MACHINE LEARINING

LAB ASSESSMENT – VI

15BCE0082 VOLETI RAVI

Implement Gaussian Mixture Model Using the Expectation Maximization.

CODE:

```
import numpy as np


class GMM:


    def __init__(self, k = 3, eps = 0.0001):
        self.k = k ## number of clusters
        self.eps = eps ## threshold to stop `epsilon`


        # All parameters from fitting/learning are kept in a named tuple
        from collections import namedtuple


    def fit_EM(self, X, max_iters = 1000):


        # n = number of data-points, d = dimension of data points
        n, d = X.shape


        # randomly choose the starting centroids/means
        ## as 3 of the points from datasets
        mu = X[np.random.choice(n, self.k, False), :]
```

```
# initialize the covariance matrices for each gaussians
Sigma= [np.eye(d)] * self.k

# initialize the probabilities/weights for each gaussians
w = [1./self.k] * self.k

# responsibility matrix is initialized to all zeros
# we have responsibility for each of n points for eack of k gaussians
R = np.zeros((n, self.k))

### log_likelihoods
log_likelihoods = []

P = lambda mu, s: np.linalg.det(s) ** -.5 ** (2 * np.pi) ** (-X.shape[1]/2.) \
    * np.exp(-.5 * np.einsum('ij, ij -> i',\
        X - mu, np.dot(np.linalg.inv(s) , (X - mu).T).T ) )

# Iterate till max_iters iterations
while len(log_likelihoods) < max_iters:

  # E - Step

  ## Vectorized implementation of e-step equation to calculate the
  ## membership for each of k -gaussians
  for k in range(self.k):
    R[:, k] = w[k] * P(mu[k], Sigma[k])

  ### Likelihood computation
```

```
log_likelihood = np.sum(np.log(np.sum(R, axis = 1)))

log_likelihoods.append(log_likelihood)

## Normalize so that the responsibility matrix is row stochastic
R = (R.T / np.sum(R, axis = 1)).T

## The number of datapoints belonging to each gaussian
N_ks = np.sum(R, axis = 0)


# M Step
## calculate the new mean and covariance for each gaussian by
## utilizing the new responsibilities
for k in range(self.k):

   ## means
   mu[k] = 1. / N_ks[k] * np.sum(R[:, k] * X.T, axis = 1).T
   x_mu = np.matrix(X - mu[k])

   ## covariances
   Sigma[k] = np.array(1 / N_ks[k] * np.dot(np.multiply(x_mu.T,  R[:, k]), x_mu))

   ## and finally the probabilities
   w[k] = 1. / n * N_ks[k]
# check for onvergence
if len(log_likelihoods) < 2 : continue
if np.abs(log_likelihood - log_likelihoods[-2]) < self.eps: break
```

```
        ## bind all results together
        from collections import namedtuple
        self.params = namedtuple('params', ['mu', 'Sigma', 'w', 'log_likelihoods', 'num_iters'])
        self.params.mu = mu
        self.params.Sigma = Sigma
        self.params.w = w
        self.params.log_likelihoods = log_likelihoods
        self.params.num_iters = len(log_likelihoods)

        return self.params

    def plot_log_likelihood(self):
        import pylab as plt
        plt.plot(self.params.log_likelihoods)
        plt.title('Log Likelihood vs iteration plot')
        plt.xlabel('Iterations')
        plt.ylabel('log likelihood')
        plt.show()

    def predict(self, x):
        p = lambda mu, s : np.linalg.det(s) ** - 0.5 * (2 * np.pi) **\
                (-len(x)/2) * np.exp( -0.5 * np.dot(x - mu , \
                        np.dot(np.linalg.inv(s) , x - mu)))
        probs = np.array([w * p(mu, s) for mu, s, w in \
            zip(self.params.mu, self.params.Sigma, self.params.w)])
        return probs/np.sum(probs)
```

```
def demo_2d():

    # Load data
    #X = np.genfromtxt('data1.csv', delimiter=',')
    ### generate the random data
    np.random.seed(3)
    m1, cov1 = [9, 8], [[.5, 1], [.25, 1]] ## first gaussian
    data1 = np.random.multivariate_normal(m1, cov1, 90)

    m2, cov2 = [6, 13], [[.5, -.5], [-.5, .1]] ## second gaussian
    data2 = np.random.multivariate_normal(m2, cov2, 45)

    m3, cov3 = [4, 7], [[0.25, 0.5], [-0.1, 0.5]] ## third gaussian
    data3 = np.random.multivariate_normal(m3, cov3, 65)
    X = np.vstack((data1,np.vstack((data2,data3))))
    np.random.shuffle(X)
#    np.savetxt('sample.csv', X, fmt = "%.4f",  delimiter = ",")
    ####
    gmm = GMM(3, 0.000001)
    params = gmm.fit_EM(X, max_iters= 100)
    print params.log_likelihoods
    import pylab as plt
    from matplotlib.patches import Ellipse

    def plot_ellipse(pos, cov, nstd=2, ax=None, **kwargs):
        def eigsorted(cov):
            vals, vecs = np.linalg.eigh(cov)
```

```
        order = vals.argsort()[::-1]
        return vals[order], vecs[:,order]

    if ax is None:
        ax = plt.gca()

    vals, vecs = eigsorted(cov)
    theta = np.degrees(np.arctan2(*vecs[:,0][::-1]))

    # Width and height are "full" widths, not radius
    width, height = 2 * nstd * np.sqrt(abs(vals))
    ellip = Ellipse(xy=pos, width=width, height=height, angle=theta, **kwargs)

    ax.add_artist(ellip)
    return ellip

def show(X, mu, cov):

    plt.cla()
    K = len(mu) # number of clusters
    colors = ['b', 'k', 'g', 'c', 'm', 'y', 'r']
    plt.plot(X.T[0], X.T[1], 'm*')
    for k in range(K):
      plot_ellipse(mu[k], cov[k],  alpha=0.6, color = colors[k % len(colors)])



fig = plt.figure(figsize = (13, 6))
fig.add_subplot(121)
```

```
    show(X, params.mu, params.Sigma)
    fig.add_subplot(122)
    plt.plot(np.array(params.log_likelihoods))
    plt.title('Log Likelihood vs iteration plot')
    plt.xlabel('Iterations')
    plt.ylabel('log likelihood')
    plt.show()
    print gmm.predict(np.array([1, 2]))


if __name__ == "__main__":

    demo_2d()
    from optparse import OptionParser

    parser = OptionParser()
    parser.add_option("-f", "--file", dest="filepath", help="File path for data")
    parser.add_option("-k", "--clusters", dest="clusters", help="No. of gaussians")
    parser.add_option("-e", "--eps", dest="epsilon", help="Epsilon to stop")
    parser.add_option("-m", "--maxiters", dest="max_iters", help="Maximum no. of iteration")
    options, args = parser.parse_args()

    if not options.filepath : raise('File not provided')

    if not options.clusters :
        print("Used default number of clusters = 3" )
        k = 3
    else: k = int(options.clusters)
```

```
if not options.epsilon :
   print("Used default eps = 0.0001" )
   eps = 0.0001
else: eps = float(options.epsilon)

if not options.max_iters :
   print("Used default maxiters = 1000" )
   max_iters = 1000
else: eps = int(options.maxiters)

X = np.genfromtxt(options.filepath, delimiter=',')
gmm = GMM(k, eps)
params = gmm.fit_EM(X, max_iters)
print params.log_likelihoods
gmm.plot_log_likelihood()
print gmm.predict(np.array([1, 2]))
```

OUTPUT:

```
xelese@xelese-Lenovo-Y50-70: ~/Machine Learning/GMM
xelese@xelese-Lenovo-Y50-70:~/Machine Learning/GMM$ python GMM.py
GMM.py:113: RuntimeWarning: covariance is not positive-semidefinite.
  data1 = np.random.multivariate_normal(m1, cov1, 90)
GMM.py:116: RuntimeWarning: covariance is not positive-semidefinite.
  data2 = np.random.multivariate_normal(m2, cov2, 45)
GMM.py:119: RuntimeWarning: covariance is not positive-semidefinite.
  data3 = np.random.multivariate_normal(m3, cov3, 65)
[-931.457086088967, -496.48852371740338, -459.33914688320544, -452.0181991849052
6, -451.17650595193993, -448.73153830461285, -428.21188748537418, -373.731963991
36657, -296.38814029334884, -184.92988425156602, -145.12717607796412, -143.94873
831421006, -143.94886545905223, -143.9488655473433]
```