Teja Juttu

ISYE 6740 Summer 2021
Homework 1 (100 points + 2 bonus points)

1 Image compression using clustering [60 points]

In this programming assignment, you are going to apply clustering algorithms for image compression. Your task is implementing *K-means* for this purpose. **It is required you implementing the algorithms yourself rather than calling k-means from a package. However, it is ok to use standard packages such as file i/o, linear algebra, and visualization.**

Formatting instruction

Input

- `pixels`: the input image representation. Each row contains one data point (pixel). For image dataset, it contains 3 columns, each column corresponding to Red, Green, and Blue component. Each component has an integer value between 0 and 255.
- `k`: the number of desired clusters. Too high value of K may result in empty cluster error. Then, you need to reduce it.

Output

- `class`: cluster assignment of each data point in pixels. The assignment should be 1, 2, 3, etc. For $k = 5$, for example, each cell of class should be either 1, 2, 3, 4, or 5. The output should be a column vector with `size(pixels, 1)` elements.
- `centroid`: location of k centroids (or representatives) in your result. With images, each centroid corresponds to the representative color of each cluster. The output should be a matrix with K rows and 3 columns. The range of values should be [0, 255], possibly floating point numbers.

Hand-in

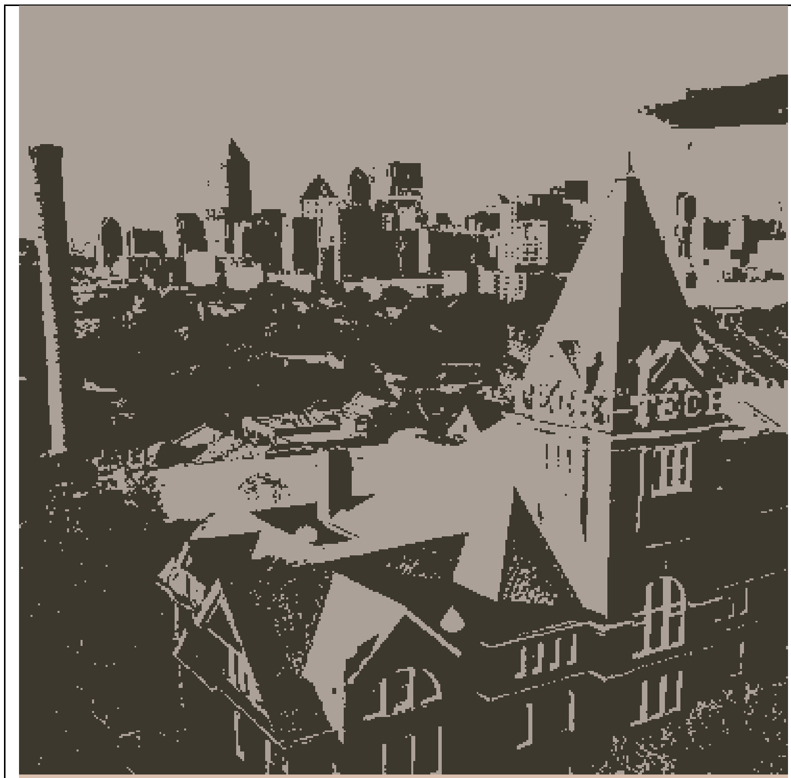
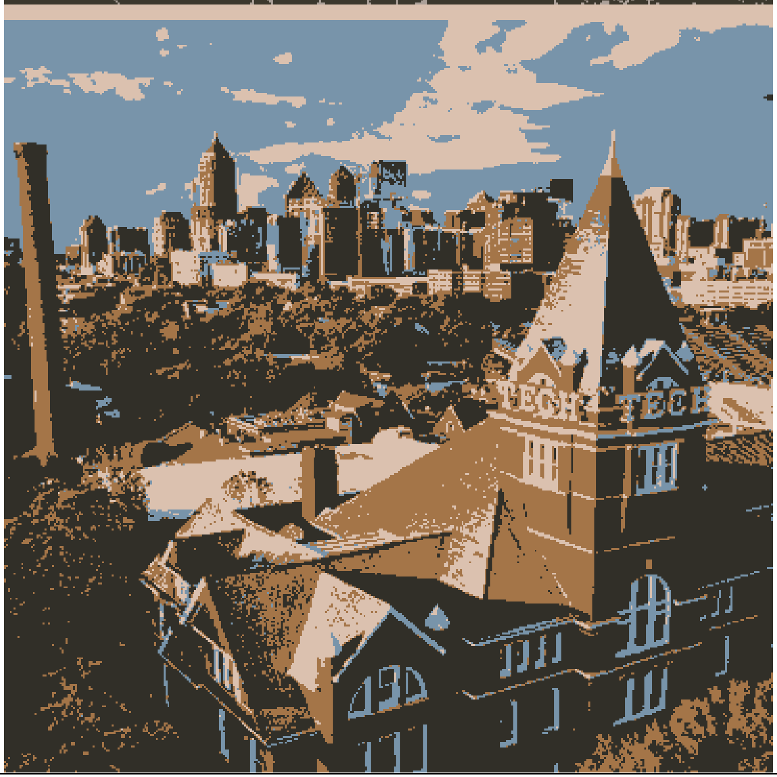
Both of your code and report will be evaluated. Upload them together as a zip file. In your report, answer to the following questions:

1. (20 points) Use k-means with squared-ℓ_2 norm as a metric, for `GeorgiaTech.bmp` and `football.bmp` and also choose a third picture of your own to work on. We recommend size of 320×240 or smaller.Run your k-means implementation with these pictures, with several different $k = 2, 4, 8, 16$. How long does it take to converge for each k (report the number of iterations, as well as actual running time)?Please write in your report, and also include the resulted compressed pictures for each k.

Image1 = GeorgiaTech.bmp			Image2 = football.bmp			Image3 = earth.jpeg		
clusters	Time	Iterations	clusters	Time	Iterations	clusters	Time	Iterations
0.0	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0
2.0	0.1247	5.0	2.0	0.1757	13.0	2.0	0.0424	8.0
4.0	0.1502	13.0	4.0	0.5495	41.0	4.0	0.1535	31.0
8.0	0.5072	32.0	8.0	0.8348	44.0	8.0	0.2957	59.0
16.0	0.9143	45.0	16.0	2.2322	76.0	16.0	0.5484	75.0
Below are the results of the compressed images generated with seed 400. The images are ordered in ascending from								

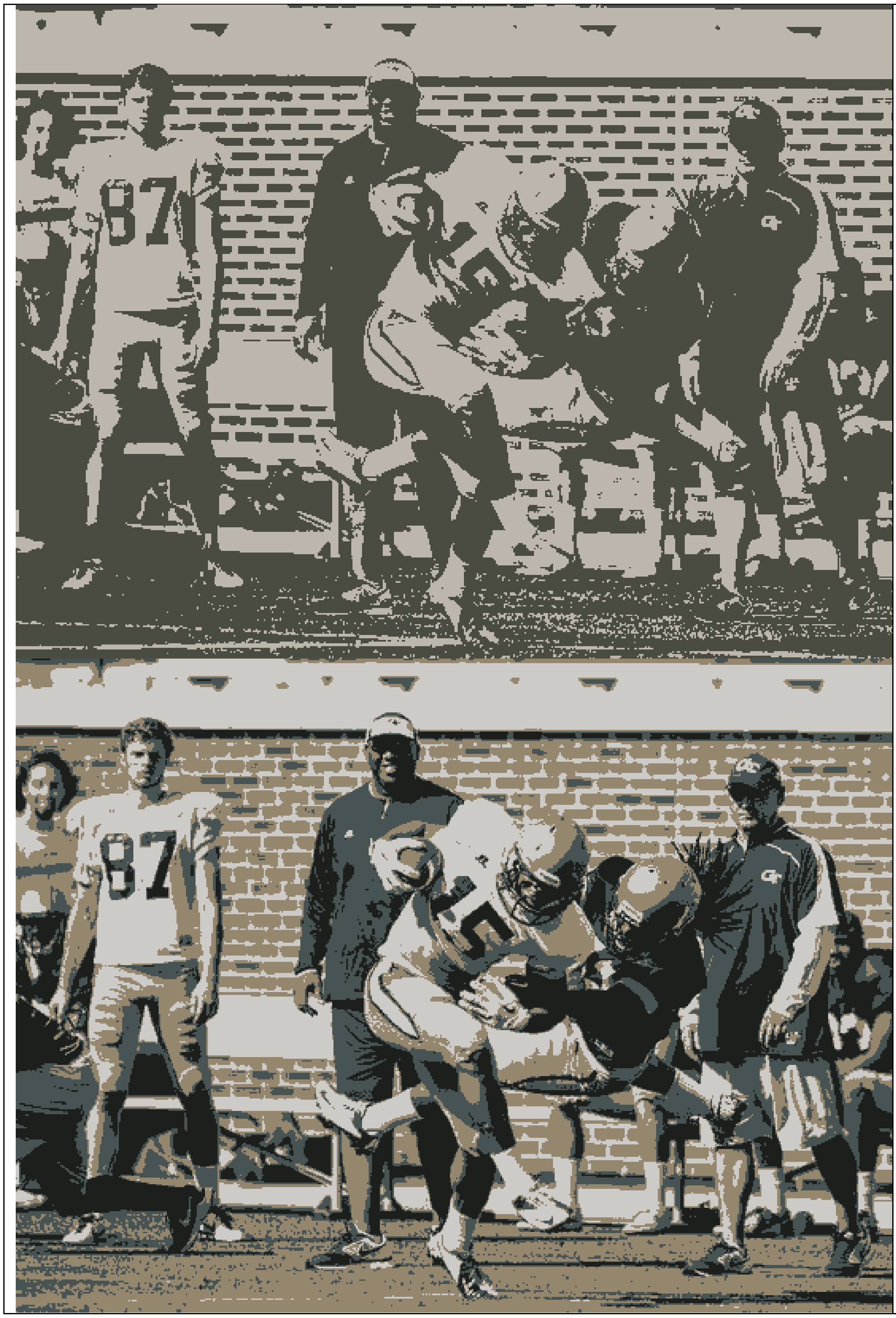
Sj

compressions with 2,4,8 and 16 clusters.

 		Image 1 = GeorgiaTech.bmp

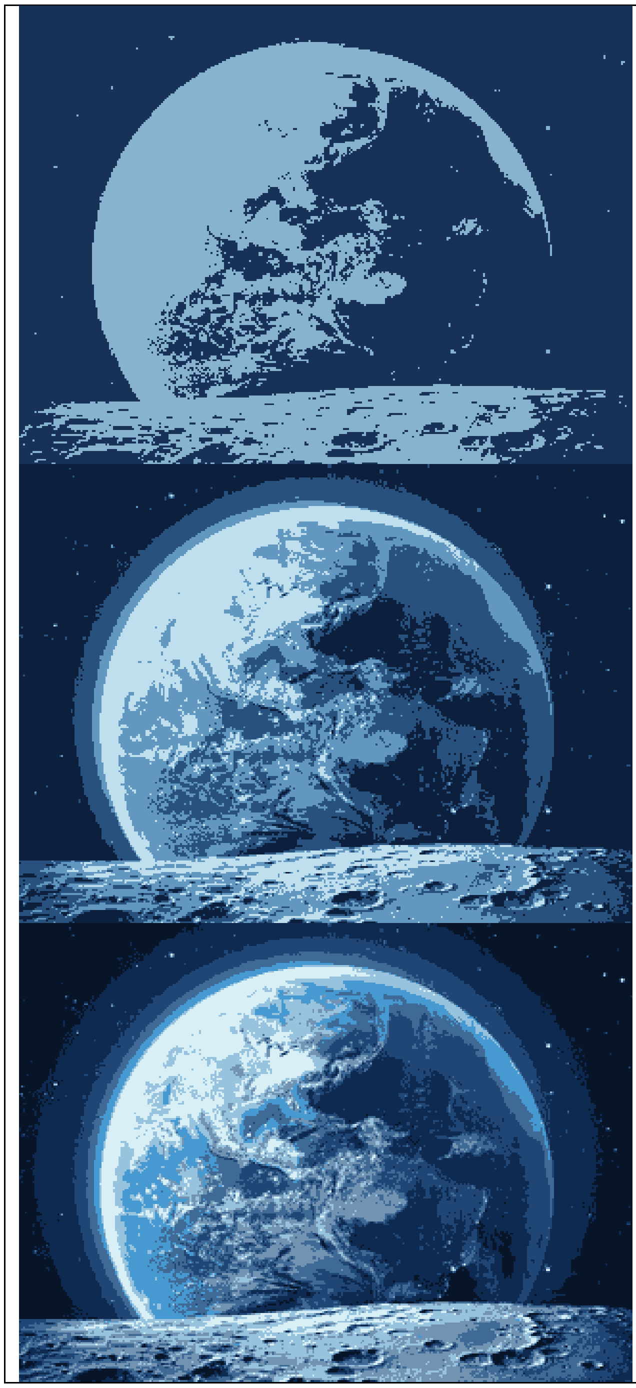
Sj

TECH TECH TECH TECH	

	Image2 = football.bmp

Sj

Sj

		Image3 = earth.jpeg

Sj

2. (20 points) Run your k-means implementation (with squared-ℓ_2 norm) with different initialization centroids. Please test two initialization strategies, compare the results (output image, running time, iterations) and report: (i) random initialization. Please try multiple time and report the best one (in terms of the image quality). (ii) poor initialization. Please design your own strategy, explain why it qualifies as a poor initialization, try multiple times, and report the results.

   How does this it affect your final result? (We usually randomize initial location of centroids in general.) Please also explain in the report how you initialize the centroid.

<u>Initializing centroid:</u>

1. Random Initialization: Assignment from existing pixels Methodology: The initial k centroids are taken randomly from the existing pixels of the image.	2. Poor Initialization: Random assignment from RGB space Methodology: The K centroids are initialized randomly from the 256*256*256 RGB space
<results> runtime and iterations Image1 = GeorgiaTech.bmp clusters Time Iterations 0.0 0.0000 0.0 2.0 0.0947 11.0 4.0 0.3419 41.0 8.0 0.9597 83.0 16.0 1.8332 104.0 Image2 = football.bmp clusters Time Iterations 0.0 0.0000 0.0 2.0 0.2672 23.0 4.0 0.3534 26.0 8.0 0.9063 50.0 16.0 4.4130 148.0 Image3 = earth.jpeg clusters Time Iterations 0.0 0.0000 0.0 2.0 0.0547 13.0 4.0 0.1893 35.0 8.0 0.3167 70.0 16.0 1.1015 153.0	<results> runtime and iterations Image1 = GeorgiaTech.bmp clusters Time Iterations 0.0 0.0000 0.0 2.0 0.0871 9.0 4.0 0.4968 62.0 8.0 1.0757 86.0 16.0 3.5843 184.0 Image2 = football.bmp clusters Time Iterations 0.0 0.0000 0.0 2.0 0.2539 20.0 4.0 0.4703 30.0 8.0 1.0793 52.0 16.0 6.7112 242.0 Image3 = earth.jpeg clusters Time Iterations 0.0 0.0000 0.0 2.0 0.0504 12.0 4.0 0.1591 40.0 8.0 0.9046 193.0 16.0 0.5613 73.0
The above results are generated with random.seed(10). I noticed that, in several of my iterations, when the cluster centers are initialized randomly from the RGB space, it takes more iterations and more time for the kmeans to converge when compared to random initialization. But sometimes the random assignments are getting lucky to land on good spots but very rarely. So all in all, I would commend the random initialization compared to poor initialization.	
All the images for these results are attached in the zip file submitted. But in short, the images seemed to be equally good. I did not find any difference in the quality of images between either of the initialization methods. The difference I could observe is only from the computation cost. But I can see how sometimes it might encounter a local optima and give not the best solutions.	

3. (20 points) Now try your k-means with the Manhattan distance (or ℓ_1 distance) and repeat the same steps in Part (1). Please note that the assignment of data point should be based on the Manhattan distance, and the cluster centroid (by minimizing the sum of deviance – as a result of using the Manhattan distance) will be taken as the "median" of each cluster. Comment on the difference of image compression results using the two methods.

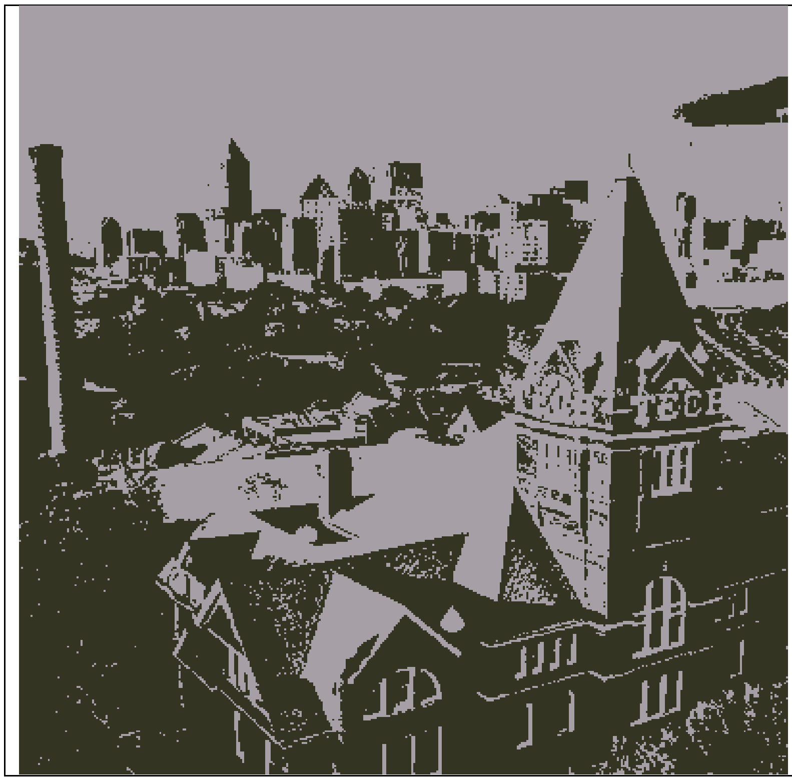
Image1 = GeorgiaTech.bmp			Image2 = football.bmp			Image3 = earth.jpeg		
clusters	Time	Iterations	clusters	Time	Iterations	clusters	Time	Iterations
0.0	0.0000	0.0	0.0	0.0000	0.0	0.0	0.0000	0.0
2.0	0.0708	4.0	2.0	0.2871	18.0	2.0	0.0430	7.0
4.0	0.3236	22.0	4.0	0.2524	11.0	4.0	0.0851	12.0
8.0	0.9437	56.0	8.0	1.1715	46.0	8.0	0.1369	13.0
16.0	1.2799	51.0	16.0	0.9387	21.0	16.0	0.3719	25.0

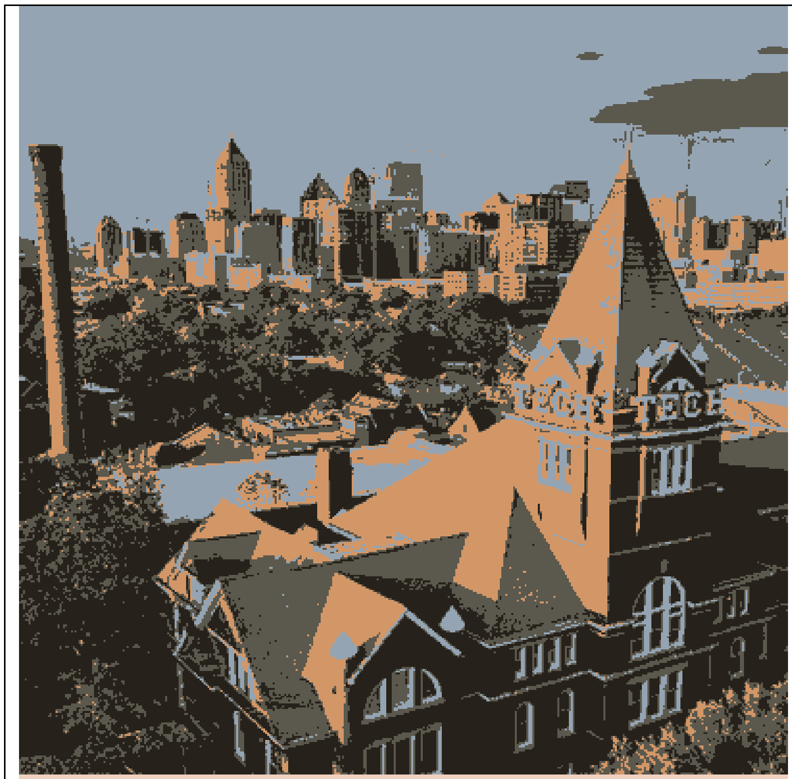
Below are the results of the compressed images generated with random.seed(100). The images are ordered in ascending from compressions with 2,4,8 and 16 clusters.

I observe the difference in compressions at 2 levels:

1. Computing speed/cost: Clearly from the time taken and the number of iterations stated for part1 v part 3, I see the k-means compression using manhattan distance is much faster than Euclidian distance based compression. It is computationally less expensive and faster
2. Quality of the image: The quality of the image howerever I feel is slightly compromised. Although not very significant, I see the images generated in part 1 are slightly more colorful and has more character than the part 3 compression images.

So depending on use case, Choose either one or the other.

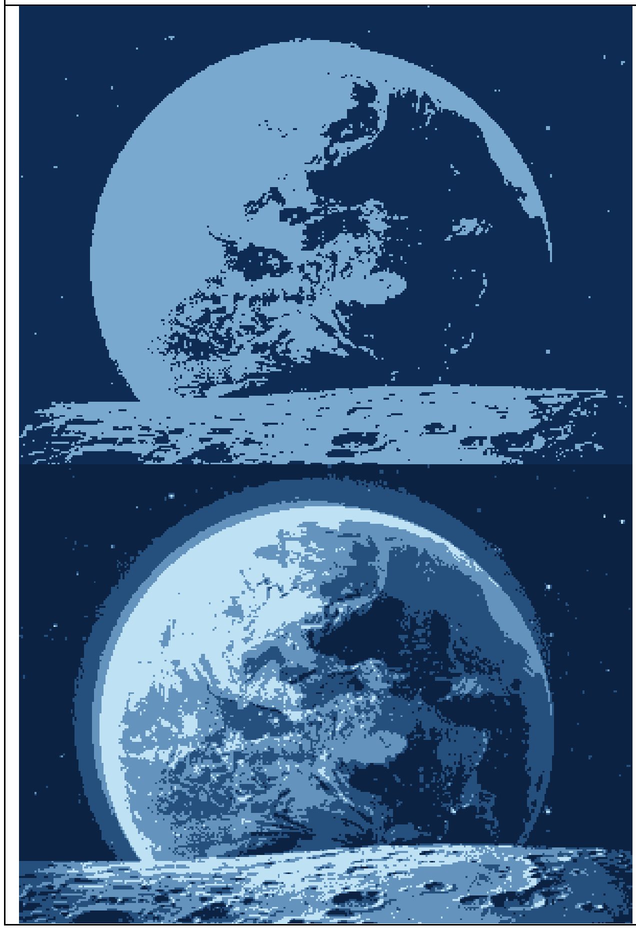
	Image 1 = GeorgiaTech.bmp

Sj

	Image 2 = football.bmp

87 15 87 15	

	Image 2 = earth.jpeg

Note

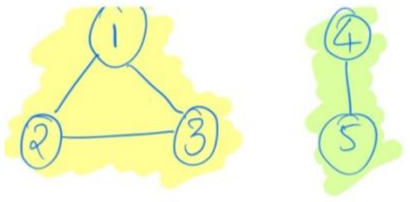
- You may see some error message about empty clusters when you use too large k. Your implementation should treat this exception as well. That is, do not terminate even if you have an empty cluster, but use smaller number of clusters in that case.
- We recommend you to test your code with several different pictures so that you can detect some problems that might happen occasionally.
- If we detect copy from any other student's code or from the web, you will not be eligible for any credit for the entire homework, not just for the programming part. Also, directly calling built-in functionsor from other package functions is not allowed.

Exception Handling:

Problems encountered	**Correction added**
Sometimes I found that after a couple of iterations from initialization, during the k-centroids updating loop, some clusters are returned empty. While I might expect this to happen, if the chosen centroid is far away from any of the pixels in the color space. This empty cluster caused problems when averaging the centroids. Specifically in L2 morn updating.	If a cluster is empty, I assigned updated the centroid to be a black pixel RGB000 to avoid nan errors in taking the mean of the cluster pixels and this avoided generating a blank image sometimes.
Using numpy element wise distance calculation was very expensive computationally. It was taking well above 1 min for 5 iterations.	Used vectorized methods to do the distance calculations and updating of the centroids. This made the process about 100 times faster. Now I can do 100 iterations in 1 second. I borrowed this idea from the piazza posts and demo code.

2 Spectral clustering and discover football colleague [40 points + 2 bonus]

First consider the following simple graph



1. (10 points) Write down the graph Laplacian matrix and find the eigenvectors associated with the zero eigenvalue. Explain how do you find out the number of disconnected clusters in graph and identify these disconnected clusters using these eigenvectors.

Adjacency Matrix

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

Captures Connectivity

Degree Matrix

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

Laplacian Matrix

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

[aka Graph Laplacian]

Eigen Vectors associated with 0 eigen value

$\downarrow$

$L \times V = 0.V$

Solving them will result the following eigen vector $\rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.71 \\ 0.71 \end{bmatrix}$

Nodes ①-②-③ & Nodes ④-⑤ are totally disconnected as referred by these unconnected blocks in the Laplacian Matrix

$$\begin{array}{c|ccccc} \text{nodes} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & -1 & -1 & 0 & 0 \\ 2 & -1 & 1 & -1 & 0 & 0 \\ 3 & -1 & -1 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & -1 \\ 5 & 0 & 0 & 0 & -1 & 1 \end{array}$$

All Eigen vectors of the Laplacian (L) are

$\downarrow$

$$\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \\ 0.58 & -0.82 & 0.43 & 0 & 0 \\ 0.58 & 0.41 & -0.82 & 0 & 0 \\ 0.58 & 0.41 & 0.39 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \\ 0 & 0 & 0 & -0.71 & 0.71 \\ {[-1} & 2 & 2 & 2 & 0] \end{array}$$

$\Leftarrow$ Eigen Values

v_5: $\left.\begin{matrix} 0 \\ 0 \\ 0 \end{matrix}\right\} 0$ $\left.\begin{matrix} 0.71 \\ 0.71 \end{matrix}\right\} 0.71$

There are 2 unique groups in row wise 0-eigen vectors of laplacian implies 2 connected components or 2 clusters

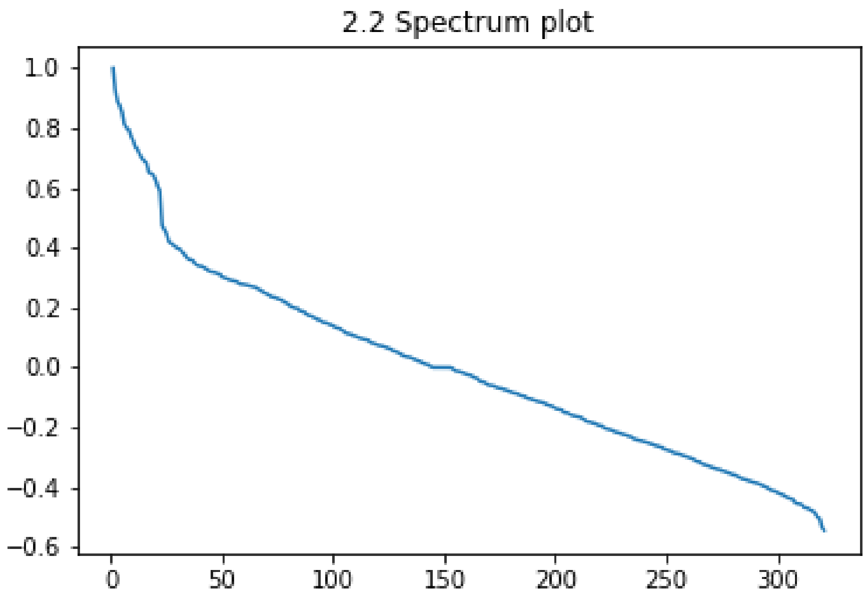
Also, these two clusters are disconnectes as per
a) Spectral theory
b) The elemental values of the eigen vectors are '0' if the other groups elements are non-zero

Implying there cannot exist any connectivity b/w these 2 Clusters

Note: To calculate the eigen vectors from Laplacian, I used python.

Now consider the football league example in the “spectral clustering” lecture. Use the data providedthere (in demo, play graph.txt, nodes.csv and edges.csv) for this question. Implement the spectral clusteringalgorithm yourself (you can borrow the idea from demo code, in particular, regarding importing data, readdata etc.)

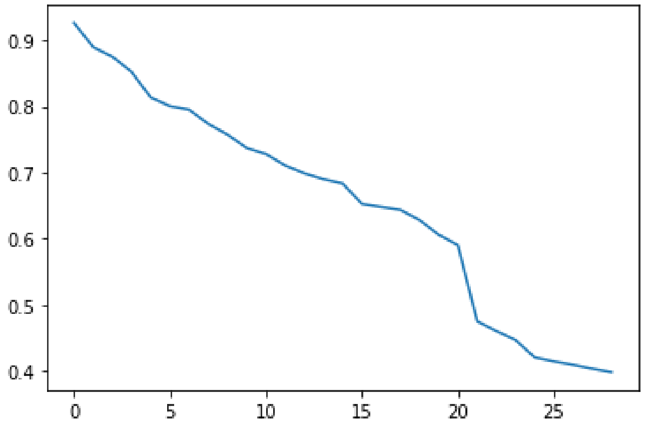
2. (10 points) For the graph Laplacian matrix and perform eigen decomposition on it. Plot the eigenvalues (ranked from the largest to the smallest), and based on the plot explain approximately how many clusters you believe there are and why.



The paper A Tutorial on Spectral Clustering — Ulrike von Luxburg proposes an approach based on **perturbation theory** and **spectral graph theory** to calculate the optimal number of clusters.

Eigengap heuristic suggests the number of clusters k is usually given by the value of k that maximizes the eigengap (difference between consecutive eigenvalues). The larger this eigengap is, the closer the eigenvectors of the ideal case and hence the better spectral clustering works.

In our case, the max Eigen gap seems to be occurring around **20.** So I believe there should be around 20 clusters in our football data. Refer the zoomed image of the spectrum plot.



Reference: I have adopted the Laplacian variant of **symmetric normalized Laplacian** as mentioned in the demo code throughout my analysis.

Sj

3. (15 points) Now perform spectral clustering, using k = 5, k = 7, k = 10, and your choice of k (based on your answer in part (1). Report the size of the largest cluster and the smallest cluster based on your result for each k. Report the results for k = 10 by listing the teams and which cluster they belongto (you can either include a table in your solution, or upload a file or spreadsheet that include your result).

Largest cluster size chosen = 20 (As chosen above from the eigen gap). The detailed team wise clustering for k=10 and k=20 is attached in the excel "Clusters2.3"

The clustering results for k= [5,7,10,20] are as below. Largest cluster is highlighted in green. Smallest in red

5 Clusters	**1**	**2**	**3**	**4**	**5**
Number of teams	106	66	36	61	52

7 Clusters	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Number of teams	37	38	28	104	45	28	41

10 Clusters	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Number of teams	24	28	62	27	27	26	18	39	30	40

20 Clusters	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Number of teams	12	24	22	14	19	21	15	18	23	18	14	15	16	14	13	12	13	14	10	14

4. (5 points) Now run the algorithm a few times for k = 10. You may notice the results are slightly different - please explain why. Also check which clusters that "Georgia Tech", "Georgia State", and "Georgia" (UGA) are in. Are they always in the same cluster or not - please explain your reasoning.

Number of teams in each of the 10 Clusters

Iteration 1 - [18 24 26 27 27 27 31 39 41 61]

Iteration 2 - [18 18 24 27 27 28 30 40 45 64]

Iteration 3 - [18 24 27 27 27 29 31 38 45 55]

Iteration 4 - [18 24 26 27 28 30 34 39 41 54]

Iteration 5 - [18 24 26 27 27 28 30 39 39 63]

.

.

Yes. Here in each iteration we see the results to be slightly different from each other. This difference is stemming from the Kmeans clustering used to cluster/group the row vectors. We know that kmeans clustering is based on heuristics of random initialization of the centroids. So, the convergences may be slightly different from different initializations.

In my multiple iterations, All three Georgia Tech, Georgia State and Georgia are found in the same cluster. This should be because of the way the teams are connected in the edges. Meaning these 3 Georgia teams had more games with each other. This makes sense because of the geographic vicinity, teams might have had more games – hence more connected and hence in the same cluster always.

5. (Bonus, 2 points.) Please report what else you can discover from this data analysis and results. Try to be creative.

   The number of teams in each cluster is evenly distributed when the clusters are large. With small number of clusters, the data points are heavily skewed and sometimes may not be of interpretable or desired combinations.

   The use of eigen gap is an excellent way to cluster. At the max drop value of k=20, the clustering is really really good with almost evenly distributed data points.

References:

- Demo codes (majorly)
- Normalised cuts https://people.eecs.berkeley.edu/~malik/papers/SM-ncut.pdf
- https://en.wikipedia.org/wiki/Laplacian_matrix
- https://towardsdatascience.com/spectral-graph-clustering-and-optimal-number-of-clusters-estimation-32704189afbe