Homework 2

Jeff Tilton

September, 15 2019

1 Image Compression Using Clustering

1.1 General K-medoids implementation

I decided to keep my k-medoids implementation simple and rely on random grid search to find the centroids with the least cost where cost is the sum of all point distacnes to nearest cluster center. The algorithm follows the steps below.

- Find centroids using k ++ find the closest pixels to these centroids and initialize with these centroids.
- Assign all points to the closest cluster using assigned distance measure
- Calculate the initial total cost which is the sum of all distances.
- Use a random grid search of all possible centroid combinations for N runs.
- Update best cluster centers if new cost ¡ previous cost else keep current cluster centers as best.
- Generate classes from best set of cluster centers

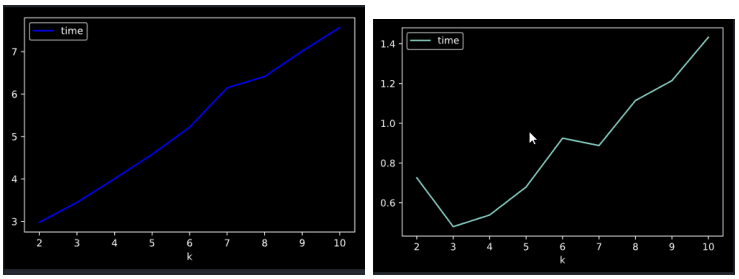
1.2 Comparison of Different K values and Distance Measures

Figure 1 below shows the outputs of the K-means and K-medoids implementations with several distance measures and K values. The picture became less compressed (looked closer to the original) the higher the k value. It is hard to tell which distance measure performed best. Average results were consistent, but differences between runs had an impact on which measure performed best within any particular run.

(a) $k = 3$ (b) $k = 15$ (c) $k = 25$

Figure 1: Comparison of 3 different distance measures (Euclidean, trace class norm, and infinite norm) with 3 diffent values for K (3, 15, 25)

I ran the algorithm as a random grid search for N runs so time to calculate distances between centroids increased linearly as a function of k for k-medoids. K-means largely followed a linear trend as well, but there were instances when a larger value of k converged faster. This is due to the randomness in the initialization.



(a) $K - medoids$ (b) $K - means$

Figure 2: Time until convergence for different values of K

1.3 Comparison with Different Initial Centroids

I implemented the k-medoids algorithm with initial centroids equal to each other. I use random grid search in my implementation so there is little to no effect in the final outcome. The second batch of clusters will most likely have a lower cost than the intentional poor assignment.

1.4 Comparison of K-means and K-medoids

Although I do not see a large difference between the two algorithms in terms of image quality, I do think K-means performs better overall. K-means is also much faster because it meets the convergence criteria where k-medoids has a predetermined amount of runs to perform random grid search.

2 Spectral Clustering

Spectral clustering considers data geometry going beyond K-means that only considers distance.

2.1 Show that there are m eigenvectors of L corresponding to eigenvalue zero, and the indicator vectors of these components $I_{A_1}, ..., I_{A_m}$ span the zero eigenspace.

Whoa.

Ok so consider a 2-node undirected graph with a single connection between A—B

The graph Laplacian is defined as $L = D - A$ where D is a degree matrix and A is an adjacency matrix. So for our example above.

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{2}$$

A property of a graph Laplacian (in perfect cases) is that the smallest eigencalue of L is 0. Therefore the above matrix, with one connected component will have a single m vector of L with a 0 eigenvalue. A graph Lapalcian with k connected components is a combination of k graph Laplacian blocks on the diagonal with zeroes filling in the non-block portions.

Each block will have a single m vector of L with a 0 eigenvalue associated with it.

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

2.2 Real Data: Political Blogs Dataset

The spectral.py file shows the steps taken to obtain the false classification rate. A 50% average false classification rate was obtained after 20 runs of eigen decomposition and K-means clustering on the dataset. This result is no better than a random guess.

3 PCA Food Consumption

3.1 Find the first two principal directions, and plot them.

The first two principal directions were found using numpy's eigendecompostion and are seen in figure 3.

3.2 Compute the reduced representation of the data point (which are sometimes called the principal components of the data). Draw a scatter plot of two-dimensional reduced representation for each country. Do you observe some pattern?

I obtained and plotted the first two principal components (PC's) using eigenvalue decomposition with the numpy and scipy python libraries. I also used numpy's Singular Value Decomposition to obtain the PC's and then compared these methods with scikit-learn's implementation all seen in figure 4 below.

I categorized the countries into regions from data I found online and found that all methods clustered the data nicely into the specific regions, but results were different. I looked at the source code for scikit-learn and see that it uses numpy's SVD under the hood, but it does some type of transformation afterwards that appears to create a mirror of my results from SVD. I am not sure what the difference between the scipy implementation and numpy implementation of eigenvalue decomposition is.

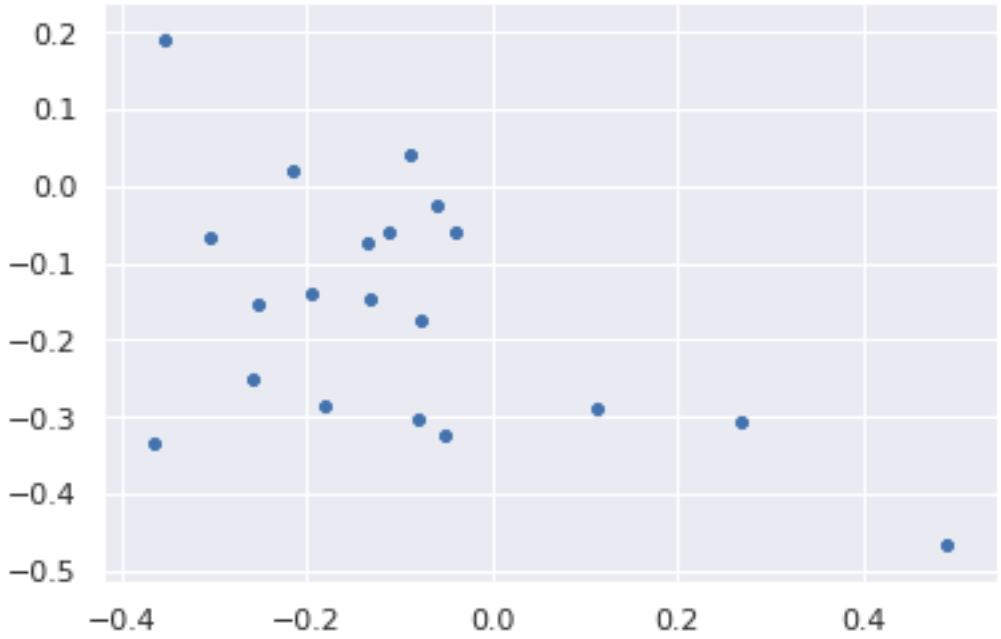
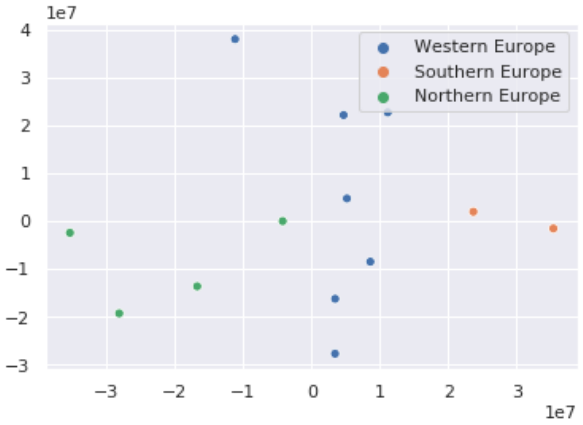
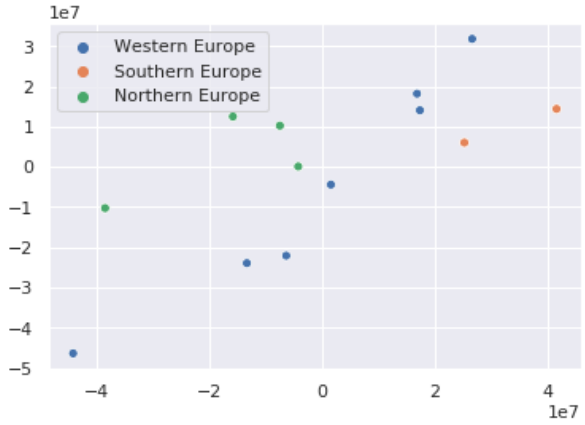


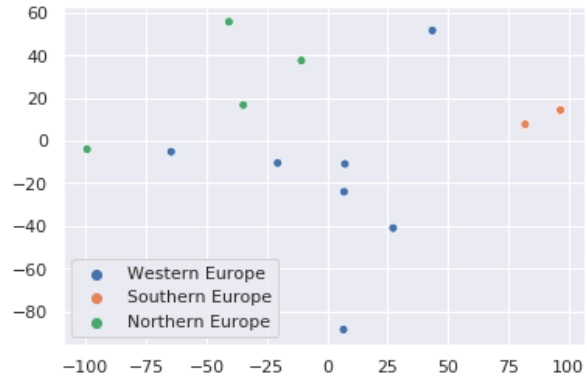
Figure 3: *The first two principal directions using eigendecomposition (numpy)*



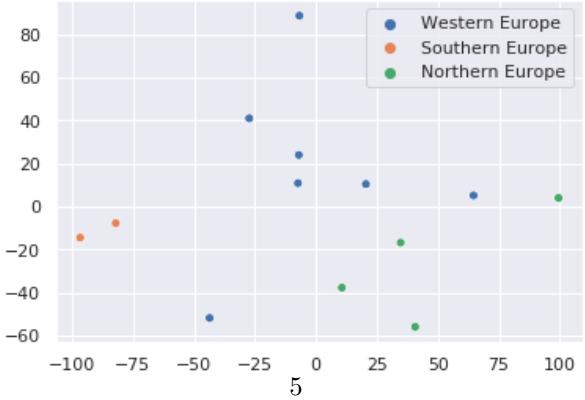
(a) *PCA using eigendecomposition (numpy)*



(b) *PCA using eigendecomposition (scipy)*



(c) *PCA using Singular Value Decomposition*



(d) *PCA using scikit-learn*

Figure 4: *Comparisons of Principal Component Analysis methods/libraries*