Kshitiz Rimal

E-mail: krimal@masonlive.gmu.edu

G#: CS482-001

Miner Username: DataMining

Score: 89%

Current Rank at the time of writing this paper: 33

18 November 2020

- ## Objectives:
  - Think about Best Metrics for Evaluating Clustering Solutions
  - Deal with Image data (processed and stored in vector format)
  - Implement the K-Means Algorithm

- ## Language Used:
  - The programming language used for this project is python. Python has variety of in-built functions which you will not find in any other programming languages. The language is best fit for anyone who is working on machine learning and data mining field. Another reason for choosing this language is because of Python's built-in libraries like numpy and sklearn helped me to compile code faster and more efficiently. Using sklean's pairwise_distances function I was able to compute distance between clusters.

- ## Implementation:

Method 1: Preprocessing

- I started by reading the iris data as we have been doing for the past 2 homework. I stored the data into a list (InputData). I appended each line by line into list using ". sprit()" which will remove unnecessary things like space and "split()" which splits data line by line.
- The reason for using list is because it is easy to append data and there isn't much hassle while appending the data. I could've used array to begin with, but array requires a lot of manipulation while adding data, so I chose list instead of array to append the raw data from file.

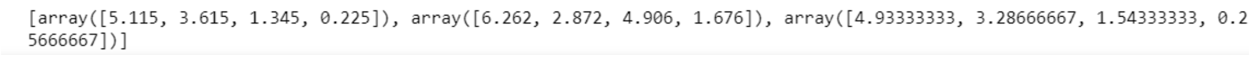
- Afterwards, I figured to convert list to array because converting data to array opened a lot of possibility for data manipulation. I used numpy's "np.array()" function to convert list to multi-dimensional array (150 x 4). Then, I also made a variable that stores total number of data which will be used later while implementing k-means algorithm.

Method 2: Implementing k-means algorithm

- First, I started by creating three random centroids because the specification mentioned to have 3 cluster ids. I created 3 rows and four columns data and saved in an array. I used "np.random.choice()" function from numpy library to create multidimensional arrays of random data points. It takes in the parameter of "total number of data", "total number of clusters" which is three, and a Boolean statement.

- Once three random data points were made, I jumped on to calculate the minimum distance from data points to each respective cluster (150 x 3). In order to do that I used sklean's "pairwise_distance" function. The metric I used to calculate the distance measure is cosine because Euclidian metric's score was lower compared to cosine metric. Then, "argmin(axis = 1)" is used to assign to each respective cluster. This function computes the minimum distance array of all three columns and outputs its index. For example, if my distance array is [3,2,5] the "argmin" function outputs index 1 because it is the smallest element in a cluster array. Each value inside distance array ([3,2,5]) are clusters that is computed by "pairwise_distance" function.

- Once assigning clusters, I went to calculate the mean of respective centroids. In order to find the minimum cluster, I used a for loop that loops three times (size of the cluster=3). I used numpy's "mean ()" function to find the minimum cluster in an array. I then appended it to a list. The example can be seen below:

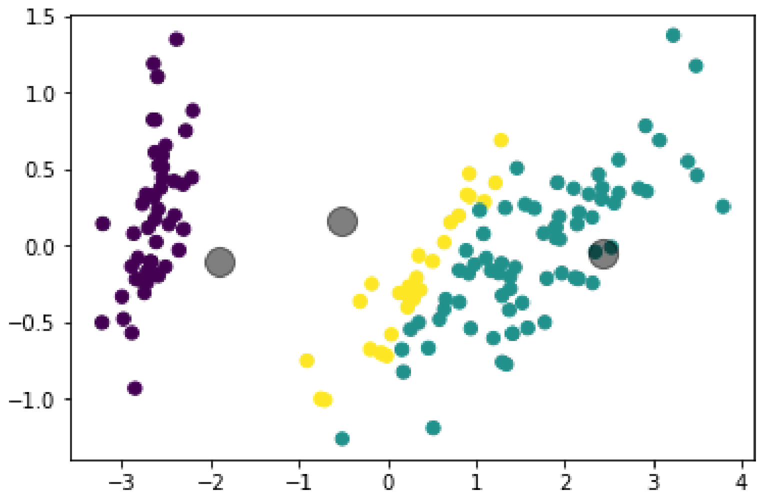


```
[array([5.115, 3.615, 1.345, 0.225]), array([6.262, 2.872, 4.906, 1.676]), array([4.93333333, 3.28666667, 1.54333333, 0.2
5666667])]
```

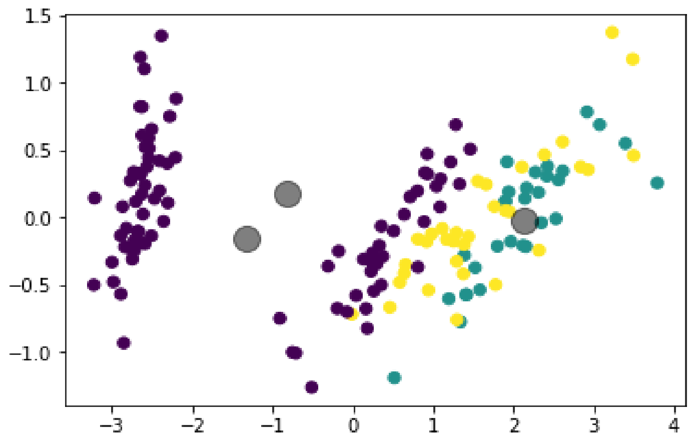
  After finding the centroid, I set old centroid's value to new centroid value by simply assigning old value = new value.

Method 3: Designing Cluster

- Once preprocessing and implementation was done, it is time to observe cluster by using diagram. I used matplotlib.pyplot library to observe the cluster. I also used sklearn.decomposition library to reduce dimension. The implemented k-means function is in 4D and we cannot observe 4D plot. So, sklearn's decomposition library gave a flexibility to reduce 4D to 2D. After that, I was able to plot the graph. The graph is just for observance to look at which cluster gives the best result. Since, in part1 there isn't any other factors that helps us to determine best cluster, using plot is easier. The plot can be seen below:



The way I looked at the plot was looking how widespread clusters are from each other. In the above diagram for example, I got a good score because there were no overlapping colors and every cluster were separated with each other. Below plot represents a bad plot where colors are overlapped with each other.



Method 4: Output

- After every step was done from prepressing, implementing k means algorithm and Designing cluster it is time to print the values into a text file. The index that was computed from "argmin" function now has to be stored in some type of file system. I used “.txt” file to store those outputs over “.csv” and other type of files. The reason for choosing “.txt” is because I feel comfortable working with it and since I have been doing this since homework1 I thought I would just go along with it. All the indexes from 0-3 are stored in a variable “final_cluster”. The data as of now is in array format and dealing with array is harder than dealing with list. So, I changed the “final_cluster” into list using “. tolist()” built-in function. Now my data is in list and I am ready to store it in the file. Next, I used the file system of python to write the data into a file and submitted the file on miner to get the result. My first result was below 40% because I was using Euclidian metric for pairwise_distance (details has been mentioned above on method 2). After couple of tries I got 89% with a good cluster imagine.