Name: Kushal Sharma
Email: kushals@usc.edu
Course: INF 552 Summer 2020
Assignment: HomeWork2 (K-means & Gaussian Mixture Model(Expectation Maximization)

Data Structure:

1. Pandas dataframe:
    1. To load initial data from the location.
    2. To find max and min range of data points to help in initialization of parameters.
    3. To conver the dataframe to numpy array.
2. Numpy:
    1. To generate random multi-dimensional numbers.
    2. To sort the data points in meshgrid to plot the gaussian form.
    3. To generate identity matrix.
    4. To take dot and matrix multiplication product.
    5. To take log of all the elements in a matrix.
    6. To take inverse of matrix
    7. To find index of maximum of minimum element in a matrix.
    8. To take sum of all elements row or column wise in a matrix.
3. Dictionary:
    1. To store the assignment of points in each cluster.
    2. To store the results from each trials/epochs and take the best result out of all.
4. Matplotlib:
    1. To scatter plot the 2D points alongwith their centroids.
    2. To scatter plot the 2D points and their ellipsoid gaussian form in 2D.
5. Scipy:
    1. To make gaussian form from given value of mean and covariance.

Code-level optimization:

- Calculation of gaussian parameters through random initialization of points as well as passing the centroid value from the best result of K-means. This shows significance of initialization of parameters.
- Use of numpy functions to carry out matrix operations.
- Numpy array is passed so that operation of conversion of dataframe to numpy is not repeated in every method.
- Best results from multiple epochs run is reported and printed.

Challenges:

- Matrix operation on covariance matrix and normal distribution value.
- Faced error like Singular matrix, overflow in exp etc. because of incompatible initialization of parameters.
- Plotting the gaussian forms with using scipy stats library, but couldn’t do it without it.
- Log likelihood estimation

K-Means:

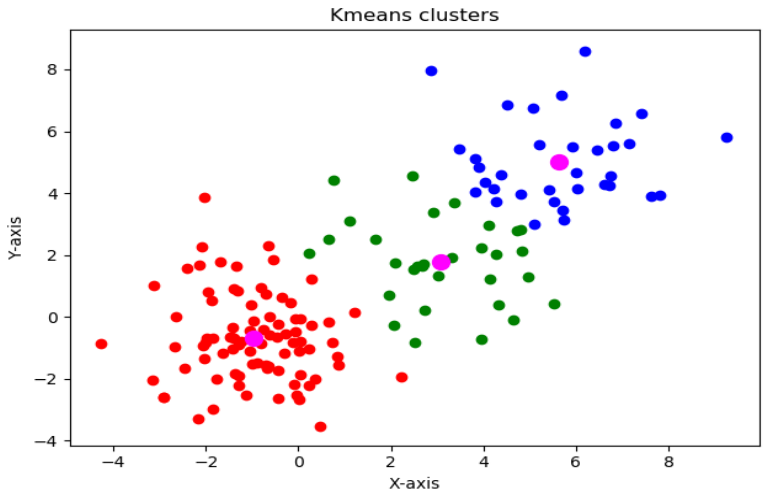
Provision has been given in code to run the algorithm multiple times for different random initialization of centroid value each time. Centroid value and sum of distance between each point and it closest centroid is reported for each epoch run as well best run out of all the epochs. Minimum value of distance is desired.

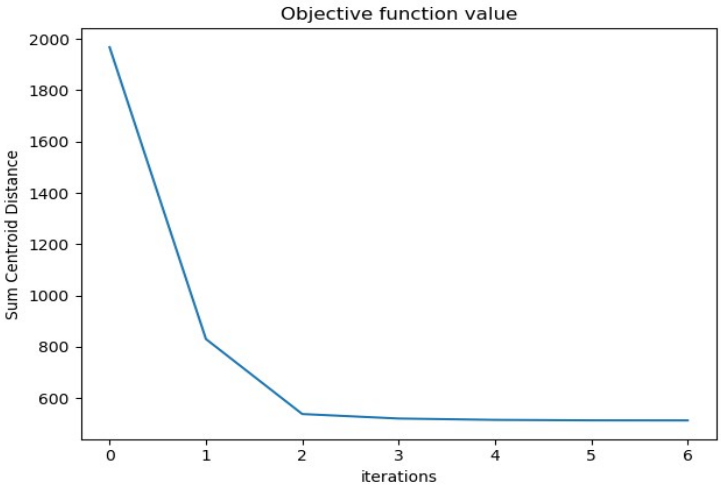
After 20 epochs:

- 5 unique distances were observed, which means there exists at least 5 sub-optimal values.
  - [513.2347825286602, 510.1492200357534, 512.1255485764601, 570.1649153203447, 630.4411691914347]
- Best out of the unique distances observed:
  - Best closest centroid distance: 510.1492200357534
  - Best centroid value:

    [[-0.97476572 -0.68419304]
    [5.62016573 5.02622634]
    [3.08318256 1.77621374]]

The same value for best centroid and best distance was observed even for epochs higher than 20. So, I assume that best values of centroid and distance given above represent optimal value.





GMM:

Two experiments were performed

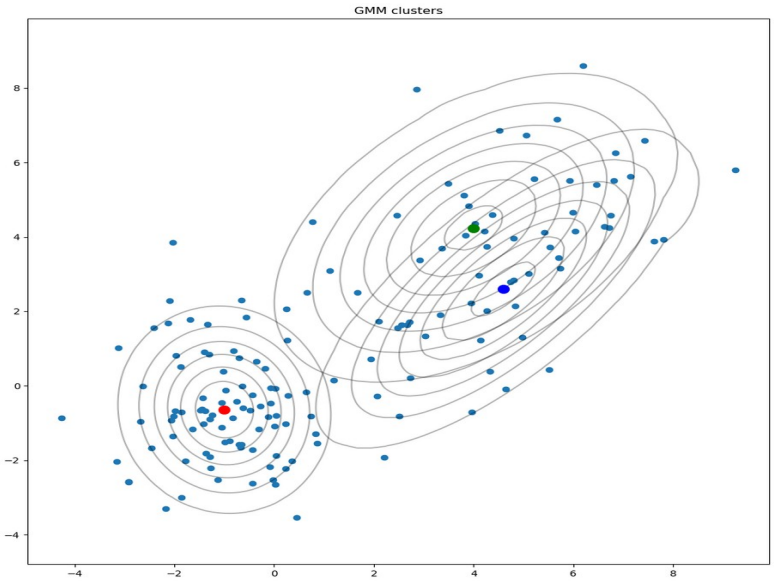
1. Passing the best centroid value from K-means to GMM as mean value for further optimization.
2. Selecting the random value for mean in GMM, run the algorithm for multiple epochs and select the best values out of all runs

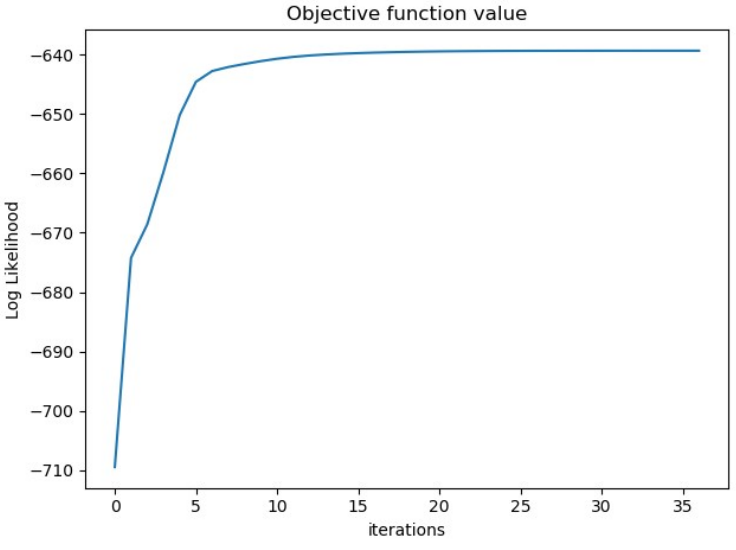
In first case, passing above mentioned best value of centroid from K-means as initial mean value for GMM, we obtain below values for gaussian parameters.

- Best Amplitude:
  [0.55778923 0.19811477 0.244096]
- Best Mean:
  [[-0.99634814 -0.6382264]
  [3.99791979 4.23174788]
  [4.59832759 2.60666628]]
- Best Covariance:
  [[[1.17556918 -0.07574472]
  [-0.07574472 1.99734995]]

  [[4.00895783 2.00548832]
  [2.00548832 4.29233924]]

  [[3.52711584 2.99214197]
  [2.99214197 4.57443295]]]
- Best log likelihood: -639.3683096656534





In second case, random initialization of mean value was selected. Covariance matrix in both case is calculated based on mean value. Amplitude's initialization value is 1 / (no. of clusters) for every data point. This was also run 20 epochs and best out of all values is displayed below. Log likelihood function value is used as termination condition and maximum value of log likelihood is desired.

Due to lot of iteration values GMM run for single epochs, the log likelihood value is matched against its previous value only upto 3 decimal points. Program terminates the iteration loop once it finds previous and new value of log likelihood as same upto 3 decimal points.

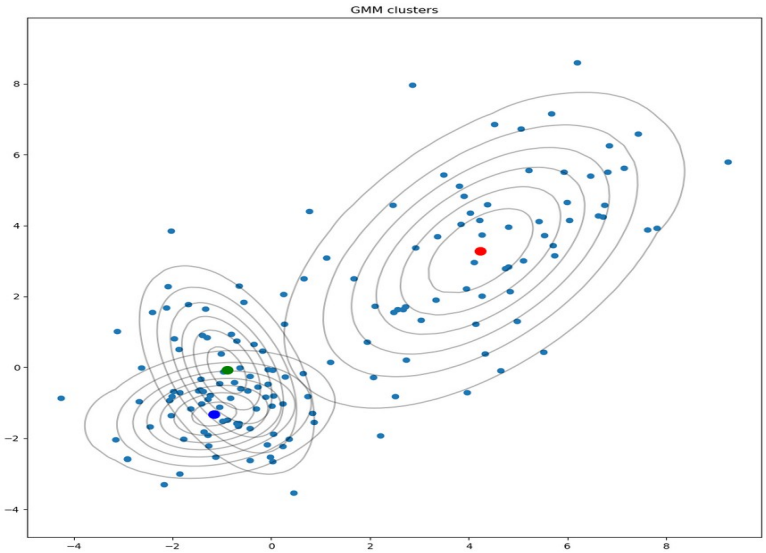
After 20 epochs:

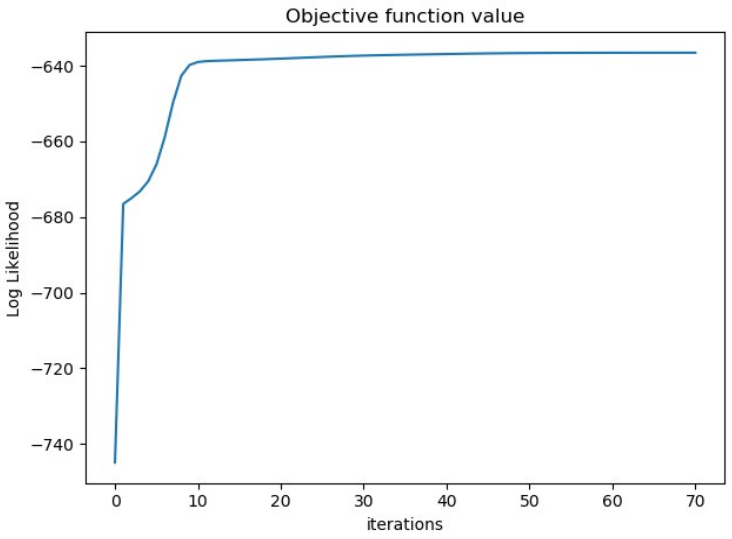
- Total 9 unique log likelihood values were observed, means atleast 9 sub-optimal values:
  [-639.359, -639.384, -636.560, -637.200, -637.199, -639.380, -635.896, -639.003, -637.201]
- Best Amplitude:
  [0.4527444 0.28151238 0.26574322]
- Best Mean:
  [[4.23863343 3.28764504]
  [-0.89804072 -0.07258949]
  [-1.15705656 -1.31471013]]
- Best Covariance:
  [[[4.09405018 2.45988735]
  [2.45988735 5.09161537]]

  [[0.82958574 -0.66297477]
  [-0.66297477 2.28236553]]

  [[1.46221507 0.22191964]
  [0.22191964 0.79986707]]]
- Best log likelihood: -635.8962724946537

Sometimes even better log likelihood value was observed during other runs. That means there exists other optimal value than mentioned above.





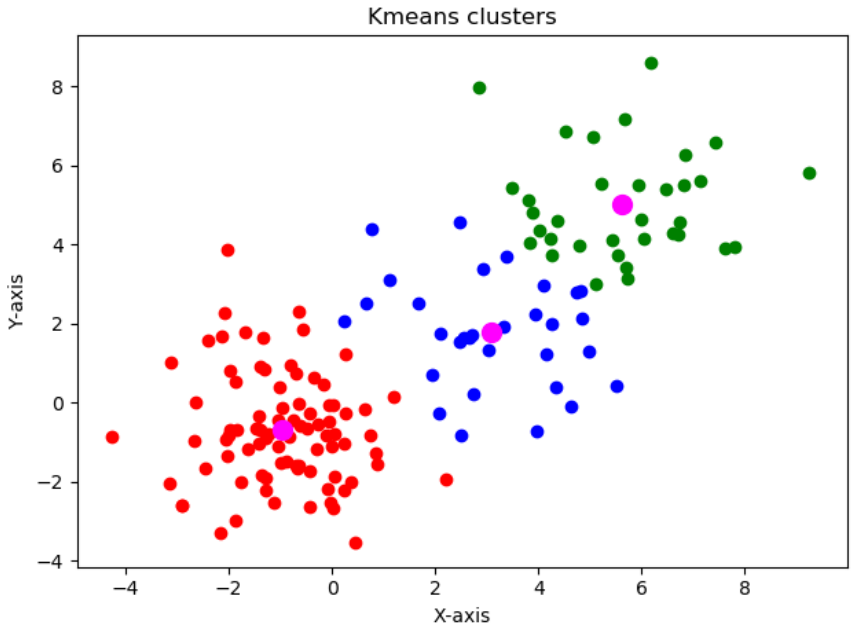
Part 2:

K-means:

By using sklearn.cluster.Kmeans library function, the optimal centroid value and distance was obtained. The result matches with results obtained from above program. Only difference seems to be in processing speed.

```
def sklearn(points, k):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(points)
    new_centroid = kmeans.cluster_centers_
    assignment, distance = kmeans_assignment(points, new_centroid, k)
    plot_clusters(new_centroid, assignment)
    print("Centroid:", new_centroid)
    print("distance:", distance)
```

Results:
Centroid: [[-0.97476572 -0.68419304]
 [3.08318256 1.77621374]
 [5.62016573 5.02622634]]
distance: 510.14922003575344



GMM:

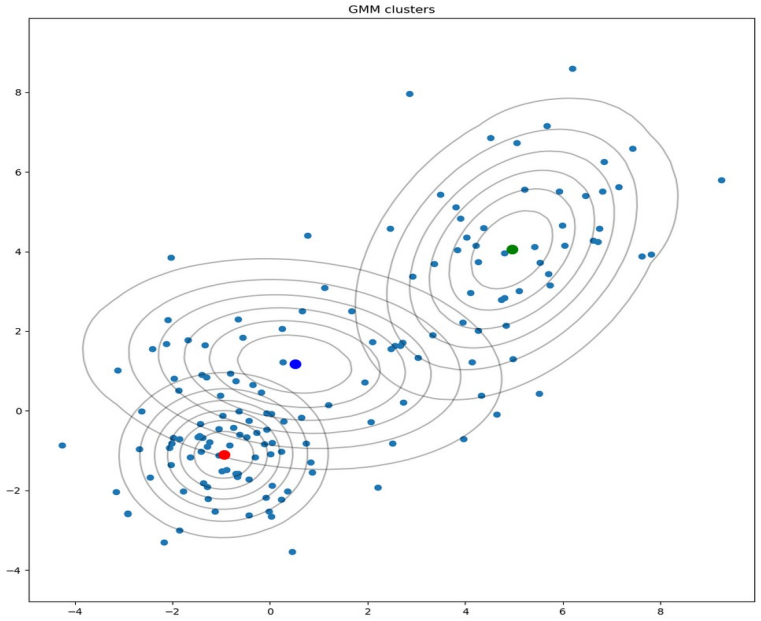
By using sklearn.mixture.GaussianMixture library function, different values of weights, mean and covariance are obtained than what are obtained from above program.

```
def sklearn_gmm(points, k):
    gmm = GaussianMixture(n_components=k)
    gmm.fit(points)
    weights = gmm.weights_
    mean = gmm.means_
    covariance = gmm.covariances_
    log_likelihood = gmm.lower_bound_
    plot_gaussian(mean, covariance, points)
    print("Sklearn_GMM:")
    print("Weights:", weights)
    print("Mean:", mean)
    print("Covariance:", covariance)
    print("Log Likelihood:", log_likelihood)
```

```
Results:
Weights: [0.45113555 0.33867502 0.21018943]
Mean: [[-0.94711092 -1.10556831]
 [ 4.95843099  4.04835699]
 [ 0.50766952  1.17197402]]
Covariance: [[[ 1.1737202   0.01415913]
  [ 0.01415913  1.09211744]]

 [[ 2.66576016  1.18525437]
  [ 1.18525437  3.69804095]]

 [[ 4.63691567 -0.40892068]
  [-0.40892068  1.80204919]]]
Log Likelihood: -4.261243095569497
```



Part 3:

K-means algorithm is very useful and popular in following applications:

- Market Segmentation – Marketers improve their customer base, target area, and segment customers based on their purchase history, interests, or activity monitoring.
- Document Classifications – Documents can be classified in multiple categories based on tags, topics, and content of document. Term frequency is used to identify commonly used terms that help classify the document.
- Image Segmentation – Classification of image into different groups. before applying K-means algorithm, first partial stretching enhancement is applied to the image to improve the quality of the image. Subtractive clustering method is data clustering method where it generates the centroid based on the potential value of the data points. So subtractive cluster is used to generate the initial centers and these centers are used ink-means algorithm for the segmentation of image. Then finally medial filter is applied to the segmented image to remove any unwanted region from the image
- Image Compression – Each pixel location is 3 8-bit integers that specify red, green and blue intensity values. Goal is to reduce the number of colors to user defined number which represent the centroid value of clusters of pixels.

GMM is also found useful in above applications described for k-means as well as below applications:

- Feature extraction from speech data - Voice samples from the subject are used to create a model or template.
- Object tracking of multiple objects - To model relative orientation and translation of pairs of objects from a very large synthetic scene dataset. Objects that are within a certain distance from each other are taken into account, reasoning that far-away objects have weaker relationships.