CS188, Winter 2017
Problem Set 4: Clustering and PCA
Due March 16,2017, 4:00pm

Submission instructions

- Submit your solutions electronically on the course Gradescope site as PDF files.
- If you plan to typeset your solutions, please use the LaTeX solution template. If you must submit scanned handwritten solutions, please use a black pen on blank white paper and a high-quality scanner app.

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

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.*

1 PCA and Image Reconstruction [4 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) 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, and 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.

(b) 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?

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

(c) Explore the effect of using more or fewer dimensions to represent images. Do this 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 and K-Medoids [16 pts]

Next, we will explore clustering algorithms in detail by applying them to a toy dataset. In particular, we will investigate k-means and k-medoids (a slight variation on k-means).

(a) 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) 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 all of the methods marked `TODO` in the `Cluster` and `ClusterSet` classes.

(c) Next, implement `random_init(...)` and `kMeans(...)` based on the provided specifications.

(d) Now test the performance of k-means on a toy dataset.

Use `generate_points_2d(...)` to generate three clusters each containing 20 points. (You can modify `generate_points_2d(...)` to test different inputs while debugging your code, but be sure to return to the initial implementation before creating any plots for submission.) You can plot the clusters for each iteration using the `plot_clusters(...)` function.

In your writeup, include plots for the k-means cluster assignments and corresponding cluster "centers" *for each iteration* when using random initialization.

(e) 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`).[2]

As before, include plots for k-medoids clustering *for each iteration* when using random initialization.

(f) Finally, we will explore the effect of initialization. Implement `cheat_init(...)`.

Now compare clustering by initializing using `cheat_init(...)`. Include plots for k-means and k-medoids *for each iteration*.

3 Clustering Faces [12 pts]

Finally (!), we will apply clustering algorithms to the image data. To keep things simple, we will only consider data from four individuals. Make a new image dataset by selecting 40 images each from classes 4, 6, 13, and 16, then translate these images to (labeled) points:

```
X1, y1 = util.limit_pics(X, y, [4, 6, 13, 16], 40)
points = build_face_image_points(X1, y1)
```

(a) Apply k-means and k-medoids to this new dataset with $k = 4$ and initializing the centroids randomly. Evaluate the performance of each clustering algorithm by computing the average cluster purity with `ClusterSet.score(...)`. 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?

Now construct another dataset by selecting 40 images each from two individuals 4 and 13.

[2] 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.

(b) 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), and compute the scores of each clustering algorithm.

So that we are only changing one thing at a time, use `init='cheat'` to generate the same initial set of clusters for k-means and k-medoids. 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 $l = 1, 3, 5, \ldots, 41$. The number of clusters $K = 2$. Then, on a single plot, plot the clustering score versus the number of components for each clustering algorithm (be sure to label the algorithms). Discuss the results in a few sentences.

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

(c) Experiment with the data to find a pair that clustering can discriminate very well and another pair that it finds very difficult (assume you have 40 images for each individual). Describe your methodology (you may choose any of the clustering algorithms you implemented). Report the two pairs in your writeup (display the pairs of images using `plot_representative_images`), and comment briefly on the results.