EM algorithm

# example code

|  |
| --- |
| import numpy as np  from scipy import random  from matplotlib.patches import Ellipse  import matplotlib.transforms as transforms  from scipy.stats import multivariate\_normal  class GMM():      def \_\_init\_\_(self, k, dim, init\_mu=None, init\_sigma=None, init\_pi=None, colors=None):          '''          Define a model with known number of clusters and dimensions.          input:              - k: Number of Gaussian clusters              - dim: Dimension              - init\_mu: initial value of mean of clusters (k, dim)                         (default) random from uniform[-10, 10]              - init\_sigma: initial value of covariance matrix of clusters (k, dim, dim)                            (default) Identity matrix for each cluster              - init\_pi: initial value of cluster weights (k,)                         (default) equal value to all cluster i.e. 1/k              - colors: Color valu for plotting each cluster (k, 3)                        (default) random from uniform[0, 1]          '''          self.k = k          self.dim = dim          if(init\_mu is None):              init\_mu = random.rand(k, dim)\*20 - 10          self.mu = init\_mu          if(init\_sigma is None):              init\_sigma = np.zeros((k, dim, dim))              for i in range(k):                  init\_sigma[i] = np.eye(dim)          self.sigma = init\_sigma          if(init\_pi is None):              init\_pi = np.ones(self.k)/self.k          self.pi = init\_pi          if(colors is None):              colors = random.rand(k, 3)          self.colors = colors        def init\_em(self, X):          '''          Initialization for EM algorithm.          input:              - X: data (batch\_size, dim)          '''          self.data = X          self.num\_points = X.shape[0]          self.z = np.zeros((self.num\_points, self.k))        def e\_step(self):          '''          E-step of EM algorithm.          '''          for i in range(self.k):              self.z[:, i] = self.pi[i] \* multivariate\_normal.pdf(self.data, mean=self.mu[i], cov=self.sigma[i])          self.z /= self.z.sum(axis=1, keepdims=True)        def m\_step(self):          '''          M-step of EM algorithm.          '''          sum\_z = self.z.sum(axis=0)          self.pi = sum\_z / self.num\_points          self.mu = np.matmul(self.z.T, self.data)          self.mu /= sum\_z[:, None]          for i in range(self.k):              j = np.expand\_dims(self.data, axis=1) - self.mu[i]              s = np.matmul(j.transpose([0, 2, 1]), j)              self.sigma[i] = np.matmul(s.transpose(1, 2, 0), self.z[:, i] )              self.sigma[i] /= sum\_z[i]        def log\_likelihood(self, X):          '''          Compute the log-likelihood of X under current parameters          input:              - X: Data (batch\_size, dim)          output:              - log-likelihood of X: Sum\_n Sum\_k log(pi\_k \* N( X\_n | mu\_k, sigma\_k ))          '''          ll = []          for d in X:              tot = 0              for i in range(self.k):                  tot += self.pi[i] \* multivariate\_normal.pdf(d, mean=self.mu[i], cov=self.sigma[i])              ll.append(np.log(tot))          return np.sum(ll)        def plot\_gaussian(self, mean, cov, ax, n\_std=3.0, facecolor='none', \*\*kwargs):          '''          Utility function to plot one Gaussian from mean and covariance.          '''          pearson = cov[0, 1]/np.sqrt(cov[0, 0] \* cov[1, 1])          ell\_radius\_x = np.sqrt(1 + pearson)          ell\_radius\_y = np.sqrt(1 - pearson)          ellipse = Ellipse((0, 0),              width=ell\_radius\_x \* 2,              height=ell\_radius\_y \* 2,              facecolor=facecolor,              \*\*kwargs)          scale\_x = np.sqrt(cov[0, 0]) \* n\_std          mean\_x = mean[0]          scale\_y = np.sqrt(cov[1, 1]) \* n\_std          mean\_y = mean[1]          transf = transforms.Affine2D() \              .rotate\_deg(45) \              .scale(scale\_x, scale\_y) \              .translate(mean\_x, mean\_y)          ellipse.set\_transform(transf + ax.transData)          return ax.add\_patch(ellipse)      def draw(self, ax, n\_std=2.0, facecolor='none', \*\*kwargs):          '''          Function to draw the Gaussians.          Note: Only for two-dimensionl dataset          '''          if(self.dim != 2):              print("Drawing available only for 2D case.")              return          for i in range(self.k):              self.plot\_gaussian(self.mu[i], self.sigma[i], ax, n\_std=n\_std, edgecolor=self.colors[i], \*\*kwargs)  import numpy as np  import matplotlib.pyplot as plt  from scipy import random  from scipy.stats import multivariate\_normal  import scipy  import matplotlib  def gen\_data(k=3, dim=2, points\_per\_cluster=200, lim=[-10, 10]):      '''      Generates data from a random mixture of Gaussians in a given range.      Will also plot the points in case of 2D.      input:          - k: Number of Gaussian clusters          - dim: Dimension of generated points          - points\_per\_cluster: Number of points to be generated for each cluster          - lim: Range of mean values      output:          - X: Generated points (points\_per\_cluster\*k, dim)      '''      x = []      mean = random.rand(k, dim)\*(lim[1]-lim[0]) + lim[0]      for i in range(k):          cov = random.rand(dim, dim+10)          cov = np.matmul(cov, cov.T)          \_x = np.random.multivariate\_normal(mean[i], cov, points\_per\_cluster)          x += list(\_x)      x = np.array(x)      if(dim == 2):          fig = plt.figure()          ax = fig.gca()          ax.scatter(x[:,0], x[:,1], s=3, alpha=0.4)          ax.autoscale(enable=True)      return x  def plot(title):      '''      Draw the data points and the fitted mixture model.      input:          - title: title of plot and name with which it will be saved.      '''      fig = plt.figure(figsize=(8, 8))      ax = fig.gca()      ax.scatter(X[:, 0], X[:, 1], s=3, alpha=0.4)      ax.scatter(gmm.mu[:, 0], gmm.mu[:, 1], c=gmm.colors)      gmm.draw(ax, lw=3)      ax.set\_xlim((-12, 12))      ax.set\_ylim((-12, 12))        plt.title(title)      plt.savefig(title.replace(':', '\_'))      plt.show()      plt.clf()  # Generate random 2D data with 3 clusters  X = gen\_data(k=3, dim=2, points\_per\_cluster=1000)  # Create a Gaussian Mixture Model  gmm = GMM(3, 2)  # Training the GMM using EM  # Initialize EM algo with data  gmm.init\_em(X)  num\_iters = 30  # Saving log-likelihood  log\_likelihood = [gmm.log\_likelihood(X)]  # plotting  plot("Iteration:  0")  for e in range(num\_iters):      # E-step      gmm.e\_step()      # M-step      gmm.m\_step()      # Computing log-likelihood      log\_likelihood.append(gmm.log\_likelihood(X))      print("Iteration: {}, log-likelihood: {:.4f}".format(e+1, log\_likelihood[-1]))      # plotting      plot(title="Iteration: " + str(e+1)) |

# testing result

|  |
| --- |
|  |