TP - Modèles de Mélanges Gaussiens

Télécharger le code your_GMM.py depuis votre intranet

En vous référant à la page 5 du cours, écrire la méthode qui génère des données et qui est conforme au modèle de mélange passé en argument

```
def my_GMM_generate(P,Mean,Cov,N,Visualisation=False):
    ''' P: les probabilité a priori des clusters'''
    '' Mean : la matrice contenant les K centroïdes''
    '' Cov : le tenseur contenant les K matrice de covariances ''
    K,p = np.shape(C)
    # insérer votre code ici

    #
    if Visualisation: #on visualise les deux premières coordonnées
        plt.figure(figsize=(8,8))
        for k in range(K):
            bornes[k+1]+=bornes[k]
            plt.plot(X[bornes[k]:bornes[k+1],0],
                     X[bornes[k]:bornes[k+1],1],
                     colors[k]+'o',markersize=4,markeredgewidth=3)
            plt.plot(Mean[k,0],Mean[k,1],'kx',markersize=10,markeredgewidth=3)
        plt.xlim(-10, 10)
        plt.ylim(-10,10)
        plt.show()

    return X, y
```

TP - Modèles de Mélanges Gaussiens

Expliquer et justifier les étapes de calcul de la méthode logsumexp()

```
def logsumexp(X):

    X_max = max(X)
    if math.isinf(X_max):
        return -float('inf')

    acc = 0
    for i in range(X.shape[0]):
        acc += math.exp(X[i] - X_max)

    return math.log(acc) + X_max
```

Déduire le calcule réalisé par la méthode LogSumExp()

```
def LogSumExp(Log_Vrais_Gauss):

    K,N = np.shape(Log_Vrais_Gauss)

    logsomme = np.zeros(N)
    for n in range(N):
        logsomme[n] = logsumexp(Log_Vrais_Gauss[:,n])

    return logsomme
```

TP - Modèles de Mélanges Gaussiens

Compléter la méthode `my_GMM_p_a_posteriori()`

```
def my_GMM_p_a_posteriori(X,K,P,Mean,Cov):

    Log_Vrais_Gauss = np.zeros((K,N))

    ############ insérer votre code ici ################

    ############
    return Proba_Clusters,LogVrais
```

Déduire le calcule réalisé par la méthode `my_GMM_predict()`

```
def my_GMM_predict(X,K,P,Mean,Cov):

    Proba_Clusters, LogVrais =
my_GMM_p_a_posteriori(X,K,P,Mean,Cov)

    y = np.argmax(Proba_Clusters,axis=0)

    return y,LogVrais
```

TP - Modèles de Mélanges Gaussiens

Compléter la méthode `def my_GMM_fit()`

```
def my_GMM_fit(X,K,Visualisation=False,Seuil=0.0000001,Max_iterations = 100):
    N,p = np.shape(X)

    # INITIALISATION D'UN PREMIER MODÈLE
    P, Mean, Cov = my_GMM_init(X,K)

    if Visualisation :
        print("P init = ",P)
        print("Mean init = ",Mean)
        print("Cov init = ",Cov)

    iteration = 0
    Log_Vrais_Gauss = np.zeros((K,N))
    Nk = np.zeros(K)
    New_Mean = np.zeros((K,p))
    New_Cov = np.zeros((K,p,p))
    New_P = np.zeros(K)

    LOGVRAIS=np.zeros(Max_iterations+1)
    LOGVRAIS[0] = -100000
```

... / ...

TP - Modèles de Mélanges Gaussiens

```
                … / …
    while iteration < Max_iterations:
        iteration +=1
        ###############################################################
        # E step : estimation des données manquantes
        #          affectation des données aux clusters les plus proches
        Proba_Clusters, LOGVRAIS[iteration] = my_GMM_p_a_posteriori(X,K,P,Mean,Cov)

        if np.abs(LOGVRAIS[iteration] - LOGVRAIS[iteration-1]) / np.abs(LOGVRAIS[iteration]) < Seuil:
            print("itération =",iteration,"BREAK")
            break

        ###############################################################
        # M Step : calcul du nouveau GMM

        # les centroïdes
        for k in range(K):
            ##### compléter ici

        # les matrices de covariance
        for k in range(K):
            ##### compléter ici

        # les proba des clusters
        ##### compléter ici

        Mean = New_Mean
        P = New_P
        Cov = New_Cov

return P, Mean, Cov, LOGVRAIS[1:iteration]
```