ISYE 6740 Homework 1

Nick DiNapoli, ndinapoli6@gatech.edu

Due: September 5, 2021

1 Conception questions

1.1 What's the main difference between supervised and unsupervised learning?

Supervised learning is one subgroup of machine learning that deals with training labeled data. In unsupervised learning, datasets are unlabeled and the algorithm is left to learn on its own without being explicitly told data point's class labels.

1.2 Will different initializations for k-means lead to different results?

Although this question arose in lecture and the answer was formally, "yes", I beg that the answer is "sometimes" (i.e. mostly yes but not necessarily). It is clear that vastly different randomized initializations will lead to some data points being classified into different clusters. I claim "sometimes" because imagine an instance when data clusters are far apart in feature space (easily defined clusters), for the two cluster scenario, cluster center initializations of c^1 and c^2 could have the exact same data point prediction as initialization $c^1 + \delta$ and $c^2 + \delta$ where δ is small. Although this is unlikely, we cannot ignore that it is true.

1.3 Give a short proof (can be in words but using correct logic) why k-means algorithm will converge in finite number of iterations.

The goal of the k-means algorithm is to minimize the within-cluster sum of squares (WCSS) and hence the objective function,

$$\frac{1}{m}\sum_{i=1}^{m}\|x^i - c^{\pi(i)}\|^2 \tag{1}$$

The two main iterative steps of the algorithm include cluster assignment (decide the cluster membership of each data point) and cluster adjustment (adjusting the cluster centers) both of which only make changes to decrease the objective function. Because of this, the objective function is monotonically decreasing and has a minimum value of 0. Because of this, the algorithm will converge in a finite number of steps ($< k^m$).

1.4 What is the main difference between k-means and generalized k-means algorithm?

In the k-means algorithm, the distance metric or dissimilarity function used in the objective function, cluster membership, and cluster center adjustment is the Euclidean distance or L2 norm. In the generalized k-means algorithm, the dissimilarity function takes on a more general form,

$$d(x^i, c^{\pi(i)}) \tag{2}$$

The main difference from k-means to generalized k-means is in the cluster center adjustment step where k-means used the Euclidean distance and generalized k-means chooses the v that minimizes the following expression,

$$\sum_{i:\pi(i)=j} d(x^i, v)^2 \tag{3}$$

1.5 Write down the graph Laplacian matrix and find the eigenvectors associated with the zero eigen-value. Explain how do you find out the number of disconnected clusters in graph and identify these disconnected clusters using these eigenvectors.

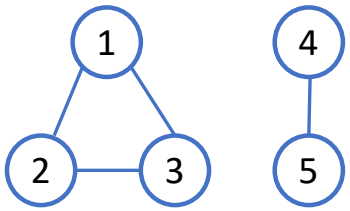


Figure 1: Consider the following simple graph

The graph Laplacian, $L = D - A$, is derived below,

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$L = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \tag{5}$$

Now, I can find the eignevectors associated with the zero eignevalue,

$$Lv_1 = \frac{1}{\sqrt{3}} \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 0 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{6}$$

$$Lv_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = 0 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \tag{7}$$

I can find the number of disconnected clusters and the specific breakdown of which nodes belong to each by putting the eigenvectors together like so,

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \tag{8}$$

I can look at the rows of this new matrix and see which are the same and hence belong to the same community. Here I clearly verify what is seen in Figure 1.

2 Image compression using clustering

In this problem, I implement the k-means clustering algorithm from scratch, rather than calling it from a package. I verify its success and examine the results by applying the algorithm to several 2D, RBG (red, green, and blue) images. In this use case, each pixel of the images is a data point with three attributes, the R, G, and B values which each lie in the range, [0, 255].

2.1 Using squared-L2 norm as a metric

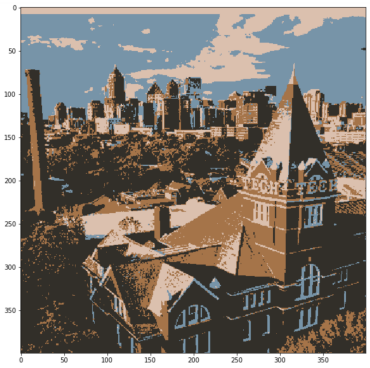
I first applied my implementation of the k-means clustering algorithm to the images shown in Figure 2. I used the L2 norm as the distance/dissimilarity metric and used randomly initialized centroids. I completed the clustering using several different values for k (k = 2, 4, 8, 16) and recorded both the number of iterations it took to converge and the compressed image. Compressed images have pixels that take on their associated class/cluster centroid value (RGB value). Figures 3-6 show the results from my implementation.

(a) GT campus (400x400) (b) GT football (620x412) (c) Portrait of me (400x400)

Figure 2: The three original images used for compression (k-means clustering)

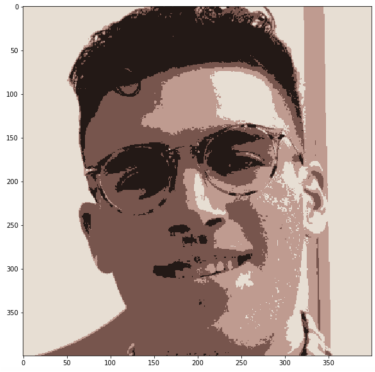
(a) number of iterations = 10 (b) number of iterations = 22 (c) number of iterations = 8

Figure 3: Compressed images after k-means clustering where k = 2



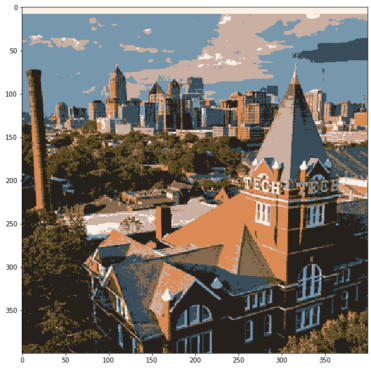
(a) number of iterations = 30

(b) number of iterations = 29



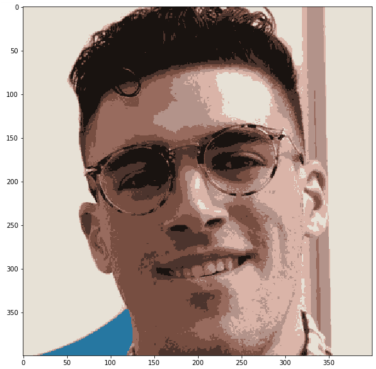
(c) number of iterations = 23

Figure 4: Compressed images after k-means clustering where k = 4



(a) number of iterations = 47

(b) number of iterations = 63

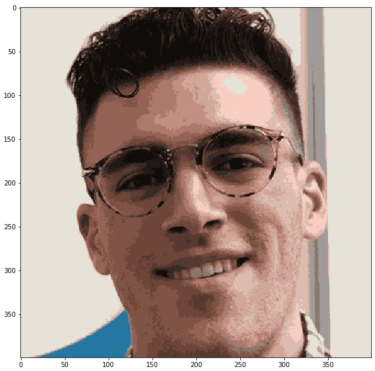


(c) number of iterations = 74

Figure 5: Compressed images after k-means clustering where k = 8

(a) number of iterations = 120

(b) number of iterations = 66



(c) number of iterations = 456

Figure 6: Compressed images after k-means clustering where k = 16

There are a few intuitive pieces of analysis that I can complete to ensure the algorithm is working as intended. The first and most obvious is that as k gets smaller, the compressed images become increasingly abstract. This is because with fewer clusters, one starts to lose sense of depth or image context. Another simple sanity check is noticing that the pixels in the k = 2 compressed image take on just one of two values (for (a) cream and dark grey qualitatively) which makes sense because there are only two clusters. Similarly

in the k = 4 compressed image, pixels are either cream, light blue, light brown, or dark brown (for image (a)). Of course there are RBG values or cluster centers associated with theses colors but I am just speaking qualitatively. Again, this verifies that the algorithm is working properly.

2.2 Using L1 distance as a metric

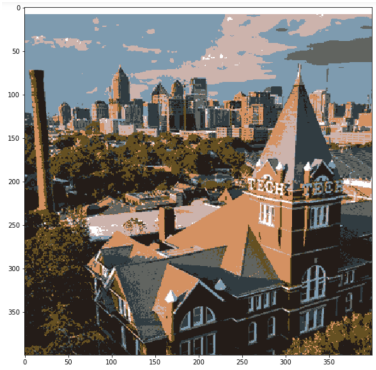
Next, I transitioned to using the generalized k-means approach using the L1 distance or Manhattan distance. I completed the exact same tasks as Section 2.1. It is worth noting again that when using the L1 norm, the cluster centers are now adjusted according to the median of each cluster's R, G, and B values respectively whereas before, the cluster center was taken as the average of each dimension. Figures 7-10 display the results of the clustering using the Manhattan distance metric.

(a) number of iterations = 5 (b) number of iterations = 10 (c) number of iterations = 6

Figure 7: Compressed images after k-means clustering where k = 2

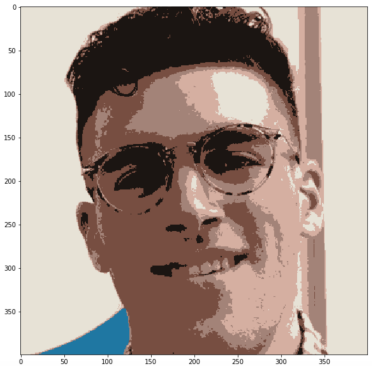
(a) number of iterations = 9 (b) number of iterations = 13 (c) number of iterations = 10

Figure 8: Compressed images after k-means clustering where k = 4



(a) number of iterations = 27

(b) number of iterations = 75

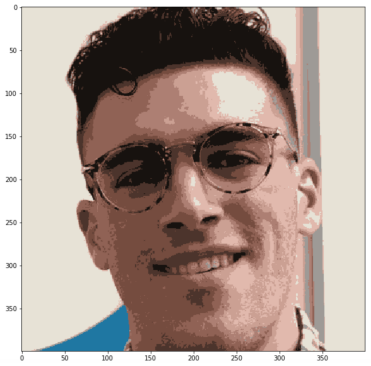


(c) number of iterations = 200

Figure 9: Compressed images after k-means clustering where k = 8

(a) number of iterations = 200

(b) number of iterations = 200



(c) number of iterations = 200

Figure 10: Compressed images after k-means clustering where k = 16

There are a few more takeaways that can be seen to ensure that the algorithm is functioning properly. Namely, the cluster centers are no longer long floating point values i.e. 123.456789... (computed using mean) but rather values from using the median values i.e. 123.0. Also the same trends touched on in Section 2.1 hold for this implementation. Regarding the comparison of using the L2 norm vs. L1 norm, I first noticed that when using the smaller number of clusters, the L1 norm seemed to converge faster. This makes mathematical sense because there is a chance that adding a few points to each cluster does not change the median value for R, G, and B, but it almost certainly changes the mean. I would've thought that this statement holds true for larger values of k but I found that I needed to cap the number of iterations to get the algorithm to converge in a reasonable amount of time. So the number of iterations for Figure 10(a-c) are capped at 200 but are would likely be significantly higher. Comparing the two qualitatively is also interesting. It is clear that there are differences for example from Figure 6(c) to Figure 10(c) which are cropped images of my portrait. The k-means implementation with L1 norm is much more pixelated (Figure 10(c)) and the L2 norm version is much "smoother" (Figure 6(c)). It is hard to say if this is due to the distance metric or difference in the number of iterations. In general the distance metric becomes more apparent for left/right skewed datasets where the mean and median differ greatly. Because these images aren't heavily skewed in one direction or another, the two implementations end up being fairly similar.

Note that my implementation in Python for this question is a bit unorthodox because I completed it before having a better understanding of numpy. Regardless, the implementation is sound, it just makes use of list comprehension and some additional helper functions to make the main kmeans function a bit more concise.

3 Political blogs dataset

The goal for this problem is to implement spectral clustering from scratch on a graph of political blog data. If successful (hypothesized), blog sites with a similar political orientation should cluster or form a community. There are 1490 nodes in this graph with undirected and unweighted edges to other nodes, but some nodes have no connections whatsoever and are isolated. For convenience and to solve the real problem at hand, I pre-processed the data by removing isolated nodes from the analysis bringing the dataset to 1224 nodes. Tables 1-4 show the outcomes of spectral clustering using k = 2, 5, 10, and 20. I report the number of points in each cluster, the majority label and the mismatch rate for each cluster.

Cluster number	Num points in cluster	Majority label	Mismatch rate
1	1222	1	0.480
2	2	0	0.000

Table 1: Results after applying spectral clustering with k = 2

Cluster number	Num points in cluster	Majority label	Mismatch rate
1	590	1	0.027
2	469	0	0.021
3	74	1	0.419
4	10	1	0.300
5	81	0	0.025

Table 2: Results after applying spectral clustering with k = 5

Cluster number	Num points in cluster	Majority label	Mismatch rate
1	57	1	0.421
2	212	1	0.028
3	128	0	0.016
4	31	0	0.419
5	30	0	0.200
6	29	0	0.034
7	364	1	0.019
8	15	1	0.467
9	309	0	0.026
10	49	0	0.041

Table 3: Results after applying spectral clustering with k = 10

Cluster number	Num points in cluster	Majority label	Mismatch rate
1	130	1	0.031
2	120	1	0.117
3	40	0	0.050
4	201	0	0.025
5	64	0	0.078
6	93	1	0.011
7	22	0	0.045
8	31	1	0.290
9	12	0	0.250
10	5	1	0.200
11	126	1	0.016
12	76	0	0.039
13	33	1	0.091
14	25	0	0.040
15	94	1	0.011
16	52	0	0.000
17	15	0	0.000
18	26	0	0.269
19	34	0	0.029
20	25	0	0.440

Table 4: Results after applying spectral clustering with k = 20

There are several conclusions one can draw from these results. First, the variance in number of points per cluster seems to decrease as I increase the number of clusters. For the k = 2 case, the results point towards the fact that there maybe are not two distinct communities in the political blogging world but rather maybe a spectrum that are a bit too connected to tell a difference. Here, one could interpret the results as there is really only one major political community filled with affiliations from both parties (labels). As I increase the number of clusters, I can see the formation of communities within the dataset where similar blogs form communities and have a higher density of edges. Another sanity check that the algorithm is working properly is noticing that around half of the clusters have an affiliation majority with one label/party and half of the cluster are affiliated with the other party. This makes intuitive sense about the political orientation of the country as a whole and can be seen in the original labeling data as well. An additional interesting conclusion that can be drawn is the fact that the average mismatch rate seems to decrease as I increase k. I use this observation to drive the question of, "what is the optimal number of clusters to achieve a reasonably good mismatch rate?"

To answer this question, I explore a few values for k around k = 20 to see how the average cluster-mismatch rate trends from an apparent good solution in k = 20. Table 5 summarizes my calculated average cluster-mismatch rate for different values of k near k = 20.

k	Average mismatch rate
10	0.171
20	0.085
25	0.090
30	0.073
35	0.082
40	0.073
45	0.066
50	0.066
55	0.061
60	0.058

Table 5: Average cluster-mismatch rate at various k values

One should be careful when tackling this sort of analysis for several reasons. In general as k approaches the number of data points (overfitting), there are too few data points in each cluster for the mismatch rate to be representative of how much sense it makes to use that k. When $k = 30$, there can still be around 40 data points per cluster on average and the average mismatch rate is very reasonably low, $\approx$7.3%. To complete this I conducted a sort of binary search around values near $k = 20$ where the mismatch rate was fairly low. This result can be interpreted as the existence of 30 reasonably separated communities within the network that mostly have the same political orientation. Increasing the number of communities further may be getting too specific and not having enough clusters (like $k = 2$) only tells us that these political blogs have connections to one another (probably because they're all about politics). There exists some middle ground around $k = 30$ where there is optimal separation between distinct groups or communities in this network.