CSE 447: Data Mining

Project 1

Tal Derei

06 March 2022

Simulation Datasets:

Both k-means and spectral clustering represent unsupervised learning algorithm techniques. And the basic goal in any machine learning algorithm is to reduce the "**cost function**". The cost function in k-means involves summing the Euclidean distances from points to their nearest cluster centroid. The cost function in spectral clustering measures the cosine of the angle between cluster points.

In regards to their constructions:

K-Means operates by:

[1] Random initialization of k-centroid values

[2] Cluster points to nearest centroid based on Euclidean distance function

[3] Calculate mean of cluster to compute new centroid value for the cluster

[4] Keep iterating until the center of the clusters converge

Running the K-means algorithm on the ”square” and ”elliptical” datasets yielded expected results for k = 2. The ”elbow” method was used to select the K-value in our clustering by characterizing the relationship between the object function of k-means (SSE) and k values in a algorithmically optimal way.

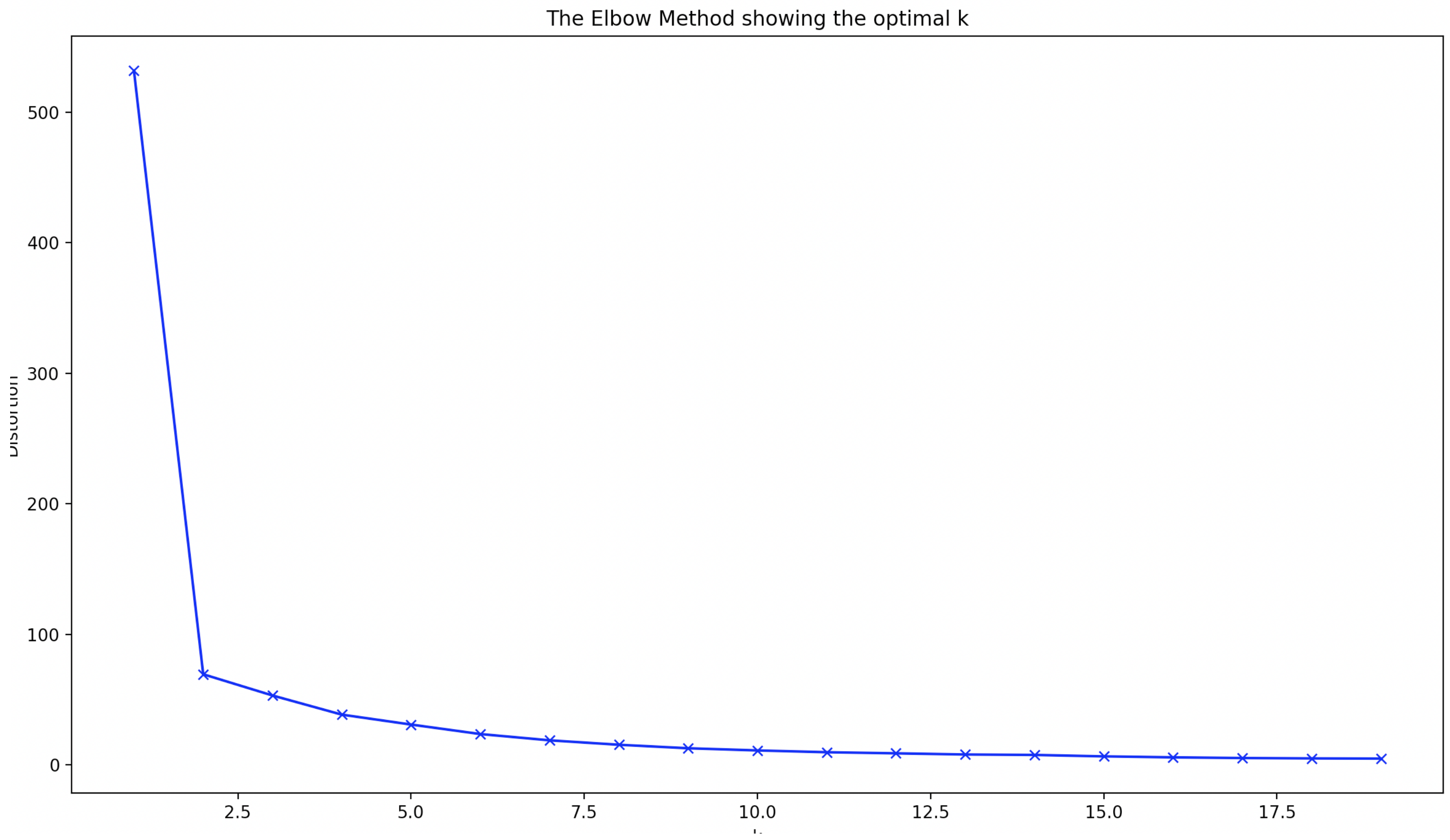


Figure 1: *Elbow Method for Determining K*

After computing the correct number of k-values, the simulated datasets were run through the k-means algorithm and plotted:

Comparing this to <u>Spectral Clustering</u>, I ran through these discrete steps:

[1] Construct similarity matrix matrix using cosine similarity metric

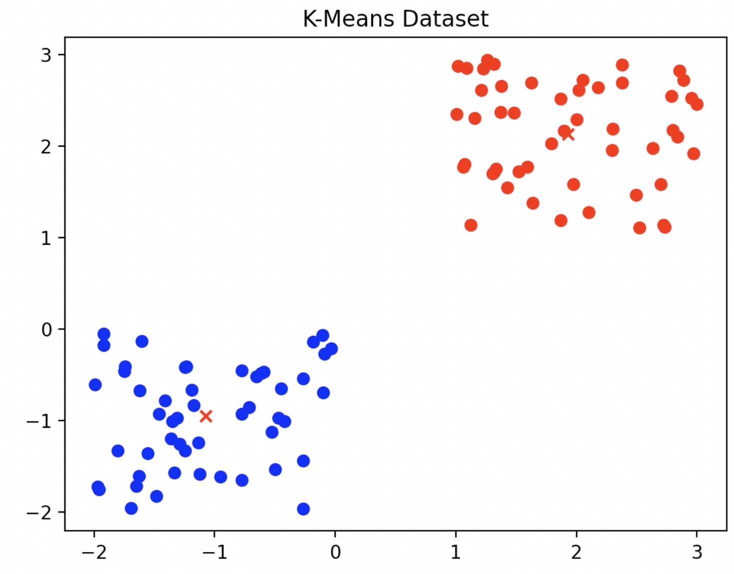


Figure 2: *K-means Algorithm w/ "sqaure" Dataset*

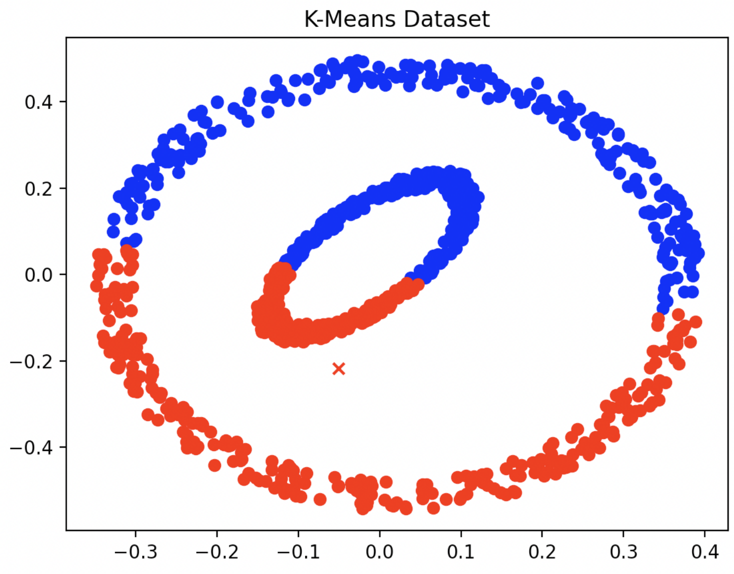


Figure 3: *K-means Algorithm w/ "elliptical" Dataset*

[2] Compute the adjacency matrix by defining some threshold, α, where you assign a 0 or 1 depending on whether the value is below/above the threshold. This adjacency matrix is used for constructing a k-nearest neighbor graph (KNN) classification

[3] Compute the degree matrix by summing all values along the diagnole

[4] Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object

[5] Run k-means on these features (i.e. eigenvectors) to separate objects into k classes. This requires performing a mapping between the eigenvectors and the original dataset. Spectral clustering transforms the original data to a new data representation, the dimension will change from n*d to n*k, so when you get k eigenvectors, it means you find k new features to represent your data

Running the Spectral Clustering on the "square" and "elliptical" datasets yielded expected results for k = 2. The primary change is seen in the "elliptical" dataset, where spectral clustering separates the concentric rings into their own clusters. Running Spectral Clustering involved iteratively ruining the threshold, represented by the hyper-parameter σ. There is no prior knowledge to indicate what values or ranges they should be for different datasets, so I adjusted them based on clustering performance. "square" had a hyperparameter of **1** and "elliptical" had a hyperparameter of **0.15**.

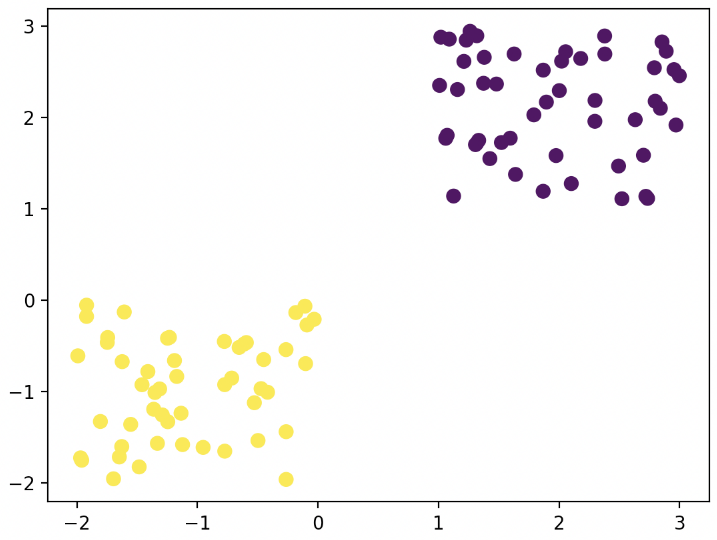


Figure 4: *Spectral Clustering Algorithm w/ "sqaure" Dataset*

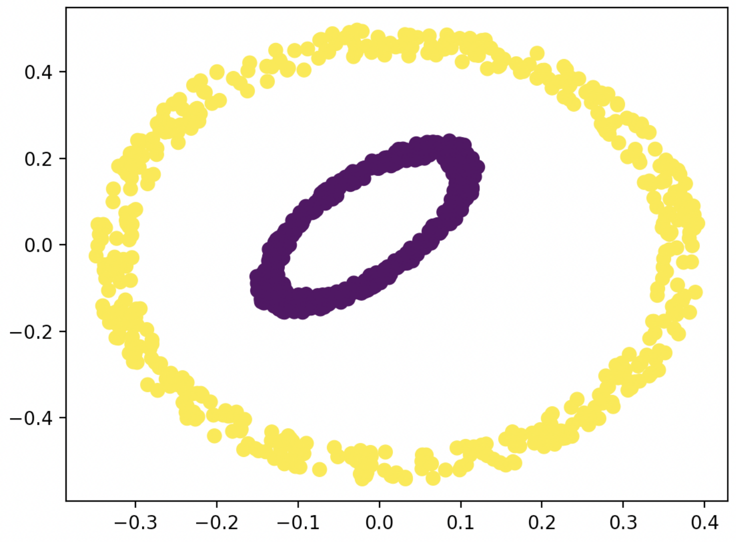


Figure 5: *Spectral Clustering Algorithm w/ "elliptical" Dataset*

Pros and Cons of K-Means and Spectral Clustering:

K-Means:

Pros: [1] Relatively simple to implement, [2] Scales to large data sets, [3] Guarantees convergence, [4] Generalizes to clusters of different shapes and sizes.

Cons: [1] Choosing K manually, [2] Clustering outlines mess with results

Spectral Clustering:

Pros: [1] Clusters not assumed to be any shape/distribution, [2] Only need the similarity/laplacian matrix

Cons: [1] Choosing K manually, [2] Costly to compute

In both K-Means and Spectral Clustering, certain parameters like clustering value K needs to be computed manually. Spectral clustering is also more expensive to run that k-means, as it inherently calls k-means from inside the algorithm in addition to all the expensive matrix transformation's is needs to compute. But while K-means cares about Euclidean distances, Spectral Clustering is more about connectivity, representing a more flexible method for clustering data sets.

Centroid Initialization on K-Means:

K-Means clustering utilizes random centroid initialization, but the final clustering results are heavily dependent on centroid initialization as a poor initialization can result in an inferior local minimum. A smarter initialization of centroids results in better clustering performance by achieving clustering divergence the fastest.

Performance Analysis for Spectral Clustering:

Cosine and Gaussian kernel similarity: To perform performance analysis metrics like Gaussian kernel similarity, I imported similarity functions from external python libraries. Both cosine similarity and gaussian similarity for construction similarity matrix yielded the correct plots seen above in figures 4 and 5. Based on my spectral cluster implementation, computing the similarity matrix with either cosine similarity or gaussian similarity had no benefits on the simulated datasets. The only difference between them were the hyperparameters that were needed to be tuned. Cosine similarity yielded higher hyperparameter figure than gaussian kernel similarity. The euclidean distance metric used in the code was fine tuned to be '1' for "square.txt", and '0.15' for "elliptical.txt".

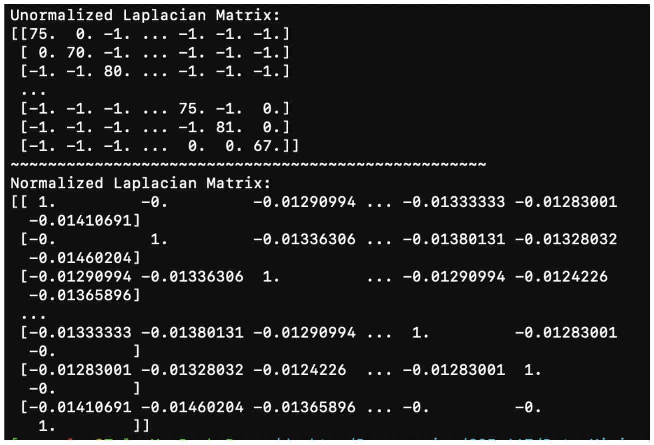
Performance Analysis for Spectral Clustering:

Unormalized Laplacian and Normalized Symmetric Laplacian: This involves transforming the unsymmetric adjacency matrix to a symmetric adjacency matrix, instead of unsymmetric laplacian matrix to symmertic laplacian matrix. Unnormalized matrix is represented by: L = D - A, while normalized and symmetric matrix is represented by: $D^{-1/2}$*L*$D^-1/2$. Both of them seem to plot the eigenvectors correctly, yet they require slightly different hyperparameters. Analyzing the matrices themselves, we see that the unnormalized laplacian has outlier values (i.e. extreme values) that are stripped away in the symmetric laplacian matrix.

Real-World Datasets:

Clustering Results:

After performing K-Means and Spectral Clustering on the real datasets, graphing them

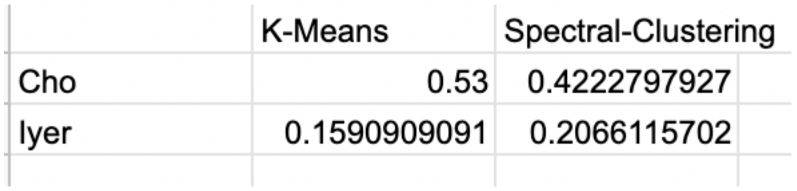


```
Unormalized Laplacian Matrix:
[[75.  0. -1. ... -1. -1. -1.]
 [ 0. 70. -1. ... -1. -1. -1.]
 [-1. -1. 80. ... -1. -1. -1.]
 ...
 [-1. -1. -1. ... 75. -1.  0.]
 [-1. -1. -1. ... -1. 81.  0.]
 [-1. -1. -1. ...  0.  0. 67.]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Normalized Laplacian Matrix:
[[ 1.         -0.         -0.01290994 ... -0.01333333 -0.01283001
  -0.01410691]
 [-0.          1.         -0.01336306 ... -0.01380131 -0.01328032
  -0.01460204]
 [-0.01290994 -0.01336306  1.         ... -0.01290994 -0.0124226
  -0.01365896]
 ...
 [-0.01333333 -0.01380131 -0.01290994 ...  1.         -0.01283001
  -0.        ]
 [-0.01283001 -0.01328032 -0.0124226  ... -0.01283001  1.
  -0.        ]
 [-0.01410691 -0.01460204 -0.01365896 ... -0.         -0.
   1.        ]]
```

Figure 6: *Unnormalized vs. Normalized Laplacian Matrix*

was meaningless because you can't display meaningful graphs past two dimentions. Instead, these algorithms computer their accuracies and compared them to truth labels, in addition to clustering data points into their respective clusters.

in order to calculate the accuracies, I used the accuracyscore library from sklearn. Combining that with a best matching algorithm between the true class labels (i.e. the ground truth values) and the obtained cluster labels still resulted in relatively low accuracies between 0.2 and 0.5. The "cho" dataset had a higher accuracy than "iyer" in both runs.

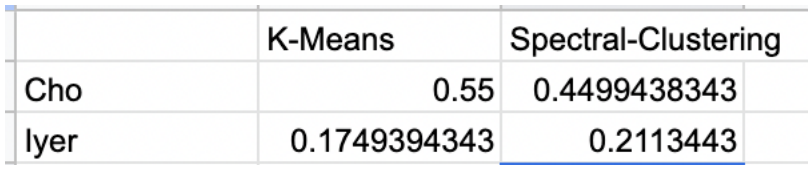


	K-Means	Spectral-Clustering
Cho	0.53	0.4222797927
Iyer	0.1590909091	0.2066115702

Figure 7: *Accuracy Metric*

<u>Data normalization on K-Means and Spectral Clustering:</u>

When applying data normalization to both datasets, the accuracy measures don't really change in terms of consistentcy across runs, and computing the accruracy rates are generally the same.

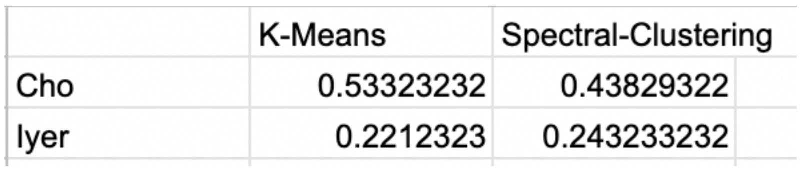


	K-Means	Spectral-Clustering
Cho	0.55	0.4499438343
Iyer	0.1749394343	0.2113443

Figure 8: *Accuracy Metric*

<u>**Noise on K-Means and Spectral Clustering:**</u>

There was little preprocessing steps that needed to occur for the "cho" dataset, but there were some outliers in the "iyer" dataset that needed to be removed. Some values had -1 in place for their truth values and therefore could not be clustered. If we compare the accuracy results to the preprocessed vs non-preproccessed data, we see that the "cho" accuracies have not changed. The "iyer" accuracies on the other hand have slightly improved, but not by much.



	K-Means	Spectral-Clustering
Cho	0.53323232	0.43829322
Iyer	0.2212323	0.243233232

Figure 9: *Accuracy Metric*