Projet

Abstract

Le présent projet se propose d'aborder les différents points vus en cours mais aussi d'explorer des notions en lien avec les modèles linéaires mais non évoqués pendant les séances.

Il se décompose en trois parties :

- la première sera consacrée au modèle linéaire multiple et modèle logistique afin de construire le modèle plus pertinent pour la tâche demandée.
- dans une seconde partie, nous étudierons la régression de type Lasso (ou ridge à voir) afin de faire de la sélection de variables et de modèles. Il s'agit d'introduire et de présenter un contexte plus orienté *Machine Learning* pour sélectionner un modèle.
- dans une dernière partie, on se propose d'étudier des modèles non paramétriques fondés sur la régression à noyaux.

Dans chaque partie, nous serons amenés à emmployer des jeux de données différents. Ces derniers sont présentés en début de chaque section du projet.

Première Partie

Dans cette première partie un institut de biologie effectue une série d'étude concernant le taux de cholestérol ou encore l'état de santé du sujet (absence ou non de diabète). Pour cela, elle a procédé à une étude sur 30 personnes sélectionnées aléatoirement dans la population. Les données de cette étude sont données ci-dessous.

```
## Jeu de données de l'étude

Diabete = c(1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0,
            1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0)

Sport = c(0, 2, 0, 2, 0, 0, 2 ,0, 1, 0, 2, 2, 0, 2, 0,
          0, 1, 1, 2, 2, 0, 1, 1, 2, 2, 0, 0, 2, 1, 2)

Cholesterol=c(354, 190, 405, 263, 451, 302, 288, 385, 402, 365,
              209, 290, 346, 254, 395, 435, 543, 345, 298, 237,
              421, 498, 348, 123, 283, 319, 361, 298, 271, 246)

Poids=c(84, 73, 65, 70, 77, 69, 63, 72, 79, 75,
        47, 89, 65, 57, 59, 65, 80, 69, 59, 78,
        103, 80, 87, 61, 63, 69, 77, 79, 81, 70)

Age=c(46, 21, 52, 30, 57, 25, 28, 36, 57, 44,
      24, 31, 52, 25, 60, 23, 67, 54, 38, 47,
      39, 21, 45, 84, 53, 59, 37, 98, 45, 56)

Taille=c(180, 190, 160, 155, 165, 170, 175, 180, 150, 165,
         160, 165, 165, 170, 165, 167, 165, 156, 169, 179,
         185, 168, 156, 182, 190, 165, 147, 166, 188, 170)
```

```r
df<-data.frame(Cholesterol,Poids,Age,Taille,Sport,Diabete)
df$Sport<-as.factor(df$Sport)
df$Diabete<-as.factor(df$Diabete)

# Extrait du jeu de données
head(df)
```

```
##   Cholesterol Poids Age Taille Sport Diabete
## 1         354    84  46    180     0       1
## 2         190    73  21    190     2       0
## 3         405    65  52    160     0       1
## 4         263    70  30    155     2       0
## 5         451    77  57    165     0       1
## 6         302    69  25    170     0       1
```

Ce jeu de données contient les informations suivantes :

- le taux de cholestérol chez l'individu en $mg.L^{-1}$,
- le poids de l'individu en kg,
- l'âge de l'individu en année,
- la taille de l'individu en cm
- l'intensité de la pratique sportive de l'individu : 0 : pas d'activité sportive, 1 : une activité sportive occasionnelle et 2 une activité sportive régulière,
- on indique enfin si le sujet est atteint du diabète (1) ou non (0).

Estimation du Cholestérol

Dans un premier temps, les biologistes cherchent à déterminer s'il existe un lien entre le taux de cholestérol chez l'individu et les caractéristiques physiques de ce dernier, *i.e.* en fonction de son *âge, poids* et de sa *taille*. Pour cela, elle modélise le problème sous forme d'une régression linéaire multiple et cherche à déterminer le meilleur modèle qui soit.

(i) Rappeler la forme d'un modèle linéaire multiple, la dimension des objets ainsi que les hypothèses relatives à ce type de modèle.

(ii) Comment sont déterminés les paramètres du modèles ? Donner l'expression et les valeurs de ces paramètres. Pour ce dernier point, on pourra utiliser la fonction *lm* de R.

On souhaite maintenant étudier le modèle pour savoir si les données récoltées par les biologistes sont pertinentes pour expliquer le taux de cholestérol chez les individus.

(iii) Commenter, analyser les sorties du modèle.

(iv) Sélectionner une variable au choix parmi les trois étudiées et retrouver toutes les valeurs associées à cette variable ("Estimate", "Std. Error", "t value" et "Pr(>|t|)"). On rappellera les expression littérales des quantités étudiées et on effectuera les calculs sous R.

Etant données les observations faites précédemment, on se propose de construire un modèle plus pertinent pour estimer la taux de cholestérol.

(v) Après avoir rappelé un critère pouvant être utilisé pour comparer des modèles, utiliser ce dernier afin de sélectionner le meilleur modèle au sens du critère choisi. Quel modèle le plus pertinent obtenez-vous ?

Etude du diabète

On souhaite maintenant regarder s'il est possible de prédire si l'individu est atteint de diabète en fonction de l'ensemble des descripteurs dont nous disposons. Comme nous avons pu le faire précédemment, on chercher à déterminer quels sont les variables les plus pertinentes pour expliquer la présence ou non de diabète chez un individu.

(vi) Quel type de de modèle proposeriez-vous pour modéliser le problème ? Rappeler brièvement son fonctionnement.

(vii) Consruire votre modèle de régression logistique à l'aide de la fonction *glm* comme suit et commenter les sorties de ce modèle. En foncton des observations effectuées, proposer un modèle qui vous semble plus pertinent. On pourra par exemple comparer et étudier les modèles suivants :

- un modèle complet (qui prend en compte l'ensemble des variables)
- un modèle qui utilise les variables Cholesterol, Poids et l'intensité de la pratique sportive
- un modèle qui utilise uniquement le taux de Cholestérol et l'intensité de la pratique sportive
- un modèle qui utilise uniquement le taux de Cholestérol

```
model<- glm(Diabete ~ ., data = df, family = "binomial")
summary(model)
```

Deuxième Partie

Dans cette deuxième partie on se propose d'étudier la *régression pénalisée de type Lasso*. Il s'agit d'une forme plus générale que la régression linéaire multiple dans lequel on introduit un *terme de régularisation* qui va induire des contraintes sur les paramètres du modèle appris. Le modèle se présente sous la même forme :

$$Y = X\beta + \varepsilon, \quad \text{où} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

avec les mêmes hypothèses que pour le modèle linéaire. En revanche, la façon dont est appris le modèle diffère; le paramètre β est cette fois-ci solution du problème d'optimisation :

$$\underset{\beta \in \mathbb{R}^p}{\arg\min} \ \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1, \tag{1}$$

où le premier terme représente notre estimateur MCO et le deuxième terme est notre terme de régularisation/pénalisation de type **Lasso**, *i.e.* il s'agit d'un terme de régularisation utilisant la norme L_1. Enfin le paramètre λ est un terme de régularisation qui va permettre de contrôler le *trade-off* entre le pouvoir prédictif du modèle et la norme des paramètres appris.

A propos de la régression pénalisée de type Lasso

Ce type de modèle est très utilisé dans l'étude de la *Statistique en Grande Dimension, i.e.* lorsque le nombre de variables du modèle p est plus grand que le nombre de données n, c'est par exemple le cas lorsque l'on étudié des données génomiques.
Cette pénalisation va ainsi permettre de faire de la **sélection de variables** et permettre l'apprentissage d'un modèle dit **parcimonieux**. Le vecteur β sera alors "creux" et comprendra plein de valeurs proches de 0, ce qui va permettre de ne conserver que des variables significatifs pour le modèle.

(i)) Supposons $n << p$ et $\lambda = 0$, que peut-on dire à propos de la matrice $(X^T X)$? La solution de l'équation (1) est-elle la même que lorsque $n > p$?

(ii)) Que se passe-t-il dans le cas limite où $\lambda \to \infty$?

On va maintenant s'intéresser au cas où $\lambda > 0$, et notre objectif sera de déterminer un bon modèle pour notre tâche de régression. Une question que l'on se pose alors est de savoir comment choisir une bonne valeur de λ pour avoir un modèle à la fois parcimonieux et performant. On va utiliser ce que l'on appelle de la cross-validation.

Apprentissage et Cross-Validation

Cette section se présente comme une une introduction au Machine Learning dont le but est de faire de la sélection de modèles.

Notre objectif va être d'apprendre un modèle à partir de données **d'entraînement** et qui sera capable d'être performant sur des données non rencontrées jusqu'à présent que l'on appelle des données **tests**.

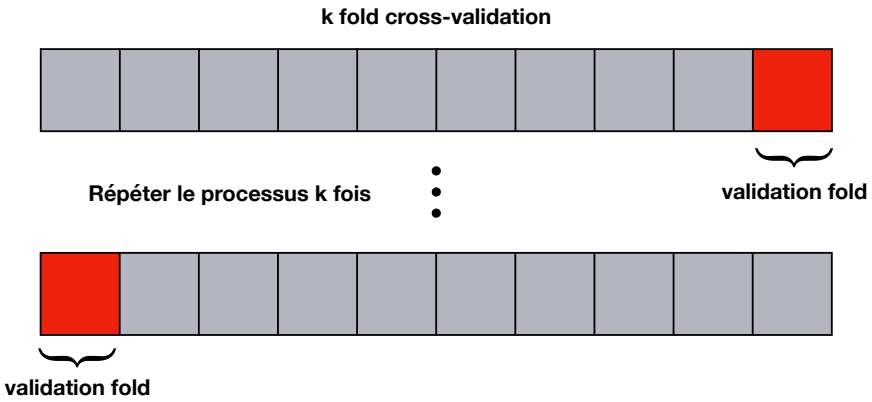
Ces données tests vont servir à évaluer la qualité de notre modèle, on va donc supposer que l'on ne diposera de ces données que lors de l'évaluation finale du modèle, elles ne sont pas prises en compte pendant la phase d'apprentissage. Il est en revanche très important de supposer, dans ce type d'approche, que ces données tests sont issues de la même distribution que les données d'entraînement.

Revenons au cas de la régression lasso, nous avons vu qu'elle dépend d'un *hyper-paramètre* λ qui contrôle l'importance du terme de régularisation. La question que l'on peut alors se poser est "qu'elle est une bonne valeur de λ" ? Une approche naïve serait de tester plusieurs valeurs et de regarder ce que cela donne en phase

de test ... cette approche naïve n'est pas réalisable en pratique (dans de cas réels ou les données ne sont pas observées) mais elle n'est pas totalement dénuée de sens.
En fait, ce que l'on va faire c'est considérer une partie de notre ensemble d'entraînement comme étant "une sorte d'ensemble test" que l'on appellera ensemble de *validation*. Ce dernier ensemble va servir à estimer les performances obtenues par notre modèle avec le paramètre λ considéré. Nous pourrons ainsi tester différentes valeurs de λ et évaluer les performances des modèles appris avec chacune des valeurs de cet hyper-paramètre sur l'ensemble de validation. In fine on retiendra le modèle associé à la valeur de λ qui maximisera le critère de performance sur notre ensemble de validation. Enfin le modèle retenu est appliqué aux données tests. On résume la procédure ci-dessous :

1. Séparer notre jeu de données en deux ensembles S_{train} et S_{test}, ainsi qu'une liste prédéfinie de valeurs $\lambda = (\lambda_i)_{i=1}^N$.
2. Séparer le jeu de données S_{train} en deux ensembles : S_{learn} et S_{valid}.
3. Pour i allant de 1 à N
   (a) apprendre un modèle M_i correspondant à la solution du problème d'optimisation à l'aide du jeu de données S_{learn}
   $$\arg\min_{\beta \in \mathbb{R}^p} \|Y - X\beta\|_2^2 + \lambda_i \|\beta\|_1,$$
   (b) évaluer les performances du modèle M_i (retournant les paramètres $\hat{\beta^{(i)}}$) sur le jeu de données de validation S_{val}, *i.e.* évaluer la quantité.
   $$\|Y - X\hat{\beta^{(i)}}\|_2^2,$$
4. Retenir le modèle M parmi l'ensemble des modèles M_i qui minimise l'erreur en régression
   $$M = \arg\min_{i=1,\dots,N} \|Y - X\hat{\beta^{(i)}}\|_2^2.$$
5. Evaluer les performances du modèles M sur les données de tests S_{test}.

L'approche précédente présente cependant l'inconvénient ne se servir qu'une partir des données pour évaluer les performances d'un modèle appris, donc d'un ensemble très restreint de la distribution. Il n'est donc pas à exclure que le modèle retenu pendant la phase de "validation" puisse être totalement différent si S_{valid} avait été tiré différemment.
Pour palier à cela on utilise un autre procédé que l'on appelle $k-$*cross-validation* et qui consiste à itérer le processus de validation présenté plus tôt en faisant également varier l'ensemble de validation (et donc l'ensemble d'apprentissage !). Le procédé est illustré ci-dessous :



Pourquoi k ? En fait cela représente le nombre de groupes G_j dans lesquels sont réparties nos données et on va successivement, pour $j = 1, ..., K$ se servir d'un groupe comme étant $S_{valid} = G_j$ et les autres groupes pour définir $S_{learn} = \cup_{k \neq j} G_j$.

On va ensuite retenir le modèle qui maximise les performances **en moyenne** sur les $k-$folds, donc sur les k groupes possibles en tant qu'ensemble de validation. Les étapes de la procédure peuvent se résumer de la façon suivante :

1. Séparer notre jeu de données en deux ensembles S_{train} et S_{test}, ainsi qu'une liste prédéfinie de valeurs $\lambda = (\lambda_i)_{i=1}^N$.
2. Séparer le jeu de données S_{train} en k folds $(G_j)_{j=1}^k$
3. Pour j allant de 1 à k
   (a) on pose $S_{valid} = G_j$ et $S_{learn} = \cup_{l \neq j} G_l$
   (b) appliquer ensuite l'étape 3) de la procédure précédente et enregister les valeurs du critère de performance associé
4. Pour chaque valeur λ_i, calculer la performance moyenne sur l'ensemble des folds
5. Retenir la valeur de λ_i, que l'on notera λ qui donne la meilleure performance moyenne sur l'ensemble des $k-$folds.
6. Réapprendre un modèle avec l'ensemble des données d'entraînement S_{train} et la valeur de λ retenue et évaluer les performances de ce modèle sur les données de tests S_{test}.

Rassurez-vous, la section suivante vous expliquera comment faire tout cela avec R.

Mise en pratique

Nous allons maintetant mettre en pratique a régression de type Lasso. Pour cela on va considérer le jeu de données que vous pourrez trouver télécharger à l'aide de la commande suivante :

```
# N'hésitez pas à taper ?Boston dans la console de R pour avoir une description du jeu
# de données. La variable que l'on cherche à prédire est "medv" : la valeur médiane
# des logements occupés par leur propriétaire.

library(MASS)
data(Boston)
```

Pour apprendre un modèle de type lasso, vous aurez besoin d'installer le package "*glmnet*" (n'oubliez pas de charger la librairie par la suite).

L'apprentissage d'un modèle se fera à l'aide de la fonction *glmnet* et on peut faire de la cross-validation *cv.glmnet*. N'hésitez pas à consulter l'aide R pour connaître le fonctionnement de ces deux fonctions.

(iii) Commencez par séparer votre jeu de données en deux ensembles S_{train} et S_{test} avec une proportion de 70% des données dans S_{train} et 30% dans S_{test}. On pourra créer une liste d'indice ("index") dans laquelle on stocke les indices des données servant d'ensemble d'entraînement.

(iv) On va regarder les données d'entraînement pour étudier l'influence du paramètre de régularisation. Regarder l'évolution des coefficients de la régression en fonction de la valeur du paramètre λ et décrivez le graphique. Est-ce en encore avec votre intuition formulée à la question (ii) ?

```
# On centre et réduit nos données
Boston.X.std<- scale(Boston[index,names(Boston)!="medv"])
```

```r
# On extrait nos jeux de données train et test
X.train<- as.matrix(Boston.X.std)
X.test<-  as.matrix(scale(Boston[-index,names(Boston)!="medv"],
                          attr(Boston.X.std, "scaled:center"),
                          attr(Boston.X.std, "scaled:scale")))
Y.train<- Boston[index, "medv"]
Y.test<- Boston[-index, "medv"]

# Apprentissage du modèle
lasso.fit<- glmnet(x=X.train, y=Y.train, family = "gaussian", alpha = 1)
plot(lasso.fit, xvar = "lambda", label=TRUE)
```

On cherche maintenant à apprendre un modèle qui est capable d'estimer le prix d'une maison (la variable Y) en fonction des variables explicatives à notre disposition (la variable X) sur notre jeu de données S_{test}. Ce modèle sera appris à partir de S_{train} en utilisant le principe de cross-validation pour trouver la meilleur du paramètre λ.

(v) Utiliser la fonction *cv.glmnet* de R en précisant que voius souhaitez utiliser 5 folds afin de trouver le meilleur paramètre λ parmi les valeurs suivantes

$$\lambda = (10^{-4}, 2.10^{-4}, 2.10^{-4}, 5.10^{-4}, 7.10^{-4}, 10^{-3}, 2.10^{-3}, 5.10^{-3}, 10^{-2}, 5.10^{-2}, 10^{-1}, 0.5, 1, 2, 5, 10, 20, 50, 100)$$

Quelle est la valeur de λ optimale retenue après cross-validation ? Est-ce qu'elle correspond à un modèle parcimonieux ? (On pourra faire un graphe des résultats obtenus par cross-validation) et on utilisera la commande "plot"

```r
lambda = c(10^(-3), 2*10^(-3),  5*10^(-3), 10^(-2), 5*10^(-2),
           10^(-1), 0.5, 1, 2, 5, 10, 20, 50, 100)
```

(vi) Estimer les performances sur votre jeu de données test en évaluant le risque quadratique moyen.

(vii) Comparer les résultats obtenus avec un modèle de régression standard et avec une régularisation de type Lasso. Avec quel modèle obtenez vous de meilleurs performances en test ? Commentez l'utilisation d'un modèle de type Lasso dans le cas présent.

Troisième Partie

Dans cette dernière partie, on se propose d'introduire un nouveau type de modèle, les modèles *non paramétriques*. Comme son nom l'indique ce type de modèle ne fera intervenir aucun paramètre et il va reposer uniquement sur les observations dont on dispose.

Pour être plus précis, ce type de modèle repose sur l'estimation de densité d'une variable aléatoire X en fonction de ces réalisations $\{x_i\}_{i=1}^n$. Pour estimer cette densité f, en un point x, on utilise ce que l'on appelle un noyau K (il s'agit d'une fonction qui possède quelques bonnes propriétés) et l'estimation $\hat{f}$ est donnée par :

$$\hat{f}(x,\sigma) = \frac{1}{n\sigma}\sum_{i=1}^{n} K_\sigma(x - x_i)$$

elle dépend donc d'un paramètre d'échelle σ qui va permettre d'accord plus ou moins d'importance aux réalisations x_i en fonction de leur distance à x. \ Le plus souvent, le noyau K employé est un noyau gaussien, qui s'écrit donc sous forme :

$$K(x - x_i) = K_\sigma(x - x_i) = \frac{1}{\sqrt{\pi}} \exp\left(-\frac{\|x - x_i\|^2}{\sigma^2}\right).$$

Rappellons maitenant que notre objectif est de déterminer une fonction m telle que :

$$m(X) = \mathbb{E}[Y \mid X = x]. \tag{2}$$

Pour rappel dans le cas du modèle linéaire multiple, nous avions $m(X) = \beta X$ où β **est le paramètre de la régression**.

Nous allons maintenant étudier deux types d'estimateurs à l'aide de jeux de données à deux dimensions pour visualiser les résultats :

- l'estimateur de Nadaraya-Watson
- l'estimateur polynomial local

Les deux estimateurs consistent en l'estimation d'une bonne pondération des exemples.

Estimateur de Nadaraya-Watson

Pour ce premier estimateur, commençons par réécrire l'équation (2) à l'aide des fonctions de densité :

$$\begin{aligned} m(X) &= \mathbb{E}[Y \mid X = x], \\ &= \int y f_{Y|X=x}(y) dy, \\ &= \frac{\int y f_{X,Y}(y) dy}{f_X(x)}, \end{aligned}$$

où $f_{X,Y}$ désigne la densité jointe des variables aléatoires X et Y et f_X est la densité marginale de la variable aléatoire X.

Considérons maintenant un jeu de données $\{(x_i, y_i)\}_{i=1}^n$ et l'estimation par méthode à noyaux des deux densités précédentes :

$$f_{X,Y} \simeq \hat{f}(x, y, \sigma) = \frac{1}{n\sigma_1\sigma_2} \sum_{i=1}^{n} K_{\sigma_1}(x - x_i) K_{\sigma_2}(y - y_i).$$

et

$$f_X \simeq \hat{f}(x, \sigma) = \frac{1}{n\sigma_1} \sum_{i=1}^{n} K_{\sigma_1}(x - x_i).$$

(i) A l'aide des deux approximations précédentes, montrer que l'on peut écrire :

$$\frac{\int y f_{X,Y}(y) dy}{f_X(x)} = \sum_{i=1}^{n} \frac{K_{\sigma_1}(x - x_i)}{\sum_{i=1}^{n} K_{\sigma_1}(x - x_i)} y_i.$$

On utilisera le fait que :

$$\frac{1}{\sqrt{\pi}\sigma} \int x \exp\left(-\frac{(x - m)^2}{\sigma^2}\right) dx = m$$

(ii) En déduire que le modèle peut alors s'écrire sous la forme

$$\hat{m}(x, \sigma) = \sum_{i=1}^{n} W_i(x) y_i,$$

où on explicitera la valeur des coefficients $W_i(x)$ et interpréter ce modèle.

On souhaite maintenant visualiser les résultats obtenus pour cet estimateur à l'aide du code ci-dessous

```r
nw <- function(x, X, Y, sigma, K = dnorm) {

  # Arguments
  # x : evaluation points
  # X : vecteur contenant les valeurs des observations x_i
  # Y : vecteur contenant les valeurs de la variable réponse
  # sigma : le paramètre d'échelle
  # K : noyau

  # Fonction qui calcule la fonction de régression
  Kx <- rbind(sapply(X, function(Xi) K((x - Xi) / sigma) / sigma))
  W <- Kx / rowSums(Kx)
  drop(W %*% Y)
}

# A nouveau, on fixe la graine pour que l'on puisse tous avoir des résultats identiques
set.seed(12345)
n <- 100
eps <- rnorm(n, sd = 2)
# La vraie fonction que l'on cherche à estimer
m <- function(x) 3*x + 5*cos(x)
# Génération des données "bruitées"
```

```r
X <- rnorm(n, sd = 2)
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# Paramètre d'échelle
sigma <- 1

# Graphe des résultats
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(x_grid, nw(x = x_grid, X = X, Y = Y, sigma = sigma), col = 2)
lines(sort(X), nw(x = sort(X), X = X, Y = Y, sigma = sigma), col = 2)
legend("top", legend = c("Foction de  régression", "Nadaraya-Watson"),
       lwd = 2, col = 1:2)
```

(iii) Tester le code précédent avec des valeurs de σ différentes. Décrire le/les graphe(s) obtenu(s) avec des petites et grandes valeurs de $\sigma > 0$.

(iv) Fixons maintenant $\sigma = 1$. Comparer l'estimateur de Nadaraya-Watson avec un modèle obtenu par régression linéaire simple (on pourra l'ajouter au graphe précédent). Avec quel modèle obtenez vous l'erreur quadratique moyenne la plus faible ? Pensez vous qu'un modèle linéaire est pertinent à utiliser sur ces données ?

(v) (Bonus) Replaçons nous dans un contexte de Machine Learning et reprenons l'analyse effectuée à la question (*iii*), quel est l'inconvénient de prendre une valeur σ faible.

Estimateur polynomial local

Dans cette dernière partie, on se propose d'étendre la notion d'estimateur de Nadaraya-Watson en la notion d'estimateurs polynomiaux locaux. L'estimateur de Nadaraya-Watson se présente en fait comme l'estimation de la partie constante d'un polynôme.

Rappellons qu'en régression notre obejctif est de trouver une fonction $\hat{m}$ qui minimise l'erreur quadratique (moyenne) :

$$\sum_{i=1}^{n} \left(y_i - \hat{m}(x_i)\right)^2 . \tag{3}$$

Le seul problème est que l'on ne dispose d'aucune hypothèse sur la forme de $\hat{m}$ dans le cas de modèles non paramétriques. Pour remédier à cela, l'idée est d'introduire une paramétrisation locale sur la fonction m par une approximation de Taylor à l'ordre p pour tout x proche de x_i :

$$\begin{aligned} m(x_i) &= m(x) + m'(x)(x_i - x) + \frac{m''(x)}{2!}(x_i - x)^2 + ... + \frac{m^{(p)}(x)}{p!}(x_i - x)^p + \mathcal{O}\left((x_i - x)^{p+1}\right), \\ &= \sum_{k=0}^{p} \frac{k!}{m^{(k)}(x)}(x_i - x)^k + \mathcal{O}((x_i - x)^{p+1}). \end{aligned}$$

En injectant ce développement dans l'équation (3) on chercher alors à minimiser la quantité

$$\sum_{i=1}^{n}\left(y_i - \sum_{k=0}^{p}\underbrace{\frac{k!}{m^{(k)}(x)}}_{=\beta_k}(x_i - x)^k\right)^2.$$

Avec cette expression, on voit que si on arrive à estimer les valeurs β_k va nous permettre d'avoir une estimation les valeurs $m^{(k)}(x)$.

Pour compléter le modèle précédent, on va ensuite pondérer le poids de chaque exemple en fonction de sa distance à la donnée étudiée x en utilisant un noyau K_σ. Notre problème final s'écrit alors :

$$\hat{\beta} = \underset{\beta\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n}\left(y_i - \sum_{k=0}^{p}\beta_k(x_i - x)^k\right)^2 K_\sigma(x - x_i). \tag{4}$$

Et ce type de formulation est très proche du problème que l'on cherche à résoudre dans l'étude des modèles linéaires

$$\hat{\beta} = \underset{\beta\in\mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n}\left(Y_i - \sum_{k=0}^{p}\beta_k X_k\right)^2.$$

(i) Montrer que le problème (4) peut s'écrire sous la forme

$$\hat{\beta} = \underset{\beta\in\mathbb{R}^{p+1}}{\arg\min} (Y - X\beta)^T W(Y - X\beta)$$

et donner la forme des objets (qui sont des matriaces) X, Y et W.

(ii) Déterminer l'expression de $\hat{\beta}$ en fonction de X, Y et W.

On peut alors montrer que le modèle $\hat{m}$, qui est un donc un estimateur local polynomial d'ordre p, peut s'écrire sous la forme :

$$\hat{m}(x, p, \sigma) = u_1^T(X^T W X)^{-1}X^T W Y = \sum_{i=1}^{n} W_i^p(x) y_i,$$

où $W_i^p(x) = u_1^T(X^T W X)^{-1}X^T W e_i$, $u_1 \in \mathbb{R}^{p+1}$ est un vecteur comprenant u la valeur 1 en première position et des 0 partout ailleurs et e_i est un vecteur comprenant des 0 partout, sauf à la $i-$ème position où il prend la valeur 1.

(iii) Montrer que lorsque $p = 0$, on retrouve bien l'estimateur de Nadaraya-Watson, *i.e.*

$$W_i(x) = W_i^0(x) = \frac{K_\sigma(x - x_i)}{\sum_{i=1}^{n} K_\sigma(x - x_i)}$$

(iv) **(Question difficile).** Supposons maintenant que $p = 1$, dans ce cas notre estimateur $\hat{m}$ est polynomial d'ordre 1. Montrer que l'on a :

$$W_i^1(x) = \frac{1}{n} \times \frac{\hat{s}_2(x, \sigma) - \hat{s}_1(x, \sigma)(x_i - x)}{\hat{s}_2(x, \sigma)\hat{s}_0(x, \sigma) - \hat{s}_1(x, \sigma)^2} \times K_\sigma(x - x_i),$$

où $\hat{s}_r(x,\sigma) = \frac{1}{n}\sum_{i=1}^{n}(x_i - x)^r K_\sigma(x - x_i)$.

(v) En étudiant les expressions de W_i^0 et de W_i^1, quel est selon le vous le principal inconvénient d'un estimateur local polynomial d'ordre 1 comparé à l'ordre 0 ?

(vi) A l'aide du code ci-dessous, comparer et commenter les différents graphiques en fonction des valeurs de σ.

```r
# Génération des données
set.seed(10)
n <- 100
eps <- rnorm(n, sd = 2)
m <- function(x) 3*x + 5*cos(x)
X <- rnorm(n, sd = 1.5)
Y <- m(X) + eps
x_grid <- seq(-10, 10, l = 500)

# KernSmooth::locpoly fits

# Ces deux lignes permettent d'estimer les valeurs prises par le modèle sur une grille
# prédéfinie de valeurs.
sigma <- 0.5
lp0 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = sigma, degree = 0,
                           range.x = c(-10, 10), gridsize = 1000)
lp1 <- KernSmooth::locpoly(x = X, y = Y, bandwidth = sigma, degree = 1,
                           range.x = c(-10, 10), gridsize = 1000)


# Graphique
plot(X, Y)
rug(X, side = 1); rug(Y, side = 2)
lines(x_grid, m(x_grid), col = 1)
lines(lp0$x, lp0$y, col = 2)
lines(lp1$x, lp1$y, col = 3)
legend("bottom", legend = c("Fonction de regression", "Localement constant (ordre 0)",
                            "Localement linéaire (ordre 1)")
)
```

Annexe

Troisième partie, question (iii)

On pourra aussi utiliser ce code en complément pour la visualisation des résultats

```
manipulate::manipulate({

  # Plot data
  plot(X, Y)
  rug(X, side = 1); rug(Y, side = 2)
  lines(x_grid, m(x_grid), col = 1)
  lines(x_grid, nw(x = x_grid, X = X, Y = Y, sigma = sigma), col = 2)
  legend("topright", legend = c("True regression", "Nadaraya-Watson"),
         lwd = 2, col = 1:2)

}, sigma = manipulate::slider(min = 0.01, max = 3, initial = 0.5, step = 0.01))
```