**CLUSTERING ALGORITHMS**

**By Sean Conway**

**Code available on GitHub:** <https://github.com/ConSeanway/ClusteringAlgorithms>

**Overview:**

The primary purpose of this assignment was to implement k-means and spectral clustering using a student’s choice of programming language. These two algorithms were to be tested on two different datasets containing genetic data (cho.txt and iyer.txt). Though I initially attempted a solution to this assignment in R, I decided that python would work better for this project. Python was a simpler language from an implementation standpoint, and it also ran faster than R for this application.

**Code Flow:**

Essentially, my code starts with the file main.py. When run, the program prompts the user to enter the dataset they would like to test (cho.txt or iyer.txt), as well as the clustering method that they’d like to use (either k-means or spectral clustering). Once the program receives a valid input from the user, it calls the main method in either kmeans.py or spectralClustering.py. How the program progresses at this point depends upon the algorithm selected by the user.

**K-means:**

For k-means clustering, either cho.txt or iyer.txt is loaded, and the ground truth clustering is extracted from the file for later use. We also create the training dataset, which is just the original dataset, less the indices column and the ground truth column. There are additional preprocessing steps for iyer.txt, which involve removing the third column of data (as it doesn’t provide us any additional information when clustering), and changing the ground truth values of -1 into their own cluster, numbered 0. Next, we need to normalize the training data so that the numerical scale of the attributes does not affect the result of our clustering (i.e. we want to weigh all factors equally for now). We then set the value of k to be equal to the number of unique items in the ground truth cluster, as well as the value of maximum iterations of the k-means algorithm to 100. Lastly, we call the k-means function on our dataset to cluster it.

The k-means function starts by checking the values of k, as well as the maximum number of iterations, as well as initializing arrays to hold the cluster numbers that each data point belongs to. We also initialize an array to hold the cluster centers, and each center is initialized to one of our data points. The actual algorithms are composed of only a few actions: recording our current cluster in a variable, reassigning the data points to the closest cluster center, and updating where the cluster centers are.

The data point reassignment method is straightforward: iterate through the dataset and find the distance between each data point and each cluster center. Find the distance between a given data point and each cluster and assign that data point to the closest cluster center. We then exit this method and move to the updateClusterPosition method. In this method, we essentially iterate through each point assigned to each cluster and take the average of each cluster’s points and move the centroid to that average position. We may need to re-pick cluster centers at this point if there are no data points assigned to a particular cluster. In this case, we simply re-initialize the cluster center to one of our data points and continue as normal. Outside of this case, we continue to move these cluster centroids and reassign data points until the clusters don’t change anymore. We print the sum squared error (SSE) at the algorithm’s termination.

**Spectral Clustering:**

If spectral clustering is selected instead of k-means, we follow a similar initial process: extract the user’s chosen dataset, pull the ground truth information into an array, create the training dataset, clean the iyer.txt data, normalize the training data, and then initialize k to the number of unique items in the ground truth dataset. Where the spectral clustering code differs, however, is in the procedure that it uses to cluster the datapoints.

With spectral clustering, we first create an undirected KNN graph of all of the data points using a method in the sklearn package. We then compute the Laplacian of the graph by finding the D matrix and subtracting the undirected KNN graph’s results from that (also called the adjacency matrix). We then determine the eigenvalues and eigenvectors of the Laplacian matrix and call the k-means algorithm on those eigenvectors. From that point forward, the process stated in k-means is applicable.

**Post-Clustering:**

In the case of both k-means and spectral clustering, we print the clustering results and compare them to the ground truth clusters. What makes this comparison difficult is the fact that the cluster numbers of the ground truth dataset and the clusters numbers assigned through the clustering algorithm are not guaranteed to be the same numbers. Thus, what we can do instead is partition the data points based upon what cluster the data point corresponds to in the ground truth. Due to the fact that the ground truth dataset is used as the basis for this partitioning, we cannot report purity statistics if the k value chosen is greater than the number of clusters in the ground truth. In that case, there would be no ground truth cluster to compare cluster k to. Similarly, if the value of k is less than the number of clusters in the ground truth, given the data we have, there is no way to accurately estimate how data points would be reassigned in the ground truth data. Thus, we can only report purity statistics if the k selected is equal to the number of clusters in the ground truth.

To report the purity, the program partitions the datapoints based upon their ground truth clustering, and then the cluster numbers each data point was assigned by the algorithm is assessed. The mode of each grouping’s cluster assignment is determined, and the number of times that the mode value appeared in the cluster is recorded. The purity of that cluster is simply the frequency of the mode value divided by the total number of elements in the cluster. This is printed to the user, and the program ends.

**Performance**

Two primary indices were used to assess the performance of the k-means and the spectral clustering algorithms. The first was an internal metric, the sum of squared errors. Essentially, this is a measure that indicates how close data points are to the cluster centers. The second, an external metric, is cluster purity, which takes into account how well the clusters represent classes in the ground truth dataset. To collect the following data, I ran each algorithm on each dataset 30 times, and took the sample mean and average. This information is reported in the following chart and an attached Excel document.

**Sum of Squared Errors (Average, Standard Deviation of 30 replications)**

|  |  |  |
| --- | --- | --- |
|  | K-Means | Spectral Clustering |
| cho.txt | 3107.525, 82.73227 | 380.996, 0.004983 |
| iyer.txt | 1603.431, 198.4556 | 505.9953, 0.005074 |

**Purity (Average, Standard Deviation of 30 replications)**

|  |  |  |
| --- | --- | --- |
|  | K-Means | Spectral Clustering |
| cho.txt | 65%, 4% | 66%, 19% |
| iyer.txt | 53%, 3% | 63%, 21% |

\* Please note that for iyer.txt for spectral clustering, two data points were altered. This is because these data points were extreme outliers. Keeping those data points unaltered in our data set made it impossible to find a solution that made data point assignments to “k” ground truth clusters. To account for this, these data points were instead given the mean value of the columns for which they were extreme outliers. For reference, these data points that were modified were items 363 and 491 of iyer.txt

**Analysis**

**Mean Analysis :**

When comparing the mean values for SSE and purity for the two algorithms, it is important to note the differences in input for each algorithm. Because spectral clustering inputs the eigenvectors of the dataset’s Laplacian into k-means when it runs, it will likely have a smaller SSE, as the inputted data is on a smaller scale. Thus, while it seems that spectral clustering reduced the SSE for both cho.txt and iyer.txt, part of this is simply due to the differences in the way the data was fed into the k-means portion of each algorithm.

Where the mean data becomes a bit more interesting is when we look at cluster purity. It seems that for cho.txt, the cluster purity did not change much between k-means and spectral clustering. Thus, we can conclude that k-means’ assumption of spherical data is mostly correct for the cho.txt dataset. However, for iyer.txt, there is a significant 10% difference in cluster purity between k-means and spectral clustering (53% and 63%, respectively). Thus, we can conclude that spectral clustering is likely more appropriate for this dataset, and that the dataset’s clustering is likely in some sort of non-convex shape. However, this mean value is only half of the story when it comes to the algorithms’ performance on the datasets given. Even more insights can be gained from looking at the standard deviation of each performance metric.

**Standard Deviation Analysis:**

To start, k-means seems to be a far more consistent algorithm when assessing the purity of a clustering result than spectral clustering is. The standard deviation of cho.txt and iyer.txt’s purity was only 3% and 4%, respectively, while for spectral clustering, the standard deviation of the data was 19% and 21% respectively. This high standard deviation means that it is very possible that the resultant purity of the clustering could be very high or very low. This attests to the fact that the results of spectral clustering are highly dependent upon the initial data points selected as centroids. To help ensure that we find the best clustering solution possible, if I were to implement spectral clustering again, I would want to add a “multistart” capability.

Essentially, a “multistart” capability would allow the clustering to restart after it has settled on a locally optimal clustering. Multistart would allow the algorithm to restart, saving its best current solution for the clustering, and compare any future solutions against that. In this way, we are more likely to find a globally optimal solution as opposed to a locally optimal clustering. This helps to eliminate the effect of bad initial point choice, which seems to really impact spectral clustering (though it certainly impacts both spectral and k-means clustering).

However, an interesting observation is that while k-means appears to be more consistent when estimating cluster purity, it seems that spectral clustering is far more consistent than k-means when it comes to assessing the SSE. The standard deviation values for the cho and iyer datasets clustered with k-means, respectively, are 82.73227 and 198.4556. On the other hand, when using spectral clustering, these values reduce to 0.004983 and 0.005074. However, this smaller standard deviation is due in part to a lower mean value for spectral clustering’s SSE, which is due to the fact that the values for the eigenvectors tend to be smaller than the actual numerical data point values. To account for this, instead of using the raw values for standard deviation, we can calculate the standard deviation as a percentage of the mean. Upon doing this, the difference is even more evident. For the cho dataset, the standard deviation is 2.7% of the mean for SSE when using k-means, while it is roughly 0% for spectral clustering. For the iyer dataset, the standard deviation is approximately 12.4% of the mean value for SSE, while for spectral clustering, ,the deviation from the mean is also roughly 0%.

The SSE is far more consistent for spectral clustering than it is for k-means because it has already identified which points are more connected than others. The primary driver of the value of the SSE for spectral clustering are the outliers in the dataset, which have likely already been identified in through the adjacency matrix and Laplacian. Essentially, before and during the k-means portion of spectral clustering, the algorithm has some idea of the outliers and can cluster the remaining points together to attempt to minimize the effect of that outlier.

**Pros/Cons chart:**

|  |  |
| --- | --- |
| K-Means Clustering | |
| Pros | Cons |
| Faster than spectral clustering, as there’s no need to calculate the Laplacian, eigenvalues and eigenvectors | Assumes that points assigned to a cluster are spherical about some center, thus non-spherical data does not work well |
| Guarantees convergence to some value | Does not capture connectivity in the dataset |
| Can generalize to elliptical clusters | Need to select a hyperparameter “k” manually |
| Can easily adapt to new examples, and can use warm-starting to do so | Inconsistent and highly dependent upon initial cluster positions. I tried to mitigate this by selecting random data points as cluster centers, but even then, we can see that the mean SSE varied quite a bit because of these initial positions |

|  |  |
| --- | --- |
| Spectral Clustering | |
| Pros | Cons |
| Allows us to find clusters with non-convex boundaries, and makes no assumption about the shape of the clusters | Slower than k-means due to costly calculations, especially for large datasets (due to eigenvalue/vector calculation) |
| Captures connectivity between points with the graph. K-means doesn’t do this, and this connectivity allows spectral clustering to be extremely flexible | Highly affected by extreme outlier points, to the point that I actually had to tamper with the iyer data in order for the spectral clustering to give reasonable results |
| Can cluster 1-dimensional data | Still need to choose a value for hyperparameter k |
| Don’t necessarily need the full dataset, just the similarity/distance matrix or Laplacian | Inconsistent, as spectral clustering relies on k-means (we see this inaccuracy in the cluster purity variation) |
| Easy to implement and generally gives good results |  |

**Other Insights:**

The cho.txt dataset was rather straightforward. Based upon the performance of the cho.txt dataset in k-means clustering and its performance with spectral clustering, I would predict that most of the data in cho.txt is contained in higher-dimensional spheres. This is because k-means seemed to predict cluster purity quite accurately, although it struggled a bit to get a consistent SSE.

The iyer.txt dataset was more of a handful. Not only were there issues with the class labels (there was a -1 and no 0 label), but there were also a few outliers in that dataset. The outliers were so large compared to the rest of the data that I actually had to tamper with them in order to get the spectral clustering to work. It was because of these values that I also had difficulty implementing the normalized Laplacian function. It was a very irregular dataset, but to be honest, I imagine that this is how most datasets would look like in a real environment. Essentially, the data just required a lot of cleaning and assumption-making on my part (i.e. that I would be able to replace the outlier data values with the mean from that column). Unfortunately, even though this may not be correct, this is required in most data mining tasks. In a non-academic environment, these outlier data points would be one of the first parts of this project that I would ask about, as they are certainly anomalies and would require further explanation.

The other major insight that I had regarding iyer.txt is that because it tended to perform better with spectral clustering than K-means, the ground truth data clusters are likely of some sort of non-convex shape. Moreover, in the iyer dataset, there is likely a large group of similarly labeled data, and outside of that, datapoints are relatively sparse. This is because I noticed that when I used spectral clustering with the iyer dataset, there tended to be one very large cluster with many data points. Even if I set k equal to 10, I would have one cluster with hundreds of points, and another cluster with only one or two points. This actually makes the purity metric rather useless when assessing the performance of the algorithm, as just weeding out the outliers and having one large cluster does not do much to further our information gain.

In retrospect, I wish I would have had more information regarding what each attribute represented in the cho.txt and iyer.txt datasets. It would have made the associations and clustering far more meaningful to know this. The clustering would have been more concrete, and perhaps it would have been a good opportunity to learn more about the domain. Overall, however, I thought that this assignment was a good way to learn about the different clustering algorithms and how to implement them on a functional level.