Programming Assignment 2 Report

PART 1: IMPLEMENTATION

K-Means Algorithm:

Data Structures:

- We utilized the pandas DataFrame structure to read in and parse the input file.
- There were 2 columns ‘x’ and ‘y’ that held the coordinates of each data point.
- We also used a list of integer lists to hold the centroids.
- In the implementation with libraries, we also used the pandas DataFrame structure for holding the points.
- The library also stored the cluster centers in a list of integer lists.

Code-level optimizations:

- We found the min and max of the data points to give a better range for the randomization of the initial centroid cluster points.
- We utilized the distance function to calculate the distance between the points and each cluster in order to find the closest cluster centroid.

Challenges:

- Since we assign our initial cluster centroids randomly, the results often vary between each run of the program since the centers still converge.
- It is possible that they get stuck in a local minimum.
- In order to limit the range of values used to randomize the initial cluster centers, we found the minimum and maximum of the data points to use as the range to randomize from.
- Since the K-Means algorithm assumes the spherical shape of clusters, it does a poor job in clustering the data when the geometrical shape of clusters deviate from spherical shapes.
- The number of clusters must also be pre-defined.

Output:

- The centroid clusters found were (3.083, 1.776), (-0.975, -0.684), and (5.620, 5.026).
- However, these results vary since the initial cluster centers are randomized and don’t always converge to the same centroid.

EM Algorithm:

Data Structures:

- In the implementation without libraries, we used numpy.ndarray object to store the data points.
- This stored each point as an array within an array where each inside array stored the x & y coordinate of a datapoint.
- We also used 1-D and 2-D arrays for storing the means, amplitudes and covariance matrix. We have also used 2-D arrays for storing weights and the posterior probability x_i.
- In the implementation with libraries, we have used dataframe structure from Pandas to read and store the input points.
- There were two columns 0&1 representing the x & y coordinates of each data point.
- We also used arrays to store the means, amplitudes and variances returned by the library functions.

Code-level optimizations:

- We have used the loadtxt function from NumPy to read the data and store them into numpy.ndarray objects as they are faster than using normal lists.

- Additionally, the data was also stored in mathematical notation which helped in improving the accuracy when compared to using normal numbers.
- We have initially initialized mu, sigma and alpha(means, covariance and amplitude respectively) using random values but later updated them to use values from a k-means algorithm in order to improve efficiency and accuracy.
- By using these optimizations, our results differed from the library implementation results by a maximum of 0.2.

Challenges faced:

- The values being generated by our algorithm were initially not too accurate when the variables were initialized with random values.
- The number of iterations taken for the likelihood to converge was also quite high when compared to the library implementation.
- By performing various code refactorings along with the above optimizations, especially through the use of values derived from k-means for initializing the variables, we were able to get more accurate values.

Output:

- The values generated for mean, amplitude and covariance matrices are as follows:

  Mean:

  1. With library:
     [[2.95794243 1.5172694]
      [-0.97675717 -0.65978187]
      [5.35474763 4.71166217]]
  2. Without library:
     [[-0.97569452 -0.64458101]
      [5.44055714 4.80565151]
      [3.10891774 1.67404492]]

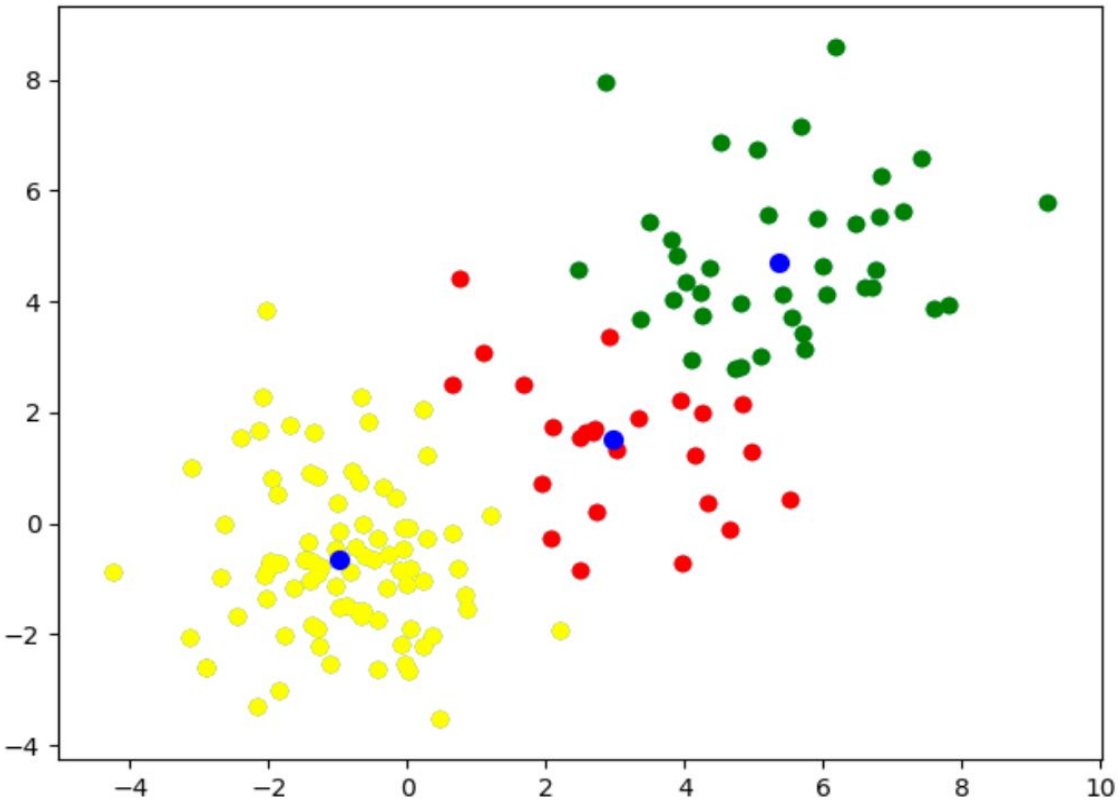
  Amplitude:

  1. With library: [0.17477342 0.56497181 0.26025477]
  2. Without library: [0.56680006 0.24230539 0.19089455]

  Covariance matrices:

  1. With library:
     [[[2.05537151 -0.26590085]
       [-0.26590085 2.08519189]]
      [[1.21681206 -0.11996906]
       [-0.11996906 1.97577318]]
      [[2.3640313   0.45069033]
       [0.45069033 2.33636882]]]
  2. Without library:
     [[[1.21202573e+00 -1.02968692e-01]
       [-1.02968692e-01 2.00921936e+00]]
      [[2.35584460e+00 3.74290084e-01]
       [3.74290084e-01 2.28515063e+00]]
      [[1.97947717e+00 1.07700359e-03]
       [1.07700359e-03 2.38748205e+00]]]

- The plot showing the clusters and means which were obtained in the library implementation is as shown below:



PART 2: SOFTWARE FAMILIARIZATION

K-Means Algorithm

Libraries

- pandas: This library was used for reading in the input file into a DataFrame structure.
- sklearn.cluster KMeans class: To use the KMeans class, you have to initialize it with the number of clusters to use and then utilize the fit() method to fit the data points. In order to output the centroid clusters, you can access the cluster_centers_ attribute.

Implementation Comparison

- The sklearn KMeans library selects the initial centroids with the k-means++ initialization scheme implemented by the scikit-learn library to speed up convergence.
- It initializes centroids to be generally distant from each other which leads to more uniform results.
- Our implementation without the library selects the initial centroids randomly from the range of the min and max of the data points.
- Therefore, our results for the centroid clusters tend to vary between each run, since it is possible that the centroid clusters can converge to a local minimum.

Result Comparison

- The centroid clusters computed in the library implementation were (3.083, 1.776), (-0.975, -0.684), and (5.620, 5.026).
- In our implementation without the KMeans library, the centroid clusters were also sometimes the same as the library implementation.

```
(base) Vanessas-MacBook-Pro-2:PA2 vanessatan$ python kmeans_withoutLibrary.py
Centroids
(-0.9606529070232558, -0.6522184128604651)
(3.2888485605151514, 1.9326883657575757)
(5.738495346032258, 5.164838081193548)
(base) Vanessas-MacBook-Pro-2:PA2 vanessatan$ python kmeans_withLibrary.py
```

- Some runs would vary in the centroids like (3.364, 2.253), (-1.114, -0.631), and (6.555, 5.619). This variation in results could be because the initial cluster centers are randomized, we sometimes get stuck in a local minimum.

```
(base) Vanessas-MacBook-Pro-2:PA2 vanessatan$ python kmeans_withoutLibrary.py
Centroids
(3.3640812818846153,2.2528303625)
(-1.1139953484875,-0.6310498948875)
(6.555367191166667,5.618595879)
(base) Vanessas-MacBook-Pro-2:PA2 vanessatan$ python kmeans_withLibrary.py
Centroids:
(3.083182557032258,1.776213738032258)
(-0.9747657180823526,-0.6841930411764712)
(5.620165734970588,5.026226344176472)
(base) Vanessas-MacBook-Pro-2:PA2 vanessatan$
```

Possible Improvements

- Some possible improvements we could make in our implementation without the library is to initialize our centroid clusters in a similar way to the library initialization.
- We could randomly assign centroid clusters, but also ensure that they are generally distant from each other.

EM Algorithm

Libraries

- pandas: This library was used for reading in the input file into a DataFrame structure in the library implementation part.
- Matplotlib: We used this library for visualizing the scatter plots of the data and the clusters that they were assigned to along with their means in the library implementation part.
- sklearn.mixture GaussianMixture class: To use this library we had to initialize it with the number of clusters to use, and fit it with the data to predict the labels. We were able to output the mean, amplitude and covariance matrix by accessing the means_, weights_ and covariances_ functions of the GaussianMixture class in the library implementation part.
- Numpy: We used the numpy library to implement arrays for storing the various variables and zeros array for initializing the weights. We used the numpy library's linear algebra functions for calculating the determinants, dot products, inverse, sum of arrays and logarithmic values as well in the algorithm implementation without the libraries part.
- Math: We used the math function to get a precise value of pi so that our results are more accurate in the algorithm implementation without the libraries part.

Implementation Comparison

- We initially randomly assigned values for the weights, means, and precisions, but we found that we achieved better accuracy and efficiency by following the sklearn library method of initialization with k-means.
- On implementing the library function, we realized that the actual library function selects points randomly and more uniformly thereby giving different values in different orders on running multiple times.

Result Comparison

- The results from our implementation using libraries were quite similar to the algorithm we wrote without using libraries with a maximum difference of 0.2.

- However, sometimes the order of the data was different. This is because the algorithms assign different data points to different clusters in each implementation.
- The results obtained are as follows:
  With libraries:

```
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw2/EMAlgorithm_withLibrary.py',
The means are:
[[-0.97675717 -0.65978187]
 [ 5.35474763  4.71166217]
 [ 2.95794243  1.5172694 ]]

The amplitudes are:
[0.56497181 0.26025477 0.17477342]

The covariance matrix is:
[[[ 1.21681206 -0.11996906]
  [-0.11996906  1.97577318]]

 [[ 2.3640313   0.45069033]
  [ 0.45069033  2.33636882]]

 [[ 2.05537151 -0.26590085]
  [-0.26590085  2.08519189]]]
```

  Without libraries:

```
>>> runfile('/Users/sudeepthamouniganji/PycharmProjects/hw2/EMAlgorithm_withoutLibrary.py
The means are:
 [[-0.97569452 -0.64458101]
 [ 5.44055714  4.80565151]
 [ 3.10891774  1.67404492]]

The amplitudes are:
 [0.56680006 0.24230539 0.19089455]

The covariance matrix is:
 [[[ 1.21202573e+00 -1.02968692e-01]
  [-1.02968692e-01  2.00921936e+00]]

 [[ 2.35584460e+00  3.74290084e-01]
  [ 3.74290084e-01  2.28515063e+00]]

 [[ 1.97947717e+00  1.07700359e-03]
  [ 1.07700359e-03  2.38748205e+00]]]
```

Possible Improvements

- Some possible improvements we could implement are adding a precision matrix like in the library version.
- A precision matrix is the inverse of the covariance matrix and it would make computing the log-likelihood of new samples more efficient.
- We could also use iteratively find and store the different possible means using k-means and recursively use them during the implementation of the EM algorithm for getting more precise values and a quicker likelihood convergence.

PART 3: APPLICATIONS

K-Means Algorithm

- **Market segmentation:** We can utilize the clustering of similar customers to understand patterns in behaviors, attributes, spending habits, etc. This allows marketing teams to better understand customers to more effectively target and market their products for increasing sales.
- **IT performance monitoring:** We can conduct a real-time performance monitoring of a company by using k-means. Various metrics are given as input to the algorithm, which groups the data into different clusters of unlabeled data. It can then be used to find different patterns in data as well as find any unusual activity. By using k-means, we also have the flexibility of deciding which data to be used in different situations. We can also use the whole data in case we are unclear about what to use in a particular situation or to see how the whole data is combinedly affecting various trends.
- **Gene Therapy:** We can use clustering at different depths to understand how RNA sequencing affects the aging process. The cells are clustered into four different groups by the k-means algorithm. By examining the clusters together, we can understand which RNA sequencing is the best. On further, examining each cluster, the researchers were able to determine how the presence and number of each cell inside each cluster affected the vascular aging process. For example, the number of B lymphocytes was found to be higher in the clusters. They were also able to examine the expression of genes that are associated with atherosclerosis, arthritis, heart failure, osteoporosis, or amyotrophy in all four clusters.
- **Prediction of students' academic performance:** By using the k-means clustering algorithm, we can understand the key characteristics affecting students' performance and use them for future prediction. The k-means algorithm was used along with the euclidean distance in a deterministic model to analyze the dataset of students in a private school having nine courses per semester. The results obtained were used to improve existing methods and perform prediction on the students' performance.

EM Algorithm

- **Structural engineering:** The algorithm can be utilized to help with structural identification by identifying natural vibrations properties of structural systems with sensor data
- **Audio-Visual Scene Analysis:** When audio and visual observations are represented in the same Euclidean space, sound-source localization methods can be utilized to perform sound-source estimation direction onto an image plane. Data clusters of different types of data: audio-visual, visual only, and audio-only can help to identify if there is an active speaker present.
- **Models with latent (unobserved) features:** Hidden Markov Models use latent features and have many real-world applications in speech recognition, handwriting recognition, time series analysis, etc.
- **Psychometrics:** The EM algorithm helps to estimate item parameters and the latent abilities of item response theory models.
- **Medical image reconstruction:** A variant of the EM algorithm called ordered subset EM is used for positron emission tomography, single-photon emission computed tomography, and x-ray computed tomography.

Group Members and Contributions

- Our group members for this programming assignment were Sudeeptha Mouni Ganji (ID: 2942771049) and Vanessa Tan (ID: 4233243951).
- Vanessa and Sudeeptha worked together to research and understand the algorithms.

- After implementation, Sudeeptha and Vanessa worked together to write the report.

Programming:

K-Means algorithm ● Without libraries ● With libraries	 Vanessa Tan Vanessa Tan
EM algorithm using GMM ● Without libraries ● With libraries	 Sudeeptha Mouni Ganji Sudeeptha Mouni Ganji

Report Writing:

Part 1: ● K-Means algorithm ● EM algorithm using GMM	 Vanessa Tan Sudeeptha Mouni Ganji
Part 2: ● K-Means algorithm ● EM algorithm using GMM	 Vanessa Tan Sudeeptha Mouni Ganji
Part 3: ● K-Means algorithm ● EM algorithm using GMM	 Sudeeptha Mouni Ganji Vanessa Tan