

# Homework 1

## 1.) Concept Questions

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

The main difference between supervised and unsupervised learning is having the labels of the training data prior to modeling. In supervised learning, you would be classifying a test data set after training on data that was already labeled or classified. Unsupervised learning, such as k-means, would be learning the clusters or classifications from unlabeled data. Unsupervised learning also can call for adding subjectivity or domain knowledge into the analysis to make sense of the output.

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

Yes, due to the fact that it is a non-convex optimization problem. This could lead to multiple local solutions and we would only end up in one of them. Depending on the starting point, if there are multiple local solutions you could end up at different solutions based on initialization. This also means that it is not guaranteed to find the global minimal optimization solutions.

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

Since there are a fixed number of data points, there are only a finite number of combinations to calculate the centroids. The algorithm will try to reduce a certain metric, and you can continue to decrease until it is not able to decrease any further. This could take many iterations, but eventually it will always converge.

### 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 difference everything to do with the similarity measure. K-means uses Euclidean distance, where the generalized k-means algorithm can use any similarity measure. Replacing the Euclidean distance with a general definition can allow it to be a convex optimization problem. This also allows customization for different types of datasets where Euclidean is not the best solution for the objective.

### 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.

```
In [19]: 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
L
```

```
Out[19]: array([[ 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]])
```

```
In [20]: U, V = np.linalg.eig(L)
```

#### Eigenvectors

```
In [21]: v
```

```
Out[21]: array([[ 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]])
```

#### Eigenvalues

```
In [22]: u
```

```
Out[22]: array([ 3.00000000e+00, -3.77809194e-16,  3.00000000e+00,  2.00000000e+00,
                0.00000000e+00])
```

#### Identifying the clusters

To find the clusters from the eigenvectors, the eigenvectors with eigenvalue 0 contain cluster assignment information. The rows that have similar eigenvectors are from similar communities, and clustered together.

## 2.) Image compression using clustering

### 1.) Euclidean

Utilizing the Kmeans cluster algorithm, I was able to create image compressions by clustering the individual pixels of the image. Each image begins with randomized centroids based on the  $k$  clusters passed to the method. It then iterates assigning each row to the closest centroid based on the  $L2$  distance. The centroids are then updated by taking the average of each cluster group. This continues until convergence where the centroids no longer change (stopping when the centroids are less than 0.001 in difference than the previous iteration).

For the analysis, each image was ran 5 times per cluster, on  $k = [2, 4, 6, 8, 16]$ .


As the number of clusters increased, the iterations and time it took to converge also increased. For the Football image, lower  $k$  completed in an average of 11 to 18 seconds, where higher  $k$  finished around 40 seconds. The variance was much higher as you increase  $k$ . Depending on the randomized centroids, the solution could converge quickly, or it could take quite a while relative to other runs due to poor starting points. It is possible to select the exact sample pixel (not point, but same pixel elsewhere in the image) and that would cause an error with the algorithm. To handle this, if the number of unique initialized centroids did not equal  $k$ , they were re-initialized until that held true.

The Georgia Tech image had interesting results. For  $k = 2$ , it converged quickly with minimal variance. For  $k = 4$ , it still converged quickly in an average of 22 iterations, but the variance was 133. I don't believe it is a property internal to the image, but it does show that centroid selection to begin can have a large affect on the model run time.

I then ran the algorithm on two other images, of my cat and child. The image of the baby was notable because the variance was much higher for  $k = 8$  than that of  $k = 16$ . They did not seem to be as complex as the provided images, converging much faster than the other 2.

Outputs of the images below. The higher the  $k$ , the more pixelated the image. It depends on the tradeoff you are looking for as to which works best for you. If this was to input thousands of images into a model, it might be better to use a lower  $k$ , and capture the rough outlines.

$k = 6$   football image with 6 clusters and euclidean distance

$k = 16$   georgia tech image with 16 clusters and euclidean distance

$k = 6$   baby with 6 clusters and euclidean distance

$k = 4$   cat with 4 clusters and euclidean distance

### 2.) Manhattan

The next part of the analysis was replicating the algorithm, except this time using the Manhattan distance. This would be taking the  $L1$  norm, where the distance is calculated by taking the sum of length of the  $Y$  axis direction and  $X$  axis direction. Manhattan is the nickname due to it's similarity of a taxi driving through blocks in Manhattan. Centroids were updated using the median values for each iteration.

By using the Manhattan distance, there was an increase in the time it took to converge relative to the Euclidean distance for the Football, Baby, and Cat images. However, the Georgia Tech image converged quicker with Manhattan.

$k = 16$

$k = 4$

## 3.) Political blogs dataset

### 1.)

The data provided contained all possible nodes, as well as connected edges. To perform spectral clustering, a Laplacian matrix needed to be derived from a degree and adjacency matrix. The degree matrix is a diagonal matrix, where the diagonal is the size of the node (number of unique edges connected to it). The rest of the values are 0. Almost the opposite, the adjacency matrix is a binary matrix where a value signifies that there was a connection between the two nodes.

Spectral clustering was done on values of  $k = [2, 5, 10, 20]$ . The assigned value was given based off of the majority in that cluster.

$k = 2$

- cluster 1: 46.7%
- cluster 2: 49%

$k = 5$

- cluster 1: 11.4%
- cluster 2: 1.7%
- cluster 3: 14.7%
- cluster 4: 15%
- cluster 5: 6.5%

$k = 10$

- cluster 1: 7%
- cluster 2: 6.9%
- cluster 3: 13%
- cluster 4: 18.4%
- cluster 5: 13.7%
- cluster 6: 19.5%
- cluster 7: 3.1%
- cluster 8: 1%
- cluster 9: 12.6%
- cluster 10: 3.3%

$k = 20$

- cluster 1: 0%
- cluster 2: 8.3%
- cluster 3: 15%
- cluster 4: 5.1%
- cluster 5: 3.2%
- cluster 6: 13.6%
- cluster 7: 2.5%
- cluster 8: 2.5%
- cluster 9: 1.8%
- cluster 10: 10%
- cluster 11: 10.6%
- cluster 12: 6.1%
- cluster 13: 1.5%
- cluster 14: 31%
- cluster 15: 4.3%
- cluster 16: 12.6%
- cluster 17: 20%
- cluster 18: 12%
- cluster 19: 1.9%
- cluster 20: 50%

### 2.) Tunning $k$ values

To tune for the best  $k$  parameter value, I iterated through the spectral\_graph function on a list of values of  $k$  from 2 to 40, incremented by 2. The metric used was the total observations misclassified divided by the total observations, to create an overall misclassification rate.

When  $k$  is small, the misclassification rate was high. As  $k$  increased, the misclassification rate decreased, until the range of 18 - 22, after that there was a steady increase in the misclassification rate. This is typical in other machine learning models, where the bias and variance tradeoff sees diminishing returns as overfitting takes place.

```
In [ ]:
```