# Homework 1

## 1. Concept Questions

**1. What's the main difference between supervised and unsupervised learning?**

Supervised learning uses labeled datasets to train models/algorithms in order to classify data or predict outcomes. Unsupervised learning uses unlabeled data sets and allows machine learning algorithms to find patterns and information within the data. Results from supervised learning is shaped by the labelled input while results from unsupervised learning are less predictable.

**2. Will different initializations for k-means lead to different results?**

Yes. Minimizing the average square distances from each data point to its respective cluster is a convex optimization problem that may have several local minima. Choosing different initializations may lead to different local minima.

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

K-means will always converge in a finite number of iterations because there is a finite number of ways a data set can be segmented into k different clusters. The number may be extremely large but it is still finite.

**4. What is the main difference between k-means and generalized k-means algorithm? Explain how the choice of the similarity/dissimilarity/distance will impact the result.**

The main difference between k-means and generalized k-means algorithm is in the way the generalized algorithm adjusts the cluster centers. Instead of adjusting the center to be cluster's mean like in k-means, the generalized approach instead solves an optimization problem to find the next centroid.

**5. Consider the following simple graph. 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.**

First, I calculate A, D and L.

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad L = D - A = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

To find out the number of disconnected clusters in the graph, I used the python code in the cell below to perform eigenvalue decomposition. Then I found the columns in the eigenvector matrix that correspond to a 0 eigenvalue. To identify the disconnected clusters, I take the indexes of the 0 eigenvalues and find the corresponding column in the eigenvector matrix. Based on the resulting matrix, I can conclude that nodes 1, 2 and 3 are connected in one cluster while nodes 4 and 5 are connected in another.

In [1]:

```python
import numpy as np

# Set up matrices
A = np.array([[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]])
D = np.array([[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]])
L = D-A

# Eigenvalue decomposition
s, v = np.linalg.eig(L)

# Get index of eigenvalues that are 0
zero_evalue_index =  [i for i, x in enumerate(np.around(s,1)) if x == 0]
print("Eigenvalues: " + str(np.around(s,1)))
print("Indexes of 0 eigenvalues: " + str(zero_evalue_index))

# Columns corresponding to the eigenvalues that are 0
print("Eigenvectors: ")
print(str(v))
print("Cluster Assignment:")
print(str(v[:, zero_evalue_index]))
```

```
Eigenvalues: [ 3. -0.  3.  2.  0.]
Indexes of 0 eigenvalues: [1, 4]
Eigenvectors:
[[ 0.81649658 -0.57735027  0.30959441  0.          0.        ]
 [-0.40824829 -0.57735027 -0.80910101  0.          0.        ]
 [-0.40824829 -0.57735027  0.49950661  0.          0.        ]
 [ 0.          0.          0.          0.70710678  0.70710678]
 [ 0.          0.          0.         -0.70710678  0.70710678]]
Cluster Assignment:
[[-0.57735027  0.        ]
 [-0.57735027  0.        ]
 [-0.57735027  0.        ]
 [ 0.          0.70710678]
 [ 0.          0.70710678]]
```

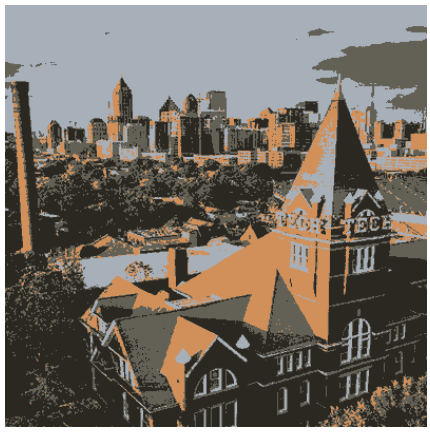
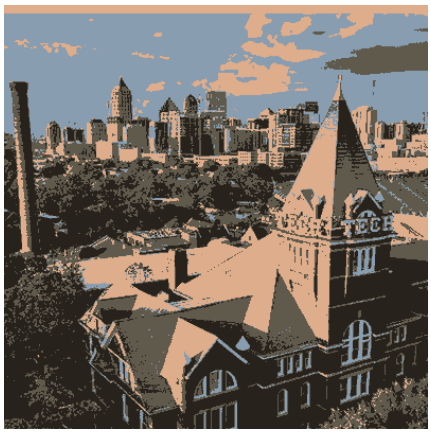
# 2. Image Compression Using Clustering

### 1. Georgia Tech

| k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|
| 2 | 5 | 72 | 5 | 59 |
| 4 | 13 | 230 | 13 | 298 |
| 8 | 35 | 1018 | 68 | 2432 |
| 16 | 33 | 2054 | 50 | 3656 |

Image Comparison:

| k | Original | L2 | L1 |
|---|---|---|---|

| k | Original | L2 | L1 |
|---|---|---|---|
| 2 |  |  |  |
| 4 |  |  |  |
| 8 |  |  |  |
| 16 |  |  |  |

## 2. Football

| k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|
| 2 | 10 | 169 | 10 | 204 |

| k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|
| 4 | 13 | 431 | 10 | 351 |
| 8 | 20 | 1125 | 19 | 1354 |
| 16 | 33 | 3114 | 33 | 4259 |

Image Comparison:

| k | Original | L2 | L1 |
|---|---|---|---|
| 2 |  |  |  |
| 4 |  |  |  |
| 8 |  |  |  |
| 16 |  |  |  |

### 3. Beach

| k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|
| 2 | 5 | 38 | 6 | 56 |
| 4 | 14 | 168 | 8 | 147 |
| 8 | 11 | 205 | 10 | 259 |
| 16 | 30 | 905 | 23 | 1021 |

Image Comparison:

| k | Original | L2 | L1 |
|---|---|---|---|

| k | Original | L2 | L1 |
|---|---|---|---|
| 2 |  |  |  |
| 4 |  |  |  |
| 8 |  |  |  |
| 16 |  |  |  |

### Comparing L1 and L2 Norm for Compression

There were several differences I found when using the Euclidean vs Manhattan distance for image compression:

1. Number of Iterations & Run Time
   Across all values of k, there were several instances where the run time using L1 norm was longer even though compression was completed in the same or fewer number of iterations. In general, using the L2 norm to compress images was faster and completed in fewer iterations.

Examples of where the L1 norm had a longer run time in the same or fewer iterations compared to L2:

| Image | k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|---|
| Georgia Tech | 4 | 13 | 230 | 13 | 298 |
| football | 2 | 10 | 169 | 10 | 204 |

| Image | k | (L2) Iterations | (L2) Run Time (Seconds) | (L1) Iterations | (L1) Run Time (Seconds) |
|---|---|---|---|---|---|
| football | 8 | 20 | 1125 | 19 | 1354 |
| football | 16 | 33 | 3114 | 33 | 4259 |
| beach | 8 | 11 | 205 | 10 | 259 |
| beach | 16 | 30 | 905 | 23 | 1021 |

2. Image Quality & Colours
   In lower values of k (2,4,8), the colours chosen by the L1 norm are more cool toned, less vibrant and have less contrast than the colours chosen by L2. With the highest value of k, the images look more comparable. However, looking very closely at the Georgia Tech and Beach images, it appears that the L1 norm has more trouble distinguishing shades of blue.

# 3. Political Blogs Dataset

## Q1 Code Output

k = 2
Majority: {0: 1, 1: 0}
Mismatch Rates: {0: 0.449, 1: 0.4636}
Average of Mismatch Rates = 0.45630000000000004
Spread of Mismatch Rates = (0.449, 0.4636) = 0.014600000000000002

k = 5
Majority: {0: 1, 1: 1, 2: 0, 3: 0, 4: 1}
Mismatch Rates: {0: 0.0656, 1: 0.1311, 2: 0.0904, 3: 0.075, 4: 0.0175}
Average of Mismatch Rates = 0.07592
Spread of Mismatch Rates = (0.0175, 0.1311) = 0.11359999999999999

k = 10
Majority: {0: 1, 1: 0, 2: 1, 3: 1, 4: 1, 5: 0, 6: 1, 7: 1, 8: 0, 9: 0}
Mismatch Rates: {0: 0.0275, 1: 0.033, 2: 0.1711, 3: 0.3111, 4: 0.3889, 5: 0.0702, 6: 0.1235, 7: 0.0806, 8: 0.0827, 9: 0.0659}
Average of Mismatch Rates = 0.13545000000000001
Spread of Mismatch Rates = (0.0275, 0.3889) = 0.3614

k = 20
Majority: {0: 0, 1: 1, 2: 0, 3: 1, 4: 0, 5: 1, 6: 0, 7: 0, 8: 0, 9: 1, 10: 1, 11: 0, 12: 1, 13: 0, 14: 1, 15: 0, 16: 1, 17: 0, 18: 1, 19: 0}
Mismatch Rates: {0: 0.2083, 1: 0.0943, 2: 0.0536, 3: 0.027, 4: 0.0588, 5: 0.1299, 6: 0.0952, 7: 0.4286, 8: 0.2909, 9: 0.025, 10: 0.2, 11: 0.1091, 12: 0.0, 13: 0.0982, 14: 0.0682, 15: 0.1333, 16: 0.0526, 17: 0.0317, 18: 0.2308, 19: 0.0143}
Average of Mismatch Rates = 0.11749
Spread of Mismatch Rates = (0.0, 0.4286) = 0.4286

## Q2

**Tune your k and find the number of clusters to achieve a reasonably small mismatch rate. Please explain how you tune k and what is the achieved mismatch rate. Please explain intuitively what this results tells about the network community structure.**

As the number of clusters gets larger, the spread (max-min) of the cluster mismatch rates gets wider and the average of the mismatch rates also increases. This tells me that breaking up the blog site list into 10 to 20 communities is too many.

In order to tune k, I will test breaking up the list of blog sites into 2 to 9 communities and compare metrics. Based on the results below, the size of community that has the lowest average mismatch rate and a small spread mismatch rate is 5. I didn't choose k = 4 (it had the smallest spread) because the minimum mismatch rate is more than 2x the minimum mismatch rate of k = 5.

These results tell me that in the remaining 1224 political blogs (266 were removed because they weren't connected to any other sites), there are 5 communities where the connected blog sites share the same political orientation. At most, 13% of sites that belong to a community may not share the same political orientation.

The achieved mismatch rates are as follows:

- Cluster 0 = 0.1311
- Cluster 1 = 0.0171
- Cluster 2 = 0.0775
- Cluster 3 = 0.0856
- Cluster 4 = 0.1159

**Code Output**

k = 2
Majority: {0: 1, 1: 0}
Mismatch Rates: {0: 0.449, 1: 0.4636}
Average of Mismatch Rates = 0.45630000000000004
Spread of Mismatch Rates = (0.449, 0.4636) = 0.014600000000000002

k = 3
Majority: {0: 0, 1: 0, 2: 1}
Mismatch Rates: {0: 0.45, 1: 0.0912, 2: 0.0624}
Average of Mismatch Rates = 0.2012
Spread of Mismatch Rates = (0.0624, 0.45) = 0.3876

k = 4
Majority: {0: 1, 1: 1, 2: 0, 3: 0}
Mismatch Rates: {0: 0.135, 1: 0.0382, 2: 0.0921, 3: 0.1121}
Average of Mismatch Rates = 0.09435
Spread of Mismatch Rates = (0.0382, 0.135) = 0.09680000000000001

k = 5
Majority: {0: 1, 1: 1, 2: 0, 3: 0, 4: 1}
Mismatch Rates: {0: 0.1311, 1: 0.0171, 2: 0.0775, 3: 0.0856, 4: 0.1159}
Average of Mismatch Rates = 0.08544
Spread of Mismatch Rates = (0.0171, 0.1311) = 0.11399999999999999

k = 6
Majority: {0: 1, 1: 0, 2: 1, 3: 0, 4: 1, 5: 1}
Mismatch Rates: {0: 0.1169, 1: 0.0973, 2: 0.0221, 3: 0.0535, 4: 0.1408, 5: 0.1489}
Average of Mismatch Rates = 0.09658333333333334
Spread of Mismatch Rates = (0.0221, 0.1489) = 0.1268

k = 7
Majority: {0: 1, 1: 0, 2: 0, 3: 1, 4: 1, 5: 1, 6: 1}
Mismatch Rates: {0: 0.1261, 1: 0.0232, 2: 0.0536, 3: 0.0214, 4: 0.145, 5: 0.3143, 6: 0.1357}
Average of Mismatch Rates = 0.11704285714285714
Spread of Mismatch Rates = (0.0214, 0.3143) = 0.29290000000000005

k = 8
Majority: {0: 1, 1: 1, 2: 0, 3: 1, 4: 1, 5: 0, 6: 1, 7: 0}
Mismatch Rates: {0: 0.1858, 1: 0.0697, 2: 0.0376, 3: 0.2941, 4: 0.0219, 5: 0.1382, 6: 0.1185, 7: 0.1685}
Average of Mismatch Rates = 0.1292875
Spread of Mismatch Rates = (0.0219, 0.2941) = 0.2722

k = 9
Majority: {0: 1, 1: 1, 2: 1, 3: 1, 4: 0, 5: 0, 6: 0, 7: 0, 8: 1}
Mismatch Rates: {0: 0.2551, 1: 0.1495, 2: 0.0855, 3: 0.0248, 4: 0.0741, 5: 0.0403, 6: 0.1064, 7: 0.0887, 8: 0.0595}
Average of Mismatch Rates = 0.09821111111111111
Spread of Mismatch Rates = (0.0248, 0.2551) = 0.2303