Apprentissage supervisé - TP1

Les différentes librairies qui seront utilisées pour ce TP sont listées ici. Le code R correspondant est en ligne sous Moodle.

```
library("caret") # ensemble de meta-fonctions nécessaires au processus d'ap
prentissage supervisé
library("class") # fonction knn
library("e1071") # librairie nécessaire pour la fonction tune.knn
library("FactoMineR") # ACM ou AFDM
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 :

Cas 1 :
On a simulé un échantillon de taille n=1000 selon le modèle :
Pour i=1 à 1000, $X_i \sim \mathcal{N}(0,1)$ $U_i \sim \mathcal{U}[0,1]$ et $Y_i = \begin{cases} \mathbb{1}_{U_i \leq 0.1}, & si \quad X_i \leq 0 \\ \mathbb{1}_{U_i > 0.2}, & si \quad X_i > 0 \end{cases}$

Cas 2 :
On a simulé un échantillon de taille n=1000 selon le modèle :
Pour i=1 à 1000, $X_i \sim \mathcal{N}(0,1)$ $U_i \sim \mathcal{U}[0,1]$ et $Y_i = \begin{cases} \mathbb{1}_{U_i \leq 0.2}, & si \quad X_i \leq 0 \\ \mathbb{1}_{U_i > 0.4}, & si \quad X_i > 0 \end{cases}$

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 :

$$f^*(x) : \quad \mathbb{R} \to \{0;1\}$$
$$x \mapsto f^*(x) = \begin{cases} 1 & si \quad P(Y=1|X=x) > 0.5 \\ 0 & sinon \end{cases}$$

Cas 1 :

- Pour $X_i \leq 0$:
  $P(Y_i = 1|X_i = x_i) = P(U_i \leq 0.1) = 0.1$ d'où $f^*(x_i) = 0$
- Pour $X_i > 0$:
  $P(Y_i = 1|X_i = x_i) = P(U_i > 0.2) = 0.8$ d'où $f^*(x_i) = 1$

On a donc la **règle de Bayes** : $f^*(x) = \begin{cases} 1 & si \quad x > 0 \\ 0 & sinon \end{cases}$

Risque de Bayes :

$$\begin{aligned} R_p^* &= P(Y \neq f^*(X)) \\ &= P(Y = 1 \cap f^*(X) = 0) + P(Y = 0 \cap f^*(X) = 1) \\ &= P(Y = 1 \cap X \leq 0) + P(Y = 0 \cap X > 0) \\ &= P(Y = 1|X \leq 0)P(X \leq 0) + P(Y = 0|X > 0)P(X > 0) \\ &= 0.1 \times 0.5 + 0.2 \times 0.5 = 0.15 \end{aligned}$$

Cas 2 : **Règle de Bayes** : $f^*(x) = \begin{cases} 1 & si \quad x > 0 \\ 0 & sinon \end{cases}$

Risque de Bayes :

$$\begin{aligned} R_p^* &= P(Y \neq f^*(X)) \\ &= 0.2 \times 0.5 + 0.4 \times 0.5 = 0.3 \end{aligned}$$

Cas 3

La loi jointe $\boldsymbol{P}$ de (X,Y) n'étant 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 est un indicateur de la complexité du problème. Plus le risque de Bayes est important, plus on considère le problème comme complexe. Ainsi le cas 2 est plus complexe que le premier.

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.

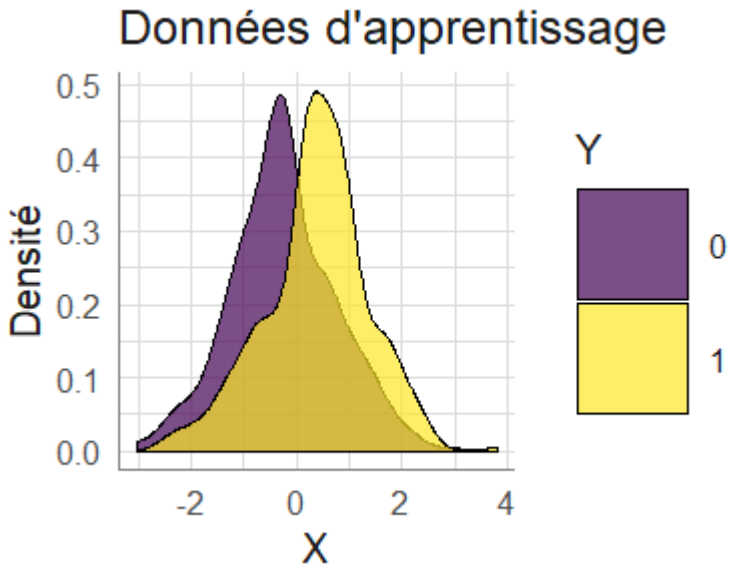
```
# Nombre d'éléments dans l'échantillon
n_train <- 1000

# 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 tu
ne.knn dans la suite
Y <- as.factor(Y1)

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

Sur le graphe suivant on peut voir la densité de X selon la valeur prise par Y

.

```
ggplot(
  data = data,
  mapping = aes(x = X, fill = Y)
) +
  geom_density(
    alpha = .7
  ) +
  labs(
    x = "X",
```

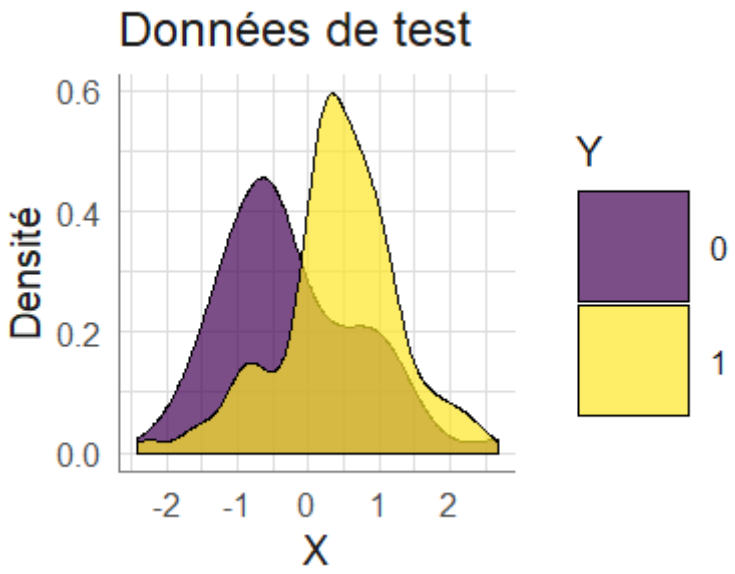
```
    y = "Densité",
    title = "Données d'apprentissage",
    fill = "Y"
  )
```



De la même manière on peut simuler des données de test qui seront utilisées pour estimer le risque moyen i.e. l'erreur de généralisation de nos modèles de prédictions issus de différentes familles de modèles.

Rappel : l'échantillon d'apprentissage sert à apprendre une règle de prédiction ***f*** qu'on va choisir dans une famille ***F*** de modèles (par exemple la famille des Knn), l'échantillon de validation (validation simple ou croisée) sert à sélectionner la meilleure règle de prédiction dans ***F*** .

```
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"
  )
```

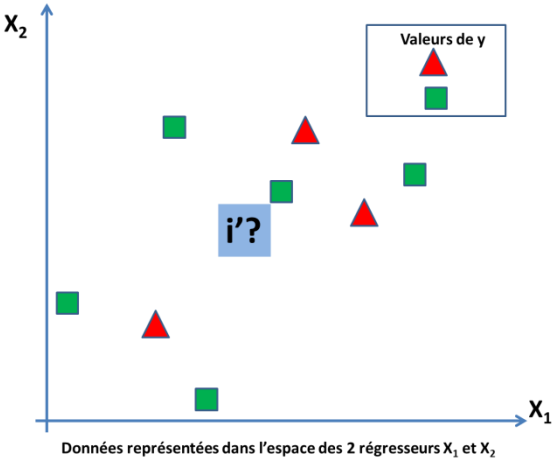


Question 4 - Knn

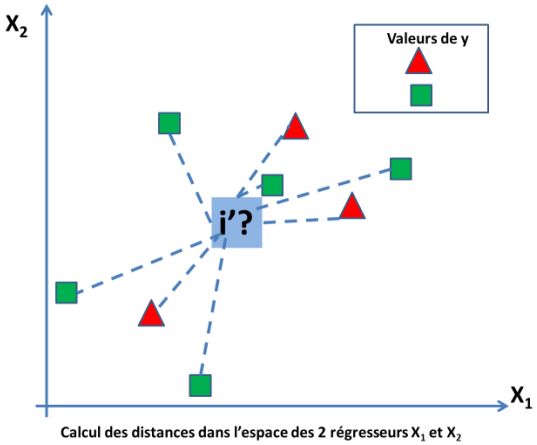
Quelle est la règle de décision associée à l'algorithme des k plus proches voisins ? Mettre en oeuvre les k plus proches voisins. Justifier le choix des paramètres et commenter les résultats en validation et sur l'échantillon test.

Le modèle KNN

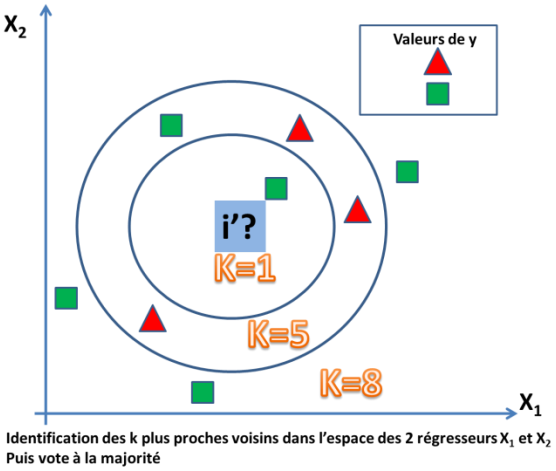
Le modèle des K plus-proches-voisins (*K-nearest-neighbours* en anglais) est un modèle d'apprentissage supervisé utilisable en classification et en régression.
En classification (variable à prédire Y qualitative), pour prédire la classe (ou modalité) d'une nouvelle observation i′, on calcule la distance qui sépare cette observation de toutes les observations des données d'apprentissage. Cela nécessite donc le choix préalable d'une distance.
On sélectionne les k points les plus proches, puis on détermine la classe à prédire en faisant un vote à la majorité.



Données représentées dans l'espace des 2 régresseurs X_1 et X_2

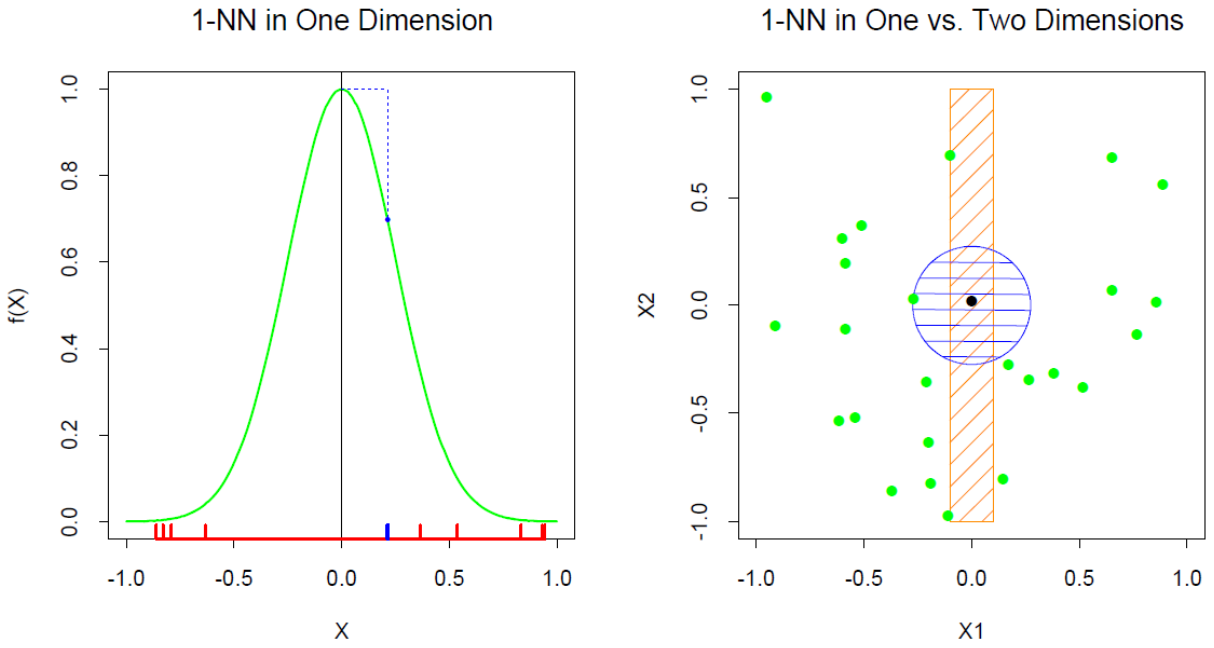


Calcul des distances dans l'espace des 2 régresseurs X_1 et X_2



Quelques points d'attention :

- Les Knn sont de complexité croissante en fonction de 1/k. Le choix de k est donc un élément central de ce modèle. On choisit sa valeur grâce aux techniques de validation simples ou croisées.
- On peut choisir la distance euclidienne si les régresseurs sont quantitatifs, sinon voir les autres distances vues en cours.
- Enfin attention au fléau de la dimension : plus il y a de prédicteurs plus les plus « proches » voisins seront potentiellement éloignés.



Elements of Statistical Learning (2nd Ed.) Hastie, Tibshirani et Friedman, 2009

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. Pour k=5 par exemple, si 3 voisins ont Y=0 et 2 voisins ont Y=1 alors la prédiction sera $\hat{Y} = 0$

Un premier modèle à titre d'exemple

Pour s'initier à cette méthode, on commence par lancer un modèle de k plus-proches-voisins avec un choix arbitraire pour k, 10 par exemple. Pour des questions de reproductibilité on fixe là aussi une graine. 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.

```
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.

Le choix de k

La valeur de k a été définie de façon arbitraire dans ce premier modèle, mais on peut optimiser ce choix en essayant différentes valeurs et en choisissant la meilleure au regard du coût estimé par des techniques de validation. Si on utilise la validation simple, on découpe l'échantillon en une partie apprentissage et une partie validation. La règle de décision est construite sur la partie apprentissage et les prédictions sont faites sur la partie validation. On peut ainsi estimer l'erreur de généralisation sur des données qui n'ont pas servi à construire la règle de décision : les données de validation. Pour cela on utilise la fonction **`e1071::tune.knn()`** qui prend 4 paramètres en entrée :

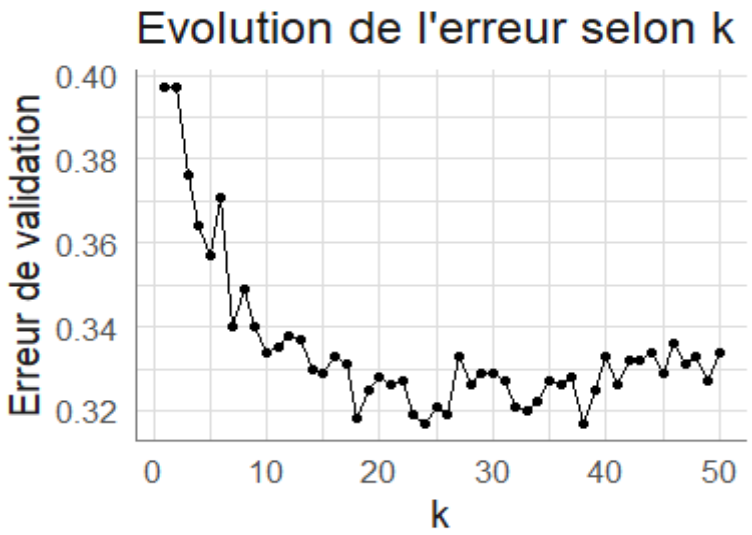
- **`x`** : les prédicteurs
- **`y`** : le vecteur cible
- **`k`** : un vecteur de valeurs de kk à tester
- **`tunecontrol`** : la méthode de validation, sous la forme d'une fonction **`tune.control(sampling = "cross")`** pour de la validation croisée (dans ce cas il faut également spécifier l'argument **`cross = 10`** pour faire de la 10-fold cv) ou **`tune.control(sampling = "boot")`** pour faire du bootstrap.

```
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 = 10 # 10 blocs
)

summary(knn_cross_results)
```

```
Parameter tuning of ‘knn.wrapper’:

- sampling method: 10-fold cross validation

- best parameters:
  k
 24

- best performance: 0.317

- Detailed performance results:
    k error dispersion
1   1 0.397 0.02540779
2   2 0.397 0.03802046
3   3 0.376 0.05541761
4   4 0.364 0.05947922
5   5 0.357 0.05907622
…
22 22 0.327 0.04522782
23 23 0.319 0.04040077
24 24 0.317 0.04595892
25 25 0.321 0.04228212
26 26 0.319 0.04677369
…
49 49 0.327 0.04398232
50 50 0.334 0.05378971
```

On peut visualiser les taux erreurs obtenus en validation croisée avec ce programme

```
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"
  )

# On pourrait aussi utiliser simplement plot(knn_cross_results)
```



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

Remarque : la valeur optimale de k *peut varier à chaque fois que la commande est lancée (sélection aléatoire des différents échantillons apprentissage/validation). On verra dans l'exercice suivant comment on peut contourner ce problème.*

On peut relancer la prédiction avec le k sélectionné et calculer le taux d'erreur 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 obtenue est très proche de l'erreur de Bayes calculée à la question 1. Pour k=30, avec les mêmes graines, l'erreur estimée est inférieure au risque de Bayes. Cela est dû au fait que

l'erreur de Bayes est une espérance or on a ici un faible nombre d'observations simulées (200). On peut simuler un nombre supérieur de données, 2000 par exemple, et recalculer l'erreur.

```
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.

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

Ce modèle se base sur la formule de Bayes :

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

Pour estimer $P(Y = y|X = x)$ (le but de tout modèle d'apprentissage supervisé) il faut donc estimer $P(Y = y)$, $P(X = x|Y = y)$ et $P(X = x)$. Précisément on cherche quelle classe prédire, c'est à dire le y qui correspondra à la plus grande probabilité.

On cherche donc :

$$\arg\max_{y \in \mathcal{Y}} \widehat{P}(Y = y|X = x)$$

C'est la méthode du Maximum a posteriori (MAP).

Comme $P(X = x)$ ne dépend pas de y on cherche donc :

$$\arg\max_{y \in \mathcal{Y}} \widehat{P}(Y = y) \times \widehat{P}(X = x|Y = y)$$

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

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

On recherche la valeur y :

$$\underset{y \in \mathcal{Y}}{\arg\max}\ \widehat{P}(Y = y) \times \prod_{j=1}^{J} \widehat{P}(X_j = x_j | Y = y)$$

La règle de décision

C'est la règle du maximum a posteriori (MAP) : on retient la modalité y de Y dont la probabilité a posteriori estimée $\hat{P}(Y = y\,|X = x)$ est la plus forte. C'est à dire celle qui est supérieure à 0,5 dans le cas où Y est binaire.

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 estimer l'erreur sur des données non utilisées pour apprendre la fonction de prédiction.

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

Comme précédemment on peut estimer 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 :

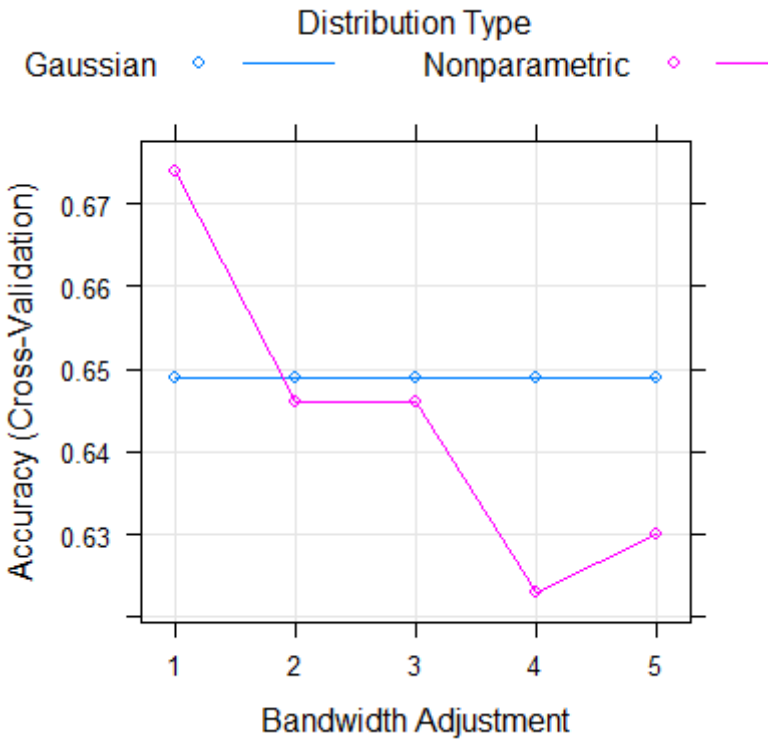
- l'utilisation ou non d'un noyau pour estimer les densités des régresseurs quantitatifs, **`usekernel`**
- la largeur de bande utilisée par le noyau, **`adjust`**
- la correction de Laplace à appliquer pour les régresseurs qualitatifs, **`fL`**.

Pour choisir la meilleure combinaison d'hyper-paramètres pour le modèle bayésien naïf, on va utiliser la validation croisée 10 blocs. Le but est d'estimer l'erreur de généralisation par validation

croisée pour chaque combinaison d'hyper-paramètres et de sélectionner la combinaison avec l'erreur estimée 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 régresseurs quantitatifs.

```
grid<- expand.grid (
 usekernel=c(TRUE,FALSE), # si TRUE utilisation d'un noyau sinon gaussien
 fL=0, # lissage de Laplace
adjust=seq(1,5,by=1)) # Largeur de bande du noyau

control<-trainControl(method="cv",number=10) # validation croisée 10 blocs

names(donnees)<-c("X","Y")
x <- as.data.frame(donnees[,1])
names(x)=c("X") #contrainte de caret, il faut que x ait un nom ici "X"
y <-donnees[,2] # train model
NB1<-train(x=x,y=y, method="nb",trControl=control,tuneGrid=grid) #plot sear
ch grid results avec library(ggplot2)
plot(NB1)
```



Le meilleur modèle est obtenu avec l'utilisation d'un noyau pour estimer la densité et une largeur de bande de 1.

On va donc utiliser ces hyper-paramètres pour effectuer une prédiction sur les données de test et ainsi estimer l'erreur de généralisation du modèle bayésien naïf.

```r
NB1.pred<-predict(NB1,test)

# accuracy = taux de succes sur echantillon test
sum(NB1.pred==test$Y)/200

## [1] 0.68

# 1-accuracy = taux d erreur sur echantillon test
1-sum(NB1.pred==test$Y)/200

## [1] 0.32
```

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.

On peut nettoyer la session avant de poursuivre avec l'exercice 2.

```r
rm(list = ls())
```

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.

Pour connaître plus précisément la description de Vehicle, on peut taper "Vehicle" dans l'onglet Help du panneau de droite dans Rstudio. Le libellé des variables apparaît en clair. Il y a 18 régresseurs. La variable à expliquer est Class, variable qualitative à 4 modalités.

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)`**. 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.

```r
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 (standardiser = réduire les régresseurs avec la fonction scale) pour que les variables de plus grande variance n'influencent pas plus que les autres le calcul des distances entre observations.

Question 2 - Fonction de coût

Quelle fonction de coût utiliser ?

Dans le cas d'une classification, la fonction de coût est $l(y, \hat{y}) = \mathbf{1}_{y \neq \hat{y}}$

```
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. Justifier 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 régresseurs doivent être standardisés dans les données de train et 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 les données train.

```
best_k <- numeric(length = 20)
for (i in 1:20) {
  knn_cross_results <- tune.knn(
    x = train_scaled,
    y = train$Class,
    k = 1:50,
    tunecontrol = tune.control(sampling = "cross"),
    cross = 10
  )
  best_k[i] <- knn_cross_results$best.parameters[[1]]
```

```
  # print(sprintf("Itération %s : meilleur K = %s", i, knn_cross_results$best.parameters))
}

summary(as.factor(best_k))
```

Valeur de K	3	4	6	7	8	9	10
Nombre d'occurrences	10	4	1	1	1	1	2

On utilise la valeur la plus fréquente, ici k=3, pour construire notre modèle de plus-proches-voisins et ainsi estimer 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 de la variable Class en apprentissage
  k = 3 # nombre de voisins
)
```

L'erreur estimée 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 (modalité de Y) prédite.

Le choix des hyper-paramètres

On va ici utiliser la libraire **caret** qui comporte un ensemble de fonctions 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 estimer l'erreur de généralisation pour différentes combinaisons d'hyper-paramètres.

On définit l'ensemble des combinaisons à tester.

```
hyperparam_grid <- expand.grid(
  usekernel = c(TRUE,FALSE), # 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
```

```
)
```

On spécifie l'argument **method** à **"cv"** pour faire de la validation croisée et l'argument **number** à 10 pour utiliser 10 blocs.

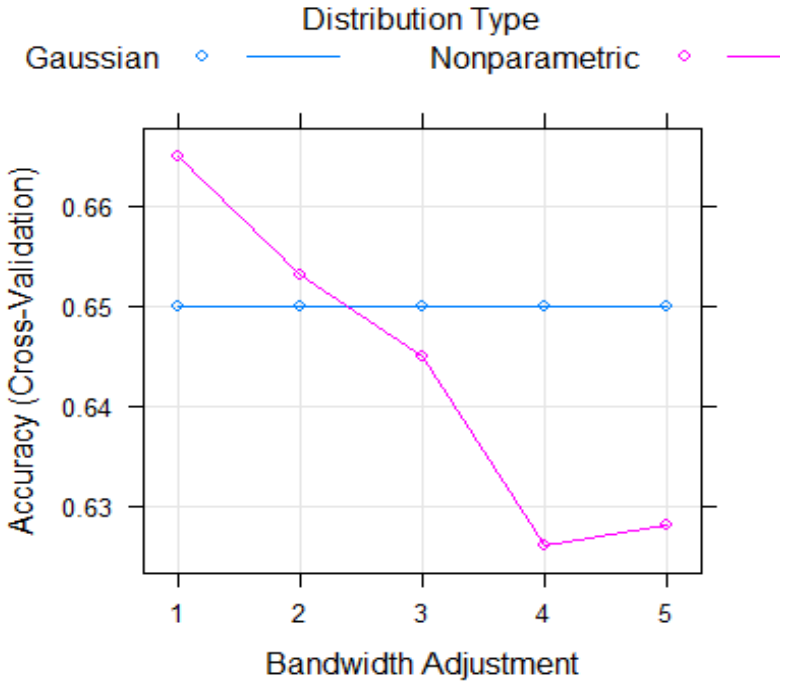
```
control <- trainControl(method = "cv", number = 10)
```

On peut ensuite entraîner le modèle avec la fonction **caret::train()**. Un modèle par combinaison sera entraîné et l'erreur sera estimée par validation croisée. 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
)

plot(naive_bayes)
```



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`. Dans le cas des régresseurs tous numériques, cela arrive,avec des observations qui présentent des valeurs extrêmes (outliers).

On peut utiliser la commande `predict(naive_bayes, test, type = "prob")` pour avoir les probabilités a posteriori estimées directement et non pas les classes (modalités prédites de Y).

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 find 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 = "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) et neighbourhood group. 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 :

$$l(y, \hat{y}) = |y - \hat{y}|^2$$

```
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.

En régression pour prédire la valeur sur Y d'une nouvelle observation i′, on calcule la distance qui sépare cette observation de toutes les observations des données d'apprentissage. On sélectionne les k points les plus proches, puis on calcule la valeur prédite en faisant la moyenne des Y mesurés sur ces plus proches voisins.

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.

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).

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 :

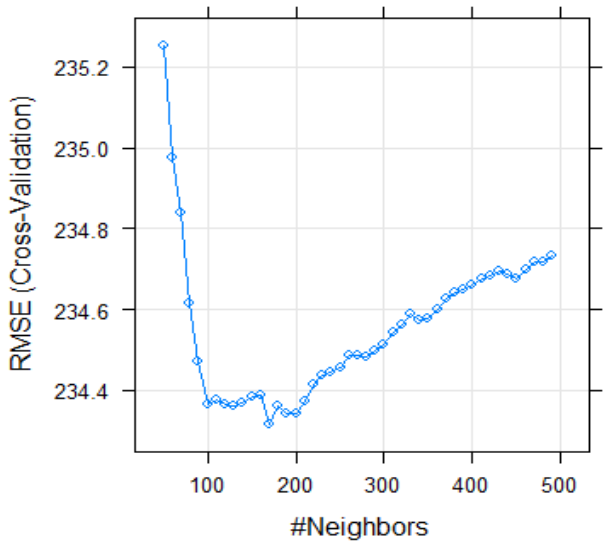
- Effectuer une ACM sur les régresseurs qualitatifs **et les régresseurs quantitatifs discrétisés** puis calculer les distances euclidiennes sur les **coordonnées factorielles non réduites**. L'inconvénient ici est une perte d'information lors de la phase de discrétisation.
- Utiliser la distance de Hamming qui est applicable pour tout type de données (ou tout autre distance avec les mêmes propriétés)
- Utiliser uniquement les variables continues mais là on peut avoir une perte d'information importante en n'utilisant pas les régresseurs qualitatifs
- Utiliser **une analyse factorielle de données mixtes (AFDM)** pour calculer des coordonnées factorielles sans avoir à discrétiser les régresseurs quantitatifs (contrairement à l'ACM), puis calculer les distances euclidiennes sur les **coordonnées factorielles non réduites (utilisées alors comme de nouveaux régresseurs quantitatifs).** Cette analyse factorielle qui permet de mélanger comme variables actives des régresseurs quantitatifs et qualitatifs est implémentée dans FactomineR. Cette méthode AFDM+Knn est sans doute la plus aboutie des 4.

NE SURTOUT PAS TRANSFORMER ARTIFICIELLEMENT LES REGRESSEURS QUALITATIFS EN FAUSSES VARIABLES QUANTITATIVES dont les valeurs numériques seraient des numéros de niveau des facteurs initiaux. Un calcul de distance euclidienne sur ces numéros de niveau n'aurait aucun sens statistique.

Variables continues uniquement à titre d'exemple

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 = 10) # 10-fold CV
)
plot(knn_fit)
```



L'erreur estimée en validation croisée par RMSE (Root Means Square Error = racine carrée de l' Erreur Quadratique Moyenne) est de 212.363

Analyse Factorielle de Données Mixtes (AFDM)

A la place de ces variables quantitatives d'origine on pourrait utiliser les coordonnées factorielles **non réduites (pas de scale)** d'une AFDM. Le choix du nombre d'axes à retenir peut se faire également par validation croisée.

On effectue une AFDM sur les données d'apprentissage en conservant les 8 premières dimensions dans le résultat.

```
famd <- FAMD( train[, c("neighbourhood_group","room_type","latitude","longi
tude")],
  ncp = 8,
  graph = FALSE
)
```

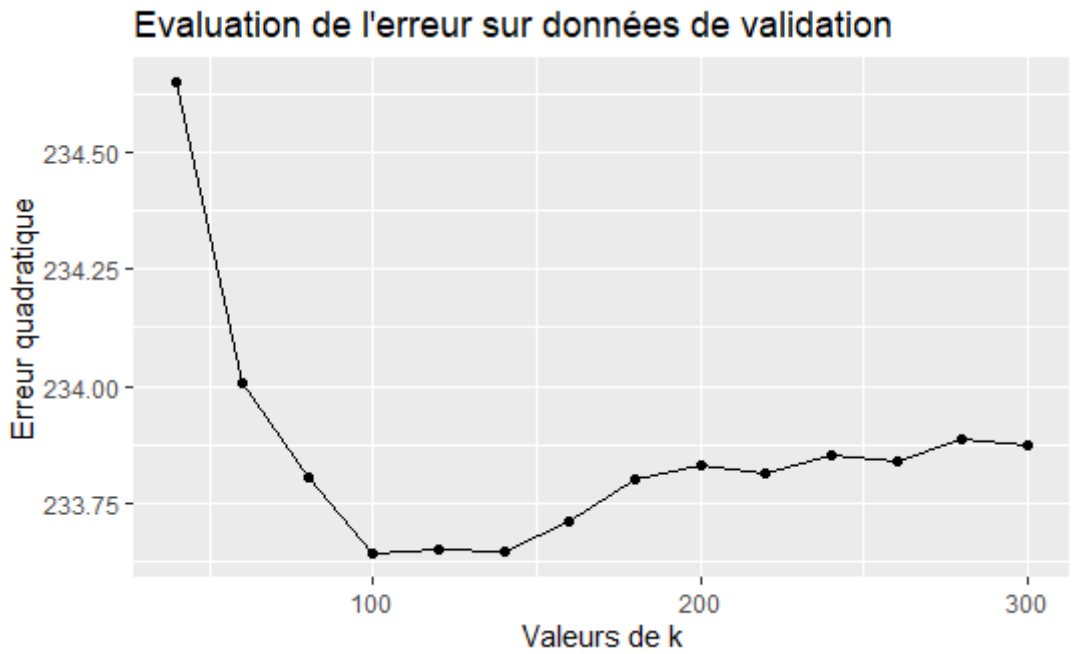
A partir des résultats on a va construire un nouveau jeu de données, dont les régresseurs (coordonnées factorielles) ne seront pas réduits.

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

```
# Entraîner le modèle de knn
knn_fit <- train(
  x = famd$ind$coord, # 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 = 10) # 10-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"
  )
```



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

```
# Préparer les données test pour faire une prédiction
famd_pred <- predict.FAMD(
  object = famd,
  newdata = test[, c("neighbourhood_group","room_type",
                     "latitude","longitude")]
)

colnames(famd_pred$coord)<-colnames(famd$ind$coord)
# Effectuer une prédiction sur les données test
knn_pred <- predict(
  object = knn_fit,
  newdata = famd_pred$coord)

# Calcul de l'erreur
knn_acm_error <- sqrt(mean((knn_pred - test$price)^2))
```

Avec 8 axes on obtient un RMSE de 207,17 en validation croisée 10 blocs. Avec 7 axes, on obtient on obtient un RMSE de 207,27.