Mélange de Bernoulli

Jingzhuo HUI, Gabriel Moran

24/10/2018

Modèle

Considérons un vecteur aléatoire binaire $\boldsymbol{x} \in [0,1]^p$ de p variables x_j suivant chacune une distribution de Bernoulli $\mathcal{B}(\mu_j)$. La distribution du vecteur s'exprime comme:

$$p(\boldsymbol{x}|\boldsymbol{\mu}) = \prod_{j=1}^{p} \mu_j^{x_j}(1-\mu_j)^{1-x_j},$$

avec $\boldsymbol{x} = (x_1, \cdots, x_p)^T$ et $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_p)^T$.

Soit une distribution mélange à K composantes de Bernoulli

$$p(\boldsymbol{x}|\boldsymbol{\pi}, \boldsymbol{M}) = \sum_{k=1}^{K} \pi_k p(\boldsymbol{x}|\boldsymbol{\mu}_k)$$

où les π_k sont les proportions du mélange et les $p(\boldsymbol{x}|\boldsymbol{\mu}_k)$ sont des distributions de Bernoulli multivariées de paramètres $\boldsymbol{\mu}_k = (\mu_{k1}, \cdots, \mu_{kp})^T$, et $M = \{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_K\}^T$ la matrice des paramètres des densités de classes.

Dans la suite nous considérerons

- un échantillon observé $X = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ issu de cette distribution mélange,
- des variables latentes $Z = \{z_1, \cdots, z_n\}$ indiquant la composante d'origine de chaque $\boldsymbol{x}_i$.

Exercice 1

Considérons un mélange à 3 composantes de Bernoulli mélangées en proportions égales $\pi_1 = \pi_2 = \pi_3$.

Simulons une matrice M de proportions dont les 3 lignes et les 50 colonnes décrivent 3 vecteurs des proportions d'un mélange de Bernoulli dans un espace de dimension 50.

```
set.seed(3)
K<-3
p<-50
n<-200
pi<-matrix(c(1/3,1/3,1/3))
M<-matrix(runif(K*p),K,p)
M[K,]<-1-M[1,]
```

Simulons $Z = \{z_1, \cdots, z_n\}$ pour $n = 200$.

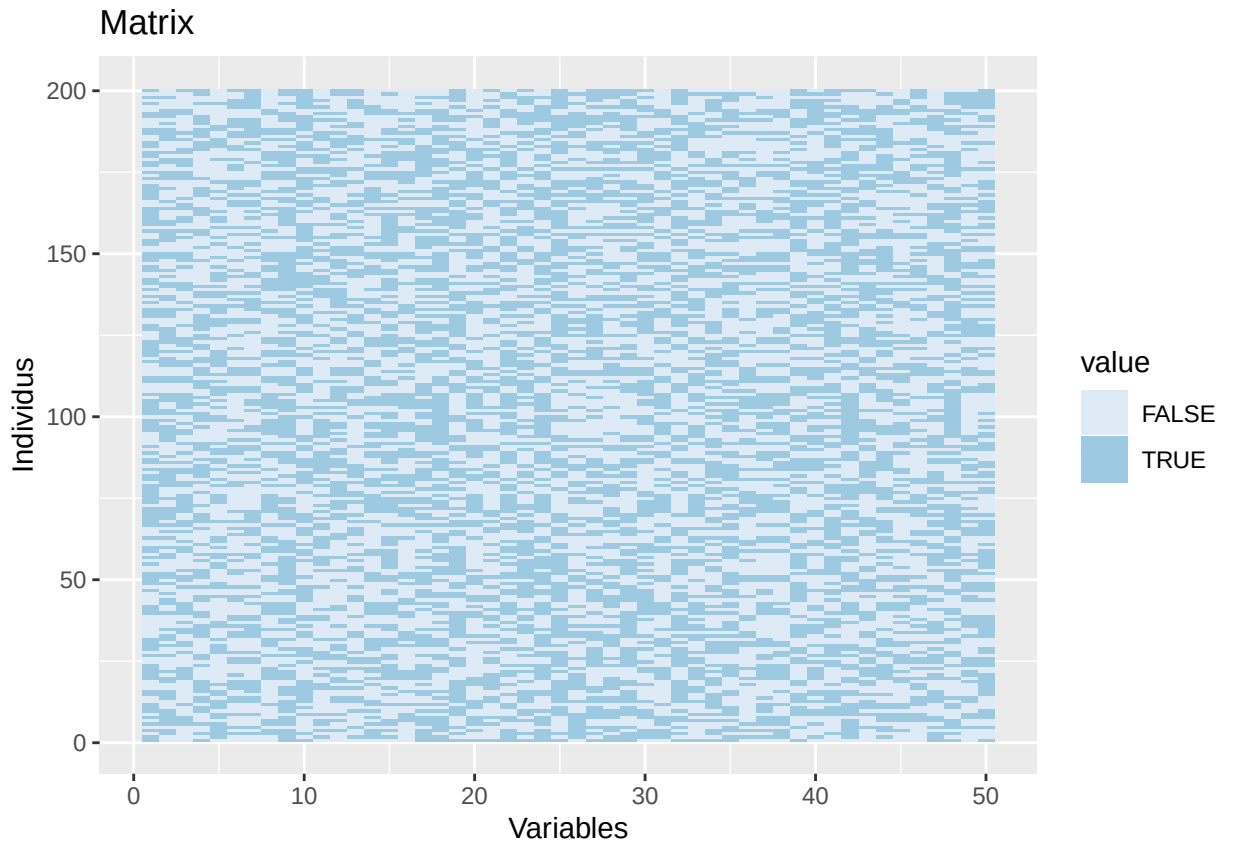
```
nks<-rmultinom(1,200,prob = pi)
Z<-apply(diag(x=1,K,K),MARGIN = 2,FUN=rep,times=nks)
```

Simulons $X|Z$.

```
X <-do.call(rbind,
                  mapply(function(nk,k){
                    matrix(rbernoulli(nk*p,p=M[k,]),
                           nrow = nk,
                           ncol=p,
                           byrow = TRUE)}, nks,1:K))
```

Permutons les lignes de la matrice X à 200 lignes et 50 colonnes et visualisons la matrice ainsi obtenue.

```
permutation <- sample(nrow(X))
X <- X[permutation,]
ggplot(melt(X), aes(x = Var2, y = Var1)) +
  geom_raster(aes(fill=value)) +
  scale_fill_brewer(aesthetics = "fill") +
  labs(x="Variables", y="Individus", title="Matrix")
```



On observe que l'échantillon a bien été mélangé et qu'il est impossible de distinguer graphiquement 3 motifs. On va désormais appliquer l'algorithme des kmeans à X avec 3 classes et produire la matrice Z des variables latentes correspondantes.

```
kmeans(X,3,nstart = 10)->res.kmeans
Z_kmeans<-matrix(0,nrow=n,ncol=K)
for (j in 1:K){
  Z_kmeans[res.kmeans$cluster==j,j]<-1
}
```

On remarque déjà que la taille des clusters obtenues correspond bien celles des composantes de notre mélange de Bernoulli (donnée par nks) :

```
c(nks)
```

```
## [1] 72 50 78
```

```
res.kmeans$size
```

```
## [1] 78 72 50
```

L'algorithme des kmeans convergeant vers un minimum local, on choisi nstart=10 initialisations différentes pour optimiser la réponse. L'algorithme des kmeans minimise l'inertie intra-classe et maximise l'inertie inter-classes. En assignant l'échantillon à 3 classes au lieu d'une seule, la réduction de variance expliquée par les clusters vaut (en pourcentage):

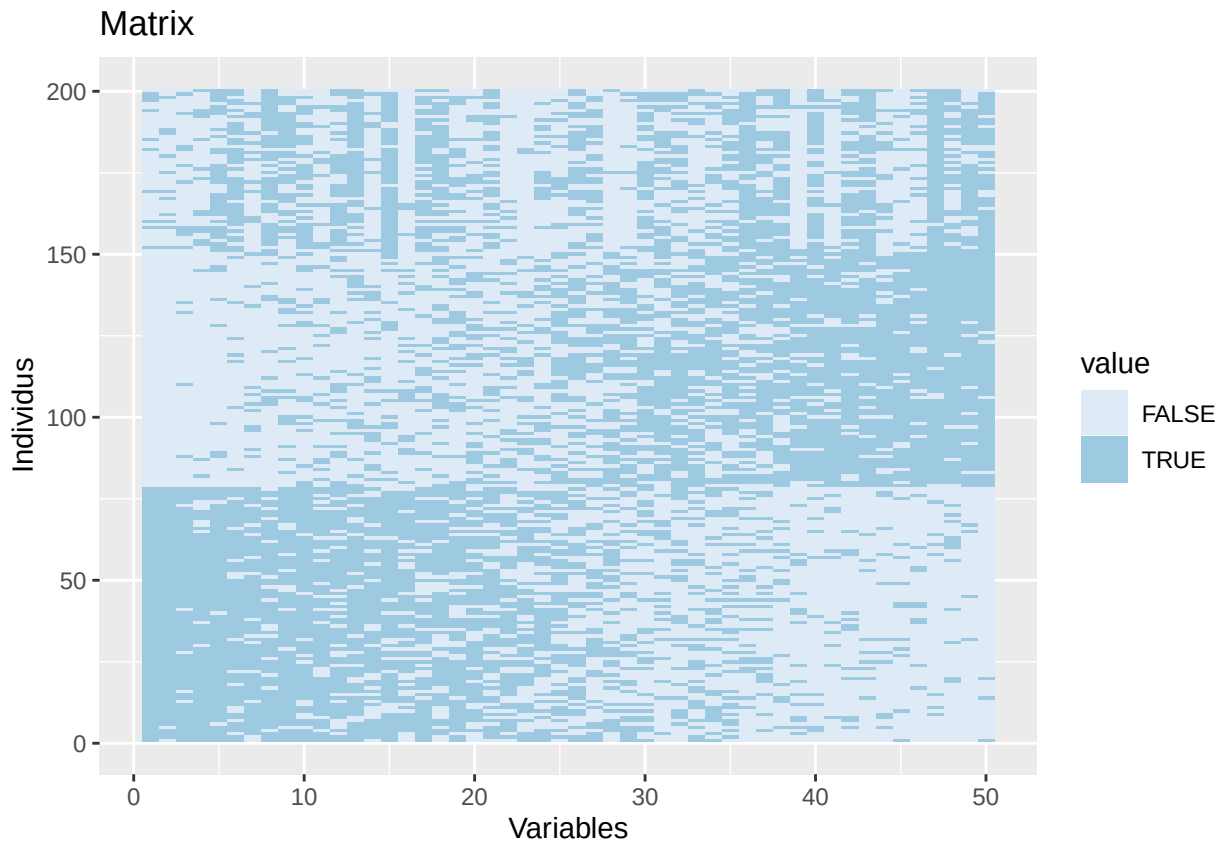
```
100*res.kmeans$tot.withinss/res.kmeans$totss
```

```
## [1] 67.92565
```

Visualisons la matrice classée :

```
tidyData<-melt(X[order(res.kmeans$cluster),order(M[1,])])

ggplot(tidyData, aes(x = Var2, y = Var1)) +
  geom_raster(aes(fill=value)) +
  scale_fill_brewer(aesthetics = "fill") +
  labs(x="Variables", y="Individus", title="Matrix")
```



On observe bien les 3 motifs distincts.

Exercice 2

Q1. Calculons la log-vraisemblance complète $\ln P(\mathbf{X}, \mathbf{Z}|\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{M}\})$.

Calculons d'abord la vraisemblance complète. On a :

$P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M}) = P(\mathbf{X}|\mathbf{Z}, \pi, \mathbf{M})P(\mathbf{Z}|\pi, \mathbf{M})$

Or, les observations étant indépendantes, en notant $z_{i,k} = \mathbf{1}_{Z_i=k}$: $P(\mathbf{X}|\mathbf{Z}, \pi, \mathbf{M}) = \prod_{i=1}^{n} P(\mathbf{x_i}|\mathbf{z_i}, \pi, \mathbf{M}) = \prod_{i=1}^{n}\prod_{k=1}^{K} P(\mathbf{x_i}|\mu_\mathbf{k})^{z_{i,k}} = \prod_{i=1}^{n}\prod_{k=1}^{K}\left(\prod_{j=1}^{p}\mu_{k,j}^{x_{i,j}}(1-\mu_{k,j})^{1-x_{i,j}}\right)^{z_{i,k}}$

Par ailleurs, comme $\pi_k = P(z_i = k)$,on a: $P(\mathbf{Z}|\pi) = \prod_{i=1}^{n} P(\mathbf{z_i}|\pi) = \prod_{i=1}^{n}\prod_{k=1}^{K} \pi_k^{z_{i,k}}$

On obtient donc pour la vraisemblance complète :

$\log P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M}) = \prod_{i=1}^{n}\prod_{k=1}^{K}\left(\pi_k \prod_{j=1}^{p} \mu_{k,j}^{x_{i,j}}(1-\mu_{k,j})^{1-x_{i,j}}\right)^{z_{i,k}}$

En prenant le logarithme, cela donne :

$\ln P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M}) = \sum_{i=1}^{n}\sum_{k=1}^{K} z_{i,k}\left(\ln \pi_k + \sum_{j=1}^{p} x_{i,j}\ln \mu_{k,j} + (1-x_{i,j})\ln(1-\mu_{k,j})\right)$

Q2. Calculons $t_{ik}^q = \mathbb{E}[z_{ik}]$ par rapport à la loi $p_{\theta^q}(\mathbf{Z}|\mathbf{X})$. On a, en utilisant le théorème de Bayes et le fait que $P(\mathbf{x_i}|z_{i,k}) = P(\mathbf{x_i}|\mu_k)$:

$t_{ik}^q = \mathbb{E}[z_{i,k}] = P(z_{i,k}|\mathbf{x_i}, \pi, \mathbf{M}) = \frac{P(z_{i,k})P(\mathbf{x_i}|\mu_\mathbf{k})}{P(\mathbf{x_i})}$.

Puis, par la formule des probabilités totales :

$$t_{i,k}^q = \frac{\pi_k P(\mathbf{x_i}|\mu_\mathbf{k})}{\sum_{m=1}^{K}\pi_m P(\mathbf{x_i}|\mu_\mathbf{m})} = \frac{\pi_k \prod_{j=1}^{p}\mu_{k,j}^{x_{i,j}}(1-\mu_{k,j})^{1-x_{i,j}}}{\sum_{m=1}^{K}\pi_m \prod_{j=1}^{p}\mu_{m,j}^{x_{i,j}}(1-\mu_{m,j})^{1-x_{i,j}}}$$

Q.3 On en déduit $Q(\theta^q|\theta)$, l'espérance de cette log-vraisemblance par rapport à la loi $p_{\theta^q}(\mathbf{Z}|\mathbf{X})$:

$Q(\theta^q|\theta) = \mathbb{E}_{Z|X,\theta^q}[\ln P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M})] = \sum_{i=1}^{n}\sum_{k=1}^{K} t_{i,k}^q\left(\ln \pi_k + \sum_{j=1}^{p} x_{i,j}\ln \mu_{k,j} + (1-x_{i,j})\ln(1-\mu_{k,j})\right)$

Q.4 Pour déterminer $\theta^{q+1} = argmax_\theta Q(\theta^q|\theta)$, on peut commencer par maximiser l'argument par rapport à $\boldsymbol{\mu}_k$ en annulant sa dérivée :

$\frac{\partial}{\partial \mu_{k,j}} Q(\theta^q|\theta) = \sum_{i=1}^{n} t_{i,k}^q(\frac{x_{i,j}}{\mu_{k,j}} - \frac{1-x_{i,j}}{1-\mu_{k,j}}) = \sum_{i=1}^{n} t_{i,k}^q \frac{x_{i,j}-\mu_{k,j}}{\mu_{k,j}(1-\mu_{k,j})} = 0 \Leftrightarrow \mu_{k,j} = \frac{1}{n_k}\sum_{i=1}^{n} x_{i,j}t_{i,k}^q$

où $n_k = \sum_{i=1}^{n} t_{i,k}^q$ correspond au nombre de points affectés au cluster k. Ainsi, pour $k \in \{1, ...K\}$, le maximum de $Q(\theta^q|\theta)$ par rapport à $\boldsymbol{\mu}_k$ est :

$$\boldsymbol{\mu}_k^{q+1} = \frac{1}{n_k}\sum_{i=1}^{n} \boldsymbol{x}_i t_{i,k}^q$$

Pour maximiser la fonction $Q(\theta^q|\theta)$ par rapport à π, sous la contrainte $\sum_{k=1}^{K}\pi_k = 1$, on peut introduire le multiplicateur de Lagrange. Le problème d'optimisation devient alors de maximiser la fonction $\Lambda(\theta, \lambda) = Q(\theta^q|\theta) + \lambda(\sum_{k=1}^{K}\pi_k - 1)$. En dérivant succéssivement par rapport à π_k puis λ, on obtient : $\frac{\partial}{\partial \pi_k}\Lambda(\theta, \lambda) = \frac{1}{\pi_k}\sum_{i=1}^{n} t_{i,k}^q + \lambda = 0 \Leftrightarrow \pi_k = -\frac{n_k}{\lambda}$,

$\frac{\partial}{\partial \lambda}\Lambda(\theta, \lambda) = \sum_{k=1}^{K}\pi_k - 1 = 0 \Leftrightarrow \sum_{k=1}^{K}\pi_k = 1$.

En combinant ces deux résultats, on obtient $\lambda = -\sum_{k=1}^{K} n_k = -n$ et donc :

$$\pi_k^{q+1} = -\frac{n_k}{\lambda} = \frac{n_k}{n}$$

Q.5 On commence l'algorithme EM en initialisant θ. Ensuite, l'étape E de l'algorithme EM consiste à calculer les $t_{i,k}^q = \mathbb{E}[z_{ik}]$ pour mettre à jour la distribution des variables latentes z_i. Puis l'étape M consiste à affecter à θ le maximum de $Q(\theta^q|\theta) = \mathbb{E}[\ln P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M})]$ par rapport à θ. Le θ estimé est obtenu lors de la convergence de la log-vraisemblance.

Q.6 On veut calculer $-\mathbb{E}[\ln p_{\theta^{q+1}}[Z|X]]$ par rapport à la loi $p_{\theta^{q+1}}[Z|X]$. On a :

$-\mathbb{E}[\ln p_{\theta^{q+1}}[Z|X]] = -\sum_{i=1}^{n}\mathbb{E}[\ln p_{\theta^{q+1}}[\mathbf{z}_i|\mathbf{x}_i]] = -\sum_{i=1}^{n}\mathbb{E}[\ln \prod_{k=1}^{K} p_{\theta^{q+1}}[\mathbf{z}_{i,k}|\mathbf{x}_i]^{z_{i,k}}] = -\sum_{i=1}^{n}\sum_{k=1}^{K} t_{i,k}^{q+1}\ln t_{i,k}^{q+1}$

Q.7 La forme de la log-vraisemblance $\ln p_{\hat{\theta}}[\mathbf{X}|\theta = \{\pi, \mathbf{M}\}]$ pour le paramètre $\hat{\theta}$ estimé est :

$$\ln p_{\hat{\theta}}[\mathbf{X}|\theta=\{\pi,\mathbf{M}\}] = \sum_{i=1}^{n} \ln p_{\hat{\theta}}[\mathbf{x_i}|\theta=\{\pi,\mathbf{M}\}] = \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \hat{\pi}_k P(\mathbf{x_i}|\hat{\mu}_\mathbf{k}))$$

avec $P(\mathbf{x_i}|\hat{\mu}_\mathbf{k}) = \prod_{j=1}^{p} \hat{\mu}_{k,j}^{x_{i,j}} (1-\hat{\mu}_{k,j})^{(1-x_{i,j})}$

Q.8 Le critère BIC correspond à $-2 \times \max logvraisemblance + v_K \ln n$

où v_K est le nombre de paramètres libres dans le modèle à K composantes. Le premier terme est donné par la question précédente et $v_K = K(p+1)$, car il a Kp paramètres $\mu_{k,j}$ et K paramètres π_k. Ainsi : le critère BIC associé à un modèle à K classes s'écrit :

$$BIC = -2 \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \hat{\pi}_k P(\mathbf{x_i}|\hat{\mu}_\mathbf{k})) + K(p+1) \ln n$$

avec $P(\mathbf{x_i}|\hat{\mu}_\mathbf{k}) = \prod_{j=1}^{p} \hat{\mu}_{k,j}^{x_{i,j}} (1-\hat{\mu}_{k,j})^{(1-x_{i,j})}$

9. Le critère ICL correspond à $-2 \times logvraisemblancecomplète - \frac{v_K}{2} \ln n$ où la log-vraisemblance complète est calculée pour $\hat{\theta}$ maximisant la log-vraisemblance, en remplaçant $\mathbf{Z}$ par $\tilde{\mathbf{Z}} = MAP(\hat{\theta})$ le maximum a posteriori de $\hat{\theta}$. Or, on a aussi pour que le critère ICL est égal au critère BIC pénalisé par le terme $-\sum_{i=1}^{n}\sum_{k=1}^{K} t_{i,k} \ln t_{i,k}$ ce qui correspond au terme calculé à la question 6 au moment de la convergence. Ainsi :
$$ICL = -2 \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \hat{\pi}_k P(\mathbf{x_i}|\hat{\mu}_\mathbf{k})) + K(p+1) \ln n - \sum_{i=1}^{n}\sum_{k=1}^{K} t_{i,k} \ln t_{i,k}$$

10. La convergence de l'algorithme EM est définie dès lors que la différence en valeur absolue de la log-vraisemblance entre deux itérations consécutives est inférieure à un certain ϵ assez petit. Ainsi, l'algorithme EM d'estimation des paramètres du modèle de mélange de Bernoulli est :

Algorithm 1 Calculate $\hat{\theta}$

Ensure: $\theta = \hat{\theta}$
  Initialisation au hasard de θ^0
  $q \leftarrow 0$
  $L_0 \leftarrow \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \pi_k^0 P(\mathbf{x_i}|\mu_\mathbf{k}^\mathbf{0}))$
  $\epsilon \leftarrow 10^{-3}$
  repeat
    $t_{i,k}^q \leftarrow \frac{\pi_k \prod_{j=1}^{p} \mu_{k,j}^{x_{i,j}} (1-\mu_{k,j})^{1-x_{i,j}}}{\sum_{m=1}^{K} \pi_m \prod_{j=1}^{p} \mu_{m,j}^{x_{i,j}} (1-\mu_{m,j})^{1-x_{i,j}}}$
    $n_k \leftarrow \sum_{i=1}^{n} t_{i,k}^q$
    $\boldsymbol{\mu}_k^{q+1} \leftarrow \frac{1}{n_k} \sum_{i=1}^{n} \boldsymbol{x}_i t_{i,k}^q$
    $\pi_k^{q+1} \leftarrow \frac{n_k}{n}$
    $\theta^{q+1} \leftarrow \{\pi_k^{q+1}, \boldsymbol{\mu}_k^{q+1}\}$
    $L_{q+1} \leftarrow \sum_{i=1}^{n} \ln(\sum_{k=1}^{K} \pi_k^{q+1} P(\mathbf{x_i}|\mu_k^{q+1}))$
    $q \leftarrow q+1$
  until $|L_q - L_{q-1}| \leq \epsilon$

11. L'algorithme EM consiste à maximiser la vraisemblance du paramètre θ. L'algorithme CEM consiste à optimiser, non pas la vraisemblance du paramètre, mais la vraisemblance complète. On rappelle que $\ln P(\mathbf{X}, \mathbf{Z}|\pi, \mathbf{M}) = \sum_{i=1}^{n} \sum_{k=1}^{K} z_{i,k} \left(\ln \pi_k + \sum_{j=1}^{p} x_{i,j} \ln \mu_{k,j} + (1 - x_{i,j}) \ln(1 - \mu_{k,j}) \right)$. A chaque étape de l'algorithme CEM, on maximise la log-vraisemblance complète en fonction de $\mathbf{Z}$ puis θ. Comme $z_{i,k} \in \{0, 1\}$, on obtient $z_{i,k}^{q+1} = \sum_{i=1}^{n} \max_k \left(\ln \pi_k + \sum_{j=1}^{p} x_{i,j} \ln \mu_{k,j} + (1 - x_{i,j}) \ln(1 - \mu_{k,j}) \right)$. Puis, par un calcul analogue à celui fait question 4, on a pour la forme de θ^{q+1} : $\boldsymbol{\mu}_k^{q+1} = \frac{1}{n_k} \sum_{i=1}^{n} \boldsymbol{x}_i z_{i,k}^{q+1}$ et $\pi_k^{q+1} = \frac{n_k}{n}$ avec $n_k = \sum_{i=1}^{n} z_{i,k}^{q+1}$

La convergence de l'algorithme CEM est définie dès lors que la différence en valeur absolue de la log-vraisemblance complète entre deux itérations consécutives est inférieure à un certain ϵ assez petit. L'algorithme CEM associé au même mélange est :

Algorithm 2 Calculate $\hat{\theta}, \hat{Z}$

Ensure: $\theta = \hat{\theta}, Z = \hat{Z}$
  Initialisation au hasard de θ^0
  Initialisation au hasard de $\mathbf{Z}^0$
  $q \leftarrow 0$
  $L_0 \leftarrow \sum_{i=1}^{n} \sum_{k=1}^{K} z_{i,k}^0 \left(\ln \pi_k^0 + \sum_{j=1}^{p} x_{i,j} \ln \mu_{k,j}^0 + (1 - x_{i,j}) \ln(1 - \mu_{k,j}^0) \right)$
  $\epsilon \leftarrow 10^{-3}$
  repeat
    $z_{i,k}^{q+1} \leftarrow \sum_{i=1}^{n} \max_k \left(\ln \pi_k + \sum_{j=1}^{p} x_{i,j} \ln \mu_{k,j} + (1 - x_{i,j}) \ln(1 - \mu_{k,j}) \right)$
    $n_k \leftarrow \sum_{i=1}^{n} z_{i,k}^{q+1}$
    $\boldsymbol{\mu}_k^{q+1} \leftarrow \frac{1}{n_k} \sum_{i=1}^{n} \boldsymbol{x}_i z_{i,k}^{q+1}$
    $\pi_k^{q+1} \leftarrow \frac{n_k}{n}$
    $\theta^{q+1} \leftarrow \{\pi_k^{q+1}, \boldsymbol{\mu}_k^{q+1}\}$
    $L_{q+1} \leftarrow \sum_{i=1}^{n} \sum_{k=1}^{K} z_{i,k}^{q+1} \left(\ln \pi_k^{q+1} + \sum_{j=1}^{p} x_{i,j} \ln \mu_{k,j}^{q+1} + (1 - x_{i,j}) \ln(1 - \mu_{k,j}^{q+1}) \right)$
    $q \leftarrow q + 1$
  until $|L_q - L_{q-1}| \leq \epsilon$

Exercice 3

Q.1 Ecrivons une fonction E-step qui produit les $t_{i,k}$ à partir de Θ.

```
E_Step<-function(X_,M_,pi_){
  n_ <- nrow(X_)
  K_ <- nrow(M_)
  #Calcul de P(X|M)
  p_x_mu <- matrix(0,nrow=n_,ncol=K_)
  for (i in 1:n_){
    for (k in 1:K_){
      p_x_mu[i,k]<-prod(M_[k,]^X_[i,]*(1-M_[k,])^(1-X_[i,]))
    }
  }
  #Calcul de T la matrice des t_i,k
  T<-matrix(0,nrow=n_,ncol = K_)
  for (i in 1:n_){
    for (k in 1:K_){
      T[i,k]<-pi_[k]*p_x_mu[i,k]/(t(pi_)%*%p_x_mu[i,])
    }
  }
  return (T)
}
```

Nous voulons vérifier les résultats en injectant les vrais paramètres de notre simulation et en comparant les $t_{i,k}$ estimé par rapport aux variables latentes Z de notre simulation (ce qui correspond à notre Z_kmeans). Si contrairement aux variables latentes $z_1, .., z_n$, les t_i ne sont pas des variables binaires, on s'attend à avoir le même maximum de responsabilité pour chaque x_i, $i \in \{1, .., n\}$. On peut donc montrer dans un premier temps que les t_i et z_i ont le même maximum de responsabilité, puis, dans un second temps, on peut comparer la norme de la différence des deux matrices (qui doit être "proche", au sens de la norme matricielle, de la matrice nulle). Or, on ne sait pas dans quel ordre l'algorithme kmeans numérote les clusters et donc si les numérotations des composantes et des clusters correspondent. On va donc devoir comparer pour chaque permutation de la numérotation des clusters. Dès que tout les t_i et z_i ont le même maximum de responsabilité, on sait que l'on aura obtenu la bonne numérotation.

```
T<-E_Step(X,M,pi)
Tmax<-apply(T,which.max,MARGIN = 1)
library(combinat)
```

```
##
## Attaching package: 'combinat'
```

```
## The following object is masked from 'package:utils':
##
##     combn
```

```
cluster_permutations <- permn(1:K)
for (i in 1:K){
  copycluster<-res.kmeans$cluster
  for (k in 1:K){
    copycluster[res.kmeans$cluster==k]<-cluster_permutations[[i]][k]
  }
  if(sum(copycluster==Tmax)==n){
    print("Mêmes maximums de responsabilité pour tout x_i")
    print("Norme de la différence des matrices Z et T:")
    print(norm(T[,cluster_permutations[[i]]]-Z_kmeans))
    break;
```

```
  }
}
```

```
## [1] "Mêmes maximums de responsabilité pour tout x_i"
## [1] "Norme de la différence des matrices Z et T:"
## [1] 0.02897998
```

On observe que la valeur de la norme obtenue est proche de 0, ce qui est cohérent.

Q.2 Ecrivons une fonction M-step qui produit les estimateurs des Θ à partir des données observées et des $t_{i,k}$. On rappelle les résultats de l'exercice 2 :

$$\boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{i=1}^{n} \boldsymbol{x}_i t_{i,k}^q$$

$$\pi_k = \frac{n_k}{n}$$

où $n_k = \sum_{i=1}^{n} t_{i,k}^q$

```
M_Step<-function(X_,T_){
  K_<-ncol(T_)
  n_<-nrow(X_)
  #Calcul des mu_k
  new_mu <- matrix(1:K_,ncol = 1)
  new_mu <- t(apply(new_mu,MARGIN=1,function(k){colSums(X_*c(T_[,k]))/colSums(T_)[k]}))
  #Calcul des pi_k
  new_pi <- colSums(T_)/n_
  return (list("new_pi"=new_pi,"new_mu"=new_mu))
}
```

De manière analogue à la question précédente, on peut tester la fonction sur notre simulation dont on connait tous les paramètres et calculer les normes des différences pour voir si elles approchent la matrice nulle :

```
new_theta <- M_Step(X,T)
```

```
## [1] "Norme de la différence des matrices M et new_mu:"
## [1] 0.3596519
## [1] "Norme de la différence des matrices pi et new_pi:"
## [1] 0.1665247
```

On obtient des normes proches de 0, ce qui est cohérent.

Q.3 Ecrivons l'algorithme EM qui estime les paramètres d'un mélange à K classes.

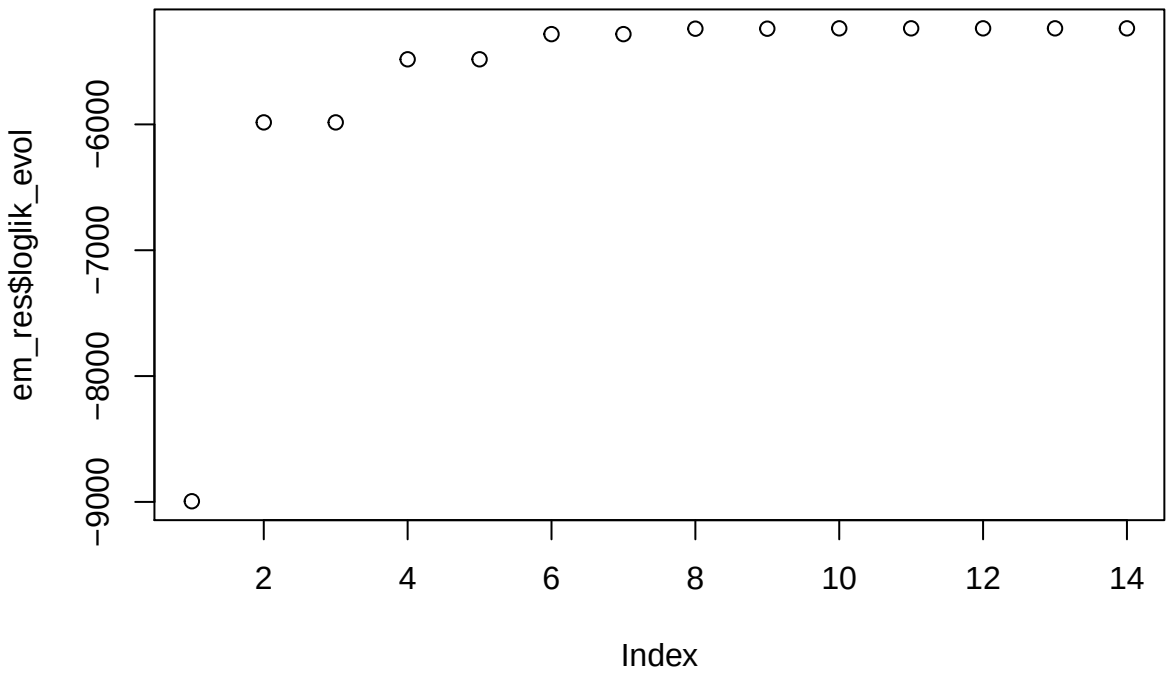
```
EM <- function(X_,K_){
  n_ <- nrow(X_)
  p_ <- ncol(X_)
  M_<-matrix(runif(K_*p_),nrow=K_,ncol=p_)
  pi_<-matrix(runif(K_),nrow=K_)
  epsilon <- 10^-3
  L_prev <- loglikelihood(X_,pi_,M_)
  loglik_evolution <- c()
  repeat{
    #Expectation
    T_<-E_Step(X_,M_,pi_)
```

```
    L_semi <- loglikelihood(X_,pi_,M_)
    loglik_evolution <- c(loglik_evolution,L_semi)
    #Maximisation
    new_theta <- M_Step(X_,T_)
    M_ <- new_theta$new_mu
    pi_ <- new_theta$new_pi
    L_curr <- loglikelihood(X_,pi_,M_)
    loglik_evolution <- c(loglik_evolution,L_curr)
    #convergence criterion
    if (abs(L_curr - L_prev)<=epsilon){
      break;
    }
    L_prev<-L_curr
  }
  return (list("pi_hat"=pi_,"M_hat"=M_,"maxloglik"=L_curr,"loglik_evol"=loglik_evolution,"T_final"=T_))
}

loglikelihood <- function(X_,pi_,M_){
  n_<-nrow(X_)
  p_<-ncol(X_)
  K_<-nrow(M_)
  #P_ matrice des p(x_i|mu_k)
  P_<-matrix(0,nrow = K_,ncol = n_)
  for (k in 1:K_){
    for (i in 1:n_){
      product<-1
      for (j in 1:p_){
        product <- product * (M_[k,j]^X_[i,j] * (1-M_[k,j])^(1-X_[i,j]))
      }
      P_[k,i]<-product
    }
  }
  res<-0
  for (i in 1:n_){
    res <- res + log(t(pi_)%*%P_[,i])
  }
  return (res)
}
```

Q.4 Traçons l'évolution de la log-vraisemblance à chaque demi étape (E et M) lorsqu'on applique l'algorithme aux données simulées :

```
em_res <- EM(X,3)
plot(em_res$loglik_evol)
```



On observe que la log-vraisemblance croît à chaque étape, ce qui est cohérent. Par ailleurs, on observe que la log-vraisemblance ne change pas entre avant et après l'étape "expectation", ce qui est cohérent.

Q.5 Programmons la fonction BIC :

```
myBIC <- function(EM_res_){
  K_ <- nrow(EM_res_$M_hat)
  p_ <- ncol(EM_res_$M_hat)
  n_ <- nrow(EM_res_$T_final)
  return (-2 * EM_res_$maxloglik +K_*(p_+1)*log(n_))
}
myBIC(em_res)
```

```
##          [,1]
## [1,] 11284.03
```

Q.6 Programmons la fonction IC :

```
myICL <- function(EM_res_){
  penalization <- sum(EM_res_$T_final * apply(EM_res_$T_final,FUN = log,MARGIN=c(1,2)),na.rm = TRUE)
  return (myBIC(EM_res_) - penalization)
}
myICL(em_res)
```

```
##          [,1]
## [1,] 11284.24
```

Comme $x \ln x \to_{x \to 0} 0$ et que certains des $t_{i,k}$ estimés par l'algorithme EM sont numériquements égaux à 0, on peut ignorer les valeur NaN dans la somme des $t_{i,k} \ln t_{i,k}$ (correspondant au résultat de 0 * -Inf).

Exercice 4

On charge les données.

```
state_firearms <- read.table("raw_data.csv",sep = ",",header = TRUE)
```

Eliminons les variables non-binaires

```
state_firearms_data <- as.matrix(state_firearms[,-c(1,2,136)])
```

Le problème est désormais d'évaluer le nombre de composantes K. Le critère BIC limite l'overfitting avec le terme $K(p+1)\ln n$. Cependant, en calculant le maximum de la log-vraisemblance, le critère BIC se positionne dans une optique d'estimation, sans considération de la classification faite a posteriori. En revanche, le critère ICL, qui calcule la log-vraisemblance complète, optimise le partionnement des données. On peut donc traçer l'évolution des critères ICL et BIC en fonction du nombre de composantes. Le plus faible sont chacun de ces critères, le mieux est la capacité prédictive du modèle. En l'absence de minimum, on peut déterminer la grande variation de la somme cumulée.

Appliquons l'algorithme au jeu de données sur les lignes pour K composantes, $K \in \{1, .., 5\}$:

{r} bic_evol <- c() icl_evol <- c() for (i in 1:5){ sf_em_res <- EM(state_firearms_data,i) bic_evol <- c(bic_evol,myBIC(sf_em_res)) icl_evol <- c(icl_evol,myICL(sf_em_res)) }

Affichons l'évolution des critères BIC et ICL en fonction du nombre de composantes.

{r} plot(bic_evol) plot(icl_evol)

Appliquons l'algorithme au jeu de données sur les lignes pour K composantes, $K \in \{1, .., 5\}$:

{r} tbic_evol <- c() ticl_evol <- c() for (i in 1:5){ tsf_em_res <- EM(matrix(state_firearms_data,nrow=133,ncol=1350),i) tbic_evol <- c(tbic_evol,myBIC(tsf_em_res)) ticl_evol <- c(ticl_evol,myICL(tsf_em_res)) }

Affichons l'évolution des critères BIC et ICL en fonction du nombre de composantes.

{r} plot(tbic_evol) plot(ticl_evol)

NOTE : En raison d'un temps de compilation trop important pour l'ordinateur sur lequel la version finale du projet a été compilée, le code ci-dessus a été mis en commentaire.

Sur un ordinateur plus performant, on obtient pour l'algorithme appliqué sur les lignes un critère BIC et ICL minimal pour $K = 2$. On peut émettre l'hypothèse que cette valeur optimale du nombre de composantes $K = 2$ pour le partitionnement des données fait transparaître le clivage entre états traditionnellemnt républicains (régulation sur les armes plus libérale) et démocrates (régulation sur les armes plus réstrictive).