Apprentissage supervisé

TP1 : les k plus proches voisins

Exercice 1. Les objectifs :

- Appliquer les knn sur un exemple.
- Découvrir les notions de taux d'erreur d'apprentissage et de taux d'erreur test.
- Découvrir l'importance du choix du paramètre k.

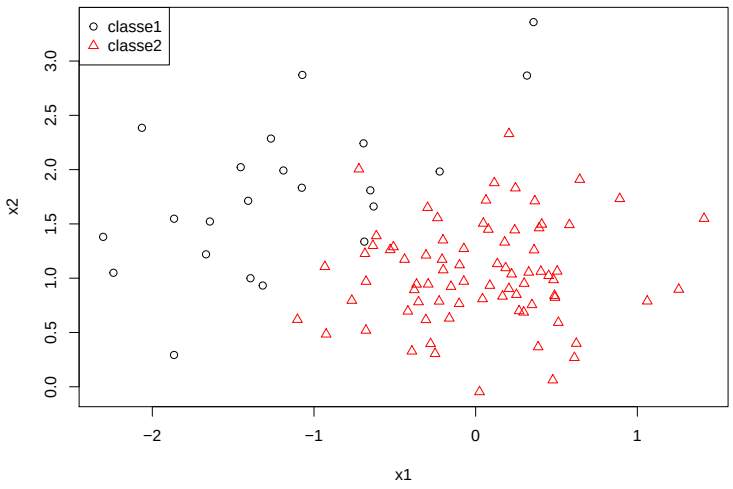
Récupérer les jeux de données synth_train.txt et synth_test.txt. On a $Y \in \{1, 2\}$ et $X = (X^1, X^2) \in \mathbb{R}^2$. On dispose de 100 données d'apprentissage et 200 données test.

1. Charger le jeu de données d'apprentissage dans R avec la commande `read.table`. Afficher les données d'apprentissage.

```
train <- read.table(file="../data/synth_train.txt", header=TRUE)
Xtrain <- train[,-1]
Ytrain <- train$y
```

2. Représenter graphiquement les observations à l'aide de la fonction `plot`. On pourra colorier les points en fonction de leur classe à l'aide du paramètre `col`, modifier le symbole avec le paramètre `pch` (=**p**oint **ch**aracter) et rajouter une légende à l'aide de la fonction `legend`.

```
plot(Xtrain, pch=Ytrain, col=Ytrain)
legend("topleft", legend=c("classe1", "classe2"), pch=1:2, col=1:2)
```



3. Appliquer la fonction `knn` du package `class` avec $k = 30$ voisins pour prédire les classes des points de coordonnées (0,0) et (-2,1). Vérifiez sur le graphique que les prédictions sont cohérentes. Recommencer avec l'argument `prob=TRUE`. A quoi correspondent ces probabilités ?

```
library(class)
Xtest <- matrix(c(0,0,-2,1), nrow=2, byrow=TRUE)
pred <- knn(Xtrain,Xtest, Ytrain, k=30, prob=TRUE)
pred
```

```
## [1] 2 1
## attr(,"prob")
## [1] 1.00 0.57
## Levels: 1 2
```

```
attr(pred, "prob")
```

```
## [1] 1.00 0.57
```

4. Programmez votre propre fonction knn et vérifiez que vous retrouvez les résultats précédents.
5. Appliquer la fonction **knn** pour prédire les données de l'ensemble d'apprentissage avec $k = 30$ voisins. Calculer le taux d'erreur d'apprentissage.

```
# avec k=30 voisins
pred_train <- knn(Xtrain, Xtrain, Ytrain, 30)
sum(pred_train!=Ytrain)/length(Ytrain) # taux d'erreur d'apprentissage
```

```
## [1] 0.1
```

6. Charger le jeu de données test dans R. Appliquer la fonction **knn** pour prédire les données de l'ensemble test avec $k = 30$ voisins. Calculer le taux d'erreur test et comparer au taux d'erreur d'apprentissage.

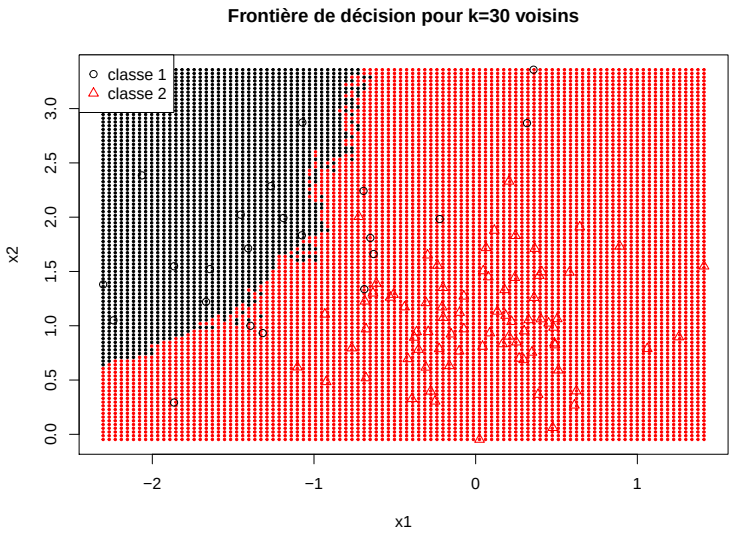
```
test <- read.table(file="../data/synth_test.txt", header=TRUE)
Xtest <- test[,-1]
Ytest <- test$y
# avec k=30 voisins
pred_test <- knn(Xtrain,Xtest,Ytrain,30)
sum(pred_test!=Ytest)/length(Ytest) # taux d'erreur test
```

```
## [1] 0.17
```

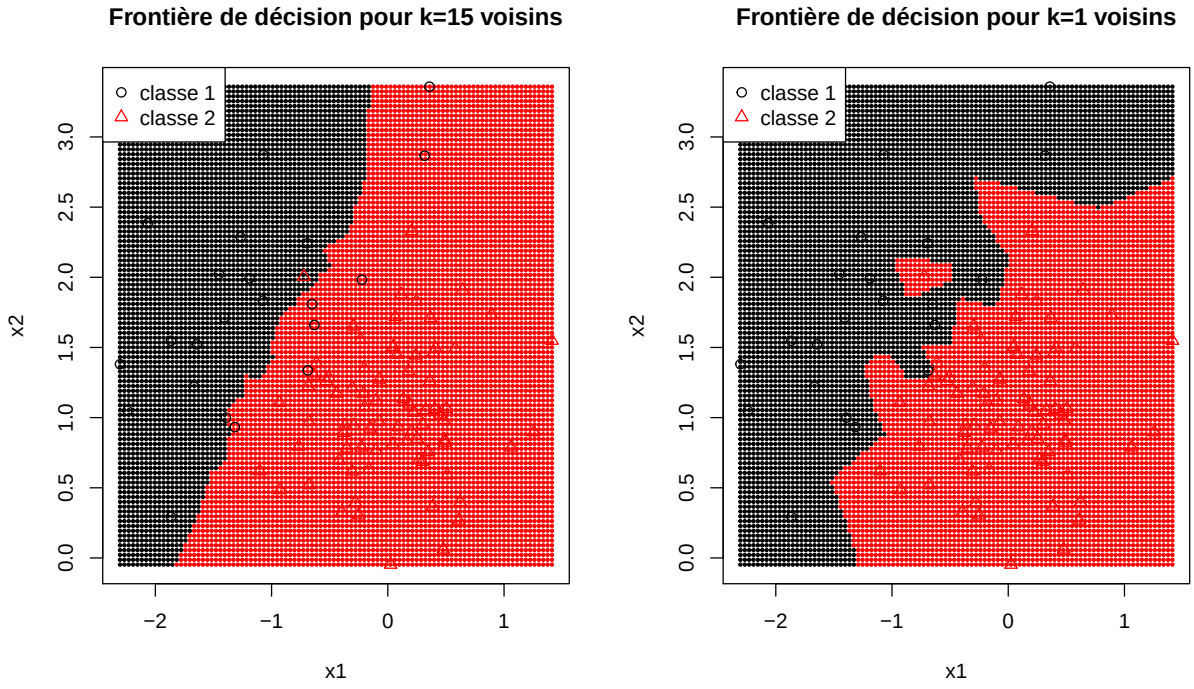
7. Calculer maintenant le taux d'erreur d'apprentissage et le taux d'erreur test sur les mêmes données avec $k = 15$ voisins puis $k = 1$ voisin.
8. On représente graphiquement la frontière de décision pour $k = 30$ voisins. Pour cela, on construit une grille de points, on prédit les classes de ces points et on représente les points et leurs prédictions sur un graphique.

```
#constuction de la grille de points
a <- seq(from=min(Xtrain$x1), to=max(Xtrain$x1), length.out=100)
b <- seq(from=min(Xtrain$x2), to=max(Xtrain$x2), length.out=100)
grille <- NULL
for (i in a){
  grille <- rbind(grille, cbind(i,b))
}
colnames(grille) <- c("x1","x2")
```

```
# avec k=30 voisins
pred_grille <- knn(Xtrain, grille, Ytrain, 30)
plot(grille, pch = 20, col = pred_grille, cex = 0.5,
     main="Frontière de décision pour k=30 voisins")
points(Xtrain, pch=Ytrain, col=Ytrain)
legend("topleft", legend=c("classe 1", "classe 2"), pch=1:2, col=1:2, bg="white")
```



9. Recommencer avec $k = 15$ voisins puis $k = 1$ voisins pour retrouver les graphiques ci-dessous. En quoi le choix du paramètre k semble important pour la qualité de la prédiction ?



Exercice 2. Les objectifs :

- Simuler des données à partir d'un mélange de lois Gaussiennes.
- Appliquer de manière empirique la règle de Bayes avec la méthode knn.
- Evaluer le taux d'erreur test avec plusieurs découpages.
- Comparer ce taux d'erreur test au taux d'erreur de la règle du "max à priori".
- Découvrir les notions de VP (vrais positifs), FP (faux positifs), VN (vrais négatifs), FN (faux négatifs) en classification binaire et les notions de sensibilité (TVP), de spécificité (TVN), de précision et de F-mesure.
- Découvrir le problème des données déséquilibrées.

— Appliquer de manière empirique la règle de Bayes avec une matrice de coûts.
— Evaluer la F-mesure par validation croisée 5-folds.

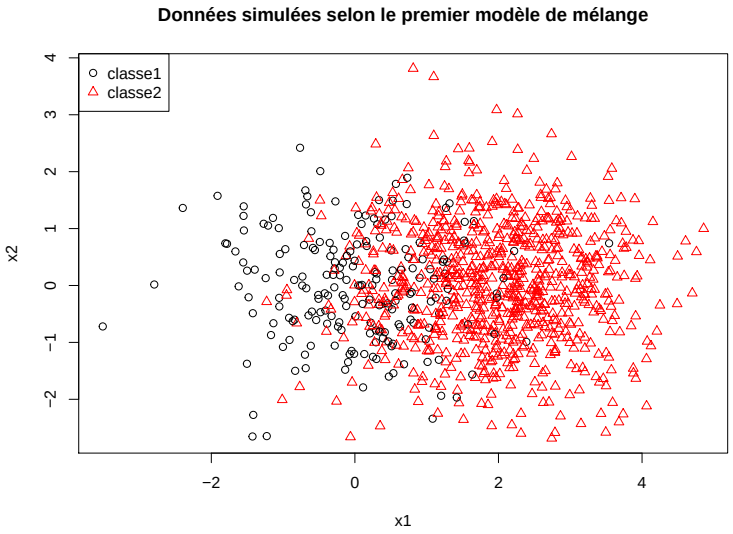
Le code ci-dessous permet de simuler des données à partir du mélange Gaussien bivarié suivant :

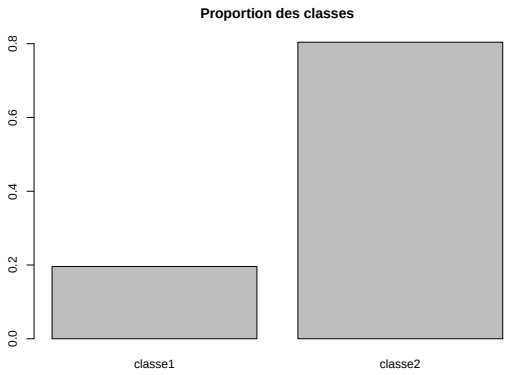
$$0.2 \times \mathbb{N}(\mu_1, \Sigma) + 0.8 \times \mathbb{N}(\mu_2, \Sigma)$$

où $\mu_1 = (0, 0)$, $\mu_2 = (2, 0)$ et Σ est la matrice identité 2×2.

```
library(mvtnorm)
n <- 1000
set.seed(10)
U =runif(n)

X <-  matrix(NA,n,2)
Y <- rep(NA,n)

for(i in 1:n) {
  if(U[i] < .2) {
    X[i,] <- rmvnorm(1,c(0,0),diag(c(1,1)))
    Y[i] <- 1
  } else {
    X[i,] <- rmvnorm(1,c(2,0),diag(c(1,1)))
    Y[i] <- 2
  }
}
```

On utilisera l'instruction `set.seed(10)` pour avoir les mêmes données et les mêmes découpages aléatoires.





1. On considère d'abord que le coût de mauvaise classification dans la classe 1 est égal au coût de mauvaise classification dans la classe 2 (matrice de coûts 0-1).

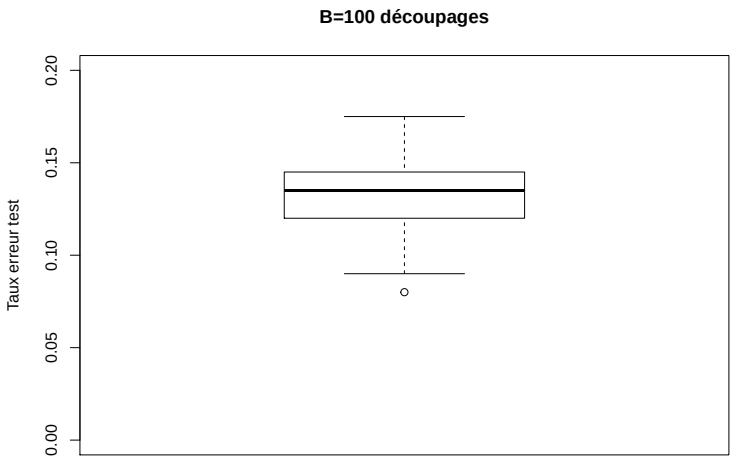
   (a) Comment la méthode knn applique de manière empirique la règle de Bayes ? Quelle est la mesure de performance minimisée ?

   (b) Evaluer le taux d'erreur de la méthode knn ($k = 10$ voisins) en découpant aléatoirement les données en 80% de données d'apprentissage (800 données) et 20% de données test (200 données) avec le code ci-dessous.

```
set.seed(10)
tr <- sample(1:n,800)
Xtrain <- X[tr,]
Ytrain <- Y[tr]
Xtest <- X[-tr,]
Ytest <- Y[-tr]
```

   (c) Recommencer en supprimant la commande `set.seed(10)` pour observer que la valeur du taux d'erreur test est modifiée. Pourquoi ?

   (d) Evaluer le taux d'erreur de la méthode knn ($k = 10$ voisins) en effectuant cette fois $B = 100$ découpages aléatoire des données. Représentez les taux d'erreurs test dans un boxplot et calculer le taux d'erreur test moyen.



```
print("Taux d'erreur test moyen")

## [1] "Taux d'erreur test moyen"

mean(err)

## [1] 0.13
```

(e) Comparer ce taux d'erreur à celui de la règle de classification qui consisterait à toujours prédire la classe la plus probable à priori ?

(f) On veut maintenant savoir si la méthode knn prédit aussi bien les données de la classe 1 que celles la classe 2. Calculer sur les données test du premier découpage (`set.seed(10)`) le taux de vrai positifs (TVP), le taux de vrai négatif (TVN) et la F-mesure avec ici positif="prédire 1". Quelle est la classe la mieux prédite ? D'après vous pourquoi ?

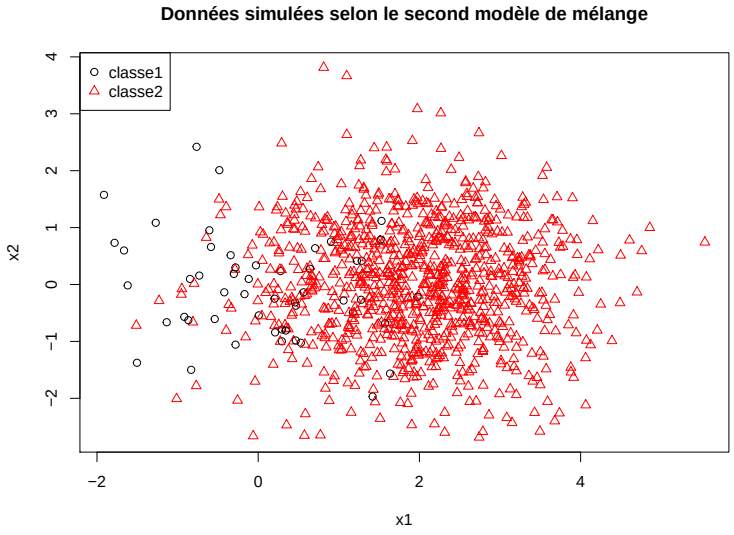
(g) Attention, la F-mesure dépend du choix de la modalité "positive". Calculer la F-mesure en prenant cette fois positif="prédire 2". Donner une interprétation de la F-mesure.

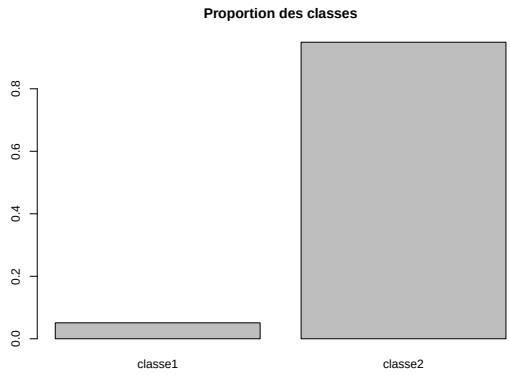
2. On reste dans le cadre d'une matrice de coûts 0-1 mais on se place maintenant dans le cadre de données très déséquilibrées.

(a) Simuler un échantillon de taille $n = 1000$ à partir du modèle de mélange bivarié suivant :

$$0.05 \times \mathbb{N}(\mu_1, \Sigma) + 0.95 \times \mathbb{N}(\mu_2, \Sigma)$$

qui est celui de la question 1 mais avec des probabilités à priori $\pi_1 = 0.05$ et $\pi_2 = 0.95$.





(b) Faire un découpage aléatoire de ces données en 80% de données d'apprentissage et 20% de données test en utilisant le code de la question 1(b).

(c) Appliquer la méthode knn ($k = 10$ voisins) pour prédire les données test et calculer le taux d'erreur test. Comparer ce taux d'erreur à celui de la règle du "max à priori".

(d) Calculer le TVP, TVN et la F-mesure sur les données test en prenant positif="prédire 1". Est-ce que ces résultats confirment les conclusions de la question 1(f) ?

(e) Calculer la F-mesure en prenant cette fois positif="prédire 2". Est-ce que ces résultats confirment les conclusions de la question 1(g) ?

3. On reste dans le cadre des données très déséquilibrées mais on considère maintenant que le coût de mauvaise classification d'une donnée de la classe 1 est 5 fois plus important le coût de mauvaise classification d'une donnée de la classe 2.

(a) Quelle matrice de coût utiliser ?

(b) Comment utiliser la méthode knn pour appliquer de manière empirique la règle de Bayes ? Quelle est la mesure de performance minimisée ?

(c) On construit maintenant une fonction `knn_cout` (code ci-dessous) qui utilise les knn pour appliquer de manière empirique la règle de Bayes avec une matrice de coût (2×2). Appliquer cette fonction aux données (très déséquilibrées) de la question 2 pour prédire les données test.

```
knn_cout <- function(Xtrain, Ytrain, Xtest, C, k)
{
  #proba à posteriori d'être dans la classe 1 : P(Y=1/X=x)
  prob <- rep(NA, nrow(Xtest))
  res <- knn(Xtrain, Xtest, Ytrain, k=k, prob=TRUE)
  prob[res==1] <- attr(res,"prob")[res==1]
  prob[res==2] <- 1-attr(res,"prob")[res==2]
  #risque conditionnel de prédire la classe 1 : R(1/x)=C21*P(Y=2/X=x)
  R1 <- C[2,1]*(1-prob)
  #risque conditionnel de prédire la classe 2 : R(2/x)=C12*P(Y=1/X=x)
  R2 <- C[1,2]*prob
  #prediction avec la règle de Bayes et la matrice de coûts
  pred <- rep(NA, nrow(Xtest))
  pred[R1 < R2] <- 1
  pred[R2 < R1] <- 2
  return(pred)
}
```

(d) Calculer sur les données test le taux d'erreur, le risque empirique, le TVP, le TVN et la F-mesure. Est-ce que l'utilisation d'une matrice de coût a permis de mieux prédire les données ? de mieux prédire les données de la classe 1 ?

(e) On veut maintenant évaluer la performance de la méthode `knn_cout` en calculant la F-mesure non pas sur un seul échantillon test mais par validation croisée 5-fold (code ci-dessous).

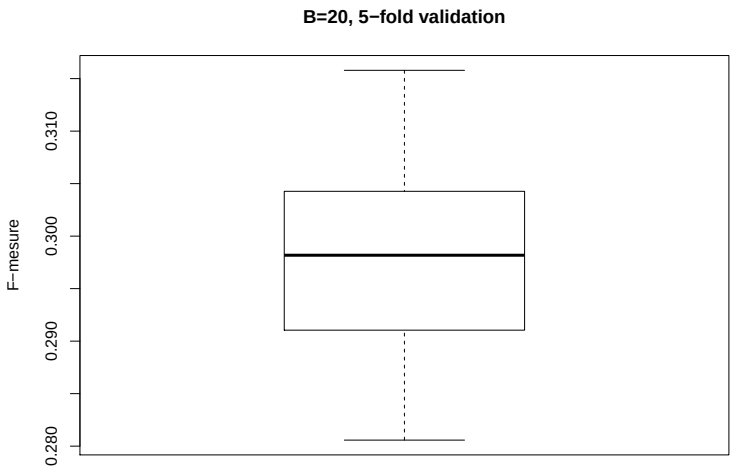
```
Fmesure <- function(Y,pred)
{
  tab <- table(Y,pred)
  tvp <- tab[1,1]/(tab[1,1]+tab[1,2])
  precision <- tab[1,1]/(tab[1,1]+tab[2,1])
  2*precision*tvp/(precision+tvp)
}
```

```
set.seed(10)
n_folds <- 5
folds_i <- sample(rep(1:n_folds, length.out = n))

pred <- rep(NA, nrow(X))
for (k in 1:n_folds) {
  test_i <- which(folds_i == k)
  Xtrain <- X[-test_i, ]
  Xtest <- X[test_i, ]
  pred[test_i] <-  knn_cout(Xtrain, Y[-test_i], Xtest, C, k=10)
}
```

```
# F-mesure calculé par validation croisée 5-folds
Fmesure(Y, pred)

## [1] 0.29
```

(f) Recommencer en supprimant la commande `set.seed(10)` pour observer que la valeur de la F-mesure est modifiée. Pourquoi ?

(g) Evaluer la F-mesure de la méthode `knn_cout` en effectuant cette fois $B = 20$ découpages aléatoires 5-folds. Représenter les F-mesures (de validation croisée 5-folds) dans un boxplot et calculer la F-mesure moyenne.



```
## [1] "F-mesure moyenne"
## [1] 0.3
```

Exercice 3. Les objectifs :

— Apprendre à choisir (calibrer) le paramètre k.
— Comprendre la notion d'échantillon de validation et son utilisation pour choisir un paramètre.

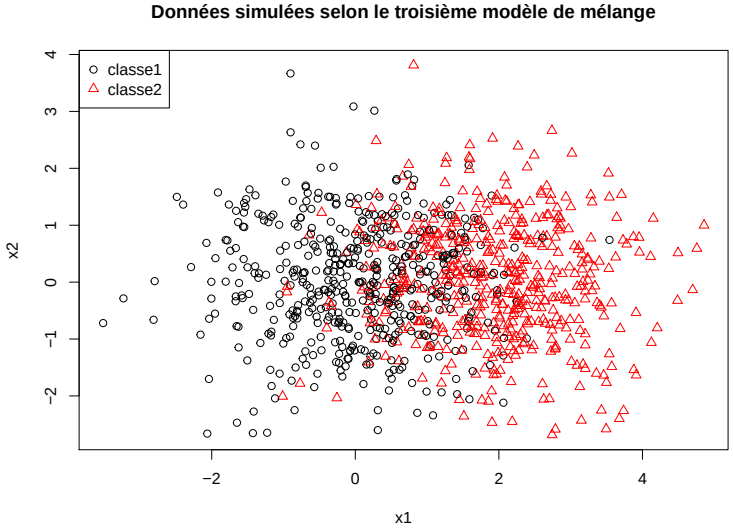
La règle fondamentale pour choisir le nombre k de voisins est que les données test (utilisées pour évaluer la méthode de classification) ne doivent pas être utilisées pour choisir k.

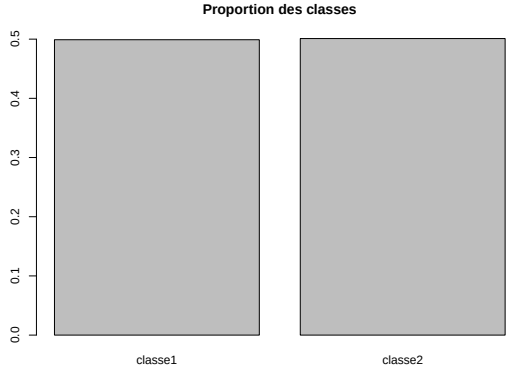
Dans cet exercice, on travail avec des données simulées à partir du mélange Gaussien bivarié suivant :

$$0.5 \times \mathbb{N}(\mu_1, \Sigma) + 0.5 \times \mathbb{N}(\mu_2, \Sigma)$$

qui est celui de l'exercice 2 mais avec des probabilités à priori $\pi_1 = 0.5$ et $\pi_2 = 0.5$.

1. Simuler un échantillon de taille $n = 1000$ selon le modèle de mélange Gaussien ci-dessus en adaptant le code de l'exercice précédent.





2. Découpez aléatoirement les données en 800 données d'apprentissage et 200 de données test.

```r
set.seed(10)
tr <- sample(1:n,800)
Xtrain <- X[tr,]
Ytrain <- Y[tr]
Xtest <- X[-tr,]
Ytest <- Y[-tr]
```

3. Une procédure pour le choix du paramètre k consiste à choisir k qui minimise l'erreur dite de **validation**.

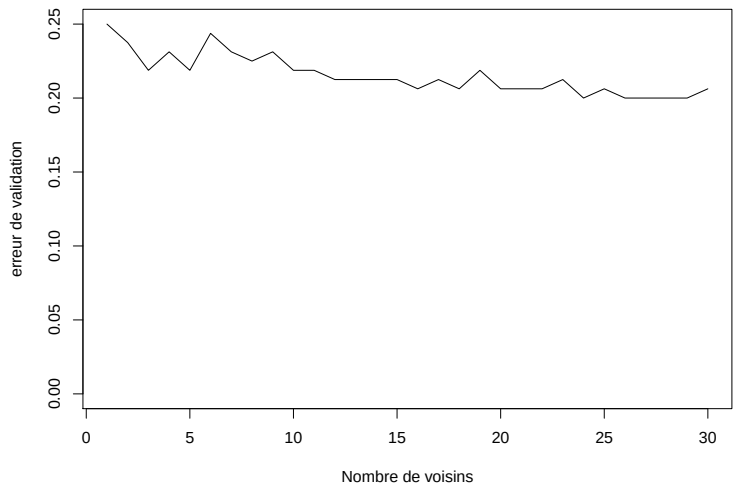
   (a) A partir du code ci-dessous, découper aléatoirement les 800 données d'apprentissage en deux parties : une partie avec 640 données (80% pour l'apprentissage) et une partie avec 160 données (20% pour la validation). Cet échantillon de taille 160 est appellé **échantillon de validation**.

```r
set.seed(10)
tr <- sample(1:800,640)
Xtrain_i <- Xtrain[tr,]
Ytrain_i <- Ytrain[tr]
Xvalid <- Xtrain[-tr,]
Yvalid <- Ytrain[-tr]
```

   (b) A partir du code ci-dessous, calculer le **taux d'erreur de validation** pour k variant de 1 à 30. Tracer la courbe des erreurs de validation en fonction de k. Quelle valeur de k minimise cette erreur ?

```r
kmax <- 30
err <- rep(NA,kmax)
for (k in 1:kmax)
  {
    pred <- knn(Xtrain_i,Xvalid,Ytrain_i,k)
    err[k] <- sum(pred!=Yvalid)/length(Yvalid)
}
```

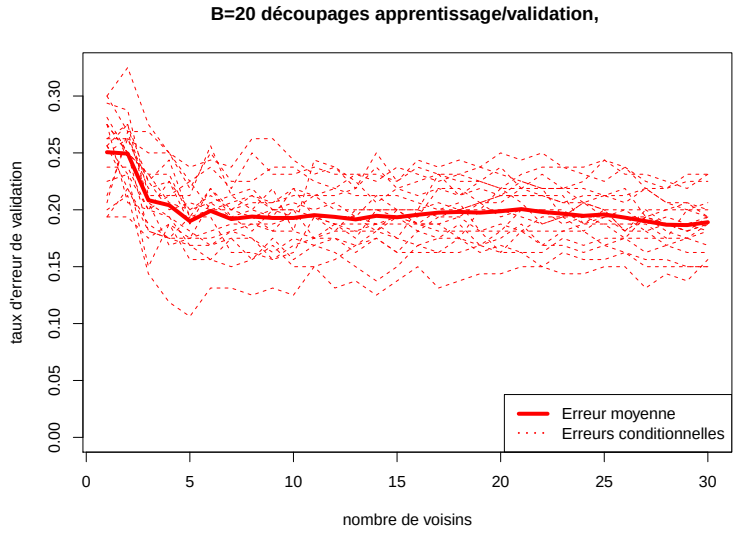
```r
plot(err,type="l",xlab="Nombre de voisins", ylab="erreur de validation",
     ylim=c(0,max(err)))
```



   (c) Recommencer avec plusieurs découpages des 800 données d'apprentissage. Que constatez-vous ?

(d) Recommencer pour $B = 20$ découpages en conservant à chaque fois les erreurs de validation dans une matrice `err` de dimension 30×20. Représenter ensuite ces erreurs et l'erreur moyenne de validation sur un graphique. Quelle valeur de k minimise l'erreur moyenne de validation ?

```
mean_err_valid <- apply(err,1,mean)
lim <-c(0,max(err))
matplot(err,type="l",lty=2,col=2,ylim=lim,
        xlab="nombre de voisins",ylab="taux d'erreur de validation",
        main="B=20 découpages apprentissage/validation, ")
matpoints(mean_err_valid,type="l",col=2,lwd=4)
legend("bottomright", lty=c(1,3),lwd=c(4,2),col=c(2,2),
       legend=c("Erreur moyenne", "Erreurs conditionnelles"))
```

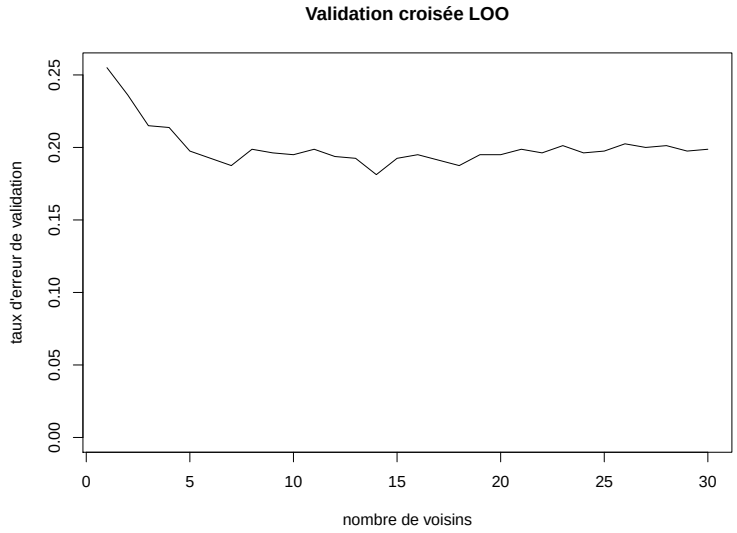


(e) Prédire les données test avec la méthode knn et la valeur de k ainsi choisie. Attention : il faut apprendre sur les 800 données d'apprentissage. Calculer alors le taux d'erreur test.

4. L'erreur de validation pour le choix k peut aussi être calculée par validation croisée LOO (Leave One Out). Attention : uniquement les 800 données d'apprentissage doivent être utilisées pour choisir k.

   (a) Prédire avec la fonction `knn.cv` les 800 données d'apprentissage pour $k = 10$. Quelle est l'erreur de validation ?

   (b) Calculer de la même manière l'erreur de validation croisée L00 pour k variant de 1 à 30. sur les données de d'apprentissage. Représenter ensuite la courbe de ces erreurs en fonction de k. Quelle valeur de k minimise l'erreur de validation croisée LOO ?



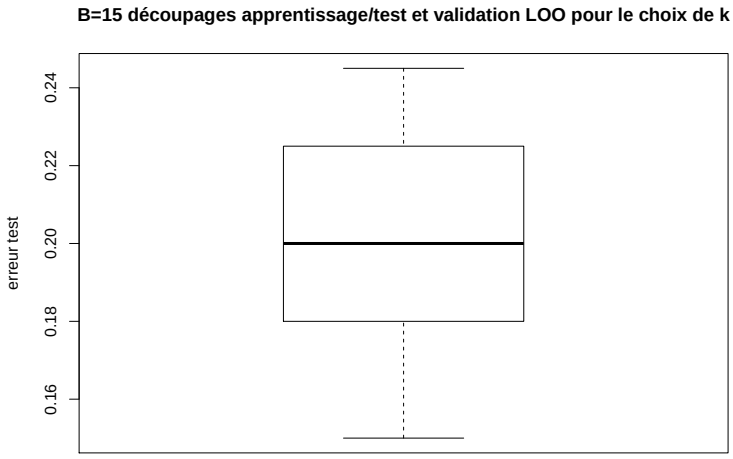
(c) Prédire les données test avec la méthode knn et la valeur de k ainsi choisie. Attention : il faut apprendre sur les 800 données d'apprentissage. Calculer alors le taux d'erreur test.

(d) Critiquez cette approche. Dans quel cas semble-t-il judicieux d'utiliser l'erreur de validation croisée LOO pour le choix de k ?

5. Proposez une troisième approche calculer les erreurs de validation pour le choix du nombre de voisins k.

6. Nous avons vu que le découpage apprentissage/test des $n = 1000$ données a permis :
   — de choisir k sur les 800 données d'apprentissage (par minimisation de l'erreur de validation),
   — de calculer avec cette valeur de k l'erreur sur les 200 données test.

   Mais si on recommence la procédure complète avec un autre découpage aléatoire apprentissage/test, on obtiendra une erreur test différente. Afin de tenir compte de compte de cette variabilité due au découpage aléatoire, on peut par exemple évaluer l'erreur test de la méthode knn (avec choix automatique du paramètre k par validation L00 sur les données d'apprentissage) pour $B = 15$ découpages apprentissage/test. On représente alors les erreurs test dans un boxplot et on calcule finalement l'erreur test moyenne.



Exercice 4. L'objectif : appliquer la méthode knn sur des données réelles et évaluer les performances de cette méthode pour ces données.

Les données concernent $n = 1260$ exploitations agricoles réparties en $K = 2$ classes : la classe des exploitations saines et la classe des exploitations défaillantes. Pour chaque exploitation agricole on a mesuré une batterie de critères économiques et financiers et finalement $p = 4$ ratios financiers étés retenus :

- **R2** : capitaux propres / capitaux permanents,
- **R7** : dette à long et moyen terme / produit brut,
- **R17** : frais financiers / dette totale,
- **R32** : (excédent brut d'exploitation - frais financiers) / produit brut.

La variable qualitative à expliquer est la variable difficulté de paiement (0=sain et 1=défaillant). Voici les 5 premières lignes du tableau de données.

```
   DIFF    R2    R14    R17    R32
1 saine 0.622 0.2320 0.0884 0.4313
2 saine 0.617 0.1497 0.0671 0.3989
3 saine 0.819 0.4847 0.0445 0.3187
4 saine 0.733 0.3735 0.0621 0.4313
5 saine 0.650 0.2563 0.0489 0.4313
```

1. Charger le jeu de données dans R avec la commande `load`. Verifier qu'il y a bien 607 exploitations agricoles défaillantes et 653 exploitations agricoles saines.

```
load("../data/Desbois.rda")
dim(data)
```

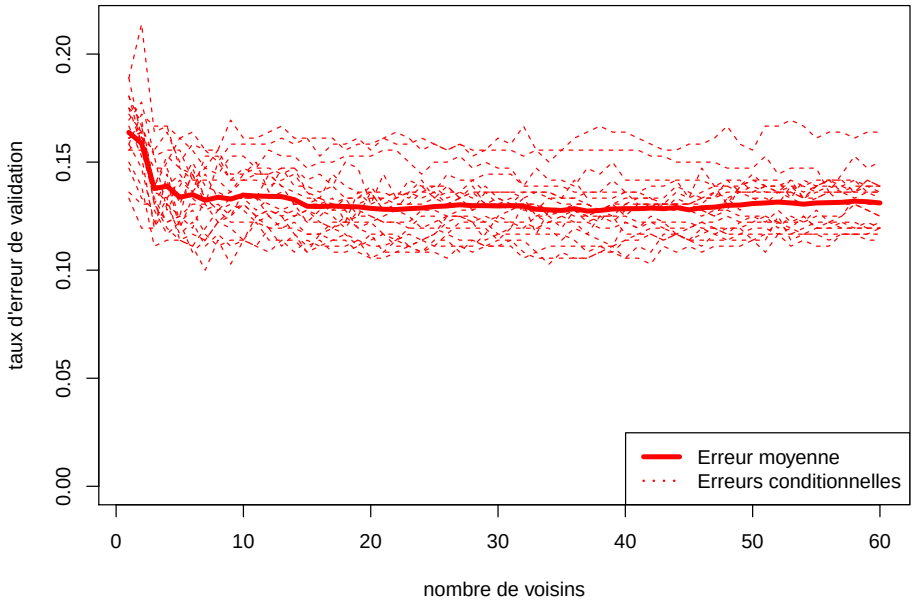
```
## [1] 1260    5
```

```
X <- data[,-1]
Y <- data$DIFF
table(Y)
```

```
## Y
##   0   1
## 653 607
```

2. La méthode knn nécessite le choix du nombre de voisins k. Pour avoir une idée de l'évolution du taux d'erreur en fonction de k, on a appliqué $B = 20$ fois la procédure suivante :
   - découpage aléatoirement des $n = 1260$ exploitations agricoles en deux parties : 900 exploitations pour l'apprentissage et 360 exploitations pour la validation,
   - calcul de l'erreur de validation pour k variant de 1 à 60.

   Construire le graphique des erreurs de validation pour chaque découpage et de l'erreur de validation moyenne en fonction de k.



En quoi ce graphique peut être rassurant avant d'appliquer une procédure de choix automatique de k ?

3. Une autre procédure de choix automatique de k (plus rapide en temps de calcul pour des petits échantillons) consiste à choisir k qui minimise l'erreur de validation croisée LOO. C'est donc cette procédure que l'on utilise maintenant pour évaluer la méthode knn et calculer les critères de performance suivants :

   — le taux de bon classement (tbc),
   — le taux de vrais positifs (tvp),
   — le taux de vrais négatifs (tvn),

   avec ici

   — prédire 1 (défaillant)=positif,
   — prédire 0 (sain)=négatif.

   Pour cela, on répète $B = 20$ fois la procédure suivante :

   — on découpe aléatoirement les $n = 1260$ données en 945 données d'apprentissage (75%) et 315 données test (25%),
   — sur les données d'apprentissage, on choisit automatiquement le nombre de voisins k (dans une grille d'entiers de 1 à 40 par exemple) qui minimise l'erreur de validation croisée LOO,
   — on prédit les données test avec la méthode knn (apprise sur les données d'apprentissage) et pour la valeur k choisie automatiquement,
   — on calcule le tbc, le tvp et le tvn avec les données test.

   (a) La première répétition (premier découpage) de cette procédure (code ci-dessous) donne les résultats suivants :

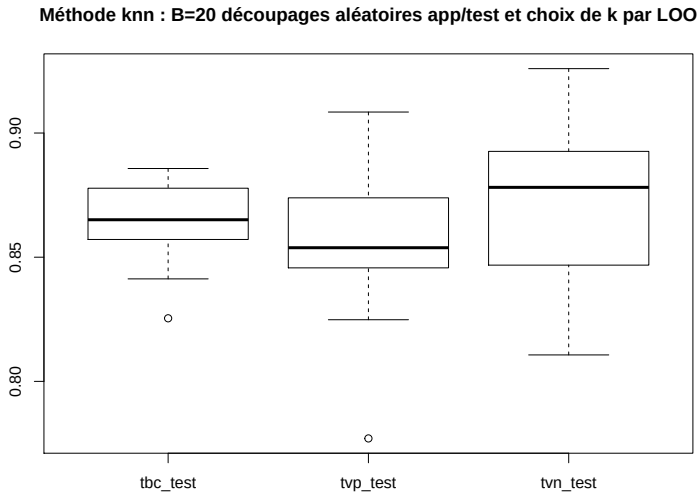
```
kmax <- 40
err_valid <- rep(NA, kmax)
set.seed(10)
tr <- sample(1:nrow(X), 945)
```

```
Xtrain <- X[tr,]
Ytrain <- Y[tr]
Xtest <- X[-tr,]
Ytest <- Y[-tr]

for (k in 1:kmax)
{
  pred <- knn.cv(Xtrain, Ytrain, k)
  err_valid[k] <- sum(pred!=Ytrain)/length(Ytrain)
}
kopt <- which.min(err_valid)
pred <- knn(Xtrain, Xtest, Ytrain, k=kopt)
C <- table(Ytest,pred)
tbc_test <-  (C[1,1]+ C[2,2])/length(Ytest)
tvp_test <- C[2,2]/(C[2,1]+C[2,2])
tvn_test <- C[1,1]/(C[1,1]+C[1,2])
```

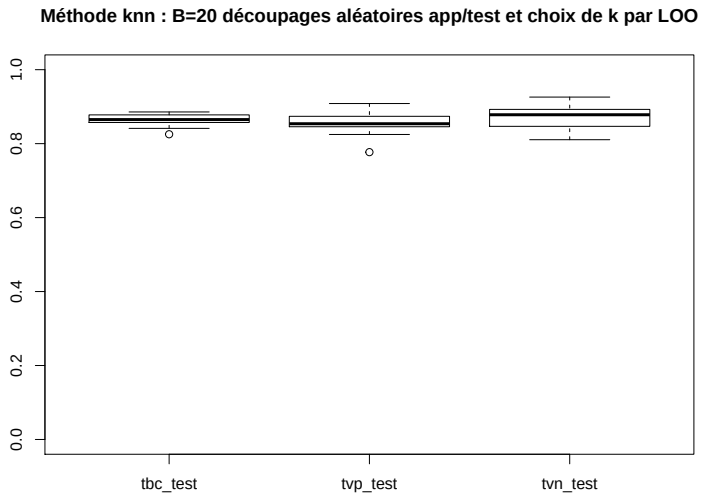
```
table(Ytest)

## Ytest
##   0   1
## 169 146

table(Ytest, pred)

##      pred
## Ytest   0   1
##     0 158  11
##     1  14 132
```

Calculer le tbc, le tvp et le tvn pour ce premier ensemble test.

(b) Les $B = 20$ répétitions de cette procédure permettent d'obtenir le graphique suivant :



Interprétez ce graphique. Comparez avec le graphique suivant :



4. On considère maintenant que les performances de la méthode knn ainsi calibrée (choix automatique de k par LOO) sont assez bonnes pour utiliser cette méthode sur de nouvelles données (dont on ne connaît pas la classe). Il faut alors calibrer k et apprendre avec **toutes les données**. Prédire ainsi la classe d'une exploitation agricole ayant les valeurs suivantes :

```
R2    R14    R17    R32
0.700 0.300 0.100 0.500
```

Exercice 5. Les objectifs :

— Constuire une score dans le cas binaire avec la méthode knn.
— Construire la courbe ROC de ce score et évaluer le critère AUC sur des données test.
— Choisir un seuil pour constuire une règle de classification.

On reprend les données sur les exploitations agricoles de l'exercice 5.

```
load("../data/Desbois.rda")
X <- data[,-1]
Y <- data$DIFF
```

On veut maintenant construire un score de détection du risque financier applicable aux exploitations agricoles. Ce score (note) devra être d'autant plus grand que la probabilité pour cette exploitation d'être défaillante est grande.

Attention : le score en classification binaire est associé à une des classes (généralement celle considérée comme positive).
Pour rappel, ici on prend :
— prédire 1 (défaillant)=positif,
— prédire 0 (sain)=négatif.

On va constuire ce score avec la méthode knn pour $k = 30$ voisins.

1. Découpez aléatoirement les $n = 1260$ exploitations agricoles en 945 exploitations pour les données d'apprentissage et 360 exploitations pour les données test.

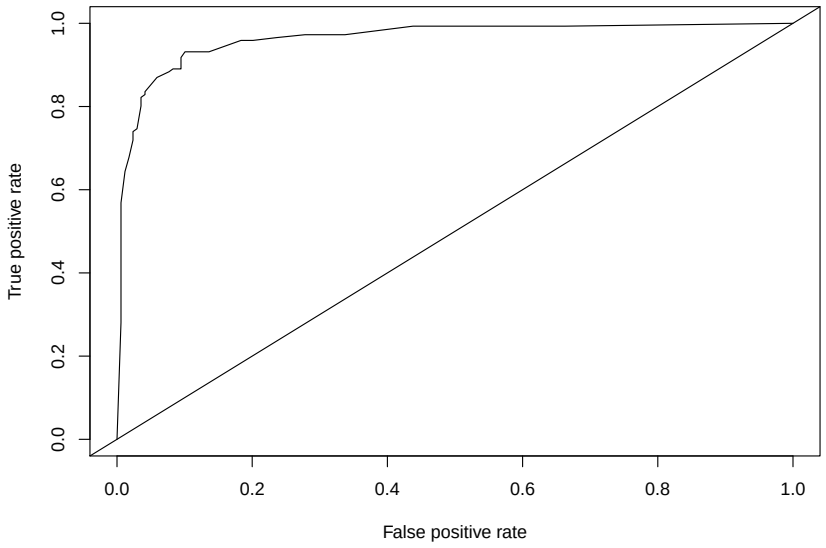
```
set.seed(10)
tr <- sample(1:nrow(X),945)
Xtrain <- X[tr,]
Ytrain <- Y[tr]
Xtest <- X[-tr,]
Ytest <- Y[-tr]
```

2. Utiliser la fonction **prob_knn** ci-dessous pour calculer le score des exploitations de l'ensemble test (leur probabilité d'être défaillante) pour $k = 30$ voisins.

```
prob_knn <- function(Xtrain, Xtest, Ytrain, k)
{
  #proba à posteriori d'être dans la classe 1 : P(Y=1/X=x)
  prob <- rep(NA, nrow(Xtest))
  res <- knn(Xtrain, Xtest, Ytrain, k=k, prob=TRUE)
  prob[res==1] <- attr(res,"prob")[res==1]
  prob[res==0] <- 1-attr(res,"prob")[res==0]
  return(prob)
}
```

3. A partir du code ci-dessous, constuire sur les donnée test la courbe ROC de la méthode knn avec $k = 30$ voisins. Comment est constuite cette courbe ? Quelle est la valeur du AUC sur les données test ? Ce score est-il de bonne qualité ?

```
library(ROCR) # 3 fonctions : prediction, performance, plot
pred <- prediction(score, Ytest, label.ordering=c("0","1"))
# object of class S4
#label.ordering : indiquer le libellé de la classe negative puis positive.
perf <- performance(pred,  "tpr", "fpr")
plot(perf) #courbe ROC
abline(a=0, b=1)
```

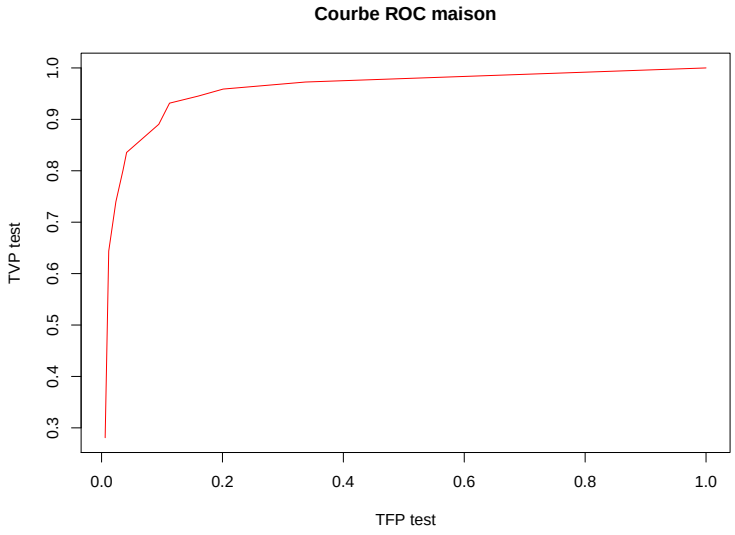


```r
auc <- performance(pred, "auc")@y.values[[1]]
auc
```

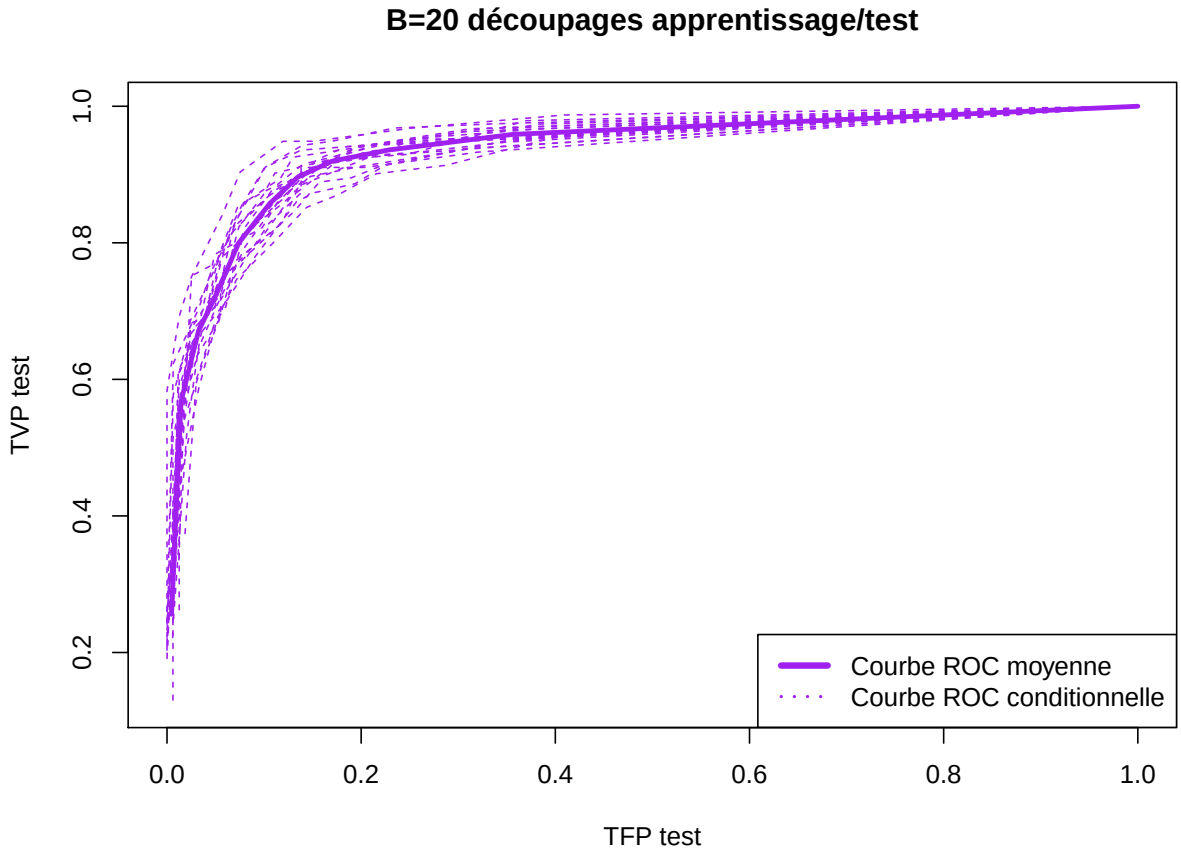
```
## [1] 0.96
```

4. Calculer le critère AUC pour 50 découpages aléatoires et retrouver le graphique ci-dessous.

5. Constuire sur les donnée test la courbe ROC de la méthode knn avec $k = 30$ voisins, **sans utiliser cette fois** le package `ROCR`.

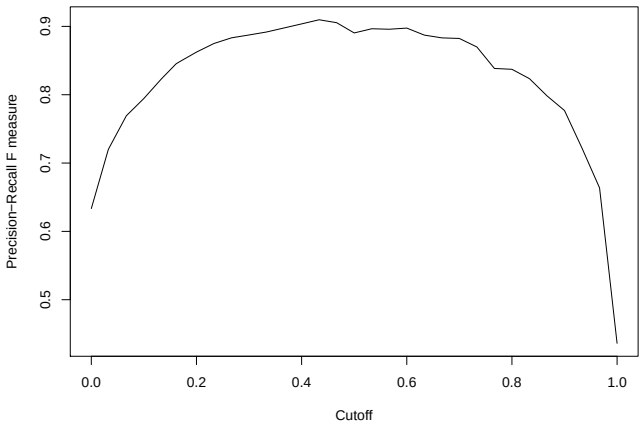


6. Recommencer pour $B = 20$ découpages apprentissage/test et représenter les 20 courbes ROC et la courbe ROC moyenne.



7. On peut utiliser le score (obtenu ici avec la méthode knn) pour définir une régle de classification. En général la règle de classification d'une nouvelle donnée x est la suivante :
   — Si $score(x) \leq 0.5$ alors x affecté à la classe 1 (défaillante).
   — Si $score(x) < 0.5$ alors x affecté à la classe 0 (saine)
   ce qui revient à affecter à la classe la plus probable à posteriori. Mais on peut aussi modifier le seuil de 0.5 pour optimiser un critère de performance.

   Quel est le seuil (pour un découpage apprentissage/test) qui maximise la F-mesure (utiliser le package `ROCR`) ?



8. Recommencer pour plusieurs découpages.