CS 613 - Machine Learning

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

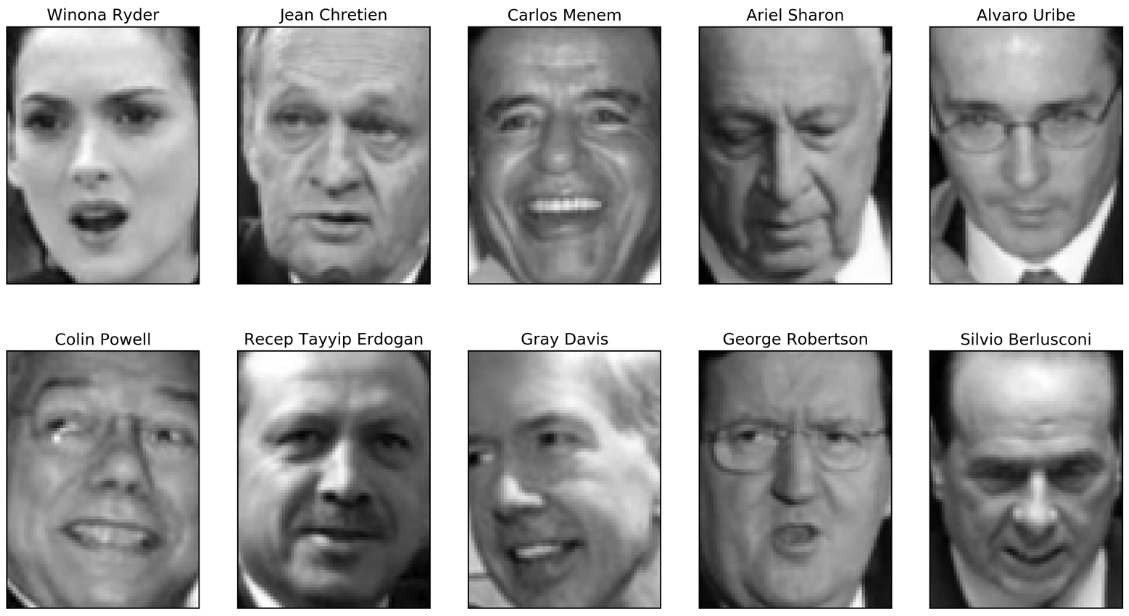
Labeled Faces in the Wild Datasaet This dataset consists of celebrities download from the Internet from the early 2000s. We use the grayscale version from sklearn.datasets.

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:

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

   (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).

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

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).

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

   (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).

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

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

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*.

Your graph should end up looking similar to Figure 1 (although it may be rotated differently, depending how you ordered things).

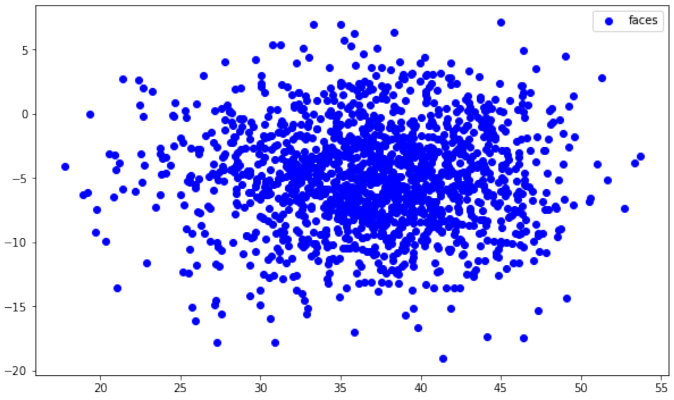


Figure 1: 2D PCA Projection of data

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 2).
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 4). 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.

Your principle eigenface should end up looking similar to Figure 2.

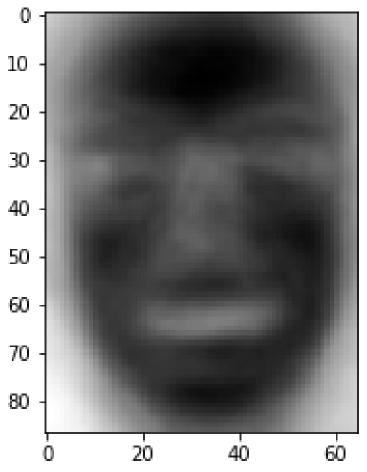


Figure 2: Primary Principle Component

Your principal reconstruction should end up looking similar to Figure 3.

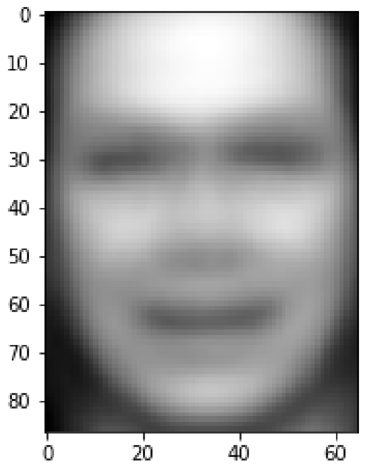


Figure 3: Reconstruction of first person

Your 95% reconstruction should end up looking similar to Figure 4.

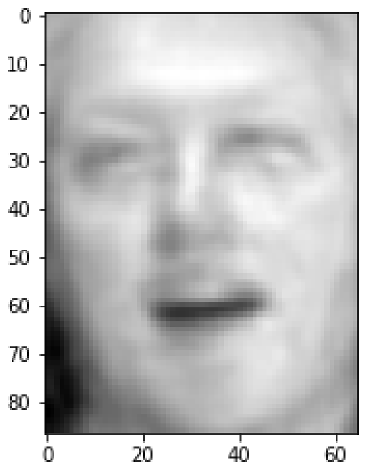


Figure 4: Reconstruction of first person)

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.

Submission

For your submission, upload to Blackboard a single zip file containing:

1. A LaTeX typeset PDF containing:
   (a) Part 1: Your answers to the theory questions.
   (b) Part 2: The visualization of the PCA result, KNN accuracies
   (c) Part 3:
      i. Visualization of primary principle component
      ii. Number of principle components needed to represent 95% of information, k.
      iii. Visualization of the reconstruction of the first person using
         A. Original image
         B. Single principle component
         C. k principle components.
   (d) Part 4: The visualization of k-means cluster centers, and the min and max images (a total of 30 images, 10 cluster centers, 2 extrema per cluster)
   (e) Source Code - python notebook