HMC CS 158, Spring 2018
Problem Set 8 Project: Famous Faces

Goals:

- To investigate one application of PCA: eigenfaces (a real word!).
- To open up the "black-box" of k-means (and k-medoids) clustering.
- To investigate dimensionality reduction and various clustering techniques towards one application of machine learning: facial recognition.

For this assignment, you can work individually though you are encouraged to work with a partner. You should sign up for partners on Canvas (People → PS8 Groups). If you are looking for a partner, try Piazza. If, after trying Piazza, you are having trouble finding a partner, e-mail Jessica.

Submission

You should submit any answers to the exercises in a single file `writeup.pdf`. This writeup should include your name and the assignment number at the top of the first page, and it should clearly label all problems. Additionally, cite any collaborators and sources of help you received (excluding course staff), and if you are using slip days, please also indicate this at the top of your document.

Your code should be commented appropriately. The most important things:

- Your name and the assignment number should be at the top of each file.
- Each class and method should have an appropriate doctsring.
- If anything is complicated, it should include some comments.

There are many possible ways to approach the programming portion of this assignment, which makes code style and comments very important so that staff can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

When you are ready to submit, make sure that your code compiles and remove any debugging statements. Then rename the top-level directory as `<username1>_<username2>_ps8`, with the usernames in alphabetical order (e.g. `hadas_yjw_ps8`). This directory should include the electronic version of your writeup and main code, any files necessary to run your code (including any code and data provided by staff), and follow the same structure as the assignment directory provided by staff. So, for this assignment, your directory should have the following structure:

- `<username1>_<username2>_ps8/`
  - `source/`
    - `cluster.py` (with your modifications)
    - `faces.py` (with your modifications)
    - `util.py` (with your modifications)
  - `writeup.pdf`
  - `cluster.pdf` (pdf printout of `cluster.py`)
  - `faces.pdf` (pdf printout of `faces.py`)
  - `util.pdf` (pdf printout of `util.py`)

Package this directory as a single `<username1>_<username2>_ps8.zip` file, and submit the archive.

This assignment is adapted from course material by Jenna Wiens (UMich).

Additionally, to aid the staff in grading, submit your `pdf`'s as separate files, and submit your predictions as a separate file as well.

Introduction

Machine learning techniques have been applied to a variety of image interpretation problems. In this project, you will investigate facial recognition, which can be treated as a clustering problem ("separate these pictures of Joe and Mary").

For this project, we will use a small part of a huge database of faces of famous people (Labeled Faces in the Wild [LFW] people dataset[1]). The images have already been cropped out of the original image, and scaled and rotated so that the eyes and mouth are roughly in alignment; additionally, we will use a version that is scaled down to a manageable size of 50 by 37 pixels (for a total of 1850 "raw" features). Our dataset has a total of 1867 images of 19 different people. You will apply dimensionality reduction using principal component analysis (PCA) and explore clustering methods such as k-means and k-medoids to the problem of facial recognition on this dataset.

Download the starter files from the course website. It contains the following source files:

- `util.py` – Utility methods for manipulating data, including through PCA.
- `cluster.py` – Code for the `Point`, `Cluster`, and `ClusterSet` classes, on which you will build the clustering algorithms.
- `faces.py` – Main code for the project.

Please note that you do not necessarily have to follow the skeleton code perfectly. We encourage you to include your own additional methods and functions. However, *you are not allowed to use any* `scikit-learn` *classes or functions other than those already imported in the skeleton code.*

Dataset warning: Before you start this assignment, you should use `util.get_lfw_data(...)` to get the LFW dataset with labels. Make sure that you have 19 classes. (You may have fewer classes since there is some randomness involved in how `scikit-learn` loads the dataset.) If you have fewer classes, you have two choices: (1) work with a partner whose dataset has 19 classes, or (2) work on your own but with a few modifications to the instructions in this assignment.

1 PCA and Image Reconstruction [8 pts]

Before attempting automated facial recognition, you will investigate a general problem with images. That is, images are typically represented as thousands (in this project) to millions (more generally) of pixel values, and a high-dimensional vector of pixels must be reduced to a reasonably low-dimensional vector of features.

(a) **(1.5 pts)** As always, the first thing to do with any new dataset is to look at it. Use `get_lfw_data(...)` to get the LFW dataset with labels. Plot a couple of the input images using `show_image(...)`. Then compute the mean of all the images, and plot it. (Remember to include all requested images in your writeup.) Comment briefly on this "average" face.

[1] http://vis-www.cs.umass.edu/lfw/

(b) **(1.5 pts)** Perform PCA on the data using `util.PCA(...)`. This function returns a matrix `U` whose columns are the principal components, and a vector `mu` which is the mean of the data. If you want to look at a principal component (referred to in this setting as an eigenface), run `show_image(vec_to_image(v))`, where `v` is a column of the principal component matrix. (This function will scale vector `v` appropriately for image display.) Show the top twelve eigenfaces:

```
plot_gallery([vec_to_image(U[:,i]) for i in xrange(12)])
```

Comment briefly on your observations. Why do you think these are selected as the top eigenfaces?

(c) **(5 pts)** Implement `apply_PCA_from_Eig(...)` and `reconstruct_from_PCA(...)` (available in `util.py`) based on the provided specifications.

Then explore the effect of using few or many dimensions to represent images by:

- Finding the principal components of the data
- Selecting a number l of components to use
- Reconstructing the images using only the first l principal components
- Visually comparing the images to the originals

To perform PCA, use `apply_PCA_from_Eig(...)` to project the original data into the lower-dimensional space, and then use `reconstruct_from_PCA(...)` to reconstruct high-dimensional images out of lower dimensional ones. Then, using `plotGallery(...)`, submit a gallery of the first 12 images in the dataset, reconstructed with l components, for $l = 1, 10, 50, 100, 500, 1288$. Comment briefly on the effectiveness of differing values of l with respect to facial recognition.

We will revisit PCA in the last section of this project.

2 K-Means [12 pts]

Next, we will explore the k-means algorithm in detail by applying it to a toy dataset.

(a) **(2 pts)** In k-means, we attempt to find k cluster centers $\boldsymbol{\mu}_j \in \mathbb{R}^d$, $j \in \{1, \ldots, k\}$ and n cluster assignments $c^{(i)} \in \{1, \ldots, k\}$, $i \in \{1, \ldots, n\}$, such that the total distance between each data point and the nearest cluster center is minimized. In other words, we attempt to find $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ and $c^{(1)}, \ldots, c^{(n)}$ that minimizes

$$J(\boldsymbol{c}, \boldsymbol{\mu}) = \sum_{i=1}^{n} ||\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{c^{(i)}}||^2.$$

To do so, we iterate between assigning $\boldsymbol{x}^{(i)}$ to the nearest cluster center $c^{(i)}$ and updating each cluster center $\boldsymbol{\mu}_j$ to the average of all points assigned to the j^{th} cluster.

Instead of holding the number of clusters k fixed, one can think of minimizing the objective function over $\boldsymbol{\mu}$, $\boldsymbol{c}$, *and* k. Show that this is a bad idea. Specifically, what is the minimum possible value of $J(\boldsymbol{c}, \boldsymbol{\mu}, k)$? What values of $\boldsymbol{c}$, $\boldsymbol{\mu}$, and k result in this value?

(b) **(2 pts)** To implement our clustering algorithms, we will use Python classes to help us define three abstract data types: `Point`, `Cluster`, and `ClusterSet` (available in `cluster.py`). Read through the documentation for these classes. (You will be using these classes later, so make sure you know what functionality each class provides!) Some of the class methods are already implemented, and other methods are described in comments. Implement the methods marked `TODO` in the `Cluster` and `ClusterSet` classes.

(c) **(7 pts)** Next, implement `random_init(...)` and `kMeans(...)` (available in `faces.py`) based on the provided specifications.

You may find it useful to test the performance of k-means on a toy dataset. We have provided one such toy dataset that contains three clusters, each containing 20 points. (You can modify the code `generate_points_2d(...)` to test different inputs while debugging your code, but be sure to return to the initial implementation before comparing against the plots below.) If you implemented `random_init(...)` and `kMeans(...)` correctly, you should generate the following plots, where color represents the cluster assignments and circles indicate the cluster "centers":

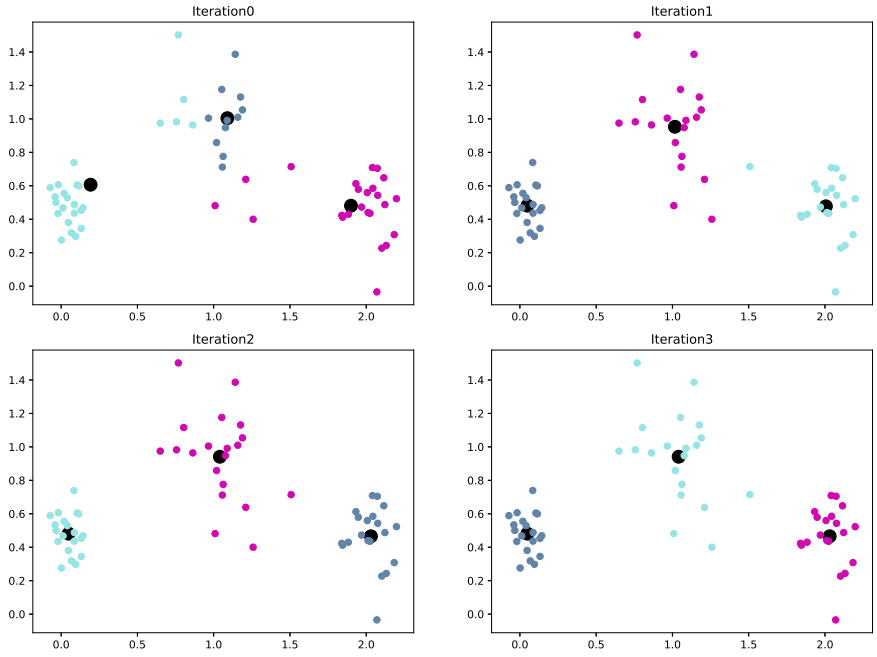

Figure 1: k-means with random initialization

Note that you may have different results based on the randomly generated toy dataset and different random initialization.

(d) **(1 pts)** Finally, we will explore the effect of initialization. Implement `cheat_init(...)` and update `kMeans(...)` to allow both types of initialization.

If you implemented `cheat_init(...)` correctly, you should generate the following plots:

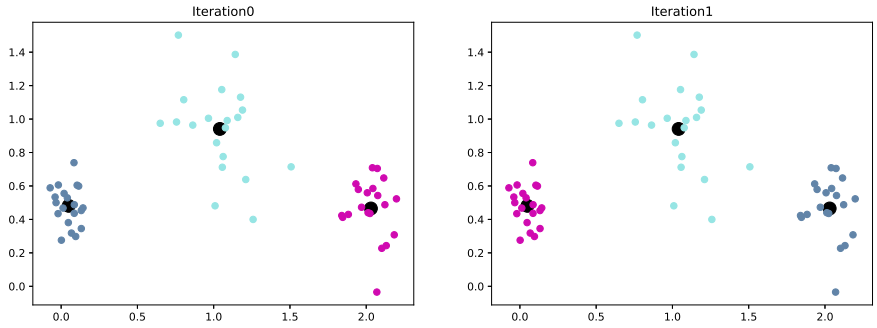

Figure 2: k-means with cheat initialization

3 Clustering Faces [10 pts]

Finally (!), we will apply clustering algorithms to the image data. To keep things simple, we will only consider data from two individuals. Make a new image dataset by selecting 40 images each from classes 4 and 13[2], then translate these images to (labeled) points:

```
X1, y1 = util.limit_pics(X, y, [4, 13], 40)
points = build_face_image_points(X1, y1)
```

(a) **(5 pts)** Explore the effect of lower-dimensional representations on clustering performance. To do this, compute the principal components for the entire image dataset, then project the newly generated dataset into a lower dimension (varying the number of principal components). Apply k-means to this lower-dimensional dataset, then compute the average cluster purity with `ClusterSet.score(...)`.

So that we are only changing one thing at a time, use `init='cheat'` to generate the same initial set of clusters. For each value of l, the number of principal components, you will have to generate a new list of points using `build_face_image_points(...)`.)

Let $l = 1, 6, 11, \ldots, 101$. Then, plot the clustering score versus the number of components for each clustering algorithm. Discuss the results in a few sentences, and provide reasoning for your observations.

Some pairs of people are more similar to one another and some more different.

(b) **(5 pts)** Experiment with the data to find a pair that clustering can discriminate very well and another pair that it finds very difficult. Describe your methodology. Report the two pairs in your writeup (display the images), and comment briefly on the results.

[2]If your dataset does not have 19 classes, then you can use the first two classes instead.

4 Extra Credit: K-Medoids [+4 pts]

Next, we will investigate k-medoids (a slight variation on k-means).

(a) **(+1 pts)** In `cluster.py`, implement `Cluster.medoid(...)` and `ClusterSet.medoids(...)`. In `faces.py`, implement `kMedoids(...)` based on the provided specification.

Hint: Since k-means and k-medoids are so similar, you may find it useful to refactor your code to use a helper function `kAverages(points, k, average, init='random', plot=True)`, where `average` is a method that determines how to calculate the average of points in a cluster (so it can take on values `ClusterSet.centroids` or `ClusterSet.medoids`).[3]

If you implemented `kMedoids(...)` correctly, it should take three iterations to cluster the toy dataset.

(b) **(+1 pts)** Construct a dataset by selecting 40 images each from individuals 4, 6, 13, and 16[4]., Apply k-means and k-medoids to this new dataset, initializing the centroids randomly, and evaluate the performance of each clustering algorithm. As the performance of the algorithms can vary widely depending upon the initialization, run both clustering methods 10 times and report the average, minimum, and maximum performance.

	average	min	max
k-means			
k-medoids			

How do the clustering methods compare in terms of clustering performance and runtime?

(c) **(+2 pts)** Let us reevaluate the effect of lower-dimensional representations on clustering performance. Regenerate your plot of clustering score versus the number of components. Put both clustering algorithms on the same plot, and be sure to label the algorithms. Compare the algorithms, and provide reasoning for your observations.

[3] In Python, if you have a function stored to the variable `func`, you can apply it to parameters `arg` by callling `func(arg)`. This works even if `func` is a class method and `arg` is an object that is an instance of the class.

[4] If your dataset does not have 19 classes, then you can use the first four classes instead.