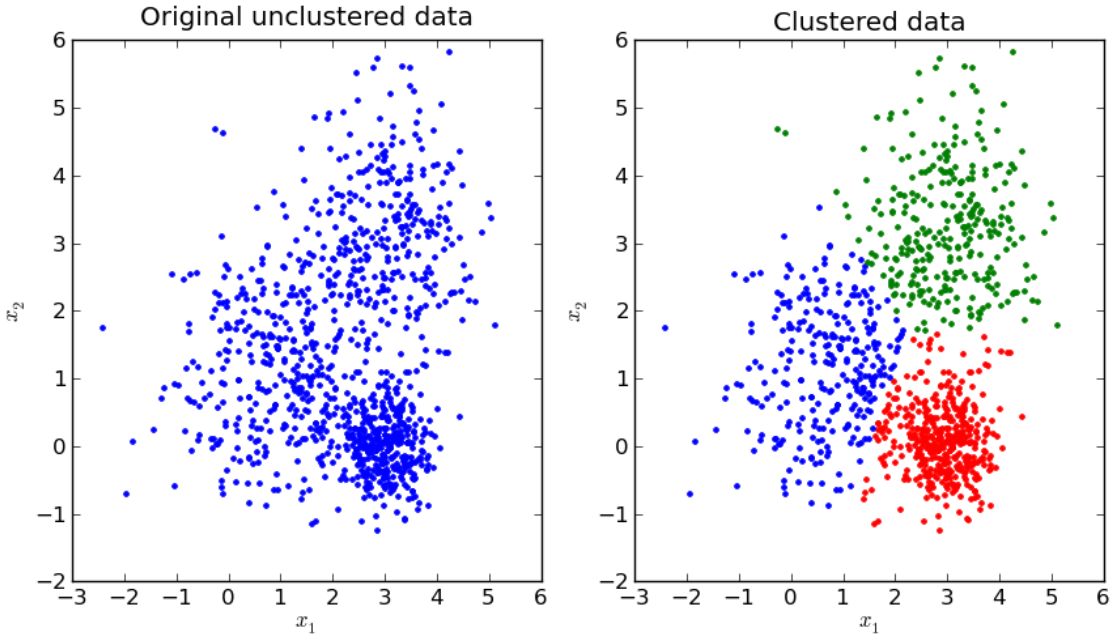
Artificial Intelligence
Intelligence k-means mini-project
Report

Rufat Huseynov
Shirin Shukurov

30 December, 2020

Introduction

Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different.



K-means is one of the popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labeled outcomes. The objective of K-means is to group similar data points together and discover underlying patterns. Unfortunately, sometimes the wrong k value can result in unexpected clustering because of random centroids.

Fortunately, there is an updated version of this algorithm called K-means++, where initial centroids are determined by assigning first centroid to the location of randomly selected data point, and then choosing the subsequent centroids from the remaining data points based on a probability proportional to the squared distance away from a given point's nearest existing centroid. The effect is an attempt to push the centroids as far from one another as possible, covering as much of the occupied data space as they can from initialization.

Program files

There are 3 files in total in this program whereas two of them are .py (k_means.py and main.py) files and one of them is .data (2d.data) file:

1) 2d.data file - x and y points stored in this file

2) k_means.py is the file where most of the functionalities of program are written in. In this file, necessary libraries are imported and K_means_plusplus class which contains 9 functions within itself is defined.

- load_data(self, file_name) receives .data file as an argument, then converts it .csv and reports it as .csv.
- calc_dist(self, X1, X2) receives 2 points as argument, then calculates the distance between 2 points.
- find_closest_centroid(self, centroid, X) receives centroid and all the data points as argument, then takes each data point within X and calculates the distance between each centroid and data point. After, the data point with minimum distance is appended to associated centroid.
- calc_centroid(self, clusters, X) receives clusters and all the data points as argument, then takes an average of all the data points of each centroid and moves the centroid to that average
- write_to_csv(self, data) receives data and converts it to csv file making file name result.csv.
- calc_entrophy(self, cluster_count, n_sample) receives number of clusters as dictionary and amount of the given data, then calculates the entropy.
- k_plus(self, data, k) receives array of data points and number of clusters as argument, then initialize the centroids for k-means++
- k_means_algorithm(self, data, centroids) receives given data set and initialized centroids that received from k-means++, then clusters the given data with received centroids from k-means++ algorithm
- run (self, k_value) receives number of clusters as an argument, then for each k_value from 2 to k_value (included), it calculates and prints entropy 32 times for each value withing the range in order to determine the best case of k_value.

3) main.py is the file which imports K_mean_plusplus class from and is mainly used to run the whole program. It creates an instance (k_means) of K_mean_plusplus class and takes the number of clusters (k_value) as input from user and and runs it with k_means.run(k_value).

User Manual

Using this program should be simple enough for everyone. All you have to do is to run the main.py program with the k_means.py and 2d.data in the same directory. You can run the software in following methods:

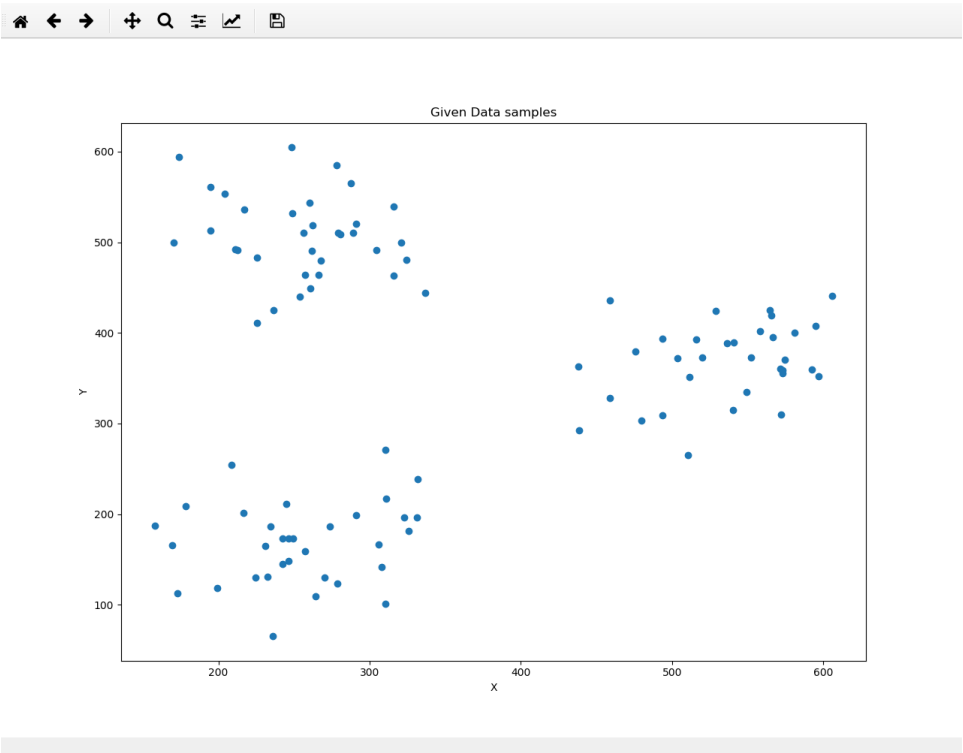
1) If you want to execute this software using command line / terminal, you have to be at the same directory with these files and have to run command “python main.py”.
2) You can execute this software just hitting “Run file (F5)” on one of the following IDE-s: Anaconda/Spyder, VS code and so on.

After accomplishing one of the above-mentioned methods, program will start executing and ask you to enter the maximum k-value. You will have to enter the integer for that purpose and the program will start calculating entropy and printing it for each value of k from 2, k_value (k_value included).

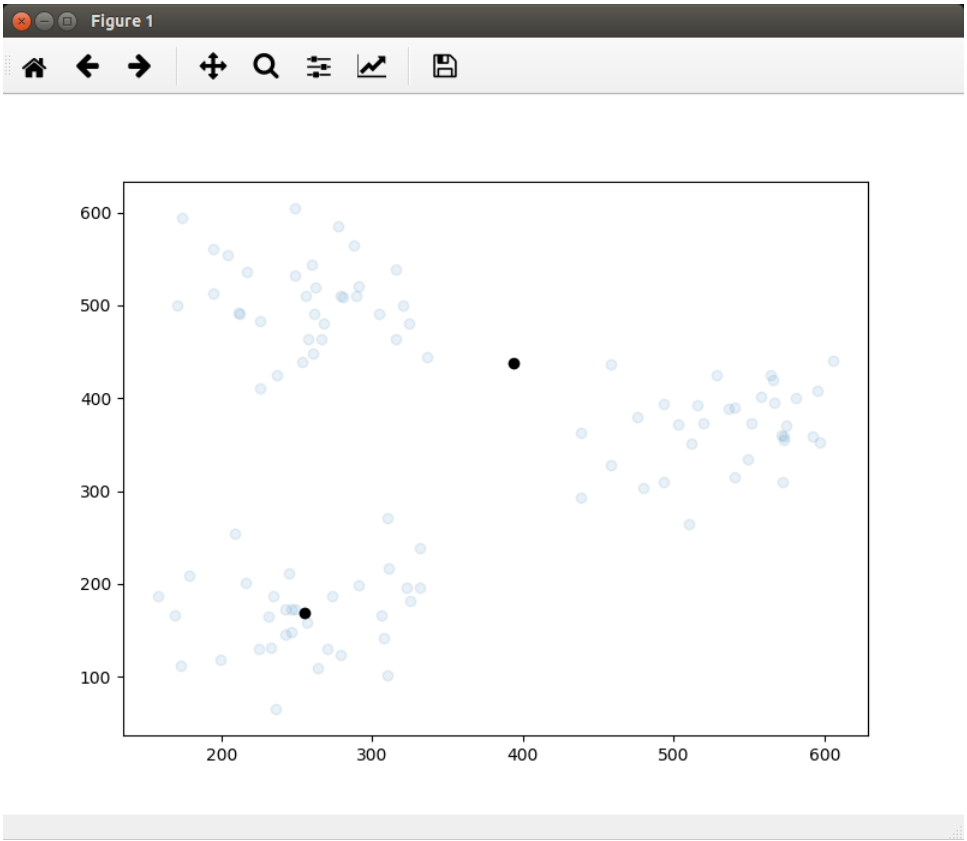
WARNING: If you enter value less than 2, program will ask you to re-enter the value with the warning message of “k-value must be equal to 2 or higher”.

How the program works?

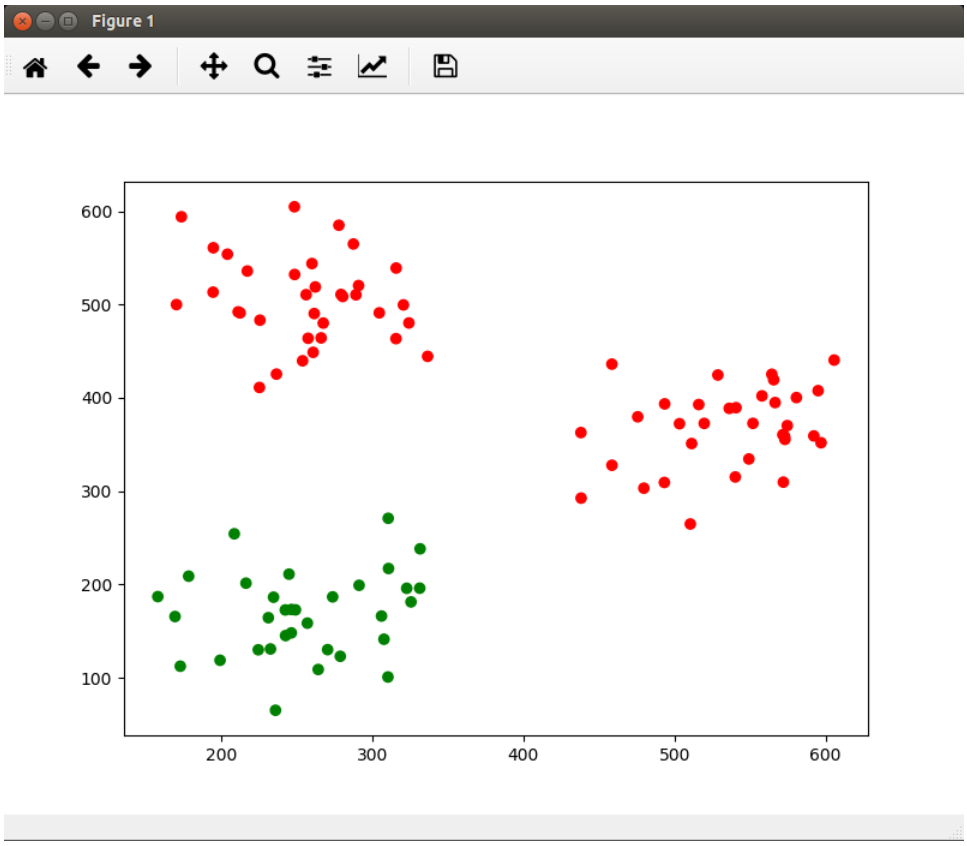
The program starts showing you the 2D representation of the given data using **matplotlib.**



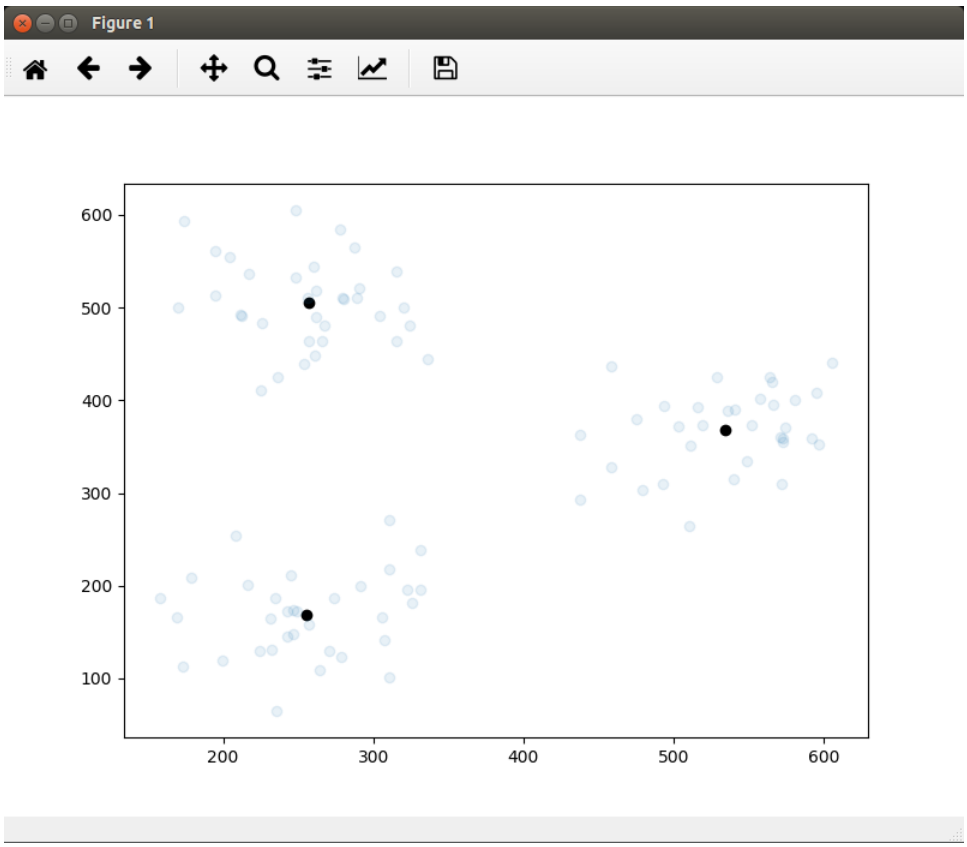
Then program ask from user to enter max k value to start evaluation of the given data. Minimum k value for k is 2. It starts looping with k values from 2 to the max value. Within each k value another for loop is called (max 32 times). Inside this loop k-means++ called firstly, then acquired centroids passed to k-means algorithm. By repeating this process, we are checking the how many different clusters can be acquired with given k value. Also, to check the diversity of the clusters we are using the entropy to calculate each time whether our cluster is changed or not. With different k values we obtained different results.



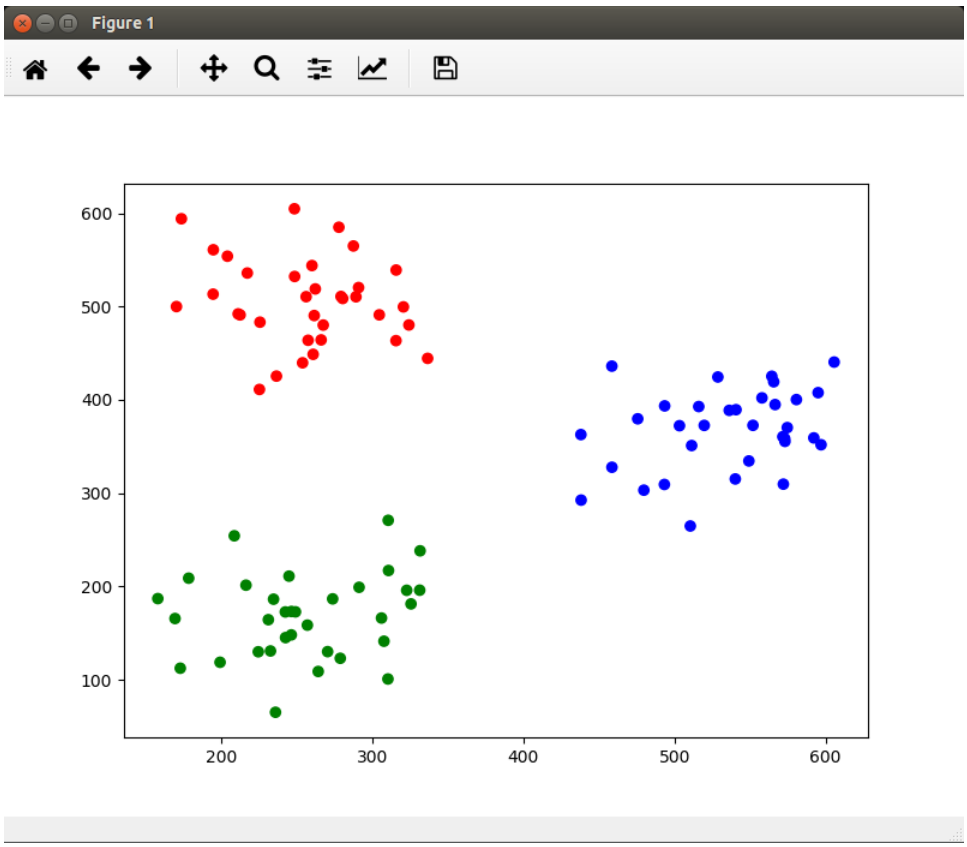
K=2 centroid



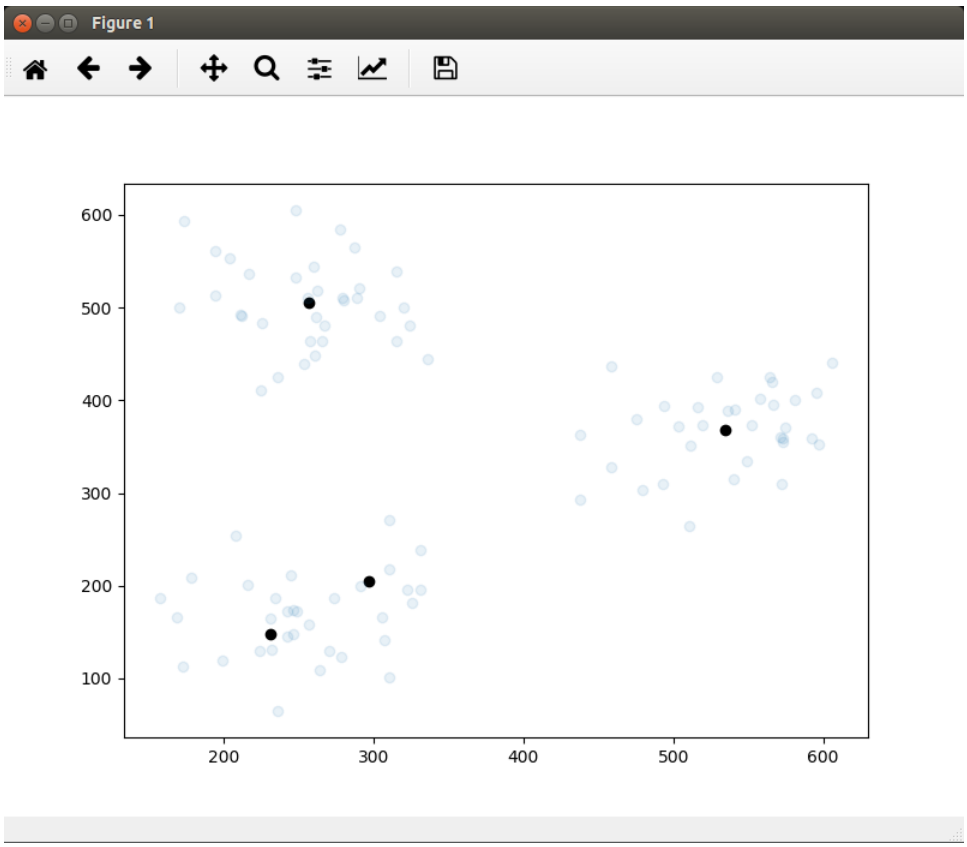
K=2 clustering



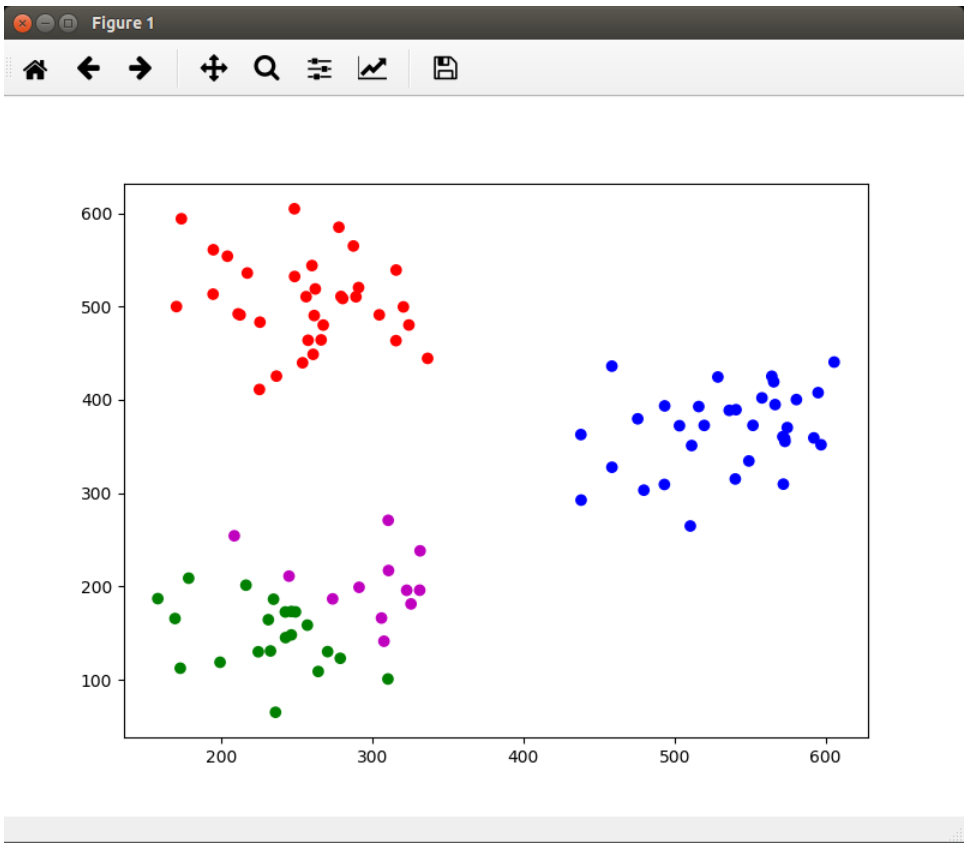
K=3 centroid



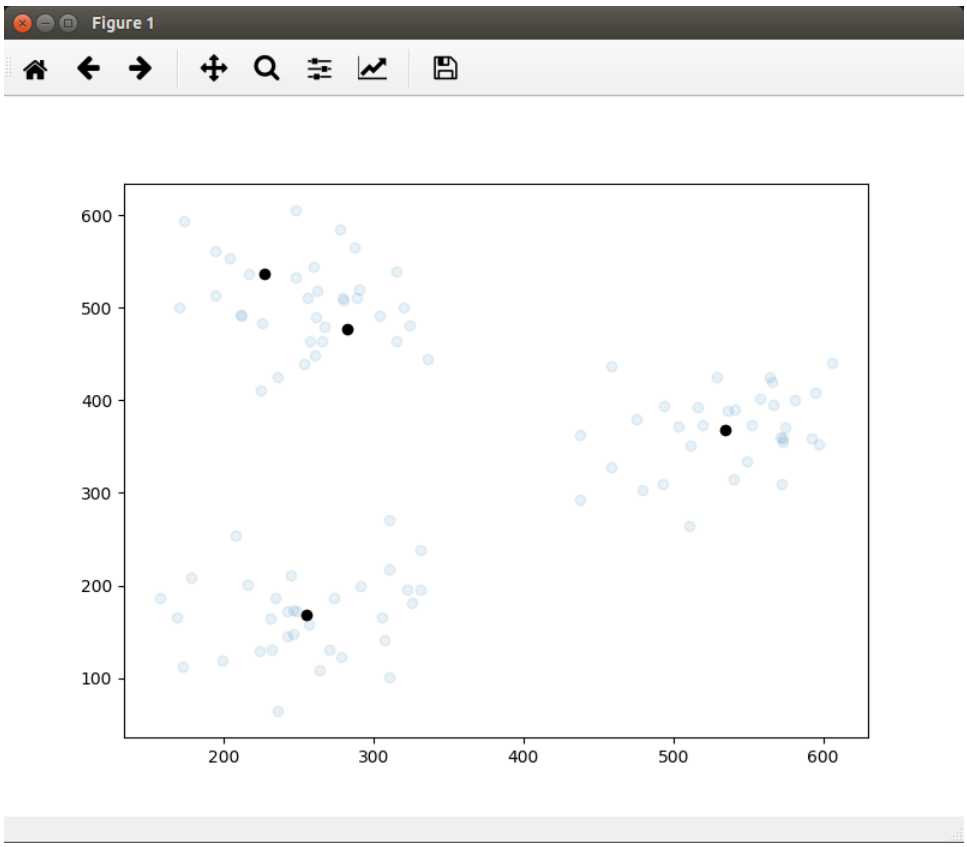
K=3 clustering



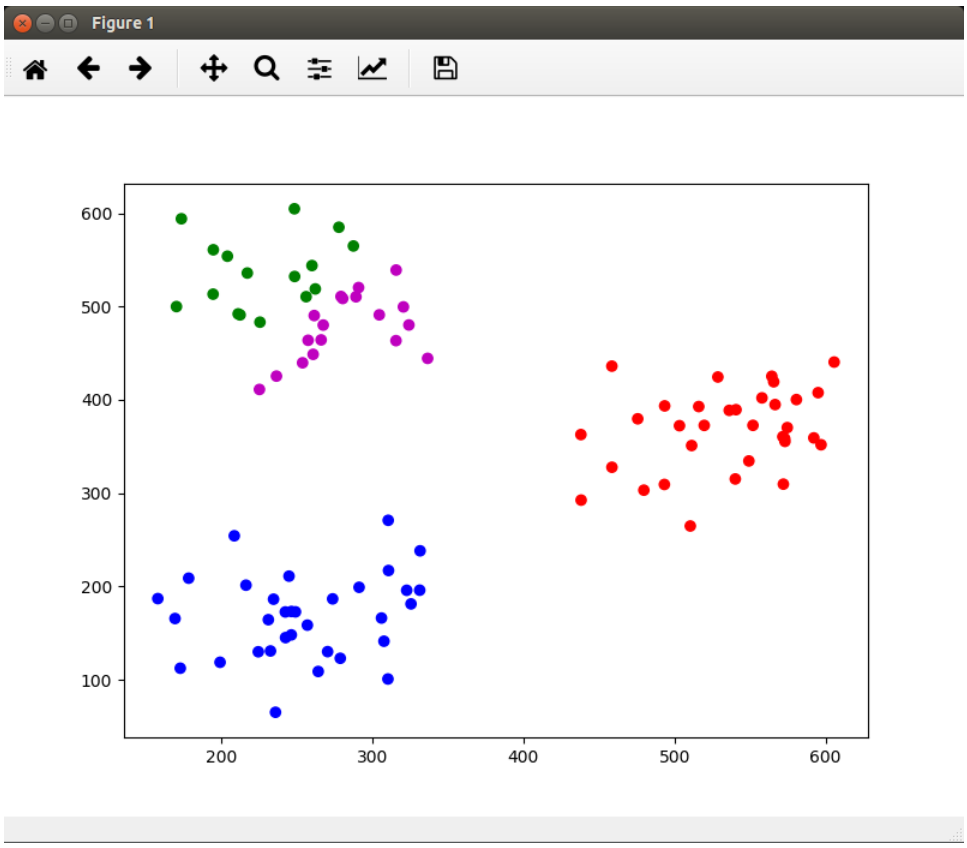
K=4 centroid



K=4 clustering



K=4 centroid_2



K=4 clustering_2

Among these results we can obviously see that K = 3 is the best option for this data.
Note: to see these graphs while running program please uncomment lines 168 ~171 and 183 ~186.

Discuss the results

Firstly, we take the maximum k-value from the user (For example: 4). Then our program starts executing the program with k-value = 2, k-value = 3, k-value = 4. It prints the obtained results on the console after running the program for each k-value as
Counter({entropy : amount of clusters with this entropy}).
For k = i (2, 3, 4) ---> Different clusters is n (n – number of different clusters obtained with 32 runs).
As we can see from the picture when we run k=2 we get **27 clusters with entropy 0.914926 and 5 clusters with entropy 0.924819.** That means we have 2 type of different cluster. But with k= 3 we have **32 clusters with entropy 1.584819 and that means we have only 1 type of cluster**. So, k = 3 is the best result for this case.

The result was satisfying as we could find the k-value where number of different clusters were equal to minimum.

```
(base) shirin@shirin-Lenovo:~/Desktop/Artificial Intelligence/Project/class$ python main.py
Enter the maximum k-value: 4


Entropy with 32 run --> (entropy : amount of clusters with this entropy)
Counter({0.914926: 27, 0.924819: 5})
For k = 2 ---> Different clusters is 2


Entropy with 32 run --> (entropy : amount of clusters with this entropy)
Counter({1.584819: 32})
For k = 3 ---> Different clusters is 1


Entropy with 32 run --> (entropy : amount of clusters with this entropy)
Counter({1.896887: 13, 1.92397: 7, 1.911112: 6, 1.83084: 3, 1.904026: 2, 1.887856: 1})
For k = 4 ---> Different clusters is 6
```

Conclusion

Generally, this project helped us to convert our theoretical knowledge about clustering into practical skills. We learned about K-means and K-means++ clustering methods and why it is better to use K-means++ for initialization of initial centroids.

The disadvantage of K-means++ is that it initializes the first centroid on a random data point. Hence, there is such possibility that first centroid might be on the bad position. That's why we are repeating this initialization 32 times with random initial center choices in order to find out the right k-value by finding the entropy 32 times for each k-value. When we get the minimum different clusters as output for associated k-value, we are confident to mention that this k-value is the best option for us.