Machine Learning Project

Suryam Arnav Kalra

Procedure :

We have used K means Clustering an unsupervised learning algorithm to create clusters of the provided dataset.

1) Importing Dataset

a) Imported the dataset from the ILPd.csv file provided
b) It contained 584 data samples, the data contained 10 features and 1 column for target values.
c) There were all real valued features except 1 (Gender) and the target value can have only two values (1 and 2)

2) Normalizing the Dataset

a) Firstly we found the missing values present in the dataset and filled those values with the average of all values for that feature,
b) In the filled dataset we did normalization as a part of data analysis.
c) Splitted the Dataset in 80:20 split , 80% of the data for training and 20% for testing.

3) Performing K means clustering

a) Chosen K random centroids for k clusters.
b) For every data point find the euclidean distance from each of the k centroids and get the minimum of those , after this put this particular datapoint in a cluster whose centroid is at minimum euclidean distance from this point.
c) After the above step we need to again calculate the centroids of the k clusters for this we now find the mean of all the points in the cluster and assign this mean as the new centroid.

d) With this new centroid we go back to step (b) and then repeat step (b) and (c) for a large number of times for better clustering.
e) For the class labels of a particular cluster we assigned it to the class label of the majority class present in that particular cluster.

4) Performance metrics calculation

a) **Performance using homogeneity** - To calculate the homogeneity for our k means clustering algorithm we used homogeneity_score from sklearn.metrics.cluster , in this we passed our testing dataset , and the prediction of this dataset using our k means algorithm.
b) **Performance using ARI** - To calculate the ARI score for aur k means clustering algorithm we used adjusted_rand_score from sklearn.metrics.cluster , in this we passed our testing dataset , and the prediction of this dataset using our k means algorithm.
c) **Performance using NMI** - To calculate the NMI score for aur k means clustering algorithm we used normalized_mutual_info_score from sklearn.metrics.cluster , in this we passed our testing dataset , and the prediction of this dataset using our k means algorithm.
d) **Performance using silhouette** - To calculate the silhouette score we found the prediction of our testing dataset and checked if every prediction comes out to the same label class we return the silhouette score to be 0 otherwise we return silhouette score that we got from silhouette_score function imported from sklearn.metrics.cluster.
e) **Performance using calinski harabasz** - To calculate the calinski harabasz score we found the prediction of our testing dataset and checked if every prediction comes out to the same label class we return the calinski harabasz score to be 0 otherwise we return silhouette score that we got from calinski_harabasz_score function imported from sklearn.metrics.cluster.
f) **Performance using ground truth** - For this we found the predictions of the testing dataset from our k means algorithm and compared it with the provided target values and returned the percentage of the accuracy.

5) Performing K means++

a) To start with this we chose a random first cluster centroid. After this we find the distance of every other point from the centroids (In the first step there will be only 1) and then choose the next centroid to be that point whose distance from all the current centroids is maximum.

b) Repeat the above step to find all the k centroids.

c) For every data point find the euclidean distance from each of the k centroids and get the minimum of those , after this put this particular datapoint in a cluster whose centroid is at minimum euclidean distance from this point.

d) After the above step we need to again calculate the centroids of the k clusters for this we now find the mean of all the points in the cluster and assign this mean as the new centroid.

e) With this new centroid we go back to step (c) and then repeat step (c) and (d) for a large number of times for better clustering.

f) For the class labels of a particular cluster we assigned it to the class label of the majority class present in that particular cluster.

6) Performing Test_A (Mentioned in the Assignment)

a) We selected a K

b) We have chosen K random points for K centroids / for a different version of Test_A that we have run on k means++ we also have chosen K initial centroid points based on heuristics.

c) We have split the training data in 80:20 split , 80% for training and 20% for testing, after this we ran k-means clustering algorithm on this training data.

d) Then we calculated the average performance metric (NMI score) by repeating the above step 50 times.

e) After this we just had to repeat (b) and (c) 50 times to get the average metric vs iteration count plot.

Functions:

1) **import_data**: Imports the data from the given ‘heart.dat’ file and converts it to a numpy array with the elements being in float data type.
2) **convert**: This function is used for column 2 of the data since it contains boolean values. We have set male as 1 and female as 0.
3) **fill_missing_values**: This function is used to fill the columns which have missing values. In our dataset, column 10 had missing values and we filled it with the mean of the values present in that column.
4) **get_validation_and_train_data**: This function randomly shuffles the data imported from the function above and does a 80:20 split to get the training and testing data. The data is further divided into train_X, train_Y, test_X, test_Y sets which denote respectively the training attributes, target values for training, testing attributes and target values for testing.
5) **normalize_data**: This function first fills the missing values in the data and then normalizes it and after doing a 80:20 split returns the train_X,train_Y,test_X,test_Y sets.
6) **get_euclidean_distance**: This function is used to get the euclidean distance between two points in the space.
7) **get_min_distance_index**: This function is used to return the index of the centroid which has the minimum distance with respect to a given data point.
8) **update_centroids**: This function updates the centroids with the mean of the data points present in the cluster.
9) **random_centroids**: This function is used to get the initial random centroids.
10) **Kmeans_clustering**: This function is used to implement the K-means clustering algorithm in which we randomly initialize the centroids and then update the centroids based on the clusters formed using the euclidean distance.
11) **performace_using_ground_truth**: This function is used to get the accuracy of the prediction using the ground truth values.
12) **performance_using_homogenity**: This function finds the homogeneity score of the clusters formed.

13) **performance_using_ARI**: This function finds the ARI (adjusted rand index) score of the clusters formed.

14) **performance_using_NMI**: This function finds the NMI (normal mutual information) score of the clusters formed.

15) **performance_using_silhouette**: This function finds the silhouette score of the clusters formed.

16) **performance_using_calinski_harabasz**: This function finds the calinski harabasz score of the clusters formed.

17) **permute_data**: This function permutes that data once and returns the 2 training sets and test sets with a 40:40:20 ratio to be used in Wang's method.

18) **wangs_method**: This function is used to implement Wang's method of cross-validation where it permutes the data C times and finds the average metric score for K values in a range to get the best k value.

19) **test_A_kmeans**: This function is used to implement the test_A mentioned in the question using K-means clustering and plots the average metric score versus the iteration count for a fixed K.

20) **test_A_kplusplus**: This function is used to implement the test_A mentioned in the question using K-means clustering with a heuristic using K-means++ and plots the average metric score versus the iteration count for a fixed K.

21) **Kmeans_plus_plus**: This function uses a heuristic to initialize the centroids And after that calls the k-means with the new centroids.

22) **get_plot_for_indexes**: This function is used to get the plot for the various external and internal indexes used for cluster evaluation.

23) **predict**: This function is used to predict the value of one single data set. It follows the path in the tree based on the attribute used at that node for splitting.

24) **get_plot**: This function is used to get the plot using the matplotlib library and save it as a jpeg file.

Data Analysis and Cleaning:

- We had 10 features in which one of them was gender whose values were not real so we replaced the two values (male, female) that feature could hold with to real values 1 for male and 0 for female.
- After the conversion of all the features to real values we then checked for missing values in the dataset and we found that for a particular feature there were 4 missing values we then calculated the average of all the remaining values of that feature and filled the 4 empty places with this average.
- Since the value of some features were not comparable it was necessary to do data normalization. Here’s the formula that we used for normalization *X’* = $\frac{X - Xmin}{Xmax - Xmin}$

Results:

1) Analyze impact of varying K using various external indices for cluster evaluation

a) **Ground truth**: We can see that with increasing K value, more clusters will be formed and so there will be a clear separation in the data which will lead to somewhat better ground truth accuracy.

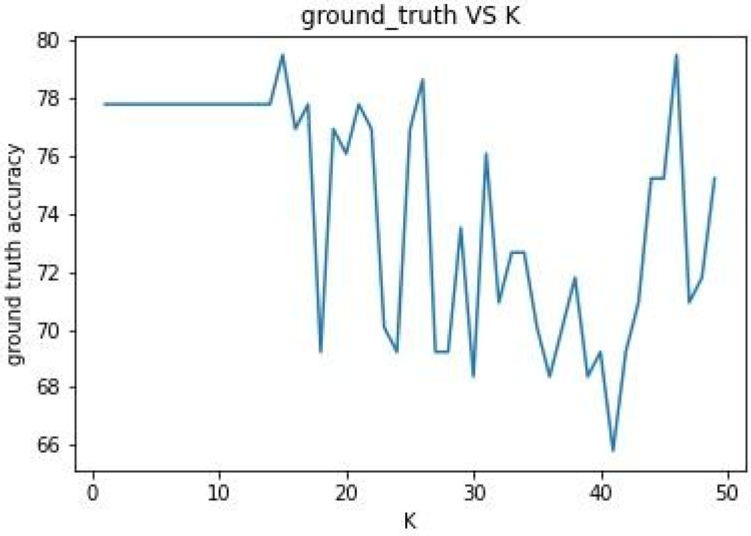

Fig a) Ground Truth vs K

b) **Homogeneity**: A clustering result satisfies homogeneity if all of the data points belonging to the cluster are members of the same class. So, with increasing K the homogeneity index should increase since this leads to more clusters as is depicted in the figure below.

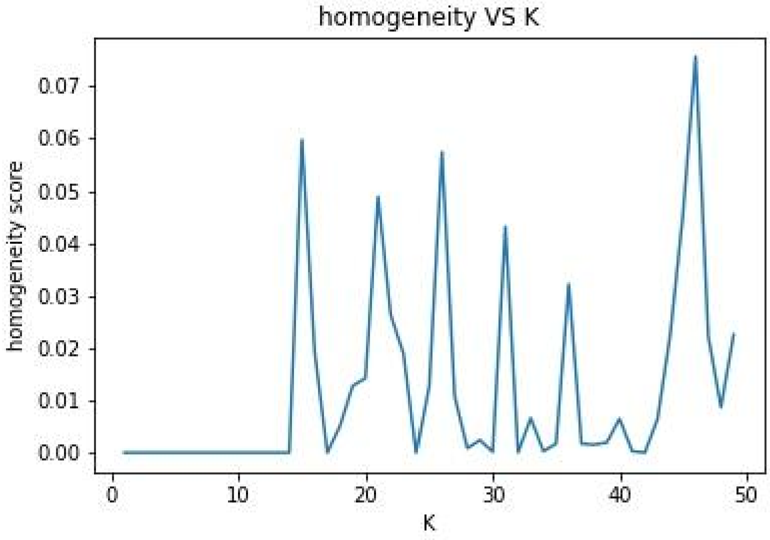

Fig b) Homogeneity index vs K

c) **Adjusted random index (ARI)**: It measures the consensus between the true, pre-existing observation labels and the labels predicted as an output of the clustering algorithm. Since the Rand index measures labeling similarity on a *0-1* bound scale, with one equaling perfect prediction labels, higher the ARI better the clustering.

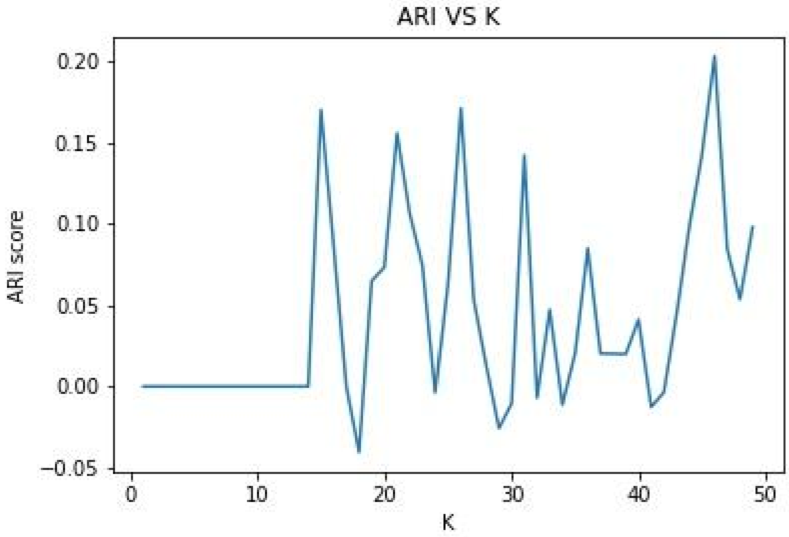

Fig c) ARI vs K

d) **Normalized mutual information (NMI)**: NMI is a measure used for determining the quality of clustering. It is an external measure because we need the class labels of the instances to determine the NMI. Higher the NMI, better is the clustering.

$$NMI(Y,C) = \frac{2 \times I(Y;C)}{[H(Y) + H(C)]}$$

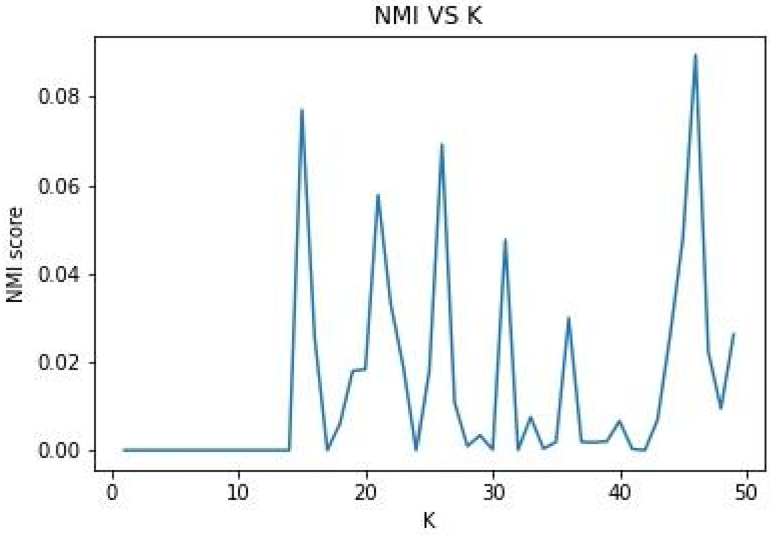

Fig d) NMI vs K

We can see that the **optimal value of K using the above indices lies in the range of 10 to 20**.

2) Analyze impact of varying K using various internal indices for cluster evaluation

a) **Silhouette Index**: Silhouette index is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1. Higher index implies clusters are well apart from each other and clearly distinguished whereas lower index signifies that the clusters are not well separated**.**

$$s = \frac{b - a}{max(a, b)}$$

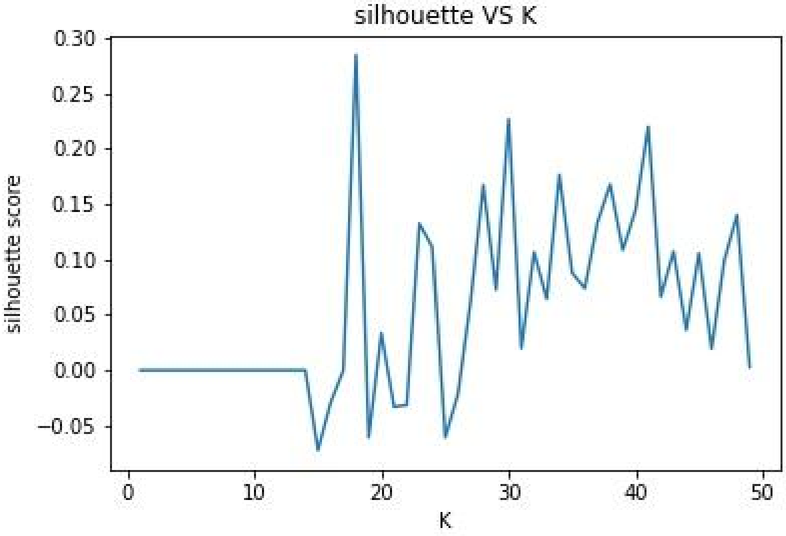

Fig e) Silhouette index vs K

b) **Calinski-Harabasz Index**: The Calinski-Harabasz index also known as the Variance Ratio Criterion, is the ratio of the sum of **between-clusters dispersion** and of **inter-cluster dispersion** for all clusters, the higher the score , the better the performances. The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.

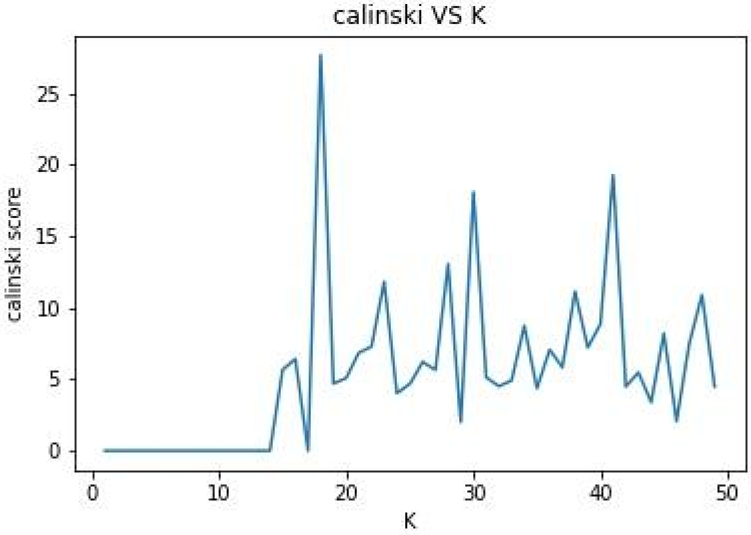

Fig f) Calinski-Harabasz index vs K

c) **Wang's method of cross-validation:** In Wang's method of cross validation, the data is permuted C times for a fixed K value to find the average metric score for a particular K. This value of K does not depend much on the random splitting of the data but reflects a more generalized picture of the clusters.

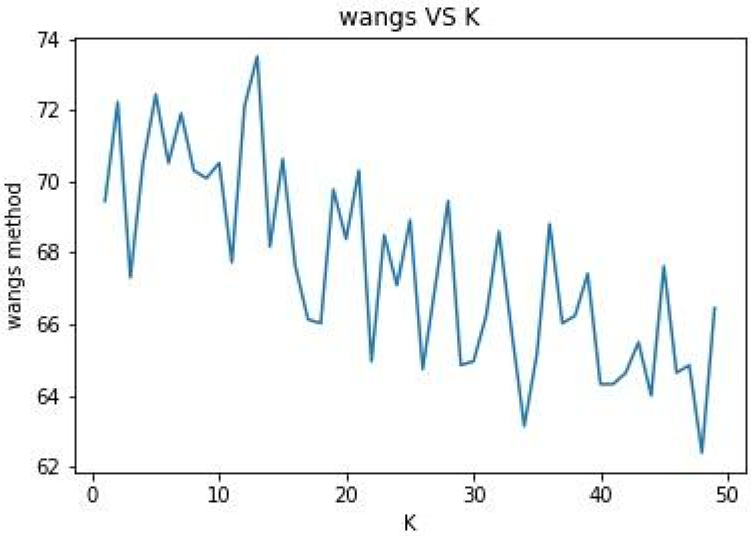

Fig g) Wang's method vs K

We can see that the **optimal value of K using the above external indices also lies in the same range of 10 to 20 as depicted by the internal indices.**

3) Effect of different random initialization of centroids in different executions

a) We pick different/random K centroids initially each time we run K-means clustering algorithm.

b) This can lead to different clusters being formed each time we run the algorithm leading to variations in cluster evaluation indices.

c) The performance of the algorithm tends to become more random, since it may find good clusters sometimes and at other times it may not.

d) We can see that for a fixed value of K, the dispersion in the metric (NMI) is quite big as discussed above.

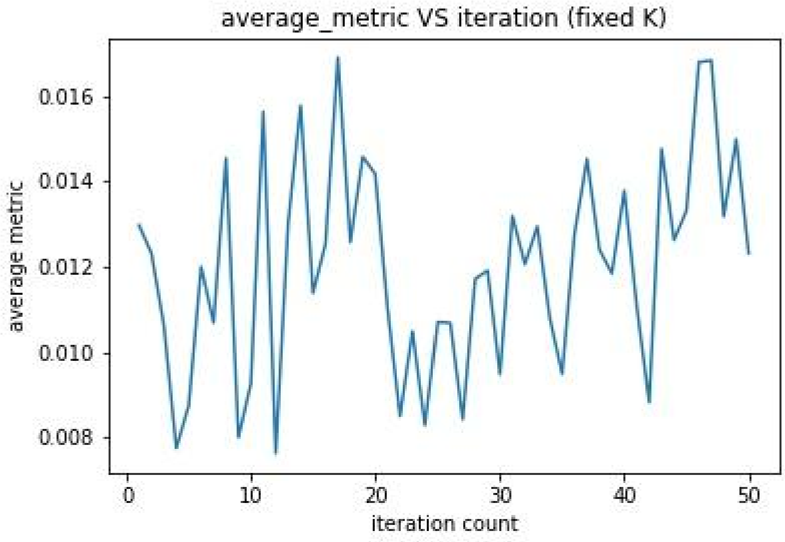

Fig h) Random initial centroids in different execution

Dispersion in metric:

- Mean = $1.2019039563358000 \times 10^{-2}$
- Standard Deviation = $2.4630946437920000 \times 10^{-3}$

4) Heuristic to select initial centroids (K-means++)

a) We have used the K-means++ heuristic to initialize the centroids of the algorithm.

b) In this method, we do not pick the K initial centroids randomly but instead only one out of the K centroids is picked at random.

c) After the first centroid is chosen, we find the distance of the remaining data points with their nearest centroid and choose the data point with the farthest distance to be a candidate for the next centroid.

d) This leads to a better clustering since the data point farthest from any of the centroids has the maximum tendency to form a new cluster with different properties than the already formed clusters (centroids).

5) Effect of using K-means++ instead of random initialization

a) When we use the heuristic (K-means++) instead of random initialization, we observe that the clusters tend to become more distinguished.

b) This is due to the fact that the heuristic chooses the data point with the maximum chance of forming a new cluster.

c) We can observe that with varying executions of the code for a fixed K, the dispersion in the metric seems to be less since the clusters formed are more stable.

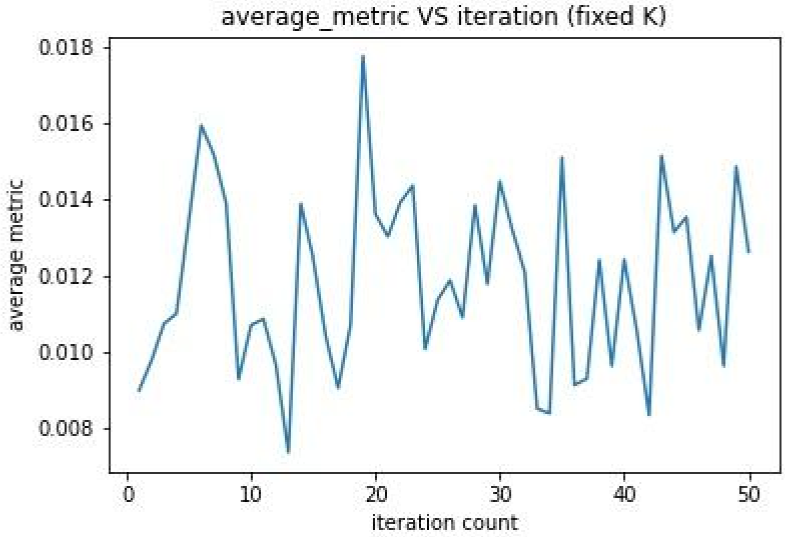

Fig i) K-means++ in different execution

Dispersion in metric:

- Mean = $1.1840529675916001 \times 10^{-2}$
- Standard Deviation = $2.321349285562100 \times 10^{-3}$

We can see that the standard deviation using the heuristic has decreased so we can say that by using this heuristic we have achieved a better K Means algorithm overall. Also, the mean of the metric has increased after using this heuristic showing a boost in performance.

Folder hierarchy:

1. main.py - This file contains all the functions necessary for this assignment.
2. requirements.txt - This contains all the packages required for this code to run
3. Report.pdf - This contains a report for this assignment.

4. ARI.jpeg - ARI Performance metric vs number of clusters
5. calinski.jpeg - calinski Performance metric vs number of clusters
6. ground_truth.jpeg - calinski Performance metric vs number of clusters
7. homogeneity.jpeg - homogeneity Performance metric vs number of clusters
8. ILPD.csv - dataset csv
9. NMI.jpeg - NMI Performance metric vs number of clusters
10. silhouette.jpeg - silhouette Performance metric vs number of clusters
11. test_A_kmeans.jpeg - average metrics vs number of iteration using k means
12. test_A_kplusplus.jpeg - average metrics vs number of iteration using k means++
13. wang.jpeg - wang Performance metric vs number of clusters

How to run the code:

1. Download this directory into your local machine
2. Copy the 'ILPD.csv' file in the Source Code Directory
3. Ensure all the necessary dependencies with required version and latest version of Python3 are available (verify with requirements.txt)

   ```
   pip3 install -r requirements.txt
   ```
4. Run the source code with the command

   ```
   python3 main.py
   ```
5.

- **If you face any difficulty in installing packages, you can run the code on google collab.**