HW 02 – K-means and GMM using EM

06/13/2020

INF 552

ISHAN JOSHI | USC ID 3451334123

K-means Clustering

K-means clustering is an unsupervised machine learning algorithm, which implies that there are no labels provided with the data. Therefore, the algorithm identifies patterns and similarities within the data points and groups them accordingly. For k-means, the grouping is done based on values of centroids. The centroids are determined by calculating the distance of each data point from all the centroids and assigning the data point to the nearest centroid. It is a local search algorithm.

K-means as an instance of the EM algorithm:

The Expectation Maximization algorithm works as follows:

- Consider Problems A and B, both of which are hard if tackled individually.
- However, if we assume a solution to Problem A and use the result to compute Problem B, we find that Problem B becomes easy to solve. The reverse also holds true when we assume the solution to Problem B.
- The Problems A and B are thus coupled.

Thus for k-means clustering,

- Problem A – Find membership for each datapoint
- Problem B – Find the centroids

If we know the centroids for a given number of clusters (Solution to Problem B), we can use those values to find membership of each datapoint, and then recompute the centroids. The repetition of the above steps will lead to a converging solution.

Algorithm:

1. The number of clusters are given (k = 3). Assign values to the three centroids on random.
2. Calculate the distance of each datapoint from each of the three clusters. The label, or cluster that the datapoint belongs to will be the cluster which is closest to the given datapoint. Mathematically,
   Let Xi = datapoint, Ci = centroid
   $$Zi = argmin(||Xi - Ci||^2)$$
   Zi corresponds to the index of the centroid that the datapoint is closest to. Therefore,
   $$Q = \sum_{i=1}^{N} ||Xi - Czi||^2$$
   Minimizing Q will give us the values on convergence of the algorithm. The steps 1 and 2 have to be repeated till convergence.

Drawbacks of K-means:

- If the datapoints are too scattered, the performance of k-means is sub-par.
- If a datapoint has an overlap between two or more clusters, the k-means algorithm is unable to detect the soft membership and assigns the datapoint to only the maximum membership cluster.
- K-means only produces spherical clusters.

These drawbacks can be solved using a more generalised instance of EM algorithms, called Gaussian Mixture Models.

Gaussian Mixture Models:

GMM have gaussian distributions in place of centroids to which the datapoints are assigned. Each gaussian distribution has a set of parameters- mean, variance and amplitude. In multiple dimensions the variance is replaced by the covariance matrix with dimensions dxd. Considering the GMM algorithm as an instance of EM, we divide the algorithm into the following coupled problems:

- Problem A – Calculate the weights of the datapoint regarding each of the k distribution curves. (k = 3 for this assignment). Mathematically,

$$Weight(i,k) = \frac{\text{Amplitude(k)} * \text{N(Xi; Mean(k),Covariance(k))}}{\sum \text{Amplitude(k)} * N(Xi; Mean(k), Covariance(k))}$$

- Problem B – Calculate the mean, covariance and amplitude using the weights calculated in Problem A.

Thus, if we assume an initial solution to Problem A with randomised weights, we can move ahead to calculate the parameters for each of the k gaussian distributions. We then calculate the likelihood or $\sum P(Xi \mid Ck = k)$ for each datapoint and continue the execution of the code until a certain point.

Algorithm:

1. Assign random, normalized weights for each point. This results in a 150x3 matrix, since we have 150 datapoints and 3 clusters as the basis for the assignment.
2. Calculate the mean, variance and amplitude of the gaussian distribution corresponding to the randomized weights.
3. Calculate the likelihood of the given gaussian distribution. To limit the number of iterations, we can decide when to stop the iterations based on the difference in likelihood values of the most recent two iterative runs.
4. If we haven't reached the limit, we update the weights using the parameters calculated for the gaussian curves and repeat step 2 and 3.

Advantages:

- Even if the datapoints are scattered, we can find membership for the curves.
- We can calculate soft membership for each datapoint as likelihood values will be non-zero results.
- Since the cross section of the gaussian curve is ellipsoid, we can create elliptical clusters.

Program Execution: k-means

Data Members:

1. Dataframe – A dataframe is a data structure within the pandas dictionary in python which makes handling and accessing large quantities of data easy for machine learning algorithms. The dataframe in the code is used to store the dataset. It is separated into two colums, x1 and x2 and a third column "label" is added to store the membership values.
2. Arrays – Arrays are used to store data of one data type in a contiguous location. Arrays are used at multiple points in the code to store data at intermediate steps within loops.

Member Functions:

1. Calc_Distance(centroids, coordinates) – This function returns the distance between a particular datapoint and one of the k centroids.
2. Kmeans() – This function is used to assign each datapoint to the nearest cluster, and then recompute the centroids based on these assignments. The function repeats the execution until we reach convergence. It also prints the centroid values at the end of each iteration and also plots the graph to visually see how the centroids move.

Output:

The output for one of the runs of the code are as follows:

Iterations 1 has centroids: [[1.59980907 -0.61556938]

[-0.41832289 -1.35138602]

[-0.16618925 0.61430076]]

Iterations 2 has centroids: [[4.42587348 2.59165257]

[-1.10950353 -1.42817182]

[0.37843434 2.43712163]]

Iterations 3 has centroids: [[4.88590624 3.75049231]

[-0.90897303 -1.11480088]

[-0.18748253 1.68687338]]

Iterations 4 has centroids: [[5.10696642 3.9859176]

 [-0.85422938 -1.20476334]

 [0.07898775 1.55570781]]

Iterations 5 has centroids: [[5.17290392 4.13591368]

 [-0.88753371 -1.17200418]

 [0.39220437 1.49456594]]

Iterations 6 has centroids: [[5.17290392 4.13591368]

 [-0.9771649 -1.16235403]

 [0.53540291 1.39416804]]

Iterations 7 has centroids: [[5.21137495 4.18235064]

 [-0.9771649 -1.16235403]

 [0.61748794 1.40924955]]

Iterations 8 has centroids: [[5.24286685 4.35793218]

 [-0.9771649 -1.16235403]

 [0.83246424 1.33895654]]

Iterations 9 has centroids: [[5.26733709 4.42759858]

 [-1.00825551 -1.06534182]

 [1.15347812 1.35651727]]

Iterations 10 has centroids: [[5.26733709 4.42759858]

 [-1.01958368 -1.00181459]

 [1.31407485 1.36494721]]

Iterations 11 has centroids: [[5.26733709 4.42759858]

 [-1.01980064 -0.85731849]

 [1.74685259 1.39115495]]

Iterations 12 has centroids: [[5.37478472 4.6631958]

 [-1.04399592 -0.80262867]

 [2.22453493 1.4067868]]

Iterations 13 has centroids: [[5.38488145 4.74702111]

 [-1.0502733 -0.7113407]

 [2.56197055 1.30400026]]

Iterations 14 has centroids: [[5.39899032 4.81380531]

 [-1.04542164 -0.67420553]

 [2.75134724 1.29868404]]

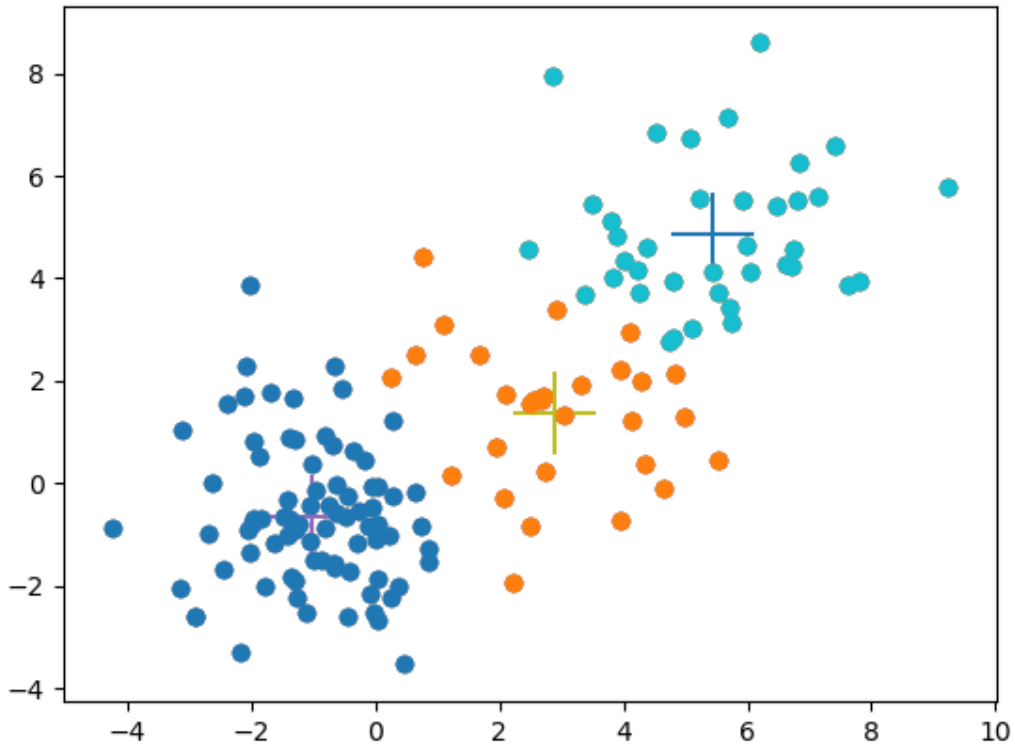
Iterations 15 has centroids: [[5.43312387 4.86267503]

 [-1.02938517 -0.65099575]

[2.88349711 1.35826195]]

Convergence Reached

The final distribution graphically is:



Program Execution- GMM:

Data Members:

1. Matrices – The initial dataset is stored as a 2x150 matrix, which makes each column of the matrix a datapoint (x1,x2). The covariance matrix is stored as a 2x2 matrix.
2. Dataframe – The dataset is input as a dataframe and separated on commas.
3. Arrays – They are used to store values of similar datatypes at contiguous locations at multiple points within the code. The final results are displayed as an array of matrices where each matrix corresponds to the parameter for one of the three clusters.

Member Functions:

1. Fit() – This runs the main execution of the code. It takes a limit as an input to find when to stop iterating through the steps. It calculated the E and M steps of the algorithm and returns the mean, covariance and amplitudes of all three gaussian curves as 1x3 arrays respectively.
2. Calc_pi() – This calculates and returns the amplitude of a particular gaussian curve.
3. Gauss_parameters() – This calculates and returns the mean and covariance matrix of the gaussian curve based on the input weights given to the function.
4. Gauss_PDF() – This returns the probability of a given datapoint for a particular gaussian curve. Essentially, P(Xi | C=k).

Output:

The following is an output for one of the runs of the program:

Iterations:1, Likelihood: -674.72741226675

Iterations:2, Likelihood: -674.518658488016

Iterations:3, Likelihood: -674.3185509862176

Iterations:4, Likelihood: -674.0427095055703

Iterations:5, Likelihood: -673.6654102371946

Iterations:6, Likelihood: -673.1688533914532

Iterations:7, Likelihood: -672.5183703013291

Iterations:8, Likelihood: -671.6173909920243

Iterations:9, Likelihood: -670.2204679735635

Iterations:10, Likelihood: -667.7717035263535

Iterations:11, Likelihood: -663.3107503087188

Iterations:12, Likelihood: -656.2740977667081

Iterations:13, Likelihood: -648.1776671041349

Iterations:14, Likelihood: -642.4573064163261

Iterations:15, Likelihood: -639.9345794938041

Iterations:16, Likelihood: -638.4722672861678

Iterations:17, Likelihood: -637.2191171805092

Iterations:18, Likelihood: -636.1823572330168

Iterations:19, Likelihood: -635.4560212764003

Iterations:20, Likelihood: -635.0200809548548

Iterations:21, Likelihood: -634.7832691831835

Iterations:22, Likelihood: -634.6585344447136

Iterations:23, Likelihood: -634.5904365253745

Iterations:24, Likelihood: -634.5499544673519

Iterations:25, Likelihood: -634.5232103784153

Iterations:26, Likelihood: -634.5036775159614

Iterations:27, Likelihood: -634.4881867271929

Iterations:28, Likelihood: -634.4751197456648

Iterations:29, Likelihood: -634.463609875079

Iterations:30, Likelihood: -634.4531737631524

Iterations:31, Likelihood: -634.4435311980959

Threshold Reached

([matrix([[1.61047448],

[2.52328406]]), matrix([[4.76630103],

[3.31466901]]), matrix([[-0.9388943],

[-0.75728827]])], [matrix([[7.95375787, 4.42705696],

[4.42705696, 6.79003256]]), matrix([[2.94785842, 2.33215549],

[2.33215549, 4.75632185]]), matrix([[1.05544532, -0.1192287],

[-0.1192287 , 1.57201431]])], [0.16989911993941056, 0.3305930600147831, 0.4995078200458062])

Challenges:

1. Initially I attempted to do all calculations based on matrices. This would mean that the dataset would be a 2x150 matrix, the weights 150x3 and so on. This would've reduced the number of loops needed to calculate means and covariance matrices in the program. However, it caused computational difficulties in calculating the covariance matrix as the output was not a 2x2 matrix. Thus, I reverted to using loops to traverse through the data arrays and calculate values.
2. Finding the number of iterations that the code should run for was a result of hit and trial. I initially tried to base the exit on the number of iterations based on the initial random assignment of weights. However, this value varied greatly and thus the determining condition was broken down to calculating the difference between the two most recent values of the likelihood. I set the difference threshold to be less than 0.01, but this can be changed to get a more detailed distribution.
3. Normalizing the weights at each E step was also challenging, as due to the dimensions of the dataset multiple nested loops had to be run.
4. For k-means, the limiting condition was based on finding the distance between the two most recent values of the centroids. This implied that the

centroids were unchanged. An initial attempt to compare the two values caused bugs which were rectified using this method.

5. I attempted to save the figures generated in each iteration of the k-means algorithm to show how the centroid points move after each iteration. This causes the code to store multiple images which need to be deleted before running the code the next time.