Rapport SY19 TP01 - Classification et melange

Thomas ABASSI & Joan MOESCH

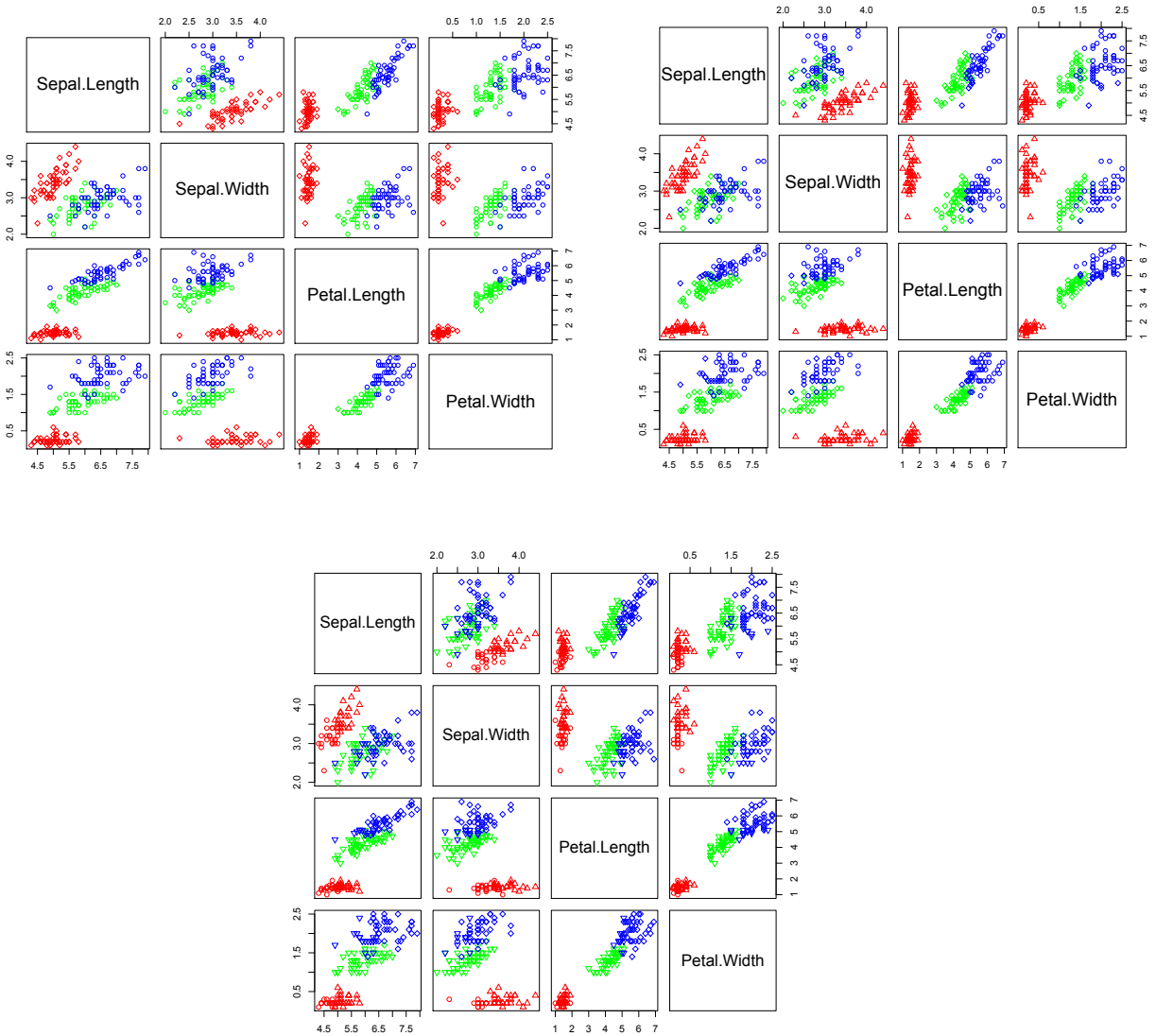
November 24, 2011

Le but de ce TP est la comprehension et l'utilisation de differentes methodes pour la classification non supervisee. En effet, nous analyserons tout d'abord l'algorithme des centres mobiles a travers la fonction *kmeans* de R.
Apres avoir apprecie les possibilites et les limites de cette methode, nous etudierons le modele de melanges grace aux algorithmes EM et CEM. Cette etude portera sur des donnees synthetiques puis sur des donnees reelles et seront effectuees sur dans le cas monodimensionnel et multidimensionnel.

1 La methode des Centres Mobiles

1.1 La fonction kmeans

Nous utilisons la fonction R *kmeans* avec un nombre de partition de K = 2,3 puis 4 . Voici les resultats obtenus :



Chaque couleur correspond aux vrais classes des donnees alors que les symboles designent respectivement les classes determinees par la methodes de K-means.
Nous remarquons que les resultats sont satisfaisants dans le cas ou le nombre de partition est egale au nombre de classe. Dans le cas contraire, une classe est typiquement scindee en 2 ou au contraire 2 classes sont fusionees. D'autre part, nous remarquons que les resultats varient sur plusieurs executions de la fonction. En effet cette derniere est trop dependante des premiers centres choisis en debut d'algorithme.
Il peut donc etre interessant de trouver une methode qui exercerait une moyenne des resultats de *kmeans* et une methode qui nous indiquerait comment choisir le bon nombre de partition.

1.2 Stabilite

Afin d'evaluer la stabilite de la methode des centres mobiles nous ecrivons la fonction suivante :

```
intra <- function()
    res <- matrix( nrow=100, ncol=1);
    for( i in 1:100)
            K <- kmeans(donneesnum, 3);
           res[i,1] <- sum(Kwithinss);
    m <- min( res[,1])
    M <- max( res[,1])
    moy <- mean(res[,1])
    return (list(intra=res, min=m, max=M, avg =moy) );
```

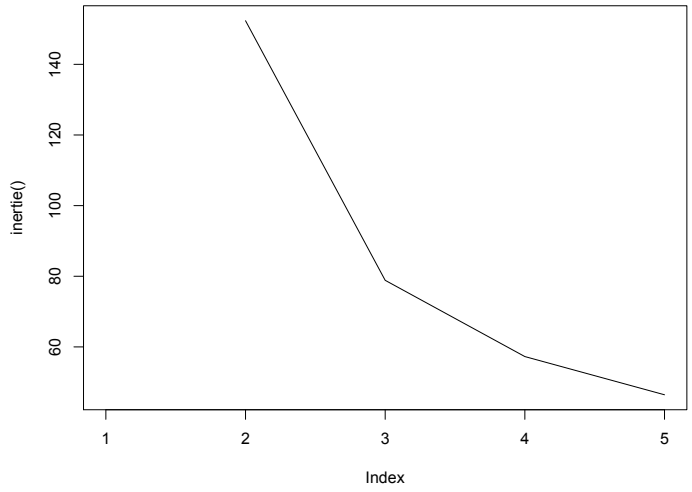
Ainsi nous obtenons les resultats suivants :
$W_{min} = 78.85144$
$W_{max} = 142.7535$
$W_{moy} = 89.7148$

Visuellement, le partitionnement est different a chaque execution de la methode k-means, certains sont meilleurs que d'autres. La comparaison des valeurs min, moy et max de l'inertie intra classe confirme la grande variation de cette methode.

1.3 Nombre de classes optimale

Afin de selectionner le nombre de classes optimale, nous implementons en une fonction R la methode du coude :

```
inertie <- function()
    res <- matrix(nrow=4, ncol=1);
    for(j in 2:5)
            for( i in 1:100 )
                K = kmeans(donneesnum, j);
                res[j] = sum(Kwithinss);
    return(res);
```



La methode du coude nous permet ainsi de visualiser que $k = 3$ classes est le nombre de classes optimal pour la classification. En effet, tant que la diminution de l'intertie intraclasse est importante, on continue d'augmenter le nombre de classes jusqu'a n'obtenir que de plus petites variations. L'abcisse de ce changement de variations est le nombre optimal de partitionnement.

1.4 Comparaison avec la partition reelle

L'inertie intra classe correspond a la variance pour chaque classe (sans le $\frac{1}{nk}$) Pour 1 classe donnee, cette inertie est donc la variance du nuage multiplie par (n-1) car VAR est variance empirique corrigee sous R.

setosa = iris[1:50,1:4]
w1 = sum(diag(49 * var(setosa))) = 15.151

versicolor = iris[51:100,1:4]
w2 = sum(diag(49 * var(versicolor))) = 30.6164

virginia = iris[101:150,1:4]
w3 = sum(diag(49 * var(virginia))) = 43.53

l'inertie totale est donc donnee par : $w = (w1 + w2 + w3) = 89.2974$
On avait $W_{moy} = 89.7148$ et $W_{min} = 78.85144$, l'inertie reelle est sans surprise inferieure a celle trouvee par les kmeans.

Si on calcule les inertie intra-classe des partitions trouvees par la methode des K-means, on obtient :

$$W_1 = 26.51234, W_2 = 33.21364, W_3 = 36.37902$$

Ces valeurs sont proches des inerties intra-classe reelles. Elles peuvent etre inferieures car la methode des k-means peut parfois trouver de meilleures classes que les reelles.

2 Algorithmes EM et CEM dans le cas monodimensionnel

2.1 Donnees synthetiques

2.1.1 Implementation des algorithmes EM et CEM

```
# modèles de mélange en 1D , K nombre de composants du modèle
gmixtmono <- function(donnees, K, param=NULL, fCEM=FALSE)
{
    # modèle de mélange gaussien unidimensionnel
    if (is.vector(donnees))
    {
        # initialisation des varibles locales
        n <- length(donnees)
        t_ik <- matrix(NA, n, K)
        c_ik <- matrix(NA, n, K)
        epsilon <- 10^(-12)
        nbiteration <- 0
        nL <- epsilon
        L <- 0

        # initialisation arbitraire des paramètres
        if (is.null(param))
        {
            for( k in 1:K )
            {
                pi_k <- 1/K
                mu_k <- mean( donnees[round((k-1)*n/K+1) : round(k*n/K)] )
                sigma_k <- var( donnees[round((k-1)*n/K+1) : round(k*n/K)] )

                param[[k]] <- list(p = pi_k, mu = mu_k, sigma = sigma_k)
            }
        }

        # tant que le point de convergence n'est pas atteint
        while( abs(nL - L) >= epsilon && nbiteration < 100 )
        {
            # Mise a jour des parametres
            nbiteration = nbiteration + 1
              print(nbiteration)
            L <- nL

            # étape E
            for( k in 1:K )
                t_ik[,k] <-param[[k]]$p*dnorm(donnees, param[[k]]$mu, param[[k]]$sigma)

            t_ik = t_ik / rowSums(t_ik)

            # étape C
            if( fCEM )
                for( i in 1:n )
                {
                        c_ik[i,] <- 0
                        c_ik[i,which.max(t_ik[i,])] <- 1
                }
            else
                c_ik <- t_ik

            # étape M
            for( k in 1:K )
            {
                pi_k <- 1/n * sum(c_ik[,k])
                mu_k <- ( t(c_ik[,k]) %*% donnees ) / sum(c_ik[,k])
                sigma_k <- sqrt( ( t(c_ik[,k]) %*% (donnees - mu_k)^2 ) / sum(c_ik[,k]) )

                param[[k]] <-list(p=pi_k, mu=as.numeric(mu_k), sigma=as.numeric(sigma_k))
            }

            # Calcul de la nouvelle vraissemblance
            fk <- matrix(0,n)
```

```
            for( k in 1:K )
            {
                pi_k <- 1 / n * sum(c_ik[,k])
                mu_k <- ( t(c_ik[,k]) %*% donnees ) / sum(c_ik[,k])
                sigma_k <- sqrt( ( t(c_ik[,k]) %*% (donnees - mu_k)^2 ) / sum(c_ik[,k]) )

                param[[k]] <-list(p=pi_k, mu=as.numeric(mu_k), sigma=as.numeric(sigma_k))
            }

            # Calcul de la nouvelle vraissemblance
              fk <- matrix(0,n)
              for( k in 1:K )
                   fk=fk+ param[[k]]$p*dnorm(donnees,param[[k]]$mu,param[[k]]$sigma)

              nL = sum(log(fk))
        }
         # Classification des donnees selon les c_ik
         clu <- matrix(rep(1:K,1), ncol = K, nrow = n, byrow = T)
         mat <- c_ik * clu
         cluster <- apply(mat,1,sum)

         param[[K+1]] <- cluster
         param[[K+2]] <- nbiteration
         return(param)
    }
}
```

2.1.2 Comparaison des algorithmes EM et CEM

A partir des donnees synthetiques $x < -c(rnorm(1000), rnorm(1000, mean = 6, sd = 5))$, nous obtenons les resultats suivants :

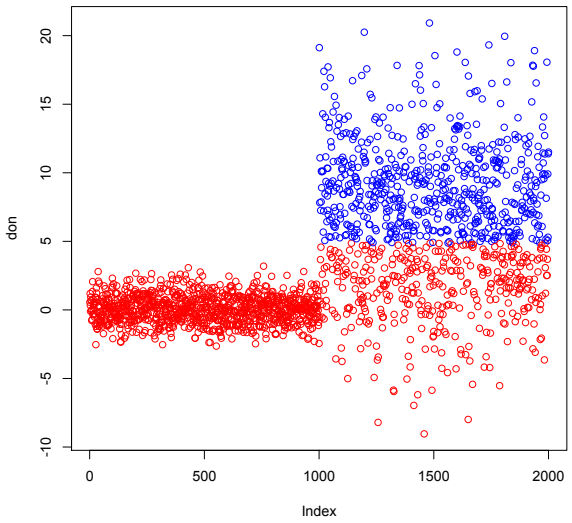
EM <- gmixtmono(x,2) : Avec une precision de 10^{-12}, l'algorithme EM termine en 49 iterations :
$\pi_1 = 0.5051534$ $\pi_2 = 0.4948466$
$\mu_1 = 0.07185754$ $\mu_2 = 6.28883$
$\sigma_1 = 1.001384$ $\sigma_2 = 4.915321$

CEM <- gmixtmono(x,2,NULL,T) : Avec une precision de 10^{-12}, l'algorithme CEM termine en 10 iterations :
$\pi_1 = 0.6045$ $\pi_2 = 0.3955$
$\mu_1 = 0.1126677$ $\mu_2 = 7.78811$
$\sigma_1 = 1.121608$ $\sigma_2 = 4.289954$

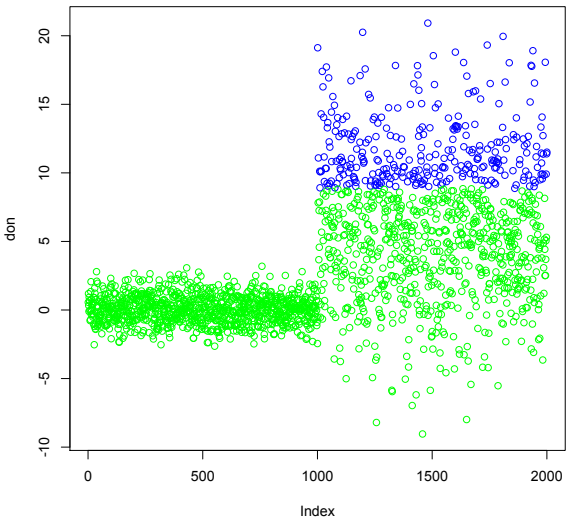
D'une maniere generale, ces algorithmes retrouvent de maniere satisifaisante les proportions ainsi que les parametres des 2 melanges gaussiens.
En comparant, on se rend compte que l'algorithme EM donne de meilleurs resulats que CEM. En effet, la moyenne et la variance trouvees par ce dernier est plus proche de la realite (0 et 1, 5 et 6).
Cependant, il est a noter que pour des resultats quand meme bons, l'algorithme CEM converge bien plus rapidement (10 iterations contre 49).

2.1.3 Comparaison des algorithmes EM, CEM et Kmeans

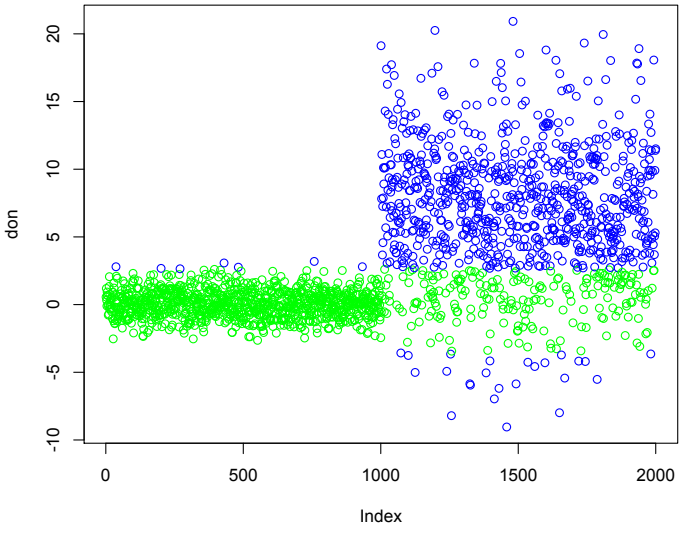
Afin de comparer les classifications entre kmeans, EM et CEM, nous ecrivons :
$EM < -gmixtmono(donnnees, 2)$
$CEM < -gmixtmono(donnnees, 2, NULL, T)$
$k < -kmeans(donnees, 2)$



$plot(donnees, col = c(\text{"}red\text{"}, \text{"}blue\text{"})[kcluster])$



$plot(donnees, col = c(\text{"}red\text{"}, \text{"}blue\text{"})[EM[[3]]])$



$plot(donnees, col = c(\text{"}red\text{"}, \text{"}blue\text{"})[CEM[[3]]])$

Le premier graphique (en rouge) represente la classification effectuee par *kmeans*. Les 2 autres (en vert) representent respectivement les classifications faites par EM et CEM.
Ainsi, on remarque que la classification selon l'algorithme CEM est la meilleure. En effet, la methode des *kmeans* donne une separation en x = 5 des donnees. Les donnees inferieures a cette droite seront du 1er melange et ceux au dessus de la droite du 2eme. De la meme maniere EM separe les donnees en x = 8.

En revanche, la classification CEM regroupe les donnees en 0 (plus ou moins 2) et le reste des donnees. Cette separation se rapproche bien plus de notre melange d'origine : un groupe de moyenne 0 avec une variance reduite (1) et un autre de 5 avec une variance plus grande (6).

2.2 Donnees reelles

2.2.1 Application des algorithmes EM et CEM pour la classification de pixels

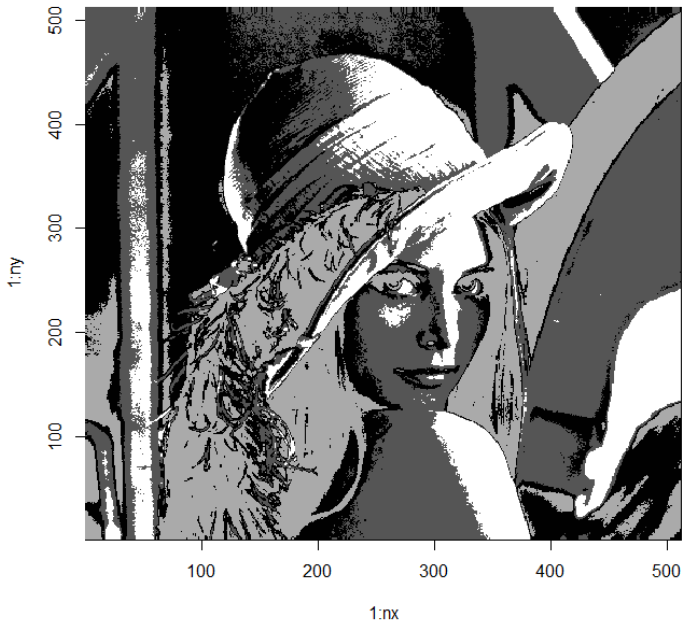
Afin de pouvoir utiliser nos algorithmes EM et CEM, nous transformons la matrice d'image lena en un vecteur, puis nous appliquons nos differents algorithmes pour un modele de K = 4 melanges :

$y < -immat2imvec(lena)$
$EM < -gmixtmono(yvec, 4)$ et $CEM < -gmixtmono(yvec, 4, NULL, T)$

Nous transformons ensuite ces resultats en matrice a 2 dimensions pour les afficher :

$imEM < -imvec2immat(EM[[5]])$ et $imCEM < -imvec2immat(EM[[5]])$
$implotbw(imEM)$ et $implotbw(imCEM)$

Nous n'avons pas pu terminer car des NaN aparraissent dans les algorithmes et les données obtenues ne nous permettent pas de tracer les images. Avec les kmeans, nous obtenons :



3 Algorithmes EM et CEM dans le cas multidimensionnel

3.1 Implementation des algorithmes EM et CEM

Ces algorithmes répètent les étapes 'Expectation', 'Maximisation' ou 'Expectation', 'Classification', 'Maximisation' Les algorithmes s'arrêtent lorsque la variation de la log vraisemblance passe au dessous d'un seuil ou au bout d'un certain nombre d'itérations.
L'implémentation de ces algorithmes fait 130 lignes de codes. Le code ne sera donc pas rapporté ici, néanmoins les points clés sont :

Calcul de t_{ik} : $t_{ik} = \frac{\Pi_k \varphi(x_i \mu_k \Sigma_k)}{\sum_l \Pi_l \varphi(x_i \mu_l \Sigma_l)}$

Calcul de Π_k : $\Pi_k = \frac{\sum_{i=1}^{n} t_{ik}}{n}$

Calcul de μ_k : $\mu_k = \frac{\sum_{i=1}^{n} t_{ik} x_i}{\sum_{i=1}^{n} t_{ik}}$

Calcul de Σ_k : $\Sigma_k = \frac{\sum_{i=1}^{n} t_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^{n} t_{ik}}$

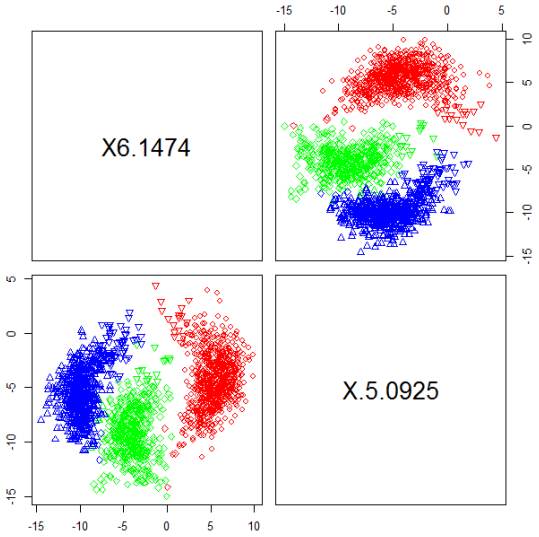
Calcul de la log vraisemblance : $L_C = \sum_{i,k} z_{i,k} log(\Pi_k \phi(x_i \mu_k \Sigma_k))$

3.2 Premiere comparaison entre l'algorithme des centres mobiles, l'algorithme EM, et l'algorithme CEM

On peut remarquer que l'algorithme CEM converge plus vite que l'algorithme EM (EM : 49 iterations pour atteindre une variation de la log vraisemblance inférieure a 10^{-6} avec 3 classes. CEM : 12 itérations pour le même epsilon avec le même nombre de classes)
On peut remarquer que l'algorithme CEM donne des résultats similaires a l'algorithme des centres mobiles. Cela peut s'expliquer par le fait que l'algorithme des centres mobiles correspond à un cas particulier simple de l'algorithme CEM.

Si les partitions obtenues grace à CEM et les centres mobiles d'une part et EM d'autre part sont differentes pour $K = 2$, en revanche pour $K = 3$ et $K = 4$, les partitions sont a peu près similaires. Nous avons stoppé l'algorithme EM à 50 iterations pour $K = 2$. Peut être qu'avec un plus grand nombre d'iterations, nous aurions obtenu des partitions plus proches.
La partition en 4 classes semble être la meilleure, même si la partition initiale etait de 3 classes. Les 3 classes initiales ont bien été reconstruites et une 4eme classe comprend les elements 'génants'.

Ci-dessous, la division en 4 classes obtenue à partir de l'algorithme CEM.



3.3 Seconde comparaison entre l'algorithme des centres mobiles, l'algorithme EM, et l'algorithme CEM

Ces methodes donnent des resultats a peu pres similaires. Les partitions obtenues respectent bien la partition originale. Ci-dessous, les differentes formes representent les differentes classes et quant au couleurs, dans la premiere figure elles sont en fonction de l'èspece réelle et dans la seconde figure elles sont fonction du sexe réel.