$$X = \begin{bmatrix} -2 & 1 \\ -5 & -4 \\ -3 & 1 \\ 0 & 3 \\ -8 & 11 \\ -2 & 5 \\ 1 & 0 \\ 5 & -1 \\ -1 & -3 \\ 6 & 1 \end{bmatrix}$$

(a) Find the principle components of the data (you must show the math, including how you compute the eivenvectors and eigenvalues). Make sure you standardize the data first and that your principle components are normalized to be unit length. As for the amount of detail needed in your work imagine that you were working on paper with a basic calculator. Show me whatever you would be writing on that paper. (5pts).

We begin by computing the mean and standard deviation of the system.

$\mu_1 = -0.9$, $\sigma_1 = 4.228$, $\mu_2 = 1.4$, $\sigma_2 = 4.247$.

Next we will use these statistics to standardize the data using the following transformation:

$$\tilde{X} = \frac{(X - \mu)}{\sigma}$$

$$\tilde{X} = \begin{bmatrix} -0.2602 & -0.0936 \\ -0.9696 & -1.2634 \\ -0.4966 & -0.0935 \\ 0.2128 & 0.3743 \\ -1.6791 & 2.2461 \\ -0.2601 & 0.8423 \\ 0.4493 & -0.3275 \\ 1.3953 & -0.5615 \\ -0.0236 & -1.0294 \\ 1.6318 & -0.0935 \end{bmatrix}$$

Next we compute the covariance matrix $\mathbf{\Sigma}$.

$$\mathbf{\Sigma} = \text{cov}(X_1, X_2) = \frac{\tilde{X}^T \tilde{X}}{N - 1} = \begin{bmatrix} 1 & -0.408 \\ -0.408 & 1 \end{bmatrix}$$

Where $N = 10$ here is the total number of records in the system.

Proceeding forward in the process of computing the principle components of the system, we will continue by computing the eigen-values and eigen-vectors of the covariance matrix Σ.

First we will construct the matrix $(\Sigma - \lambda I)$.

$$(\Sigma - \lambda I) = \begin{bmatrix} 1-\lambda & -0.408 \\ -0.408 & 1-\lambda \end{bmatrix}$$

Next, to compute the eigen-values of the system, we will need to take the determinant of $(\Sigma - \lambda I)$ and set it equal to zero.

Recall that given a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ then $\det(A) = ad - bc$.

$$\det(\Sigma - \lambda I) = (1-\lambda)(1-\lambda) - (-0.408)(-0.408) = 0$$

$$\implies 1 - 2\lambda + \lambda^2 - 0.166464 = 0$$

$$\implies \lambda^2 - 2\lambda + 0.833536 = 0$$

From here, we can leverage the quadratic formula:

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Leveraging the equation defined above, we have:

$$\lambda = \frac{-(-2) \pm \sqrt{(-2)^2 - 4(1)(0.833536)}}{2(1)} = \frac{2 \pm 0.816}{2}$$

Hence, we yield that:

$$\lambda_1 = 1.408$$
$$\lambda_2 = 0.592$$

Next we will compute the eigen-vectors associated to each eigen-value by pluging in our resulting eigen-values back into $(\Sigma - \lambda I)$ and solving the system of equations equal to zero.

For reference, recall that $(\Sigma - \lambda I) = \begin{bmatrix} 1-\lambda & -0.408 \\ -0.408 & 1-\lambda \end{bmatrix}$

Solving for e_1 we have the following:

$$(\Sigma - \lambda_1 I) = \begin{bmatrix} 1-1.408 & -0.408 \\ -0.408 & 1-1.408 \end{bmatrix}$$

$$\vec{e_1} = \begin{bmatrix} -0.408 & -0.408 \\ -0.408 & -0.408 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\implies \begin{cases} -0.408X_1 - 0.408X_2 = 0 \\ -0.408X_1 - 0.408X_2 = 0 \end{cases}$$

Solving equation 1 for X_1, we have that:

$$\begin{aligned} -0.408X_1 &= 0.408X_2 \\ \implies X_1 &= -X_2 \end{aligned}$$

Substituting this result into equation 2 and solving for X_2 we have:

$$\begin{aligned} -0.408(-X_2) - 0.408X_2 &= 0 \\ \implies 0.408X_2 &= 0.408X_2 \\ \implies X_2 &= X_2 \end{aligned}$$

Thus, we see that X_2 is a *free variable*. Given this, let us choose $X_2 = 1$. Given the relationship between X_1 and X_2 outlined above, we have that $X_1 = -1$.

Thus, we have that:

$$\vec{e_1} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Now, we will need to normalize our vector by computing the following:

$$\vec{p_{c1}} = \frac{\vec{e_1}}{\|\vec{e_1}\|}$$

where the magnitude of e is defined as follows:

$$\|\vec{e}\| = \sqrt{e_1^2 + e_2^2 + \cdots + e_n^2} = \sqrt{(-1)^2 + 1^2} = \sqrt{2}$$

Thus, the normalized eigen-vector (principle component) associate to λ_1 is $\vec{p_{c1}} = \begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.

Similarly, we will compute the normalized eigen-vector associated to λ_2 as follows:

$$(\Sigma - \lambda_2 I) = \begin{bmatrix} 1 - 0.592 & -0.408 \\ -0.408 & 1 - 0.592 \end{bmatrix}$$

$$\vec{e_2} = \begin{bmatrix} 0.408 & -0.408 \\ -0.408 & 0.408 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\implies \begin{cases} 0.408X_1 - 0.408X_2 = 0 \\ -0.408X_1 + 0.408X_2 = 0 \end{cases}$$

Solving equation 1 for X_1, we have that:

$$\begin{aligned} 0.408X_1 &= 0.408X_2 \\ \implies X_1 &= X_2 \end{aligned}$$

Substituting this result into equation 2 and solving for X_2 we have:

$$-0.408(X_2) + 0.408X_2 = 0$$
$$\implies -0.408X_2 = -0.408X_2$$
$$\implies X_2 = X_2$$

Thus, we see that X_2 is a *free variable* yet again. Given this, let us choose $X_2 = 1$. Given the relationship between X_1 and X_2 outlined above, we have that $X_1 = 1$.

Thus, we have that:

$$\vec{e_2} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Now, we will need to normalize our vector by computing the following:

$$\vec{p_{c2}} = \frac{\vec{e_2}}{\|\vec{e_2}\|}$$

where the magnitude of e is defined as follows:

$$\|\vec{e}\| = \sqrt{e_1^2 + e_2^2 + \cdots + e_n^2} = \sqrt{1^2 + 1^2} = \sqrt{2}$$

Thus, the normalized eigen-vector (principle component) associate to λ_2 is $\vec{p_{c2}} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.

Note: These normalized eigen-vectors form a basis and they are orthogonal!

(b) Project the data onto the principal component corresponding to the largest eigenvalue found in the previous part (3pts).

To project the data onto the principle component corresponding to the largest eigen-value, we will compute $\vec{z} = \tilde{X}\vec{w}$ where $\vec{w} = \vec{p_{c1}} = \begin{bmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.

Performing this operation, we have that the projection of $\tilde{X}$ onto w results in the following z-vector:

$$\vec{z} = \begin{bmatrix} 0.1177 \\ -0.2077 \\ 0.2850 \\ 0.1142 \\ 2.7756 \\ 0.7796 \\ -0.5494 \\ -1.3838 \\ -0.7112 \\ -1.2201 \end{bmatrix}$$

2. Consider the following data:

$$\text{Class 1} = \begin{bmatrix} -2 & 1 \\ -5 & -4 \\ -3 & 1 \\ 0 & 3 \\ -8 & 11 \end{bmatrix}, \text{Class 2} = \begin{bmatrix} -2 & 5 \\ 1 & 0 \\ 5 & -1 \\ -1 & -3 \\ 6 & 1 \end{bmatrix}$$

(a) Compute the information gain for each feature. You could standardize the data overall, although it won't make a difference. (5pts).

First we will construct the matrix C (Assume that Class 1 label is 1 and Class 2 label is 0):

$$C = \begin{bmatrix} -2 & 1 \\ -5 & -4 \\ -3 & 1 \\ 0 & 3 \\ -8 & 11 \\ -2 & 5 \\ 1 & 0 \\ 5 & -1 \\ -1 & -3 \\ 6 & 1 \end{bmatrix}$$

We know that (i.e. the prior):

$$H(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} (-P(v_i) \log_2 P(v_i))$$

From the classification separation provided above, we see that $p = n = 5$. Thus, we see following:

$$H(\frac{p}{p+n}, \frac{n}{p+n}) = (\frac{-5}{10} \log_2(\frac{5}{10})) + (\frac{-5}{10} \log_2(\frac{5}{10})) = 0.5 + 0.5 = 1$$

Next, we will compute:

$$E(H(X)) = \sum_{i=1}^{k} (\frac{p_i + n_i}{p+n}) H(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

First we will separate the data into sub-groups E_{f_k} for $k = 1, \ldots, 10$ for each feature. We observe from the data that for feature 1, there is only one non-zero resulting sub-group, namely $E_{f_1} = \begin{bmatrix} -2 & 1 \\ -2 & 5 \end{bmatrix}$ with labels 1 and 0 respectively. Thus, we have that:

$$E(H(X_1)) = (\frac{2}{10})(\frac{-1}{2} \log_2(\frac{1}{2}) - \frac{1}{2} \log_2(\frac{1}{2})) = 0.2$$

Similarly, computing $E(H(X_2))$, we observe from the data that for feature 2, there is also only one non-zero resulting sub-group, namely $E_{f_1} = \begin{bmatrix} -2 & 1 \\ -3 & 1 \\ 6 & 1 \end{bmatrix}$ where the first 2 row entries have labels 1 and the last row entry has label 0. Thus, we have the following:

$$E(H(X_2)) = (\frac{3}{10})(\frac{-2}{3}\log_2(\frac{2}{3}) - \frac{1}{3}\log_2(\frac{1}{3})) = 0.275$$

Next we compute the information gain for each feature. Recall that the information gain is defined as follows:

$$IG(X) = H(A) - E(H(A))$$

Leveraging the formula above, we compute the information gain for each feature:

$$IG(X_1) = 1 - 0.2 = 0.8$$
$$IG(X_2) = 1 - 0.275 = 0.725$$

(b) Which feature is more discriminating based on results in part a (1pt)?

Response:
Based on our calculation above, we see that $IG(X_1) > IG(X_2)$. Thus, feature 1 provides us with the most information.

(c) Using LDA, find the direction of projection (you must show the math, however for this one you don't have to show the computation for finding the eigenvalues and eigenvectors). Normalize this vector to be unit length (5pts).

Using LDA, we want to maximize the following:

$$J(w) = W^T(\mu_1 - \mu_2)^T(\mu_1 - \mu_2)w - \lambda(w^T(\sigma_1^T\sigma_1 + \sigma_2^T\sigma_2)w)$$

where the between scatter matrix $S_b = (\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$ and the within scatter matrix $S_w = (\sigma_1^T\sigma_1 + 2^T\sigma_2)$.

Taking the derivative with respect to w, we have:

$$\frac{dJ}{dw} = 2S_bw - 2\lambda S_ww = 0$$
$$\implies 2S_bw = 2\lambda S_ww$$
$$\implies S_bw = \lambda S_ww$$
$$\implies S_w^{-1}S_bw = \lambda w$$

Let $S_w^{-1}S_b = M$

$$\implies Mw = \lambda w$$

Now we can solve an eigen-decomposition problem.

First, let us standardize the system using the following transformation:

$$\tilde{C} = (C - \mu)/\sigma$$

$$\tilde{C} = \begin{bmatrix} -0.26015724 & -0.09359019 \\ -0.96967699 & -1.2634676 \\ -0.49666382 & -0.09359019 \\ 0.21285592 & 0.37436077 \\ -1.67919674 & 2.24616461 \\ -0.26015724 & 0.84231173 \\ 0.44936251 & -0.32756567 \\ 1.39538884 & -0.56154115 \\ -0.02365066 & -1.02949211 \\ 1.63189542 & -0.09359019 \end{bmatrix}$$

Next we will separate the standardized data into their respective classes:

$$\tilde{C}_1 = \begin{bmatrix} -0.26015724 & -0.09359019 \\ -0.96967699 & -1.2634676 \\ -0.49666382 & -0.09359019 \\ 0.21285592 & 0.37436077 \\ -1.67919674 & 2.24616461 \end{bmatrix} \quad \tilde{C}_2 = \begin{bmatrix} -0.26015724 & 0.84231173 \\ 0.44936251 & -0.32756567 \\ 1.39538884 & -0.56154115 \\ -0.02365066 & -1.02949211 \\ 1.63189542 & -0.09359019 \end{bmatrix}$$

Next we will compute the mean and variance of each class:

$$\mu_1 = \begin{bmatrix} -0.63856777 \\ 0.23397548 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} 0.63856777 \\ -0.23397548 \end{bmatrix}$$

Next, we will zero center the data of each class:

$$\dot{\tilde{C}}_1 = \tilde{C}_1 - \mu_1$$
$$\dot{\tilde{C}}_2 = \tilde{C}_2 - \mu_2$$

Compute the variance (**Note:** the sample sizes of each class is $N_1 = N_2 = 5$):

$$\sigma_1^2 = (N_1 - 1)\frac{\dot{\tilde{C}}_1^T \dot{\tilde{C}}_1}{N_1 - 1} = \begin{bmatrix} 2.08079553 & -1.64903489 \\ -1.64903489 & 6.52554745 \end{bmatrix}$$

$$\sigma_2^2 = (N_2 - 1)\frac{\dot{\tilde{C}}_2^T \dot{\tilde{C}}_2}{N_2 - 1} = \begin{bmatrix} 2.84151647 & -0.53123272 \\ -0.53123272 & 1.9270073 \end{bmatrix}$$

Next we will compute the within scatter matrix S_w:

$$S_w = \sigma_1^2 + \sigma_2^2 = \begin{bmatrix} 4.922312 & -2.18026761 \\ -2.18026761 & 8.45255474 \end{bmatrix}$$

Based on the eigen decomposition defined above, we will need to compute S_w^{-1}. Recall that given a square 2 x 2 matrix in the form $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, then $A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$ where $\det(A) = ad - bc$.

Thus, $S_w^{-1} = \frac{1}{36.853} = \begin{bmatrix} 8.45255474 & 2.18026761 \\ 2.18026761 & 4.922312 \end{bmatrix} = \begin{bmatrix} 0.22936149 & 0.05916193 \\ 0.05916193 & 0.13356776 \end{bmatrix}$

Now we will compute S_b (Recall that $S_b = (\mu_1 - \mu_2)^T(\mu_1 - \mu_2)$):

Thus, we yield the following:

$$S_b = \begin{bmatrix} 1.6310752 & -0.59763681 \\ -0.59763681 & 0.2189781 \end{bmatrix}$$

Now we compute the matrix M where $M = S_w^{-1} S_b$:

$$M = \begin{bmatrix} 0.3387485 & -0.1241197 \\ 0.01667254 & -0.00610893 \end{bmatrix}$$

We now have everything we need to compute compute the eigen-decomposition $Mw = \lambda w$. We solve for each eigen-value by solving the following:

$$\det(M - \lambda I) = 0$$

After some computation, we obtain the following results (**Note:** There will only be one non-zero eigen-value):

$$\lambda_1 = 0.3326$$
$$\lambda_2 = 0$$

Then we compute the associated eigen-vectors e_i by plugging in each eigen-value back into $(M - \lambda_i I) = \tilde{M}$ and solve the system of equations such that $\tilde{M} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ for $i = 1, 2$.

Solving these system of equations and normalizing each eigen-vector $\vec{e_i}$ we obtain our principle components $\vec{p_i} = \frac{e_i}{\|e_i\|}$. Note that the magnitude of a vector u notated $\|u\| = \sqrt{u_1^2 + \cdots + u_n^2}$.

After a little bit of work we obtain the following results:

$$p_{c_1} = \begin{bmatrix} 0.99879099 \\ 0.04915855 \end{bmatrix}$$

$$p_{c_2} = \begin{bmatrix} 0.9389552 \\ 0.04915855 \end{bmatrix}$$

From here, recall that since there is only one non-zero eigen-value, namely $\lambda_1 = 0.3326$, its corresponding eigen-vector becomes our direction of projection!

Thus, the direction of projection $w = p_{c_1} = \begin{bmatrix} 0.99879099 \\ 0.04915855 \end{bmatrix}$

(d) Project the data onto the principal component found in the previous part (3pts).

We project the data onto the principle component p_{c_1} as follows:

$$z = \tilde{C}\vec{p_{c_1}} = \begin{bmatrix} -0.26444347 \\ -1.03061487 \\ -0.50066411 \\ 0.23100161 \\ -1.56674837 \\ -0.21843588 \\ 0.43271657 \\ 1.36609725 \\ -0.0742304 \\ 1.62532168 \end{bmatrix}$$

(e) Does the projection you performed in the previous part seem to provide good class separation? Why or why not (1pt)?

Response:
Examining the results, the projection performed in the previous part does not seem to provide a perfect separation of the classes. However, LDA finds the best principle component vector such that the projection of the data onto the vector will provide the best separation possible of the data into their respective classes. We observe the the distributions of separations seem to overlap slightly as illustrated in the following figure:

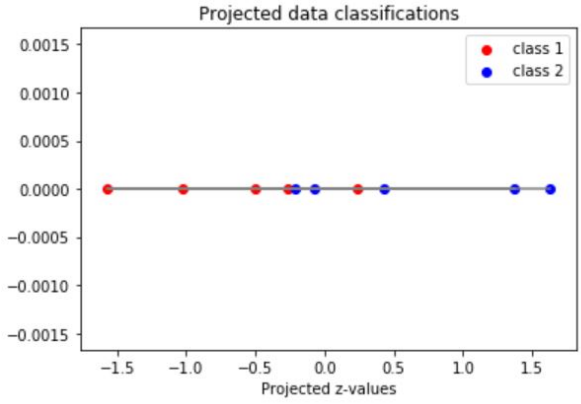


Figure 1: Projected z-values by class

2 Dimensionality Reduction via PCA

Import the data as shown above. This the labeled faces in the wild dataset.
Verify that you have the correct number of people and classes

```
print("people.images.shape: {}".format(people.images.shape))
print("Number of classes: {}".format(len(people.target_names)))
```

```
people.images.shape: (3023, 87, 65)
Number of classes: 62
```

This dataset is skewed toward George W. Bush and Colin Powell as you can verify here

```
# count how often each target appears
counts = np.bincount(people.target)
# print counts next to target names
for i, (count, name) in enumerate(zip(counts, people.target_names)):
    print("{0:25} {1:3}".format(name, count), end='   ')
    if (i + 1) % 3 == 0:
        print()
```

To make the data less skewed, we will only take up to 50 images of each person (otherwise, the feature extraction would be overwhelmed by the likelihood of George W. Bush):

```
mask = np.zeros(people.target.shape, dtype=np.bool)
for target in np.unique(people.target):
    mask[np.where(people.target == target)[0][:50]] = 1

X_people = people.data[mask]
y_people = people.target[mask]

# scale the grayscale values to be between 0 and 1
# instead of 0 and 255 for better numeric stability
X_people = X_people / 255.
```

We are now going to compute how well a KNN classifier does using just the pixels alone.

```
from sklearn.neighbors import KNeighborsClassifier
# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier using one neighbor
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Test set score of 1-nn: {:.2f}".format(knn.score(X_test, y_test)))
```

You should have an accuracy around 23% - 27%.

Once you have your setup complete, write a script to do the following:

1. Write your own version of KNN (k=1) where you use the SSD (sum of squared differences) to compute similarity
2. Verify that your KNN has a similar accuracy as sklearn's version
3. Standardize your data (zero mean, divide by standard deviation)
4. Reduces the data to 100D using PCA
5. Compute the KNN again where K=1 with the 100D data. Report the accuracy
6. Compute the KNN again where K=1 with the 100D Whitened data. Report the accuracy
7. Reduces the data to 2D using PCA
8. Graphs the data for visualization

Recall that although you may not use any package ML functions like *pca*, you **may** use statistical functions like *eig*.

Results:

1. Sklearn KNN (K=1) Accuracy: 0.23255813953488372
2. My KNN (K=1) Accuracy: 0.23255813953488372
3. My KNN (K=1) PCA 100D Accuracy: 0.25387596899224807
4. My KNN (K=1) 100D Whitened data Accuracy: 0.3313953488372093

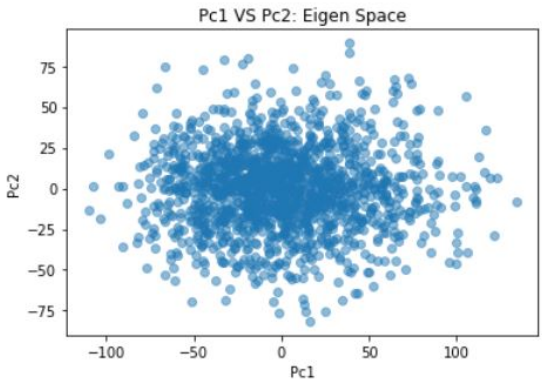


Figure 2: PC1 VS PC2 in the Eigen Space

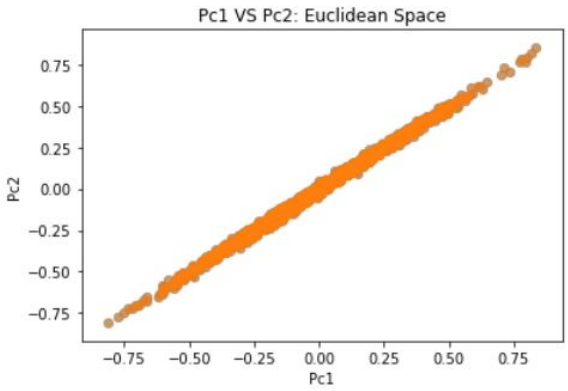


Figure 3: PC1 VS PC2 in the Euclidean Space

3 Eigenfaces

Import the data as shown above. This the labeled faces in the wild dataset.
Use the X_train data from above. Let's analyze the first and second principal components.

Write a script that:

1. Imports the data as mentioned above.
2. Standardizes the data.
3. Performs PCA on the data (again, although you may not use any package ML functions like *pca*, you **may** use statistical functions like *eig*). No need to whiten here.
4. Find the max and min image on PC1's axis. Find the max and min of PC2. Plot and report the faces, what variation do these components capture?
5. Visualizes the most important principle component as a 87x65 image (see Figure **??**).

6. Reconstructs the *X_train[0,:]* image using the primary principle component. To best see the full re-construction, "unstandardize" the reconstruction by multiplying it by the original standard deviation and adding back in the original mean.

7. Determines the number of principle components necessary to encode at least 95% of the information, k.

8. Reconstructs the *X_train[0,:]* image using the k most significant eigen-vectors (found in the previous step, see Figure **??**). For the fun of it maybe even look to see if you can perfectly reconstruct the face if you use all the eigen-vectors! Again, to best see the full re-construction, "unstandardize" the reconstruction by multiplying it by the original standard deviation and adding back in the original mean.

Results:

Min/Max image values on PC1 and PC2:
Max image on Pc1: 134.14108, Min image on Pc1: -110.21942
Max image on Pc2: 89.19568, Min image on Pc2: -81.47462

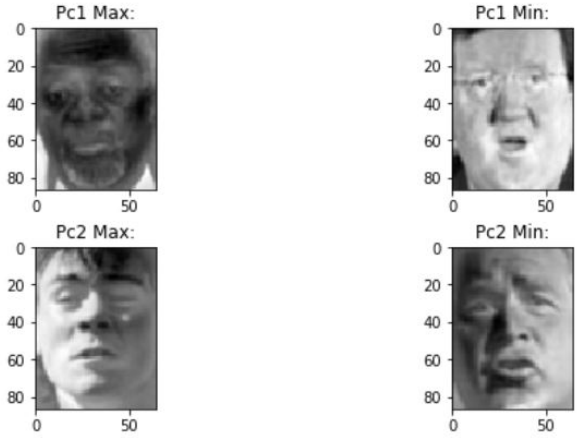
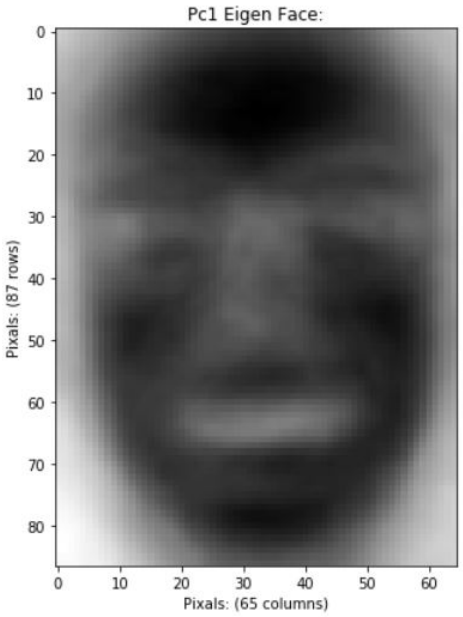
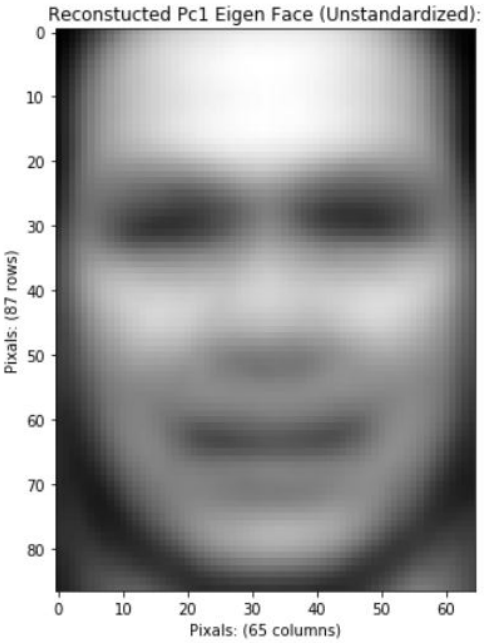


Figure 4: PC1 and PC2 projected min/max images

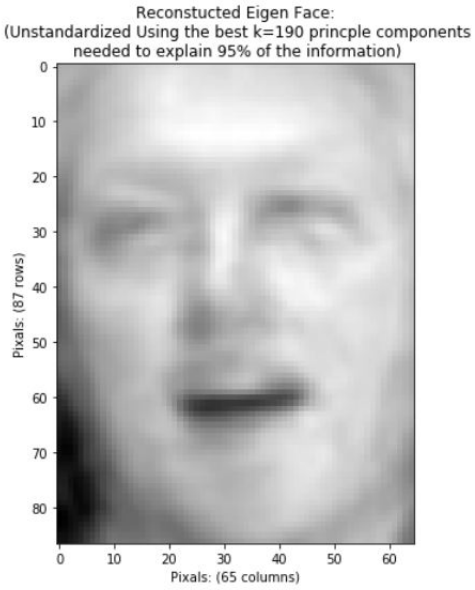
Response:
These variation components capture the tails of the distribution of the data projected on each of the principle components. These components also allow us to gleam insight into the spread of the data. We can think of these as the extreme cases of the distribution of projected images.



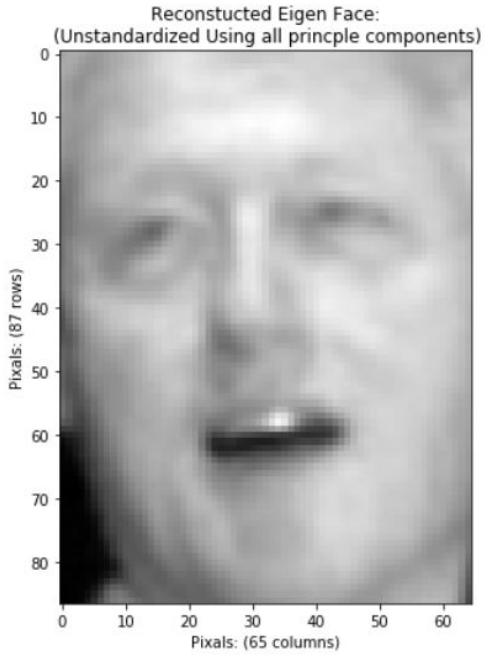
(a) Primary Principle Component



(b) Reconstructed Primary Principle



(c) Reconstruction Using Principle Components Explaining 95% of the Information (K = 190)



(d) K (All) Principle Components

Figure 5: PCA Faces

The number of principle components needed to encode 95% of the information is: 190

4 Clustering

Let's implement our own version of k-means to cluster data!

Once you have your setup complete as shown above , write a script to do the following:

1. Write your own version of K-means clustering where you use the L2 distance to compute similarity
2. Standardize your data (zero mean, divide by standard deviation)
3. Reduces the data to 100D using PCA.
4. Run K-means clustering with K = 10.
5. Report the number of images within each cluster.
6. Reconstruct the cluster centers for each of the K clusters. You will have to rotate the cluster centers back to the original space to visualize. Report these images.
7. Find the image closest to the cluster center, and furthest from the cluster center and report these images. Again, you will have to rotate the images centers back to the original space to visualize.

Implementation Details

1. Seed your random number generator with zero (do this right before running your k-means).
2. Randomly select k observations and use them for the initial reference vectors. I suggest you use randomize the indices and use the first k randomized indices to select the observations.
3. Use the L2 distance (Euclidean) to measure the distance between observations and reference vectors.
4. Terminate the training process when the sum of magnitude of change of the cluster centers (from the previous iteration to the current one) is less than $\epsilon = 2^{-23}$. That is, when $\sum_{i=1}^{k} d(a_i(t-1), a_i(t)) < \epsilon$ where k is the number of clusters, $a_i(t)$ is the reference vector for cluster i at time t and $d(x, y)$ is the L1 distance (Manhattan) between vectors x and y (as defined in the *Similarity and Distance Functions* link on BBlearn), or when you've hit 10,000 iterations.

Results:

Number of images per cluster: Cluster 0 contains 252 images, Cluster 1 contains 213 images, Cluster 2 contains 168 images, Cluster 3 contains 225 images, Cluster 4 contains 255 images, Cluster 5 contains 393 images, Cluster 6 contains 152 images, Cluster 7 contains 116 images, Cluster 8 contains 146 images, and Cluster 9 contains 143 images.

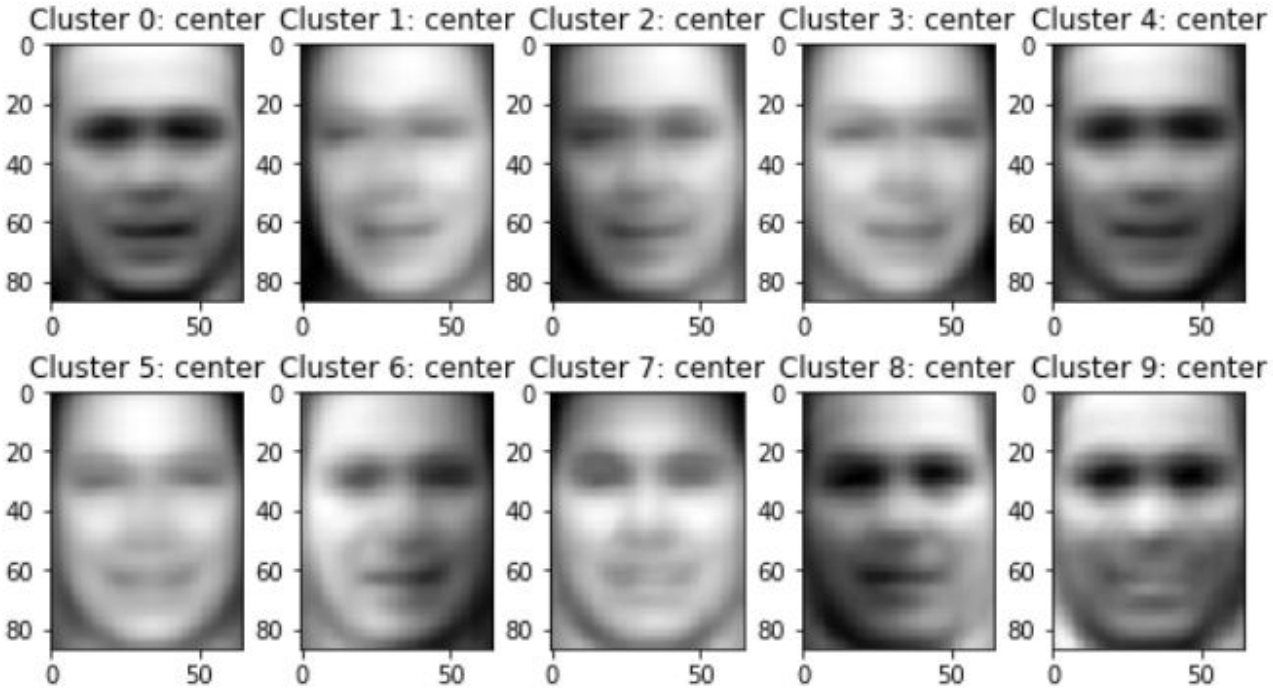


Figure 6: K-Means Cluster Centers

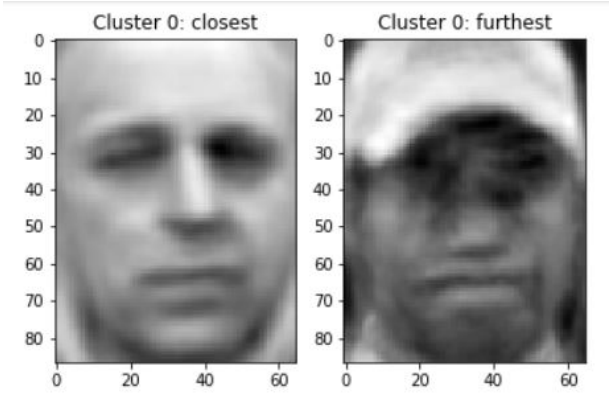


Figure 7: K-Means: Min/Max Cluster 0

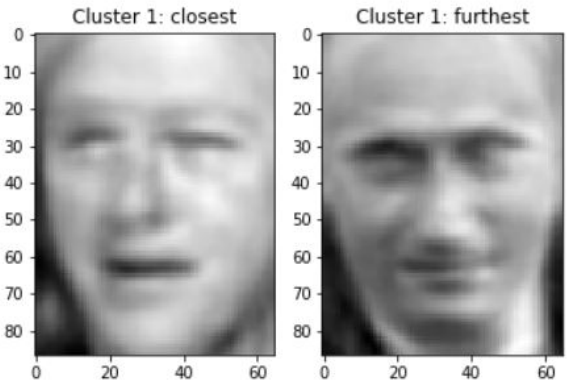


Figure 8: K-Means: Min/Max Cluster 1

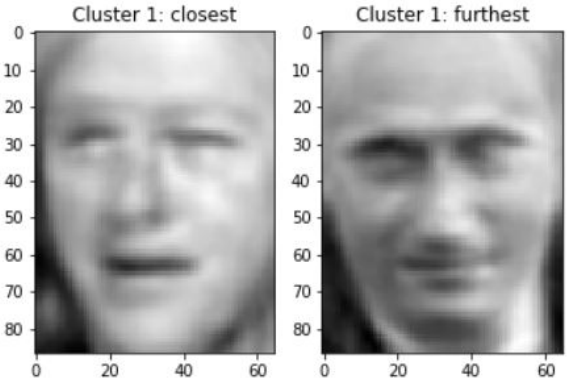


Figure 9: K-Means: Min/Max Cluster 2

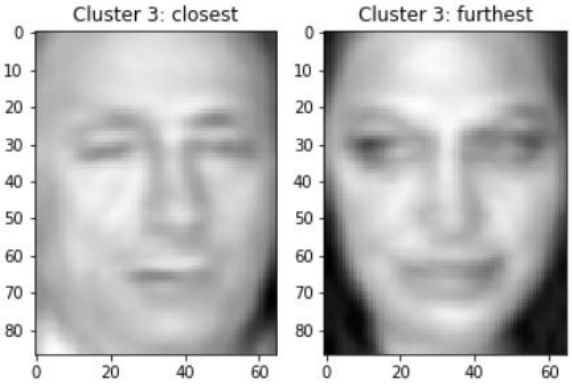


Figure 10: K-Means: Min/Max Cluster 3

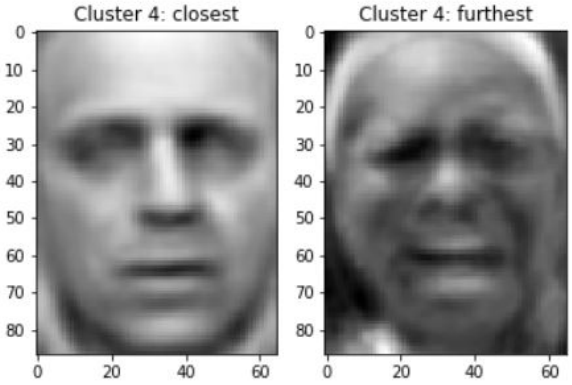


Figure 11: K-Means: Min/Max Cluster 4

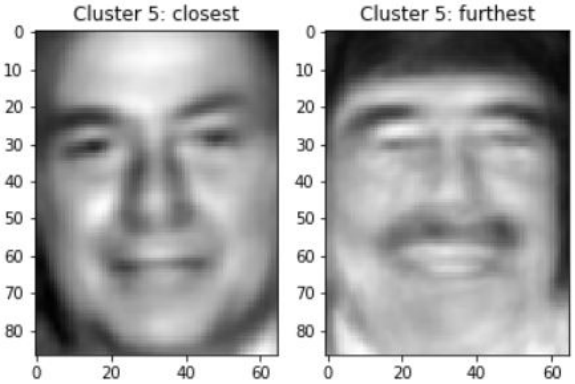


Figure 12: K-Means: Min/Max Cluster 5

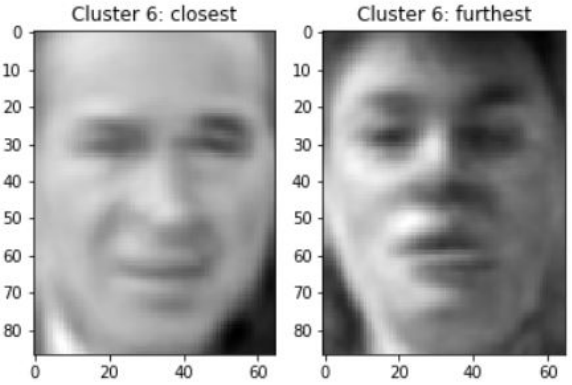


Figure 13: K-Means: Min/Max Cluster 6

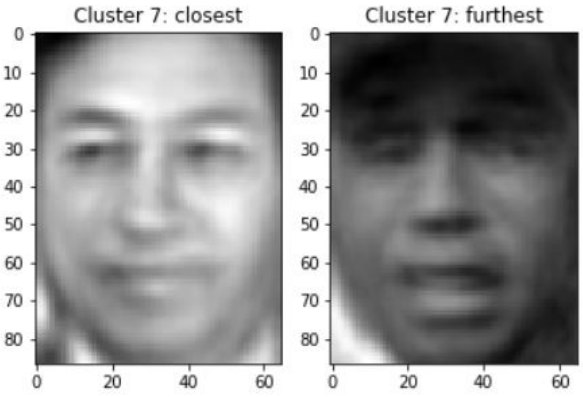


Figure 14: K-Means: Min/Max Cluster 7

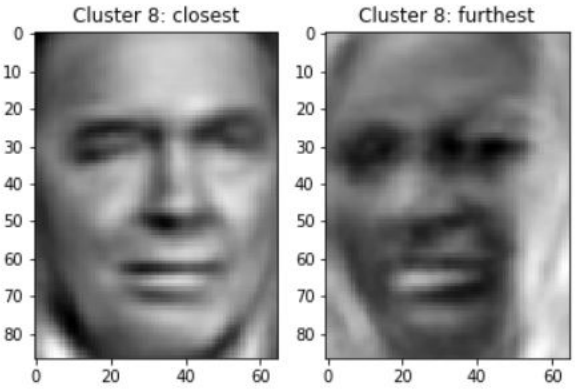


Figure 15: K-Means: Min/Max Cluster 8

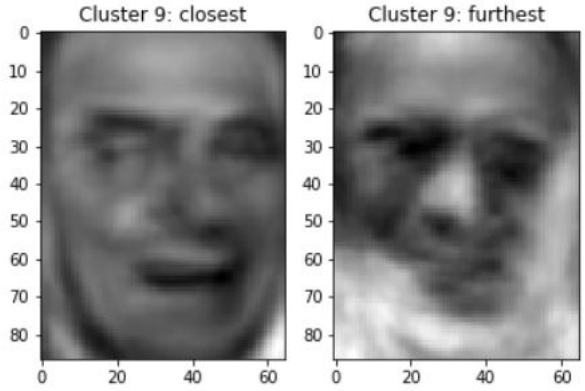


Figure 16: K-Means: Min/Max Cluster 9