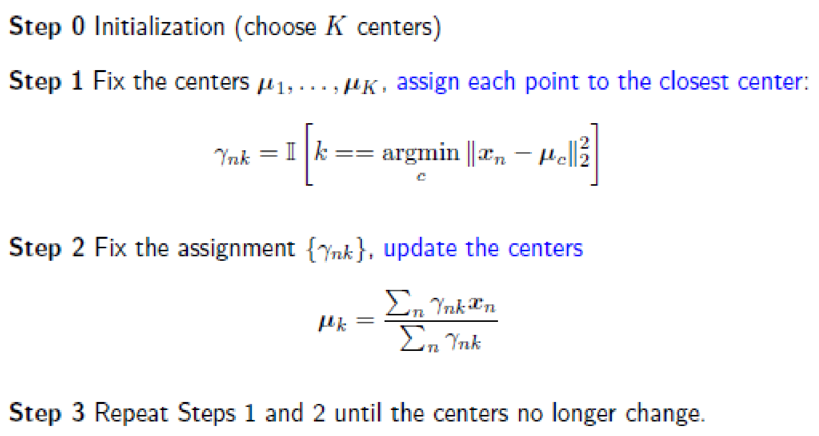
Minh Tran – HW2: K-means and Gaussian Mixture Model Report

1. **<u>Part 1: Implementation:</u>**

a. **Write a program to construct a classification using K-means algorithm:**

- Pycharm IDE is used with Python 3.6 as the programming language. Basic libraries such as numpy and random are used to facilitate the program
- Algorithm:



Step 0 Initialization (choose K centers)

Step 1 Fix the centers $\mu_1, \ldots, \mu_K$, assign each point to the closest center:

$$\gamma_{nk} = \mathbb{I}\left[k == \operatorname*{argmin}_c \|x_n - \mu_c\|_2^2\right]$$

Step 2 Fix the assignment $\{\gamma_{nk}\}$, update the centers

$$\mu_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$$

Step 3 Repeat Steps 1 and 2 until the centers no longer change.

- Data structure:
  i. Data entry is converted from text to numpy array by built-in genfromtxt function so that the training data is a 2D array (size 150*2)
  ii. Dictionary is used to represent cluster in function updateCentroids with as "Cluster Index" as keys and "List of Data Points associated with that Cluster" as values. Looping through each cluster centroid, a distance is calculated between each data point and all cluster centroids. Then, the index of the minimum distance is taken to find out which cluster that point temporarily belongs to. An example of a clustersDict is shown below:
  {0: [[x1,y1], [x5,y5]…] ,
   1: [[x4,y4], [x7,y7],…] ,
   2: [[x2,y2], [x3,y3],…]}
  iii. Cluster centroids is a list of lists: with 3 clusters and 2 dimensions → 3*2 in size.
  iv. Files included:

- ✓ kmeansMT.py - Contains the implementation of k-means from scratch
- ✓ kmeansSklearnMT.py – Contains the implementation of k-means using sklearn library
- ✓ clusters.txt - Training data in txt format.
- ✓ Hw2MinhTranReport.pdf – pdf report file

- Code-level optimization:
  i. To improve code performance, numpy built-in functions are used for numerical calculation: np.sum, np.sqrt, np.mean
  ii. Convergence criteria are three-fold: if maximum number of iterations is reached or if centroid clusters have converged (the difference between new and old centroids are less than error threshold). Here I choose the max number of iterations to be 100, error threshold to be 0.01.
- Challenges:
  i. The uses of "For Loop" may lead to slowdown of the code if the data has many more data points or dimensions. Map Reduce may tackle this challenge as my code is parallel-able.

b. Run K-means program on data file:

- Final cluster centroids:

```
[[-1.03940862 -0.6791968 ]
[ 2.88349711  1.35826195]
[ 5.43312387  4.86267503]]
```

- Number of iterations: 10

c. Implement Gaussian Mixture Models:

- Pycharm IDE is used with Python 3.6 as the programming language. Basic libraries such as numpy and random are used to facilitate the program
- Algorithm:

Step 0 Initialize $\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ for each $k \in [K]$

Step 1 (E-Step) update the "soft assignment" (fixing parameters)

$$\gamma_{nk} = p(z_n = k \mid \boldsymbol{x}_n) \propto \omega_k N(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Step 2 (M-Step) update the model parameter (fixing assignments)

$$\omega_k = \frac{\sum_n \gamma_{nk}}{N} \qquad \boldsymbol{\mu}_k = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n}{\sum_n \gamma_{nk}}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

Step 3 return to Step 1 if not converged

In step 1 (E-step), based on multivariate Gaussian distribution, the detailed expression of probability to implement is

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} \omega_k \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_k|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)}$$

The code consists of 2 main functions "maximizationAlgo" and "expectationAlgo"

- Data structure:
    i. Genfromtxt, and dictionary structure are used similarly to the k-means algorithm, thus will not be repeated here.
    ii. A 2D np array is used to represent membership weight of each data point to each cluster γ_{nk} (dimensions: numPoints * numClusters: 150*3)
    iii. Initialization step – "pseudokmeans.py":
        - ✓ Weights are defined equally for each individual GMM
        - ✓ Clusters are quickly determined using k-means algorithm
        - ✓ Means and covariances are determined accordingly based on k-means clusters.
    iv. M-step - "MaximizationAlgo.py":
        - ✓ Input is data, current membership weight γ_{nk} and number of clusters
        - ✓ Covariances and means are initialized as empty lists, filled up by looping through all points and clusters and reset to empty after looping through all clusters.

- ✓ The goal is to return means, covariances and weights (amps) of all clusters following maximum likelihood estimation's result:

$$\omega_k = \frac{\sum_n \gamma_{nk}}{N} \qquad \boldsymbol{\mu}_k = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n}{\sum_n \gamma_{nk}}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

v. E-step - "expectationAlgo.py":

- ✓ Input is data, and number of clusters
- ✓ The goal is to recalculate the membership weight γ_{nk} maximizing the probability of actual data points happening in the dataset.

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \sum_{k=1}^{K} \omega_k \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}_k|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^{\mathrm{T}}\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)}$$

- Code-level optimization:
  i. To improve code performance, numpy built-in functions are used for numerical calculation: np.sum, np.sqrt, np.mean, np.linalg.inv (to calculate inverse of a matrix), np.linalg.det (to calculate determinant), np.cov (to calculate covariance)

  ii. Convergence criteria are two-fold: if maximum number of iterations is reached or if centroid clusters have converged (the difference between new and old centroids are less than error threshold). Here I choose the max number of iterations to be 500, error threshold to be 0.001.

- Challenges:
  i. Repeatability is an issue as there are no unique solutions if the starting centroid clusters are initialized differently, as it was the case when np.random is used. Final GMM solution can consist of gaussian mixture of different means and covariances each time the code is run. This can be handled by implementing additional algorithms to optimize the initial pickings of cluster centroids.

ii. Global variables are used and this may not be ideal if the code segments grow larger as it is difficult to keep track of the evolution of these variables.

d. Run GMM program on data file:

- Initializations:

i. Weights: :

```
[0.3333333333333333,
0.3333333333333333,
0.3333333333333333]
```

ii. Starting points:

```
[[-1.86133124 -2.99168277],
 [-2.17009237 -3.29231778],
 [-1.01408097  0.38579499]]
```

iii. Initial centroid clusters:

```
[[-0.77179465 -2.07419587],
 [-3.08080151 -2.27246407],
 [ 1.92026584  1.82758165]]
```

iv. Initial Covariances

```
[[[ 0.67064729 -0.15918136]
  [-0.15918136  0.30124688]],
 [[ 0.57084942 -0.67806261]
  [-0.67806261  0.81818866]],
 [[ 9.41304942  5.42477513]
  [ 5.42477513  6.03521988]]]
```

- Number of iterations: 29
- Current weights:

```
[0.51237478 0.03249961 0.45512561]
```

- Means:

```
[[-0.91044319 -0.56424291]
[-3.09129089 -2.26088209]
[ 4.23109997  3.25456174]]
```

- Covariances:

```
[[[ 0.85954402 -0.33142804]
[-0.33142804  1.88726672]],
 [[ 0.45774367 -0.54416113]
 [-0.54416113  0.6570976 ]],
 [[ 4.07397402  2.52145984]
 [ 2.52145984  5.22110935]]]
```

2. <u>Part 2: Software familiarization</u>

a. **Sklearn for Kmeans – kmeansSklearnMT.py:**

✓ Scikit-Learn offers useful built-in methods to implement k-means.

✓ 3 simple steps are needed:

- Import data
- Create model and fit to data:

  kmeans = KMeans(n_clusters=numClusters, random_state=0).fit(data)
- Run command: python kmeansSklearnMT.py clusters.txt 3

✓ Output:

- Cluster 0 - centroid [-0.97476572 -0.68419304]
- Cluster 1 - centroid [3.08318256 1.77621374]
- Cluster 2 - centroid [5.62016573 5.02622634]

✓ The results are comparable to my implementation of k-means approach

b. **Sklearn for GMM – gmmSklearnMT.py:**

✓ Scikit-Learn offers useful built-in methods to implement k-means.

✓ 3 simple steps are needed:

- Import data
- Create model and fit to data:

  gmm = GaussianMixture(n_components=numClusters,

  tol=0.001,max_iter=500,init_params='kmeans')

  gmm.fit(data)
- Run command: python gmmSklearnMT.py clusters.txt 3

✓ Output:

- Gaussian Mixture 0:
  - Mean: [-0.96174678 -0.63618636]
  - Covariance: [[1.23856256 -0.09316563]
  - [-0.09316563 2.02291629]]
  - Weight: 0.5709532595833409
- Gaussian Mixture 1:

        - Mean: [3.2842769 1.9291939]
        - Covariance: [[1.91930032 0.14344635]
        - [0.14344635 2.57669337]]
        - Weight: 0.2156080266842785
    - Gaussian Mixture 2:
        - Mean: [5.62092993 4.99410684]
        - Covariance: [[2.2216601 0.15446244]
        - [0.15446244 2.094973]]
        - Weight: 0.21343871373238094
- ✓ The results are not comparable to my implementation of GMM approach as the initialization steps are different.

3. Applications

Because of its simplicity in coding and implementation, K-means can be applied to continuous numeric data that has a smaller number of dimensions. Clustering analysis is widely used in many fields such as market research, pattern recognition, data analysis, and image processing. A few applications are classifying documents, exploring areas prone to crime, categorizing existing customer bases, detecting and flagging fraud, analyzing public transport data, alerting security threats….

Gaussian mixture models are used a lot when the underlying populations can be explained by a normal distribution and there are many heterogeneous populations, such as height or weight of different ethnicities. The weighting factor may be the percentage of the population that are from each ethnic group as defined above. Other applications are text-independent speaker identification, texture and color for image database retrieval, image classification and segmentation, multimodal biometric verification.