Apprentissage supervisé - TP1

Sommaire

Les différentes librairies qui seront utilisés pour ce TP sont listées ici.

```
library("caret") # ensemble de meta-fonctions nécessaires au processus d'apprentissage supervisé
library("class") # fonction knn
library("e1071") # librairie nécessaire pour la fonction tune.knn
library("FactoMineR") # ACM
library("ggplot2") # data visualisation
library("klaR") # fonction Naivebayes utilisée avec la librairie caret
library("mlbench") # dataset Vehicle
```

Exercice 1 - Règle et risque de Bayes en discrimination binaire

Voici 3 problèmes de discrimination binaire, on a simulé un échantillon de taille $n = 1000$ pour les cas 1 et 2.

Cas 1

Pour $i = 1..1000$, $X_i \sim \mathcal{N}(0,1)$, $U_i \sim \mathcal{U}[0,1]$ on a :

$$Y_i = \begin{cases} \mathbb{1}_{U_i \leq 0,1} & \text{si } X_i \leq 0 \\ \mathbb{1}_{U_i > 0,2} & \text{si } X_i > 0 \end{cases}$$

Cas 2

Pour $i = 1..1000$, $X_i \sim \mathcal{N}(0,1)$, $U_i \sim \mathcal{U}[0,1]$ on a :

$$Y_i = \begin{cases} \mathbb{1}_{U_i \leq 0,2} & \text{si } X_i \leq 0 \\ \mathbb{1}_{U_i > 0,4} & \text{si } X_i > 0 \end{cases}$$

Cas 3

On travaille sur des données réelles issues d'une enquête, à partir d'un échantillon tiré au hasard de $n = 1000$ consommateurs de café. La variable à expliquer Y est qualitative binaire et prend les modalités "sucré" et "non sucré". La variable explicative à notre disposition est X qui représente le sexe. On dispose de la table avec en lignes les 1000 consommateurs et en colonnes les 2 variables X et Y. Des résultats de statistique bivariée nous donnent :

- Parmi les femmes, on a 20% qui prennent du sucre dans leur café $Y = sucré$
- Parmi les hommes, on a 10% qui prennent du sucre dans leur café $Y = sucré$

Question 1 - Règle de Bayes

> *Donner dans chacun des trois cas, si c'est possible, la règle de Bayes et le risque de Bayes.*

Dans le cas de la discrimination binaire la règle de Bayes est de la forme suivante :

$$\begin{aligned} f^*(x) : \mathbb{R} &\to \{0,1\} \\ x &\mapsto f^*(x) = \begin{cases} 1 & \text{si } P(Y = 1 | X = x) > 0,5 \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

Cas 1

On définit la règle de Bayes en calculant $P(Y = 1 | X = x_i)$ selon les valeurs de X.

Pour $X \leq 0$: $P(Y = 1 | X = X_i) = P(U_i \leq 0,1) = 0,1$ d'où $f^*(X_i) = 0$.

Pour $X > 0$: $P(Y = 1 | X = X_i) = P(U_i > 0,2) = 1 - P(U_i \leq 0,2) = 0,8$ d'où $f^*(X_i) = 1$.

La règle de Bayes est donc :

$$f^*(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

On calcule maintenant l'erreur de Bayes. Elle correspond à la différence entre la prédiction obtenue avec la règle de Bayes, $f^*(X)$, et la vraie valeur, Y.

$$\begin{aligned} \mathcal{R}_P^* &= P\big(Y \neq f^*(X)\big) \\ &= P(Y = 1 \cap f^*(X) = 0) + P(Y = 0 \cap f^*(X) = 1) \\ &= P(Y = 1 \cap X \leq 0) + P(Y = 0 \cap X > 0) \\ &= P(X \leq 0) \times P(Y = 1 | X \leq 0) + P(X > 0) \times P(Y = 0 | X > 0), \text{ avec } P(A \cap B) = P(A|B) \times P(B) \\ &= 0{,}5 \times 0{,}1 + 0{,}5 \times 0{,}2 \\ &= 0{,}15 \end{aligned}$$

Cas 2

On définit la règle de Bayes en calculant $P(Y = 1 | X = x_i)$ selon les valeurs de X.

Pour $X \leq 0$: $P(Y = 1 | X = X_i) = P(U_i \leq 0{,}2) = 0{,}2$ d'où $f^*(X_i) = 0$.

Pour $X > 0$: $P(Y = 1 | X = X_i) = P(U_i > 0{,}4) = 1 - P(U_i \leq 0{,}4) = 0{,}6$ d'où $f^*(X_i) = 1$.

La règle de Bayes est donc :

$$f^*(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{sinon} \end{cases}$$

On calcule maintenant l'erreur de Bayes. Elle correspond à la différence entre la prédiction obtenue avec la règle de Bayes, $f^*(X)$, et la vraie valeur, Y.

$$\begin{aligned} \mathcal{R}^* &= P\big(Y \neq f^*(X)\big) \\ &= P(Y = 1 \cap f^*(X) = 0) + P(Y = 0 \cap f^*(X) = 1) \\ &= P(Y = 1 \cap X \leq 0) + P(Y = 0 \cap X > 0) \\ &= P(X \leq 0) \times P(Y = 1 | X \leq 0) + P(X > 0) \times P(Y = 0 | X > 0) \\ &= 0{,}5 \times 0{,}2 + 0{,}5 \times 0{,}4 \\ &= 0{,}3 \end{aligned}$$

Cas 3

Comme la loi jointe de (X, Y) n'est pas connue il est impossible de connaître la règle de Bayes et donc l'erreur de Bayes.

Question 2 - Complexité

> *Est-il possible de donner un indicateur de la complexité de ces problèmes et ainsi de les ordonner en fonction de leur complexité ?*

Le risque de Bayes donne une idée de la complexité du problème. Plus l'erreur théorique minimale est importante plus on considère le problème comme complexe. Ainsi le cas 2 est plus complexe que le premier, théoriquement.

Question 3 - Simulation

> *Simuler les données du cas 2 et créer en plus un échantillon test de taille 200 (avec le même modèle).*

On va générer un vecteur de X et un vecteur de Y de longueur 1000. On fixe une graine pour que nos résultats soient reproductibles.

```r
# Nombre d'éléments dans l'échantillon
n_train <- 1000

# Générer X selon une loi normale
set.seed(1)
X <- rnorm(n_train)

# Générer U selon une loi uniforme
set.seed(2)
U <- runif(n_train)

# Générer le vecteur réponse Y selon le cas 2
Y1 <- rep(0, n_train)
Y1[X <= 0 & U <= 0.2] <- 1
Y1[X > 0 & U > 0.4] <- 1

# Y est transformé en "factor" pour faciliter l'utilisation de la fonction tune.knn dans la suite
Y <- as.factor(Y1)

# Créer un data.frame avec X et Y
data <- data.frame(X, Y)
```

Le graphe suivant montre la densité de X selon la valeur prise par Y. On y voit une différence entre les valeurs positives et négatives de X.

```r
ggplot(
 data = data,
 mapping = aes(x = X, fill = Y)
) +
 geom_density(
  alpha = .7
 ) +
 labs(
  x = "X",
  y = "Densité",
  title = "Données d'apprentissage",
  fill = "Y"
 )
```

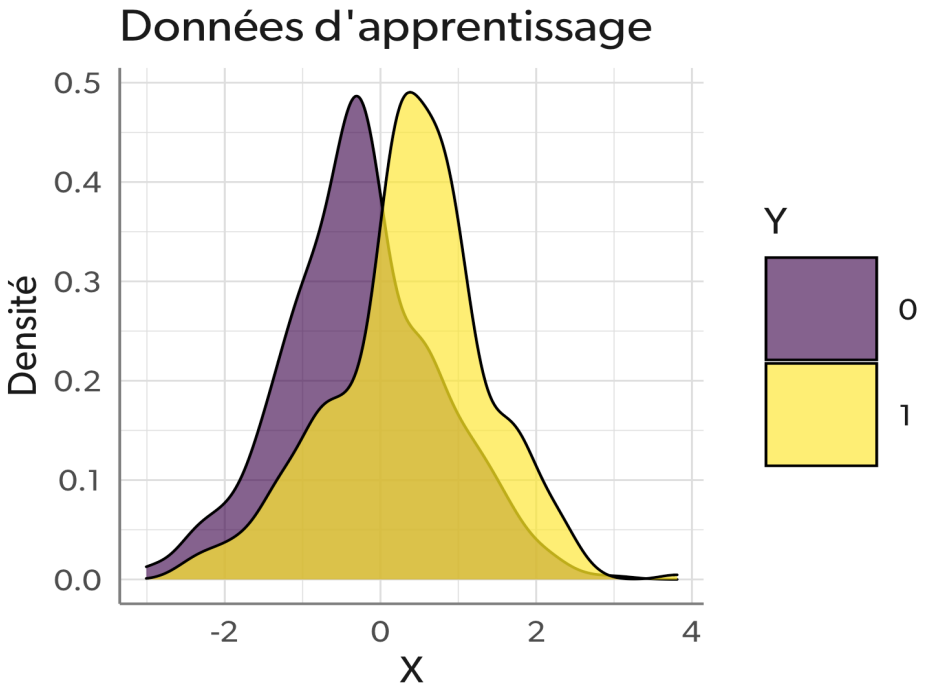


Figure 1: Distribution des données d'apprentissage selon la classe de la variable cible.

De la même manière on peut simuler des données de test qui seront utilisées pour mesurer l'erreur de nos modèles de prédictions.

```
n_test <- 200
set.seed(3)
X <- rnorm(n_test)
set.seed(4)
U <- runif(n_test)
Y2 <- rep(0, n_test)
Y2[X <= 0 & U <= 0.2] <- 1
Y2[X > 0 & U > 0.4] <- 1
Y <- as.factor(Y2)
test <- data.frame(X, Y)
ggplot(
 data = test,
 mapping = aes(x = X, fill = Y)
) +
 geom_density(
  alpha = .7
 ) +
 labs(
  x = "X",
  y = "Densité",
  title = "Données de test",
  fill = "Y"
 )
```

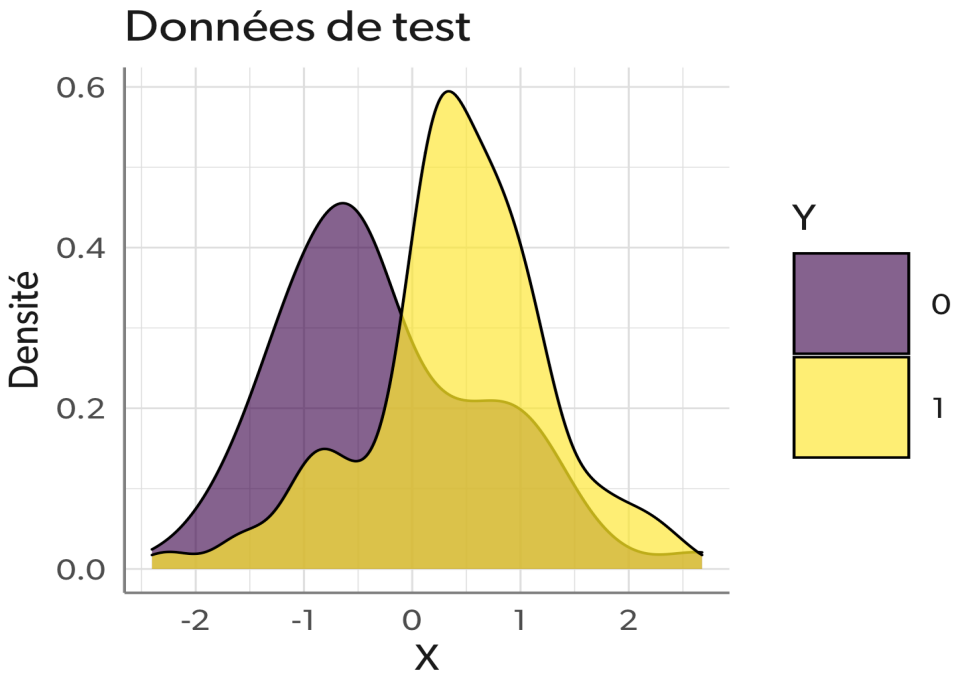


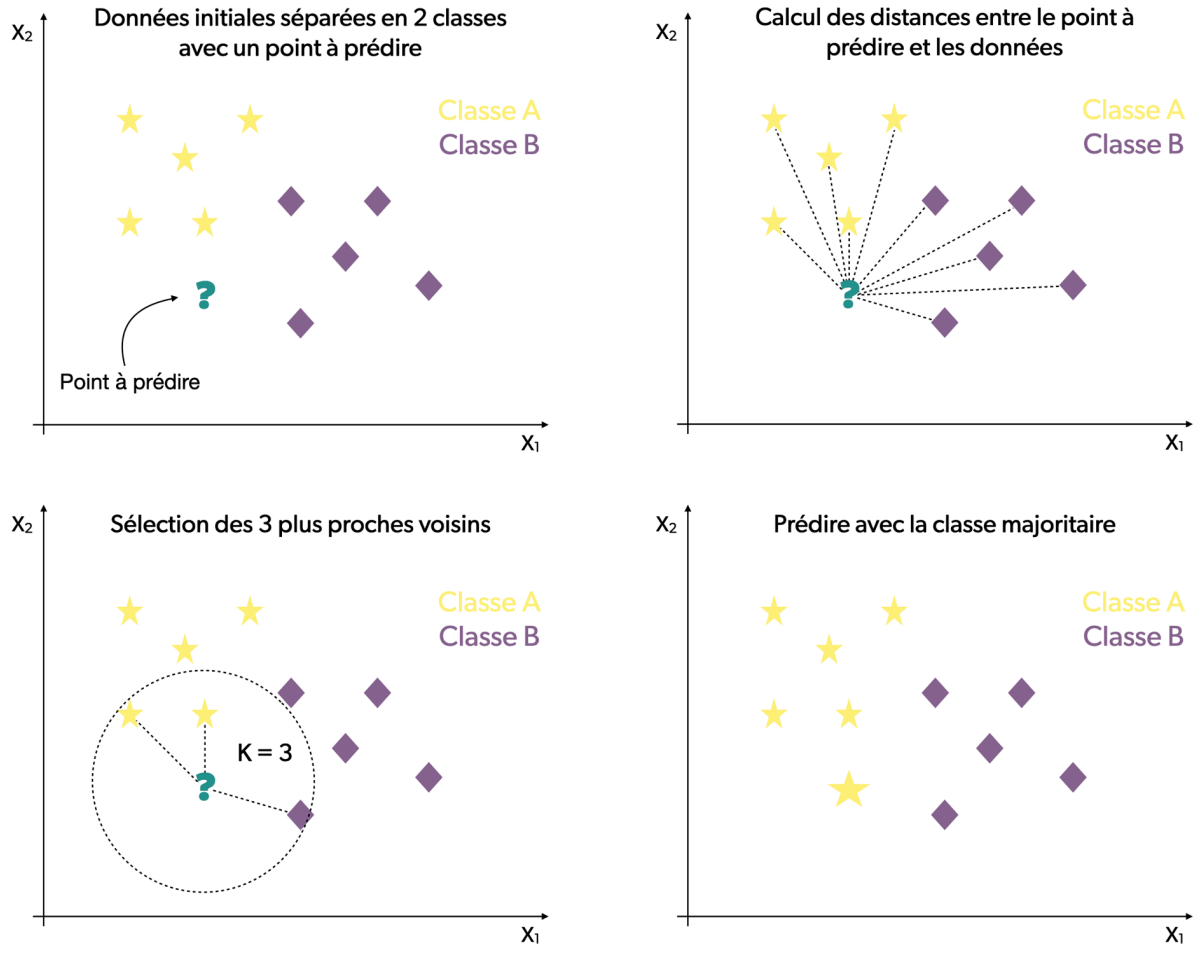
Figure 2: Distribution des données de test selon la classe de la variable cible.

Question 4 - Knn

Quelle est la règle de décision associée à l'algorithme des k plus proches voisins ? Mettre en oeuvre les k plus proches voisins. Justifier le choix des paramètres et commenter les résultats en validation et sur l'échantillon test.

Le modèle KNN

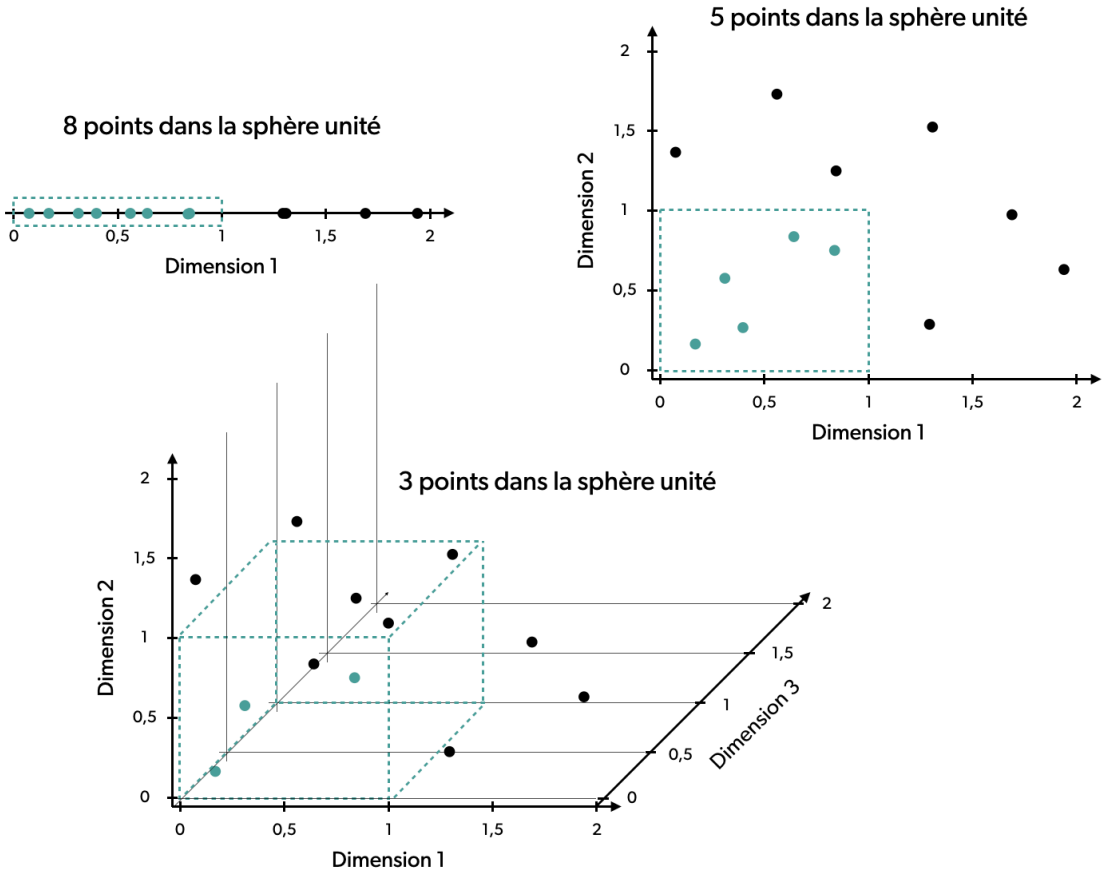
Le modèle des K plus-proches-voisins (*K-nearest-neighbours* en anglais) est un modèle d'apprentissage supervisé utilisable en classification et régression. Soient (x_i, y_i) pour $i = 1, \dots, n$ des données d'apprentissage et x le point à prédire. Pour effectuer la prédiction de la valeur y de x, on calcule la distance qui sépare x de chaque x_i. On sélectionne ensuite les k points $(x_1, \dots, x_k)$ les plus proches de x parmi les points $x_i, i = 1, \dots, n$ puis on détermine la classe à prédire en faisant un vote à la majorité sur $(y_1, \dots, y_k)$ pour la classification et en prenant la moyenne $\frac{1}{k}\sum_{j=1}^{k} y_j$ pour la régression.



Modèle des plus proches voisins expliqué visuellement.

Quelques points d'attention :

- L'hyper-paramètre k est bien sûr l'élément central de ce modèle. Il faut choisir sa valeur de manière réfléchie (on verra dans la suite comment procéder).
- La distance utilisée est aussi un point important. On choisira souvent la distance euclidienne mais il est bon de savoir qu'on peut utiliser n'importe quelle mesure de similarité selon les besoins.
- Enfin il faut savoir que plus il y a de prédicteurs plus les plus proches voisins seront potentiellement éloignés, c'est ce que l'on appelle le fléau de la dimension. Ce papier fournit des détails intéressants à ce sujet.



Le fléau de la dimension. Plus le nombre de dimensions augmente, plus le nombre de points dans la sphère unité va diminuer. Les points s'éloignent les uns des autres avec l'augmentation du nombre de dimensions.

La règle de décision

Dans le cas d'une classification, la règle de décision associée aux k plus-proches-voisins est le vote à la majorité. Exemple : pour $k = 5$, si 3 voisins ont une valeur y de A et 2 voisins ont une valeur y de B alors la prédiction sera $\hat{y} = A$. Dans le cadre de la régression c'est la moyenne des valeurs de Y des 5 voisins sélectionnés qui sera utilisée.

Un premier modèle

On commence par lancer un modèle de k plus-proches-voisins avec un choix arbitraire pour k, 10 par exemple. UNe graîne est fixée pour des questions de reproductibilité. On utilise la fonction `class::knn()` qui prend 4 paramètres en entrée :

- `train` : les prédicteurs des données d'apprentissage,
- `test` : les prédicteurs des données de tests sur lesquelles se feront les prédictions,
- `cl` : le vecteur de la variable cible des données d'apprentissage,
- `k` : le nombre de plus-proches-voisins souhaité.

Cette fonction renvoie un vecteur de prédictions effectuées sur les données de tests.

```
set.seed(123456)
knn_pred <- knn(
  train = as.matrix(data$X), # données d'apprentissage
  test = as.matrix(test$X), # données à prédire
  cl = data$Y, # vraies valeurs
  k = 10 # nombre de voisins
)
```

On peut calculer l'erreur et la précision de ce modèle.

```
# Taux de bonnes prédictions
sum(knn_pred == test$Y) / n_test

# Taux d'erreur
1 - sum(knn_pred == test$Y) / n_test
```

L'erreur est de 0.305. Cela veut dire que le modèle se trompe dans 30.5% des cas.

Il est important de noter que le modèle est construit sur les données d'apprentissage et que l'erreur est calculée à partir de prédictions effectuées sur des données différentes, les données de test. On revient dans la suite de ce document sur ce sujet.

Le choix de k

La valeur de k a été définie de façon arbitraire, avec $k = 10$. Il est bien sûr possible, et même souhaitable, d'optimiser ce choix en essayant différentes valeurs de k et en choisissant la meilleure. La notion de "meilleure valeur" est associée à l'erreur du modèle. La valeur k générant le modèle avec l'erreur la plus basse est la meilleure.

Le procédé utilisé pour trouver cette valeur est appelé "optimisation d'hyper-paramètres". Les données d'entrainement vont être découpées en un échantillon d'apprentissage et un échantillon de validation. Pour chaque valeur de k à tester, des prédictions sont effectuées sur les données de validation en prenant les données d'apprentissage comme référence et l'erreur de validation est calculée.

La fonction `e1071::tune.knn()` est utilisée pour réaliser cette optimisation de l'hyper-paramètre k. Elle prend 4 paramètres en entrée :

- `x` : les prédicteurs
- `y` : le vecteur cible
- `k` : un vecteur de valeurs de k à tester
- `tunecontrol` : la méthode d'échantillonnage, sous la forme d'une fonction. `tune.control(sampling = "cross")` définit la validation croisée. Dans ce cas il faut spécifier l'argument `cross = 5` pour faire une validation croisée 5 blocks, *5-fold cv* en anglais. D'autres méthodes d'échantillonnage sont possibles.

Cette fonction retourne une liste avec les résultats du processus de choix de k.

```
knn_cross_results <- tune.knn(
 x = data$X, # predicteurs
 y = data$Y, # réponse
 k = 1:50, # essayer knn avec K variant de 1 à 50
 tunecontrol = tune.control(sampling = "cross"), # utilisation de la cross validation
 cross = 5 # 5 blocs
)
```

Les erreurs obtenues pour chaque valeur de k sont visualisées sur le graphique suivant.

```
ggplot(
 data = knn_cross_results$performances,
 mapping = aes(x = k, y = error)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "k",
  y = "Erreur de validation",
  title = "Evolution de l'erreur selon k"
 )
```

Figure 3: Résultats de la validation croisée pour le choix de k. Pour chaque valeur de k entre 1 et 50, l'erreur de validation est représentée. La valeur de k correspondant à l'erreur la plus basse est choisie.

Le meilleur k est obtenu avec la commande `knn_cross_results$best.parameters$k`, ici 24. Le choix de k n'est ainsi plus arbitraire mais basé sur nos données et le k donnant l'erreur minimale (sur les données de validation) est choisi.

Remarque

La valeur optimale de k peut varier à chaque fois que la commande est lancée. UN contournement à ce problème est proposé dans l'exercice suivant.

La valeur sélectionnée de k est utilisée pour réaliser des prédictions sur les données de tests. Il est ainsi possible de comparer les résultats avec le modèle précédent, $k = 10$. Le taux d'erreur est calculé comme précédemment.

```
set.seed(123456)
knn_pred <- knn(
 train = as.matrix(data$X), # données d'apprentissage
 test = as.matrix(test$X), # données à prédire
 cl = data$Y, # vraies valeurs
 k = knn_cross_results$best.parameters$k # nombre de voisins
)
```

L'erreur est de 0.28.

Analyse de l'erreur

L'erreur est bien inférieure à celle obtenue avec $k = 10$. L'erreur obtenue est très proche de l'erreur de Bayes calculée à la question 1. Pour rappel elle est de 0.3. Pour $k = 30$, l'erreur est même inférieure au risque de Bayes. Le risque de Bayes est le risque théorique minimum, il n'est pas possible de construire un modèle qui produise une erreur plus basse. Comment alors expliquer cette différence ? Elle est due au fait que les données générées ne correspondent pas parfaitement à la loi définie dans l'énoncé du fait du faible nombre d'observations simulées (200). On peut simuler un nombre supérieur de données, 2000 par exemple, et recalculer l'erreur. Pour ce faire on procède comme précédemment.

```
n_test2 <- 2000
set.seed(3)
X <- rnorm(n_test2)
set.seed(4)
U <- runif(n_test2)
Y2 <- rep(0, n_test2)
Y2[X <= 0 & U <= 0.2] <- 1
Y2[X > 0 & U > 0.4] <- 1
Y <- as.factor(Y2)
test2 <- data.frame(X, Y)
set.seed(123456)
knn_pred2 <- knn(
 train = as.matrix(data$X), # données d'apprentissage
 test = as.matrix(test2$X), # données à prédire
 cl = data$Y, # vraies valeurs
 k = knn_cross_results$best.parameters$k # nombre de voisins
)
```

L'erreur est maintenant de 0.321. Elle est bien supérieure à l'erreur minimale théorique.

Importance de l'échantillon de validation pour estimer le meilleur hyper-paramètre

Le calcul de l'erreur de généralisation d'un modèle doit se faire sur des données qui n'ont jamais servi dans la construction du modèle. La raison est que dans le cas contraire l'erreur de généralisation sera sous-estimée. On pourrait alors conclure que notre modèle est bon alors que celui-ci sera mauvais sur de nouvelles données.

Reprenons les résultats de la Figure 3. L'erreur est calculée sur les données de validation, données que le modèle n'a pas vues pour être construit. Si le même graphe est refait en utilisant les données d'apprentissage pour calculer l'erreur, à quoi ressemblera la courbe obtenue ? Plus particulièrement quelle sera l'erreur pour $k = 1$? Pour $k = 200$?

Visualisons les résultats obtenus en effectuant la prédiction sur les données d'apprentissage.

```
knn_preds <- sapply(
 X = 1:50,
 FUN = function(k){
  set.seed(123456)
  knn_pred <- knn(
   train = as.matrix(data$X),
   test = as.matrix(data$X),
   cl = data$Y,
   k = k
  )
  return(1 - sum(knn_pred == data$Y) / n_train)
 }
)
ggplot(
 data = data.frame('k' = 1:50, 'error' = knn_preds),
 mapping = aes(x = k, y = error)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "k",
  y = "Erreur sur l'échantillon d'apprentissage",
  title = "Evolution de l'erreur selon k"
 )
```

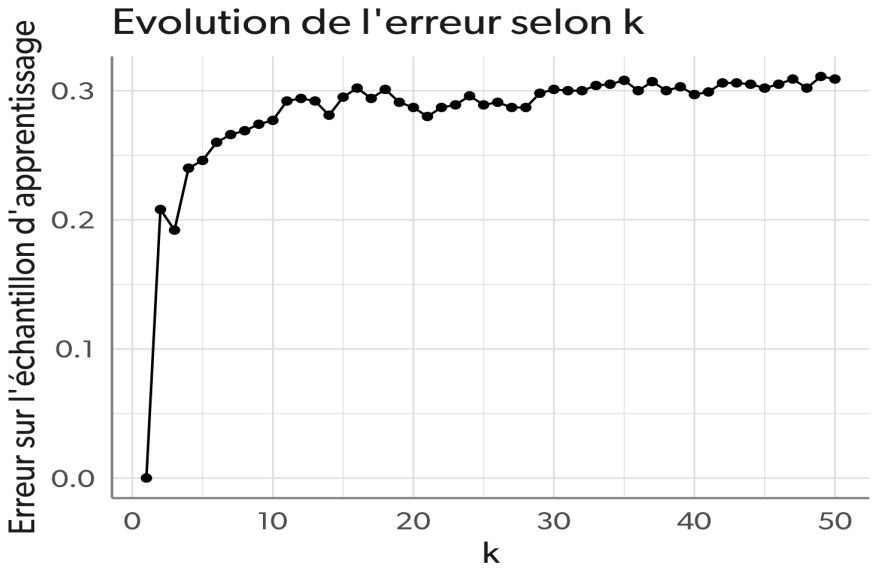


Figure 4 : Calcul de l'erreur sur l'échantillon d'apprentissage pour différentes valeurs de k.

La valeur de k avec l'erreur la plus basse est 1. En effet le plus proche voisin d'un point x de l'échantillon d'apprentissage est lui-même. Si ce modèle est utilisé pour effectuer des prédictions sur de nouvelles données, l'erreur sera bien plus importante.

Sur cet exemple simple, il apparaît clairement qu'il n'est pas possible d'utiliser les données d'apprentissage pour estimer l'erreur de généralisation du modèle. L'estimation sera sous-estimée.

Question 5 - Naive Bayes

Quelle est la règle de décision associée au bayésien naïf ? Mettre en oeuvre le Bayésien naïf. Justifier le choix des paramètres et commenter les résultats en validation et sur l'échantillon test.

Le modèle bayésien naïf

Le modèle bayésien naïf est un modèle d'apprentissage supervisé, applicable en classification, dont l'objectif est de trouver la classe y maximisant $P(Y = y|X = x)$.

$$\underset{y \in \mathcal{Y}}{\operatorname{argmax}} \ \hat{P}(Y = y|X = x) \qquad (1)$$

Autrement dit, le modèle bayésien naïf répond à la question : sachant les valeurs prises par les prédicteurs X, quelle est la classe de Y la plus probable ?

Ce modèle se base sur la formule de Bayes :

$$P(Y = y|X = x) = \frac{P(Y = y) \times P(X = x|Y = y)}{P(X = x)} \qquad (2)$$

Estimer $P(Y = y|X = x)$ revient donc à estimer $P(Y = y)$, $P(X = x|Y = y)$ et $P(X = x)$.

$P(X = x)$ ne dépendand pas de y l'Equation 1 devient :

$$\underset{y \in \mathcal{Y}}{\operatorname{argmax}} \ \hat{P}(Y = y) \times \hat{P}(X = x|Y = y) \qquad (3)$$

Le calcul de $\hat{P}(Y = y)$ se fait directement par les fréquences empiriques. $\hat{P}(X = x|Y = y)$ se calcule en faisant l'hypothèse que toutes les X_j, les différentes variables explicatives, sont indépendantes conditionnellement à Y. On obtient alors :

$$\hat{P}(X = x|Y = y) = \prod_{j=1}^{J} \hat{P}\left(X_j = x_j|Y = y\right)$$

Cette hypothèse rend le modèle bayésien naïf posible. L'estimation de $P(X = x|Y = y)$ serait impossible sans cette hypothèse "naïve". Elle n'est cependant jamais complètement vérifiée sans que cela ne détériore les performances du modèle. En conclusion l'Equation 1 se résume à :

$$\underset{y \in \mathcal{Y}}{\operatorname{argmax}} \ \hat{P}(Y = y) \times \prod_{j=1}^{J} \hat{P}\left(X_j = x_j | Y = y\right)$$

La règle de décision

Avec le modèle bayésien naïf on calcule $\hat{P}(Y = y|X = x)$ pour $y = 0{,}1$. Pour choisir la classe Y on va prendre la probabilité la plus forte, c'est à dire celle qui est supérieur à $0{,}5$.

Un premier modèle

La fonction utilisée pour réaliser un modèle bayésien naïf est la fonction `klaR::NaiveBayes()`. Elle prend 2 paramètres en entrée :

- `formula` : une formule spécifiant le modèle, ici ce sera simplement `Y ~ X`,
- `data` : un data.frame contenant les variables utilisées dans l'argument `formula`

```
naive_bayes_model <- NaiveBayes(
 formula = Y ~ X,
 data = data
)
```

On peut utiliser ce modèle pour effectuer des prédictions sur le jeu de données de test et ainsi mesurer l'erreur sur des données non utilisées pour apprendre les probabilités $\hat{P}(Y = y|X = x)$.

```
naive_bayes_pred <- predict(
 object = naive_bayes_model,
 newdata = test
)
```

Comme précédemment on peut mesurer la précision du modèle ainsi que son erreur de la façon suivante.

```
# Taux de bonnes prédictions
sum(naive_bayes_pred$class == test$Y) / n_test

# Taux d'erreur
1 - sum(naive_bayes_pred$class == test$Y) / n_test
```

Le taux d'erreur est de 0.39.

Le choix des paramètres

Le modèle bayésien naïf a été lancé avec ses paramètres par défaut. 3 hyper-paramètres peuvent être modifiés :

- l'utilisation ou non d'un noyau pour estimer les densités des variables numériques, `usekernel`
- la largeur de bande utilisée par le noyau (ou plus précisément un facteur multiplicatif qui sera appliqué à une largeur de bande pré-définie), `adjust`
- la correction de Laplace à appliquer pour les variables catégorielles, `fL`.

Pour choisir la meilleure combinaison d'hyper-paramètres pour le modèle bayésien naïf, on va diviser notre jeu de données, `data`, en 2 groupes : apprentissage et validation. Comme pour le choix de k dans le modèle des k plus-proches-voisins, le but est d'estimer l'erreur sur les données de validation pour chaque combinaison d'hyper-paramètres et de sélectionner la combinaison avec l'erreur la plus faible.

On commence par spécifier l'ensemble des combinaisons d'hyper-paramètres que l'on souhaite évaluer. Parmi les 3 hyper-paramètres nécessaires seuls 2 seront utilisés dans notre cas : `usekernel` et `adjust`. En effet la correction de Laplace sert pour les variables catégorielles or notre jeu de données ne comporte que des données numériques.

```
hyperparam_grid <- expand.grid(
 usekernel = TRUE, # si vrai utilisation d'un noyau sinon gaussien
 fL = 0, # correction avec lissage de Laplace (ici ce paramètre n'est pas nécessaire, x étant continue)
 adjust = seq(1, 5, by = 1) # largeur de bande
)
hyperparam_grid <- rbind(hyperparam_grid, data.frame(usekernel = FALSE, fL = 0, adjust = 1))
```

On construit ensuite la séparation entre données d'apprentissage et de validation.

```
set.seed(123456)
train_index <- sample(1:nrow(data), size = nrow(data) * .8)
validation_index <- setdiff(1:nrow(data), train_index)
train <- data[train_index, ]
validation <- data[validation_index, ]
```

On va maintenant entraîner autant de modèles qu'il y a de combinaisons et calculer l'erreur sur les données de validation pour chacun de ces modèles.

```
results <- cbind(hyperparam_grid, "error" = 0)
for (i in 1:nrow(results)) {
 naive_bayes_model <- NaiveBayes(
  formula = Y ~ X,
  data = train,
  usekernel = results$usekernel[i],
  fL = results$fL[i],
  adjust = results$adjust[i]
 )
 naive_bayes_pred <- predict(
  object = naive_bayes_model,
  newdata = validation
 )
 results[i, "error"] <- 1 - sum(naive_bayes_pred$class == validation$Y) / length(validation_index)
}
```

On peut visualiser les résultat sous forme graphique pour choisir la combinaison d'hyper-paramètres donnant le meilleur résultat.

```
ggplot(
 data = results,
 mapping = aes(x = adjust, y = error, color = usekernel)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "Largeur de bande",
  y = "Erreur de validation",
  title = "Comparaison des hyper-paramètres",
  color = "Utilisation d'un noyau ?"
 ) +
 theme(
  legend.position = c(.8, .22)
 )
```

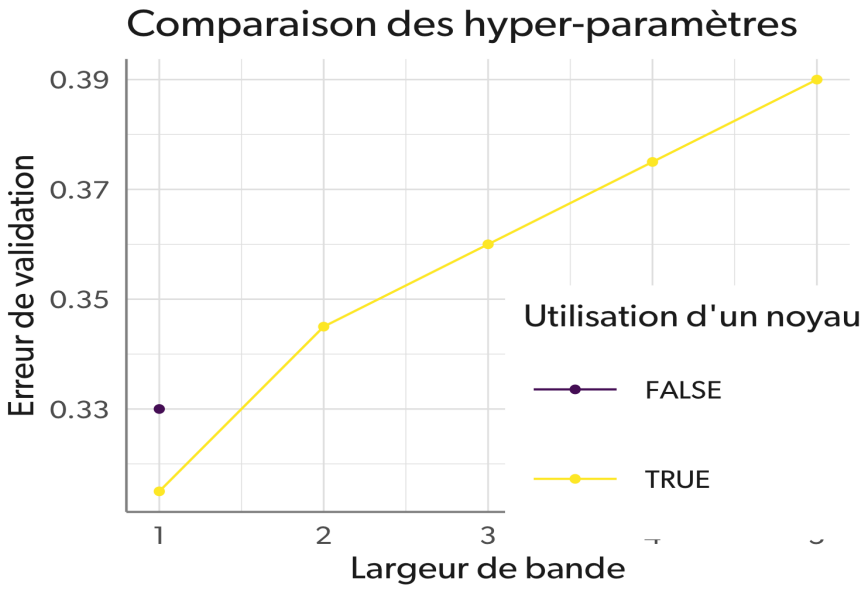


Figure 5 : Visualisation des erreurs de validation selon les hyper-paramètres choisis : largeur de bande et utilisation de noyau.

Le meilleur modèle est obtenu avec l'utilisation d'un noyau pour estimer la densité et une largeur de bande de 1 (plus précisément un facteur multiplicatif de la largeur de bande de 1).

On va donc utiliser ces hyper-paramètres pour effectuer une prédiction sur les données de test et ainsi mesurer l'erreur de généralisation du modèle bayésien naïf. Pour ce faire on va entraîner le modèle avec les hyper-paramètres sélectionnés sur l'ensemble des données train + validation puis effectuer une prédiction sur les données de test.

```
naive_bayes_model <- NaiveBayes(
 formula = Y ~ X,
 data = data,
 usekernel = TRUE,
 fL = 0,
 adjust = 1
)
naive_bayes_pred <- predict(
 object = naive_bayes_model,
 newdata = test
)
```

Comme précédemment on calcule l'erreur de généralisation. On obtient 0.32.

```
sum(naive_bayes_pred$class == test$Y) / n_test
1 - sum(naive_bayes_pred$class == test$Y) / n_test
```

Analyse de l'erreur

On remarque que l'erreur de validation et l'erreur de test sont légèrement différentes. Les données sur lesquelles l'estimation de l'erreur a été faite sont différentes : les données de validation, sur lesquelles on a choisi les meilleurs hyper-paramètres, et les données de test.

Exercice 2 - Discrimination à plusieurs classes

On utilisera les K plus proches voisins et Bayésien naïf pour discriminer les 4 types de véhicules (variable à expliquer à 4 modalités) à l'aide de différentes variables décrivant la silhouette d'un véhicule.

Question 1 - Import des données

> *Charger la table Vehicle après avoir chargé le package* **mlbench**. *Lire la description de la table. Combien de prédicteurs potentiels dans cette table ?*

On commence par charger la table **Vehicule** avec la commande **data(Vehicle)** et par avoir un aperçu des données contenues dedans. On peut regarder directement les données en affichant le data.frame, visualiser un résumé avec **summary()** ou encore utiliser **skimr::skim()**.

La variable cible est **class**, elle comporte 4 modalités. Les 18 autres variables sont toutes quantitatives et peuvent toutes être utilisées comme prédicteurs.

```
predicteurs <- names(Vehicle)[-19]
```

Points d'attention

Le modèle des k-plus-proches-voisins est sensible au nombre de prédicteurs or ici il y en 18. Une pré-sélection de variables pourrait être la bienvenue.

Toutes les variables explicatives ne sont pas sur la même échelle de valeurs. Il faudra harmoniser les échelles pour que le calcul de distance ne soit pas impacté.

Question 2 - Fonction de coût

> *Quelle fonction de coût utiliser ?*

Dans le cas d'une classification, la fonction de coût est l'erreur du modèle. Pour n observations de test on a donc :

$$\mathcal{L}(y, \hat{y}) = \mathbb{1}_{y \neq \hat{y}}$$

Pour calculer cette erreur on va séparer les données en un ensemble d'apprentissage et un ensemble de test. On définit une graine pour pouvoir reproduire les résultats et on choisit un ratio 80/20.

```
set.seed(123456)
train_index <- sample(1:nrow(Vehicle), size = nrow(Vehicle) * .8)
test_index <- setdiff(1:nrow(Vehicle), train_index)
train <- Vehicle[train_index, ]
test <- Vehicle[test_index, ]
```

Question 3 - Knn

> *Quelle est la règle de décision associée à l'algorithme des k plus proches voisins ? Mettre en oeuvre les k plus proches voisins. Justifir le choix des paramètres et commenter les résultats.*

Comme dans le cas précédent on va utiliser le vote à la majorité pour choisir la classe. Le choix de la meilleure valeur de k se fait par validation croisée avec la fonction `e1071::tune.knn()`. On a vu que la valeur retournée pouvait varier. On va ici lancer plusieurs fois cette fonction pour ensuite choisir la valeur de k la plus fréquemment retournée.

Les données doivent être mise sur la même échelle pour éviter d'introduire artificiellement de l'importance sur certaines variables. Ce traitement doit être effectué sur les données de train et sur les données de test. Pour les données de test il faut appliquer les mêmes paramètres que pour les données de train.

```
train_scaled <- scale(train[, predicteurs])
test_scaled <- scale(
 x = test[, predicteurs],
 center = attributes(train_scaled)[["scaled:center"]],
 scale = attributes(train_scaled)[["scaled:scale"]]
)
```

On entraîne le modèle sur ces données.

```
best_k <- numeric(length = 20)
for (i in 1:20) {
 knn_cross_results <- tune.knn(
  x = train_scaled,
  y = train$Class,
  k = 1:50,
  tunecontrol = tune.control(sampling = "cross"),
  cross = 5
 )
 best_k[i] <- knn_cross_results$best.parameters[[1]]
 # print(sprintf("Itération %s : meilleur K = %s", i, knn_cross_results$best.parameters))
}
```

Valeur de K	3	4	5	6	7	8	12
Nombre d'occurrences	5	2	1	4	5	2	1

Table 1 : Tableau donnant valeurs de K sélectionnées et leur fréquence parmi les 20 essais.

On utilise la valeur la plus fréquente pour construire notre modèle de plus-proches-voisins et ainsi mesurer l'erreur sur l'échantillon de test. Le modèle est entraîné sur l'ensemble des données de la table `train`.

```
set.seed(123456)
knn_pred <- knn(
 train = train_scaled, # données d'apprentissage
 test = test_scaled, # données à prédire
 cl = train$Class, # vraies valeurs
 k = 3 # nombre de voisins
)
```

L'erreur obtenue est 0.294.

Question 4 - Naive Bayes

Quelle est la règle de décision associée au bayésien naïf ? Mettre en oeuvre le Bayésien naïf. Justifier le choix des paramètres et commenter les résultats.

La règle de décision

Comme dans le cas binaire pour l'exercice 1 c'est le maximum a posteriori qui est utilisé pour choisir la classe prédite.

Le choix des hyper-paramètres

On va ici utiliser la libraire `caret` qui comporte un ensemble de fonction facilitant la création de pipelines d'entraînement en apprentissage supervisé. Ce livre en ligne et cette vignette d'introduction donnent une bonne vision du fonctionnement de ce package. On va faire la même chose que dans l'exercice précédent, à savoir évaluer l'erreur de validation pour différentes combinaisons d'hyper-paramètres mais de manière simplifiée.

On définit l'ensemble des combinaisons à tester.

```
hyperparam_grid <- expand.grid(
 usekernel = TRUE, # si vrai utilisation d'un noyau sinon gaussien
 fL = 0, # correction avec lissage de Laplace (ici ce paramètre n'est pas nécessaire, x étant continue)
 adjust = seq(1, 5, by = 1) # largeur de bande
)
hyperparam_grid <- rbind(hyperparam_grid, data.frame(usekernel = FALSE, fL = 0, adjust = 1))
```

On va définir la façon de séparer les données en apprentissage et validation avec la fonction caret::trainControl(). On spécifie l'argument method à "cv" pour faire de la validation croisée et l'argument number à 5 pour utiliser 5 blocs.

```
control <- trainControl(method = "cv", number = 5)
```

On peut ensuite entraîner le modèle avec la fonction caret::train(). Un modèle par combinaison sera entraîné et l'erreur sera calculée sur l'échantillon de validation. La fonction caret::train() prend 5 arguments :

- x : les prédicteurs
- y : le vecteur de la variable cible
- method : le nom du modèle à entraîner, ici "nb" pour Naive Bayes
- trControl : la méthode de validation
- tunegrid : la grille d'hyper-paramètres à tester

Cette fonction va choisir les meilleurs hyper-paramètres et retourner le meilleur modèle directement.

```
naive_bayes <- train(
 x = train[, predicteurs], # prédicteurs
 y = train$Class, # réponse
 method = "nb", # classifieur utilisé, ici Naive Bayes
 trControl = control, # méthode d'échantillonnage, ici 5-fold CV
 tuneGrid = hyperparam_grid # combinaisons d'hyper-paramètres à évaluer
)
```

On peut visualiser les résultats sous forme graphique. On trace ici la précision.

```
ggplot(
 data = naive_bayes$results,
 mapping = aes(x = adjust, y = Accuracy, color = usekernel)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "Largeur de bande",
  y = "Précision",
  title = "Comparaison des hyper-paramètres",
  color = "Utilisation d'un noyau ?"
 ) +
 theme(
  legend.position = c(.8, .8)
 )
```

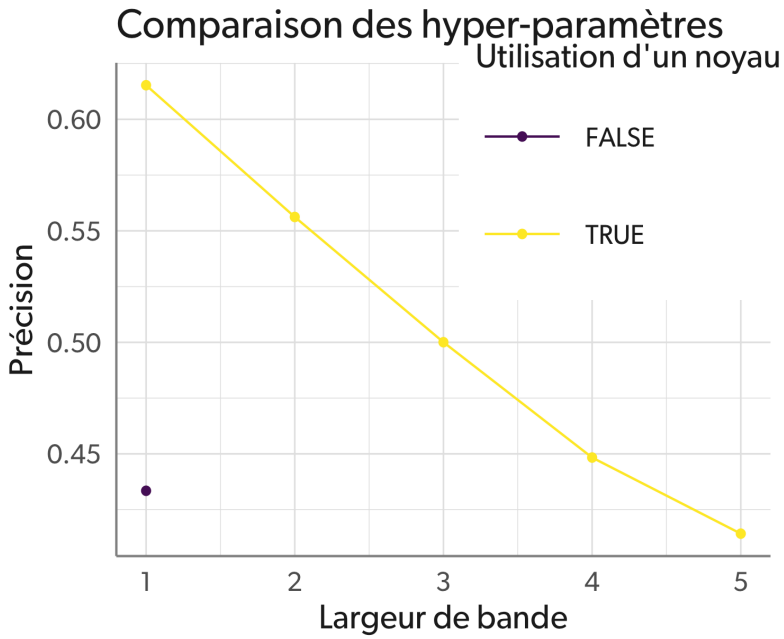


Figure 6 : Visualisation des erreurs de validation selon les hyper-paramètres choisis : largeur de bande et utilisation de noyau.

Le meilleur modèle est obtenu avec l'utilisation d'une estimation par noyau et une largeur de bande de 1.

La fonction `caret::predict()` va utiliser directement le résultat de la fonction `caret::train()` et donc les meilleurs hyper-paramètres pour effectuer la prédiction.

```
naive_bayes_pred <- predict(
  object = naive_bayes,
  newdata = test
)
```

L'erreur de généralisation est de 0.424.

Pourquoi des *warnings* ?

On peut remarquer que la fonction `predict()` génère des messages de *warning* de la forme : `Numerical 0 probability for all classes with observation`. Cela veut dire que les 4 probabilités initialement calculées par le modèle sont toutes très faibles. Il s'agit des $P(Y = y) \times P(X = x|Y = y)$. Leur comparaison n'est donc peut-être pas pertinente. La fonction renvoie ces probabilités normalisées donc ces faibles valeurs n'apparaissent pas lorsqu'on regarde les résultats du modèle (utiliser la commande `predict(naive_bayes, test, type = "prob")` pour avoir les probabilités et non pas les classes directement).

Exercice 3 - Problème de régression

Context: Since 2008, guests and hosts have used Airbnb to expand on traveling possibilities and present more unique, personalized way of experiencing the world. This dataset describes the listing activity and metrics in NYC, NY for 2019.

Content: This data file includes all needed information to nd out more about hosts, geographical availability, necessary metrics to make predictions and draw conclusions.

Acknowledgements This public dataset is part of Airbnb, and the original source can be found on this website.

Source: kaggle and Airbnb.

- id : listing ID
- name : name of the listing
- host id : host ID
- host name : name of the host
- neighbourhood group : location
- neighbourhood : area
- latitude : latitude coordinates
- longitude : longitude coordinates
- room type listing : space type
- price : price in dollars
- minimum nights : amount of nights minimum
- number of reviews : number of reviews
- last review : latest review
- reviews per month : number of reviews per month
- calculated host listings count : amount of listing per host
- availability 365 : number of days when listing is available for booking

La variable à expliquer est *price*.

Question 1 - Choix des variables

> *Quelles variables explicatives proposez-vous d'utiliser si vous êtes un particulier et que vous souhaitez prédire à quel prix vous pourriez proposer votre logement en Airbnb en vous positionnant "au prix du marché".*

On commence par importer les données.

```
airbnb <- read.csv(file = "data/AB_NYC_2019.csv")
```

Ici les seuls régresseurs utilisables pour un particulier qui n'est pas encore inscrit sur Airbnb sont les variables de localisation (peut-être une sélection parmi les 4), et le type de logement (room_type). La variable à prédire est, cette fois, une variable quantitative (price).

Question 2 - Fonction de coût

Quelle fonction de coût utiliser ?

On est ici dans le cas d'une régression (variable cible quantitative), c'est donc la perte quadratique qui sera utilisée :

$$\mathcal{L}(Y, \hat{Y}) = \| Y - \hat{Y} \|^2$$

Il s'agit simplement de la racine carrée de la moyenne des erreurs au carré. Pour estimer cette erreur on va diviser les données en un ensemble d'apprentissage et un ensemble de test.

```
set.seed(123456)
train_index <- sample(1:nrow(airbnb), size = nrow(airbnb) * .7)
test_index <- setdiff(1:nrow(airbnb), train_index)
train <- airbnb[train_index, ]
test <- airbnb[test_index, ]
```

Question 3 - Knn

Quelle est la règle de décision associée à l'algorithme des k plus proches voisins ? Mettre en œuvre les k plus proches voisins. Justifier le choix des paramètres et commenter les résultats.

Parmi les variables explicatives à notre disposition on a :

- `neighbourhood_group` : catégorielle à 5 modalités
- `neighbourhood` : catégorielle à 221 modalités
- `latitude` : numérique
- `longitude` : numérique
- `room_type` : catégorielle à 3 modalités

Il y a donc à la fois des variables continues et des variables catégorielles. On ne peut pas intégrer directement ces données dans l'algorithme KNN. En effet la distance utilisée par défaut est la distance euclidienne, or celle-ci s'applique uniquement aux données continues.

Plusieurs solutions sont possibles :

- Effectuer une ACM pour transformer les variables catégorielles en variables continues
- Utiliser la distance de Hamming qui est applicable pour tout type de données (ou tout autre distance avec les mêmes propriétés)
- Encoder les variables catégorielles en variables numériques
- Utiliser uniquement les variables continues

Variables continues uniquement

On va construire le modèle de k plus-proches-voisins en se basant uniquement sur les variables latitude et longitude. Comme précédemment on va tester différentes valeurs de k pour choisir la meilleure valeur. Le choix de la plage de valeurs à tester peut se faire de manière itérative. On commence par un ensemble de valeurs larges puis on peut réduire petit à petit.

```
knn_fit <- train(
 x = train[, c("latitude", "longitude")], # prédicteurs
 y = train[, "price"], # réponse
 tuneGrid = data.frame(k = seq(50, 500, by = 10)), # nombre de voisins
 method = "knn", # knn classifieur
 trControl = trainControl(method = "cv", number = 5) # 5-fold CV
)
```

On peut voir les résultats sur le graphique suivant.

```
ggplot(
 data = knn_fit$results,
 mapping = aes(x = k, y = RMSE)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "Valeurs de k",
  y = "Erreur quadratique",
  title = "Evaluation de l'erreur sur données de validation"
 )
```

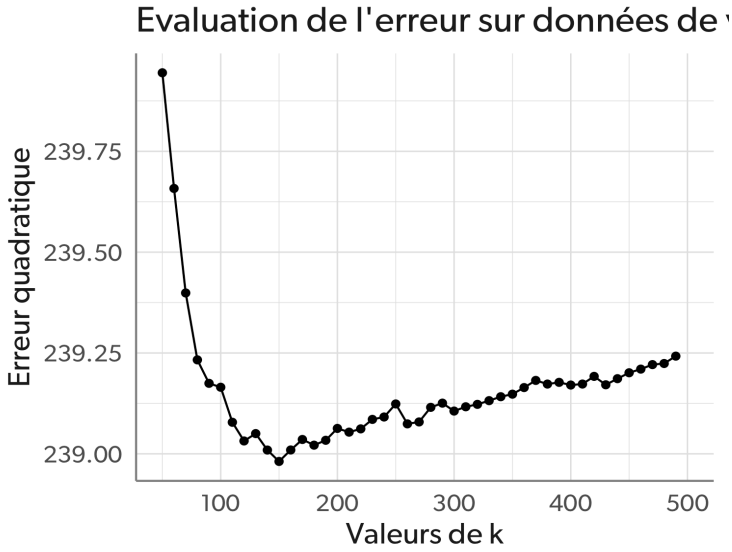


Figure 7 : Résultats de la validation croisée pour le choix de k. Pour chaque valeur de k entre 50 et 500 par pallier de 10, l'erreur de validation est représentée. La valeur de k correspondant à l'erreur la plus basse est choisie.

On utilise ensuite les données de test pour calculer l'erreur de généralisation.

```
knn_pred <- predict(
 object = knn_fit,
 newdata = test
)
knn_num_rmse_error <- sqrt(mean((knn_pred - test$price)^2))
knn_num_mape_error <- mean(abs((test$price[test$price != 0] - knn_pred[test$price != 0]) / test$p
rice[test$price != 0])) * 100
```

L'erreur RMSE est de 212. L'écart absolu moyen entre la valeur prédite et la valeur réelle est de 65%.

Variables catégorielles encodées

Les 3 variables catégorielles vont être encodées en variables continues. Les modalités sont transformées en nombre. De cette manière on pourra utiliser directement le modèle knn. Il faut faire attention à la manière avec laquelle on effectue cette transformation. En effet l'ensemble de test ne contient pas nécessairement les mêmes modalités que l'ensemble de train (cela sera surtout vrai pour les variables à forte cardinalité).

```
# Sur les données de train
train$room_type <- as.numeric(
 factor(
  x = train$room_type,
  levels = unique(airbnb$room_type)
 )
)
train$neighbourhood_group <- as.numeric(
 factor(
  x = train$neighbourhood_group,
  levels = unique(airbnb$neighbourhood_group)
 )
)
train$neighbourhood <- as.numeric(
 factor(
  x = train$neighbourhood,
  levels = unique(airbnb$neighbourhood)
 )
)

# Sur les données de test
test$room_type <- as.numeric(
 factor(
  x = test$room_type,
  levels = unique(airbnb$room_type)
 )
)
test$neighbourhood_group <- as.numeric(
```

```
 factor(
  x = test$neighbourhood_group,
  levels = unique(airbnb$neighbourhood_group)
 )
)
test$neighbourhood <- as.numeric(
 factor(
  x = test$neighbourhood,
  levels = unique(airbnb$neighbourhood)
 )
)
```

Chaque variable est transformée en factor en utilisant tous les niveaux présents dans la table initiale.

On peut ensuite, comme précédemment, construire le modèle et évaluer sa performance sur les données test. On pense bien sûr à mettre nos variables sur la même échelle avec la fonction `scale()`. On n'oublie pas de sauvegarder les paramètres de cette transformation pour appliquer les mêmes au jeu de données de test.

On va centrer et réduire les variables pour qu'elles soient sur la même échelle directement avec `caret`.

```
knn_fit <- train(
 x = train[, c("latitude", "longitude", "room_type", "neighbourhood_group", "neighbourhood")], # prédicteurs
 y = train$price, # réponse
 preProc = c("scale", "center"), # centrer et réduire
 tuneGrid = data.frame(k = seq(30, 300, by = 10)), # nombre de voisins
 method = "knn", # knn classifieur
 trControl = trainControl(method = "cv", number = 5) # 5-fold CV
)
```

On peut visualiser les résultats pour ensuite affiner la grille de recherche.

```
ggplot(
 data = knn_fit$results,
 mapping = aes(x = k, y = RMSE)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "Valeurs de k",
  y = "Erreur quadratique",
  title = "Evaluation de l'erreur sur données de validation"
 )
```

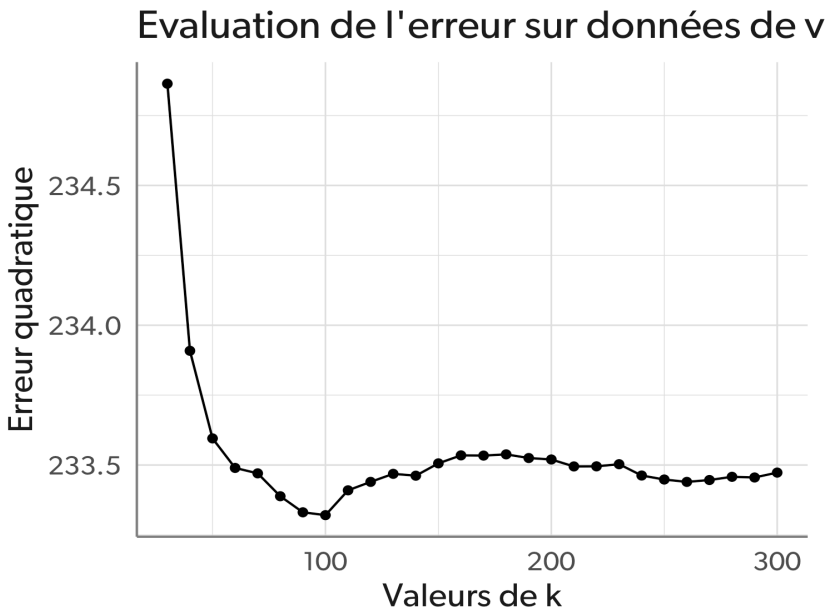


Figure 8 : Résultats de la validation croisée pour le choix de k. Pour chaque valeur de k entre 50 et 500 par pallier de 10, l'erreur de validation est représentée. La valeur de k correspondant à l'erreur la plus basse est choisie.

On effectuer les prédictions sur le jeu de données de test pour calculer l'erreur. Les paramètres pour centrer et réduire sont ceux calculés sur le train.

```
knn_pred <- predict(
  object = knn_fit,
  newdata = test
)
knn_factor_rmse_error <- sqrt(mean((knn_pred - test$price)^2))
knn_factor_mape_error <- mean(abs((test$price[test$price != 0] - knn_pred[test$price != 0]) / test$
price[test$price != 0])) * 100
```

L'erreur RMSE est de 233. L'écart absolu moyen entre la valeur prédite et la valeur réelle est de 93%.

ACM

Pour intégrer toutes les variables dans le modèle de knn on peut aussi transformer les variables catégorielles en variables numériques par le biais d'un ACM.

```
set.seed(123456)
train_index <- sample(1:nrow(airbnb), size = nrow(airbnb) * .7)
test_index <- setdiff(1:nrow(airbnb), train_index)
train <- airbnb[train_index, ]
test <- airbnb[test_index, ]
```

On effectue une ACM sur les données d'apprentissage en conservant les 5 premières dimensions dans le résultat (choix par défaut). On transforme ainsi les variables *neighbourhood_group* et *room_type* en variables continues. (La variable *neighbourhood* n'est pas utilisée car il faudrait effectuer un traitement sur les modalités à faible fréquence pour éviter des problèmes lors de la projection des individus de test sur les axes).

```
acm <- MCA(
 X = train[, c("neighbourhood_group","room_type")],
 ncp = 5,
 graph = FALSE
)
```

A partir des résultats on a va construire un nouveau jeu de données, correspondant aux axes et aux données de localisation. Ces données sont centrées et réduites.

```
train_scaled <- scale(train[, c("latitude", "longitude")])
test_scaled <- scale(
 x = test[, c("latitude", "longitude")],
 center = attributes(train_scaled)[["scaled:center"]],
 scale = attributes(train_scaled)[["scaled:scale"]]
)

# Ajouter les coordonnées aux données continues
train_acm <- cbind(train_scaled, acm$ind$coord)
```

On entraîne ensuite le modèle de knn pour choisir le meilleur k.

```
# Entraîner le modèle de knn
knn_fit <- train(
 x = train_acm, # prédicteurs
 y = train$price, # réponse
 tuneGrid = data.frame(k = seq(40, 300, by = 20)), # nombre de voisins
 method = "knn", # knn classifieur
 trControl = trainControl(method = "cv", number = 5) # 5-fold CV
)
```

Et on peut voir les résultats comme précédemment.

```
ggplot(
 data = knn_fit$results,
 mapping = aes(x = k, y = RMSE)
) +
 geom_line() +
 geom_point() +
 labs(
  x = "Valeurs de k",
  y = "Erreur quadratique",
  title = "Evaluation de l'erreur sur données de validation"
 )
```

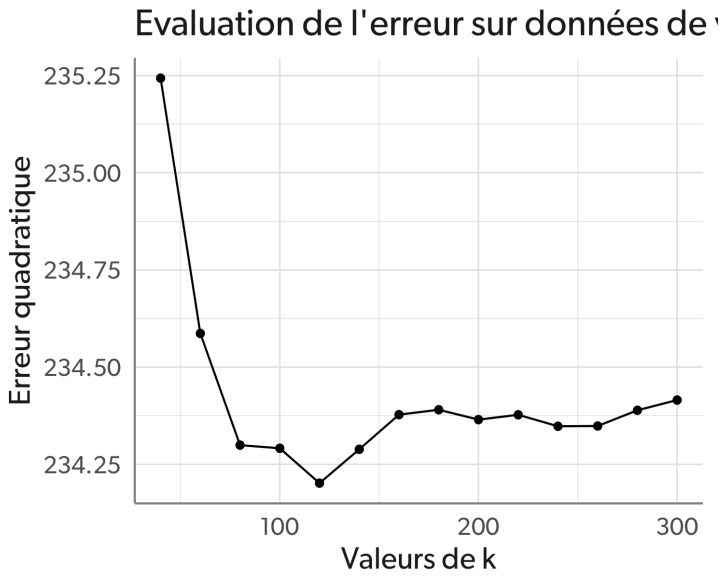


Figure 9: Résultats de la validation croisée pour le choix de k. Pour chaque valeur de k entre 50 et 500 par pallier de 10, l'erreur de validation est représentée. La valeur de k correspondant à l'erreur la plus basse est choisie.

Pour effectuer les prédictions on doit d'abord projeter les individus de test sur les axes de l'ACM. On peut ensuite effectuer une prédiction pour évaluer l'erreur.

```
# Préparer les données test pour faire une prédiction
acm_pred <- predict.MCA(
  object = acm,
  newdata = test[, c("neighbourhood_group","room_type")]
)

# Effectuer une prédiction sur les données test
knn_pred <- predict(
  object = knn_fit,
  newdata = cbind(test_scaled, acm_pred$coord)
)

# Calcul de l'erreur
knn_acm_rmse_error <- sqrt(mean((knn_pred - test$price)^2))
knn_acm_mape_error <- mean(abs((test$price[test$price != 0] - knn_pred[test$price != 0]) / test$p
rice[test$price != 0])) * 100
```

L'erreur RMSE est de 207. L'écart absolu moyen entre la valeur prédite et la valeur réelle est de 47%.

Comparaison des erreurs

On a testé 3 approches pour répondre à la problématique de prédiction. On peut voir les différentes erreurs obtenues dans le tableau suivant.

```r
data.frame(
  "Approche" = c("Données de localisation uniquement", "Transformation des variables catégoriell
es en variables numériques", "Utilisation d'une ACM"),
  "Erreur RMSE" = round(c(knn_num_rmse_error, knn_factor_rmse_error, knn_acm_rmse_error), di
gits = 0),
  "Erreur MAPE" = paste0(round(c(knn_num_mape_error, knn_factor_mape_error, knn_acm_mape
_error), digits = 0), '%'),
  check.names = FALSE
 ) %>%
kable(align = c('c', 'r'), format='latex') %>%
 kable_styling(full_width = FALSE)
```

Approche	Erreur RMSE	Erreur MAPE
Données de localisation uniquement	212	65%
Transformation des variables catégorielles en variables numériques	233	93%
Utilisation d'une ACM	207	47%

Table 2 : Tableau comparatif des approches de traitement des variables catégorielles. Les performances des modèles peuvent être comparées.