*""" ECSE 509 Fall 2020 - Project  
  
By: Alexander Fernandes  
ID: 260960205  
  
This program was written in the Python programming language to carry out the project deliverables.   
"""*# Import libraries  
import matplotlib.pyplot as plt  
from scipy.stats import expon  
import numpy as np  
  
  
def pdf\_exp(x, l):  
 *"""PDF of exponential distribution  
  
 Input* ***:param*** *x: random variable* ***:type*** *x: float* ***:param*** *l: parameter* ***:type*** *l: float  
  
 Output* ***:return****: exponential pdf* ***:rtype****: float  
 """* if x >= 0:  
 pdf = l \* np.exp(-l \* x)  
 else:  
 pdf = 0  
 return pdf  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Start of program execution  
  
 # -------------------------------------------------------------------------------------------------------------  
 # a) Plot the pdf of the exponential mixture.  
 # -------------------------------------------------------------------------------------------------------------  
  
 # Assume the given values:  
 lambda1 = 1  
 lambda2 = 3  
 pi1 = 0.25  
  
 # Setup variables to plot pdf exponential mixture  
 plot\_range = [0, 6] # x values  
 x = np.linspace(plot\_range[0], # x start  
 plot\_range[1], # x end  
 1000) # number of points in x  
  
 # Calculate exponential mixture f(x; params) over values x  
 pdf\_mix\_given = np.zeros(x.shape) # initialize vector of same length x with zeros  
 for i in range(len(x)):  
 # Here we use the pdf\_mix function using our given parameters  
 # f(x; params) = pi1\*f1(x; lambda1) + p2\*f2(x; lambda2)  
 pdf\_mix\_given[i] = pi1 \* pdf\_exp(x[i], lambda1) + (1-pi1) \* pdf\_exp(x[i], lambda2)  
  
 # plot the pdf exponential mixture  
 fig1, axa = plt.subplots()  
 axa.plot(x, pdf\_mix\_given, label='pdf')  
 plt.title('PDF of exponential mixture\nlambda1 = 1, lambda2 = 3, pi1 = 0.25')  
 plt.grid()  
 plt.legend()  
 axa.set\_xlabel('x')  
 axa.set\_ylabel('pdf(x)')  
  
 # -------------------------------------------------------------------------------------------------------------  
 # c) Write a program which applies the EM algorithm you derived.  
 # -------------------------------------------------------------------------------------------------------------  
  
 # Total number of samples  
 N = 20  
  
 # randomly sample the latent variables  
 # z latent variable = 0 or 1 for sampling from the mixture distribution  
 z = np.zeros(N)  
 for i in range(N):  
 # z = 0 with probability pi1  
 # z = 1 with probability 1-pi1  
 z[i] = np.random.binomial(1, 1-pi1)  
  
 # randomly sample from distribution using the latent variables  
 # y = g(x) observed samples from hidden variable x distribution  
 y = np.zeros(N)  
 for i in range(N): # Sample N = 20 times  
 # update y[i] values based on latent variable z[i] to sample from corresponding exponential distribution  
 if z[i] == 0: # observe from f1(x; lambda1)  
 y[i] = expon.rvs(scale=1/lambda1, size=1)  
 elif z[i] == 1: # observe from f2(x; lambda2)  
 y[i] = expon.rvs(scale=1/lambda2, size=1)  
 else:  
 print('Something is wrong with z, should only have values of 0 or 1!')  
  
 # Implement EM algorithm  
  
 totalIterations = 51 # number of iterations to run  
  
 # Three different initial parameters to try on the sampled observed data  
 l1\_0 = [0.5, 0.01, 0.1]  
 l2\_0 = [4, 10, 50]  
 p1\_0 = [0.3, 0.1, 0.6]  
  
 # plot log-likelihood iterations with different initial parameters on the same plot  
 fig2, axd = plt.subplots()  
  
 # Repeat EM algorithm using same data but with different initial parameters  
 for t in range(3):  
 # Variables used  
 l1 = np.zeros(totalIterations+1) # lambda1 parameter  
 l2 = np.zeros(totalIterations+1) # lambda2 parameter  
 p1 = np.zeros(totalIterations+1) # pi1 parameter: pi1 + pi2 = 1  
 p2 = np.zeros(totalIterations+1) # pi2 parameter  
  
 # Initialize parameters of the unknown mixture distribution  
 l1[0] = l1\_0[t]  
 l2[0] = l2\_0[t]  
 p1[0] = p1\_0[t]  
 p2[0] = 1 - p1[0]  
 print('Initial parameter set #', t)  
  
 # Run and repeat EM algorithm  
 for j in range(0, totalIterations):  
 # E-step  
 # Compute tau, the recommender membership probabilities for the corresponding mixed distributions  
 tau = np.zeros(N) # latent variable z = 0 (for pdf1)  
 for i in range(N):  
 # tau = pi2\*f2(y; lambda2) / ( pi1\*f1(y; lambda1) + pi2\*f1(y; lambda2) ) using current parameters j  
 tau[i] = (p2[j]\*pdf\_exp(y[i], l2[j])) / (p1[j]\*pdf\_exp(y[i], l1[j]) + p2[j]\*pdf\_exp(y[i], l2[j]))  
  
 # M-step  
 # Calculate next parameters[j+1] given current parameters[j]  
 # Calculate lambda1[j+1]  
 num = 0  
 den = 0  
 for i in range(N):  
 num += 1-tau[i]  
 den += (1-tau[i])\*y[i]  
 l1[j+1] = num / den # lambda1 = sum( 1-tau[i] ) / sum( (1-tau[i])\*y[i] )  
  
 # Calculate lambda2[j+1]  
 num = 0  
 den = 0  
 for i in range(N):  
 num += tau[i]  
 den += tau[i]\*y[i]  
 l2[j+1] = num / den # lambda2 = sum( tau[i] ) / sum( tau[i]\*y[i] )  
  
 # Calculate pi1[j+1]  
 num = 0  
 den = 0  
 for i in range(N):  
 num += tau[i]  
 den += (1-tau[i])  
 p1[j+1] = 1 / (num/den + 1) # p1 = 1 / ( sum(tau[i])/sum(1-tau[i]) + 1 )  
  
 # Calculate pi2[j+1]  
 p2[j+1] = 1 - p1[j+1] # p2 = 1 - p1  
  
 # -------------------------------------------------------------------------------------------------------------  
 # d) Compute the log-likelihood of the incomplete set at each iteration and plot it to verify that it increases  
 # monotonically.  
 # -------------------------------------------------------------------------------------------------------------  
  
 L = np.zeros(totalIterations) # log-likelihood function  
 for j in range(totalIterations):  
 L[j] = 0  
 for i in range(N):  
 # Compute log-likelihood function  
 # L(params) = sum( p1\*f1(y; lambda1) + p2\*f2(y; lambda2) )  
 L[j] += np.log(p1[j] \* pdf\_exp(y[i], l1[j]) + p2[j] \* pdf\_exp(y[i], l2[j]))  
  
 # plot log-likelihood iterations  
 paramstr = 'paramset ' + str(t) + ': l1\_0 = ' + str(l1\_0[t]) + ', l2\_0 = ' + str(l2\_0[t]) + ', p1\_0 = ' + str(p1\_0[t])  
 axd.plot(range(1, len(L)), L[1:], 'o-', label=paramstr)  
 axd.set\_title('log-likelihood vs iterations')  
 axd.grid()  
 axd.set\_xlabel('iteration')  
 axd.set\_ylabel('L')  
 axd.legend()  
  
 # -------------------------------------------------------------------------------------------------------------  
 # e) For some iterations plot the estimate of the exponential mixture pdf.  
 # -------------------------------------------------------------------------------------------------------------  
  
 # Setup variables to plot pdf estimate  
 pdf\_est1 = np.zeros(x.shape) # initial parameters  
 pdf\_est2 = np.zeros(x.shape) # first iteration  
 pdf\_est3 = np.zeros(x.shape) # last iteration  
  
 # Obtain the pdf values over x  
 for i in range(len(x)):  
 # Here we use the pdf\_mix function using our given parameters  
 pdf\_est1[i] = p1[0] \* pdf\_exp(x[i], l1[0]) + p2[0] \* pdf\_exp(x[i], l2[0])  
 pdf\_est2[i] = p1[2] \* pdf\_exp(x[i], l1[2]) + p2[2] \* pdf\_exp(x[i], l2[2])  
 pdf\_est3[i] = p1[-1] \* pdf\_exp(x[i], l1[-1]) + p2[-1] \* pdf\_exp(x[i], l2[-1])  
  
 # plot the pdf iterations  
 fig3, axe = plt.subplots()  
 axe.plot(y, np.zeros(N), 'x', label='observed data')  
 axe.plot(x, pdf\_est1, label='initial estimate')  
 axe.plot(x, pdf\_est2, label='first iteration')  
 axe.plot(x, pdf\_est3, label='final iteration')  
 axe.set\_title('PDF of exponential mixture of three iterations\n' + paramstr)  
 axe.grid()  
 axe.legend()  
 axe.set\_xlabel('x')  
 axe.set\_ylabel('pdf(x)')  
  
 # ---------------------------------------------------  
 # Print metrics of the data and calculated parameters  
 # ---------------------------------------------------  
  
 y0avg = 0  
 n0 = 0  
 y1avg = 0  
 n1 = 0  
 for i in range(N):  
 if z[i] == 0:  
 y0avg += y[i]  
 n0 += 1  
 else:  
 y1avg += y[i]  
 n1 += 1  
  
 print('latent variable z', z)  
 print('observed sample y', y)  
 print('l1', l1)  
 print('l2', l2)  
 print('p1', p1)  
 print('l1[0] ', l1[0])  
 print('l1[1] ', l1[1])  
 print('l1[-1]', l1[-1])  
 print(' ', n0/y0avg)  
 print('l2[0] ', l2[0])  
 print('l2[1] ', l2[1])  
 print('l2[-1]', l2[-1])  
 print(' ', n1/y1avg)  
 print('p1[0] ', p1[0])  
 print('p1[1] ', p1[1])  
 print('p1[-1]', p1[-1])  
 print(' ', 1-np.mean(z))  
  
 plt.show()  
 # End of program execution