

Clustering Algorithm Project

Zhiquan(Zac) You

Department of Computer Science

UCLA

zyou151@g.ucla.edu

Abstract

K-means clustering is an unsupervised learning algorithm that has been widely used today in the realm of machine learning and data science. This clustering technique gained its popularity due to its simple implementation and efficiency to categorize specific data points into groups even when very little information (also known as label) is given about the data. The algorithm runs iteratively trying to partition the entire data set into k presumably defined clusters. It does this by assigning each data point to a cluster such that the Euclidean distance between the point and the cluster's centroid is minimized at each iteration till convergence. In this project, I implemented the k-means algorithm from scratch and compared the result of k-means with another clustering algorithm DBSCAN from sklearn.

1. Data Preprocessing

The yeast.tsv data set has some missing values on certain rows. Although there are several ways to handle missing values, such as removing the rows, what I did was to fill out the these missing values by the median of the column that missing value belongs to. The motif behind this was that the removal of the rows containing null values might not be a good approach if the missing rows are abundant and removing bit chunk of these data would make our data align to certain trends which is inaccurate. We can observe the graphs generated from the code output that most of the data points are skewed, either to the left or right. All dimensions of the data set have skewed data points. When the data is skewed and not symmetrical, it is a good practice to use the median value for replacing the missing values. And I standardized the data as well.

2. Code Implementation

- i. **Initialize the centroids:** The very first step of the k-means algorithm is to randomly initialize k centroids to form clusters. As shown in the code, num_example would be number of rows in our data set, which is 6400. We randomly chose k indices and make points at these indices centroids.
- ii. **Assigning points to clusters:** After we randomly chose k centroids, we assign each point in the data set to its closest centroids. In my code, I calculate the Euclidean distance between each point and each centroids, and append these distances to a distance array. Suppose we have three clusters, then each element of the distance array would be tuples of three. Then we assign each point the cluster index by using `np.argmax()` function to return the smallest distance to a cluster.

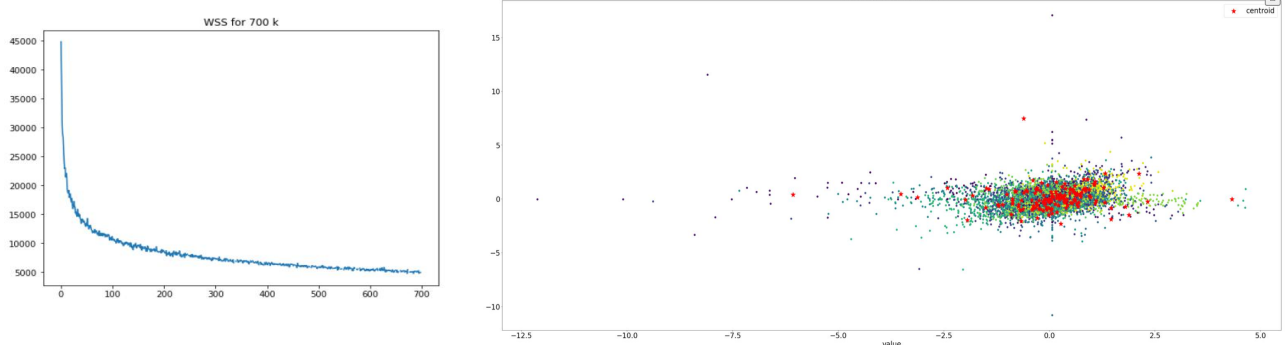
- iii. **Update the centroids based on the distance:** In this step, what I did was I re-initialized an array of centroids pre-filled with 0. Then for points assigned to each cluster in previous step, I calculated the mean of these cluster points and assign these values to new centroids. And I store these new centroids in centroids array and return the new centroids.
- iv. **Looping through max number of iterations:** In this step, we repeat the process of updating the centroids and reassigning points to their nearest clusters till we reach the maximum number of iteration specified by users.
- v. **Check convergence:** I added another stopping criteria within the for loop -- convergence. Once the centroids doesn't change at all, if this happens before the end of the maximum iteration, it would stop the loop.

3. Results

In this section, I will talk about the result of the running algorithm. I've tried several values of k and found some interesting facts about the clustering.

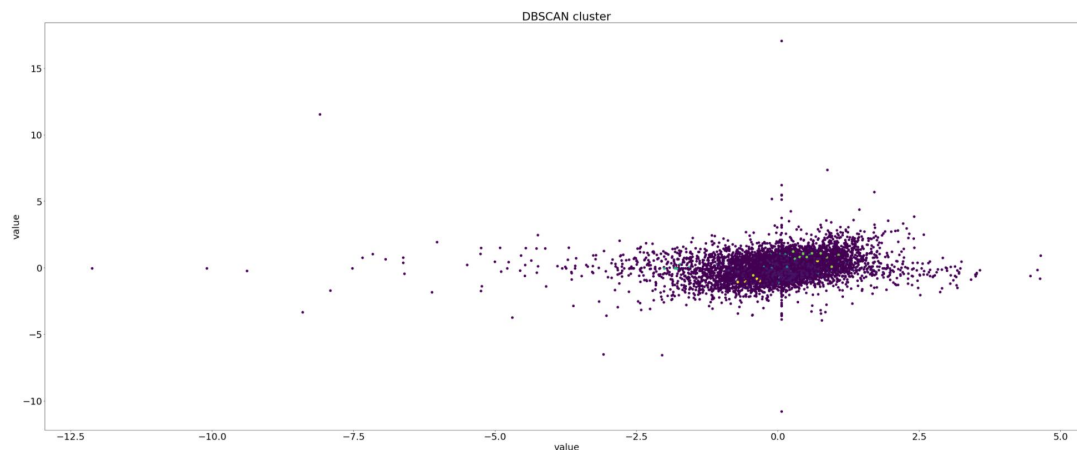
- ✧ Since there are 7 dimensions of data, I visualized dimension 1 and dimension 2 for the graphs. What I observed is that the clusters tend to overlap each other somehow. The decision boundaries for each cluster are not that clear.
- ✧ There are several outliers in the graph. Initially when k is small, the outliers were assigned evenly to the clusters that are closest to them while from eyeballing the outliers are pretty far away from the main clusters. As the number of k increases, these outliers eventually will get their own cluster centroids and form independent clusters themselves. This may imply that the model is pretty much overfitting. Because the outliers are not supposed to be a cluster. As k increases, the clusters' centers look like they mainly stayed near each other for the majorities of the points clustered in the middle, with some exceptions for which the centers are formed for these outliers.

I also created a function with sum squares to approximate number of clusters to be used, as explained by Prof. Harold in lecture -- the elbow method. I tested 700 k's and found out that after around 150 clusters the curve don't drop significantly and kind of stay stable. Reference to the code output in my notebook file. I graphed the clusters based on $k = 150$ and the observation I made was that the centers spread out to be assigned to even all the outliers and in the main cluster, there are so many clustering centers which indicates the shrink of each cluster size.



4. Comparison with DBSCAN

For comparison purposes, I used DBSCAN clustering algorithm from sklearn to cluster the dataset. Unlike k-means, DBSCAN does not require us to specify the number of clusters in the data a priori. However, the result didn't turn out well. As we can observe from the cluster below, DBSCAN returns a major cluster with some small dots (small clusters within the major cluster). The reason behind it was that DBSCAN is density based clustering algorithm, meaning that it would do well in terms of seeking areas in the data that have high density of observations and compare to the area of data that are not very dense with observations. So in our case, since we visualized dimension 1 and dimension 2 only and data from these two dimensions heavily overlap or are very close to each other, DBSCAN would just cluster one big cluster on the high density area. More interesting to note is that DBSCAN also clusters these outliers that spread out in the left into the big purple cluster. The reason why k-means form clusters on the denser data sets is that k-means are based on Euclidean distances as its measurements, so it does not take density into account, just the distances between points and centroids.



4. Summary

In summary, k-means algorithm that I implemented forms clusters on the denser data set, with the increase of k components, the cluster seems to overlap somehow, and the outliers gradually are assigned a centroid to form a cluster which indicates the possibility of overfitting. Also I used PCA as a way to reduce dimensionality of our data set (just for testing purposes) and ran the implemented k-means on the reduced components. I do observe that for the first few k's, the clusters are easily distinguishable, meaning that the decision boundary is sort of clear. DBSCAN tends to form a major cluster based on the density which our data set is highly dense on dimension 1 and 2.