TELECOM ParisTech **MDI720**
2017/2018 **Joseph Salmon et François Portier**

DEVOIR MAISON : Introduction au *boostrap* et aux méthodes gloutonnes

---

Pour ce TP de test de la plate-forme "Classgrade" vous devez déposer un **unique** fichier **anonymisé** (votre nom ne doit apparaître nulle part y compris dans le nom du fichier lui-même) sous format `ipynb` sur le site http://peergrade.enst.fr/.

Vous devez charger votre fichier, avant le dimanche 29/10/2017 23h59. Entre le lundi 30/10/2017 et le dimanche 05/11/2017, 23h59, vous devrez noter trois copies qui vous seront assignées anonymement, en tenant compte du barème suivant pour chaque question :

- 0 (manquant/ non compris/ non fait/ insuffisant)
- 1 (passable/partiellement satisfaisant)
- 2 (bien)

Ensuite, il faudra également remplir de la même manière les points de notation suivants :

- aspect global de présentation : qualité de rédaction, d'orthographe, d'aspect de présentation, graphes, titres, etc. (Question 16).
- aspect global du code : indentation, Style PEP8, lisibilité du code, commentaires adaptés (Question 17)
- Point particulier : absence de bug sur votre machine (Question 18)

Des commentaires pourront être ajoutés question par question si vous en sentez le besoin ou l'utilité pour aider la personne notée à s'améliorer, et de manière obligatoire si vous ne mettez pas 2/2 à une question. Enfin, veillez à rester polis et courtois dans vos retours.

Rappel : aucun travail par mail accepté !

EXERCICE 1. (Régression robuste)

On propose d'utiliser le bootstrap pour calibrer la fonction de perte utilisée dans le calcul de l'estimateur du coefficient de régression linéaire. L'étude proposée est basée sur des données simulées. On considère le modèle de régression suivant : $(y_i, x_i)_{i=1,\dots,n}$ est une suite de variables indépendantes et pour chaque $i = 1, \dots, n$,

$$y_i = \theta_0 + x_i^\top \boldsymbol{\theta} + \varepsilon_i,$$

où

- x_i est un vecteur colonne de taille $d \times 1$ dont les entrées sont indépendantes, chacune d'elles est distribuée selon la loi uniforme sur $[0, 1]$,
- $\boldsymbol{\theta} = (\theta_j)_{1 \leqslant j \leqslant d} \in \mathbb{R}^d$ est un vecteur colonne de taille $d \times 1$ et $\theta_0 \in \mathbb{R}$ est un scalaire,
- ε_i est de moyenne nulle et est indépendante de x_i (la distribution de ε_i sera spécifiée par la suite).

Pour chaque $\alpha > 0$, on définit l'estimateur suivant

$$\hat{\boldsymbol{\theta}}_\alpha = \underset{\theta_0 \in \mathbb{R},\, \boldsymbol{\theta} \in \mathbb{R}^d}{\arg\min} \sum_{i=1}^{n} \rho_\alpha(y_i - \theta_0 - x_i^\top \boldsymbol{\theta}),$$

où

$$\rho_\alpha(x) = \begin{cases} x^2/2 & \text{si} \quad |x| \leqslant \alpha, \\ \alpha|x| - \alpha^2/2 & \text{si} \quad |x| > \alpha. \end{cases}$$

Pour $\alpha = 0$, on prend $\rho_\alpha(x) = |x|$. L'estimation de $(\theta_0, \boldsymbol{\theta})$ est dépendante du seuil α. Le bootstrap va nous permettre de choisir convenablement α.

1) Sur un même graphique, dont l'abscisse minimal et maximal seraient -5 et $+5$, tracer ρ_α pour $\alpha = .5, 2, 5$. Décrire l'effet de alpha sur l'estimation de $\hat{\boldsymbol{\theta}}_\alpha$.
2) Générer $n = 100$ vecteurs aléatoires (y_i, x_i) selon le modèle précédent avec $d = 2$, $\boldsymbol{\theta} = (1, 1)$, $\theta_0 = 1$ et ε_i gaussien de moyenne 0 et variance 1.
3) Pour $\alpha = 2$, calculer $\hat{\boldsymbol{\theta}}_\alpha$. On pourra importer `minimize` de `scipy.optimize` et utiliser le code `minimize(f, init, method='nelder-mead',options={'xtol': 1e-5})` où `f` est la fonction (de plusieurs variables) à minimiser et `init` est le point de départ de la méthode (dans l'espace de définition de `f`, par exemple $(0, 0, 0)$). Dans la suite on va considérer d'autres valeurs de α, on pourra donc rédiger son code de façon à faciliter ce changement.
4) En utilisant le bootstrap des résidus, calculer un estimateur bootstrap $\hat{\boldsymbol{\theta}}^*_\alpha$ pour $\alpha = 2$.
5) Toujours pour $\alpha = 2$, à partir de $B = 200$ réplications d'estimateurs bootstrap $\hat{\boldsymbol{\theta}}^*_{1,\alpha}, \ldots, \hat{\boldsymbol{\theta}}^*_{B,\alpha}$ (issues de la question précédente), calculer un estimateur de la matrice de variance V_{boot} et un estimateur du biais b_{boot} de l'estimateur $\hat{\boldsymbol{\theta}}_\alpha$. On calculera aussi l'estimateur de l'erreur quadratique moyenne associée à l'estimateur, donner par

$$\mathrm{EQM}_{boot}(\alpha) = \frac{1}{B}\sum_{b=1}^{B} \|\hat{\boldsymbol{\theta}}^*_{b,\alpha} - \hat{\boldsymbol{\theta}}_\alpha\|^2 = \|\hat{\boldsymbol{\theta}}_\alpha - \overline{\hat{\boldsymbol{\theta}}^*_{\cdot,\alpha}}\|^2 + \frac{1}{B}\sum_{b=1}^{B} \|\hat{\boldsymbol{\theta}}^*_{b,\alpha} - \overline{\hat{\boldsymbol{\theta}}^*_{\cdot,\alpha}}\|^2,$$

où $\overline{\hat{\boldsymbol{\theta}}^*_{\cdot,\alpha}} = \frac{1}{B}\sum_{b=1}^{B} \hat{\boldsymbol{\theta}}^*_{b,\alpha}$ et $\|\cdot\|$ est la norme Euclidienne.
6) En remarquant que $\mathrm{EQM}_{boot}(\alpha)$ est un estimateur du risque

$$\mathbb{E}[\|\hat{\boldsymbol{\theta}}_\alpha - \boldsymbol{\theta}\|^2],$$

proposez une procédure permettant de choisir le meilleur α consiste à sélectionner le α qui minimise $\mathrm{EQM}_{boot}(\alpha)$. Sur une grille de 20 points équidistants entre 0 et 10, tracer $\mathrm{EQM}_{boot}(\alpha)$ pour α variant sur cette grille.
7) On change maintenant la distribution des erreurs. On prend une loi standard de Cauchy (générée sous `numpy` par `numpy.random.standard_cauchy(n)`). Tracer le même graphique qu'à la question précédente. Conclure.
8) Charger la base "diabetese" de `sklearn`. Appliquer la méthode précédente au choix du coefficient α pour la régression de y sur la quatrième variable de la base. On veillera à bien choisir le domaine des α pour optimiser le $\mathrm{EQM}_{boot}(\alpha)$.

Exercice 2. (Algorithmes gloutons ou *greedy*) On se place dans le cadre du modèle linéaire et l'on considère la base de données `airquality`, que l'on peut charger avec la commande

```
import statsmodels.datasets as sd
data = sd.get_rdataset('airquality').data
```

Attention, on veillera à éliminer les possibles valeurs manquantes. On prend comme variable à prédire la concentration d'Ozone, les autres variables seront les variables explicatives du modèle.

9) Standardiser les données (centrer et réduire pour obtenir une matrice X qui a des variances unitaire par colonne).

10) Écrivez une fonction `stpforward` qui prend comme argument : les observations $\mathbf{y}$, la matrice X, et enfin un paramètre M qui est le nombre maximum de variables sélectionnées (les entrees seront des `numpy` arrays. En sortie, la fonction doit renvoyer la listes `selected_variables` contenant les indices des variables sélectionnées et le vecteur $\boldsymbol{\theta} \in \mathbb{R}^p$ (dont les coordonnées sont nulles sauf les θ_j pour les indices j dans `selected_variables`) correspondant à l'estimation des moindres carrés effectuée seulement sur les variables dont les indices sont dans `selected_variables`. Une aide est donnée ici :

```
import numpy as np
from sklearn.linear_model.base import LinearModel
from sklearn.base import RegressorMixin


def stpforward(X, y, M):
    """Orthogonal Matching Pursuit model (OMP).
    X: Array-like, shape (n_samples, n_features).
        Training data.
    y: Array-like, shape (n_samples, ).
        Target values.
    M: Integer, in [1,n_features]
    """
    selected_variables = []
    residual = y
    p = X.shape[1]
    coef_selected = np.zeros(p)
    for i in range(1, M + 1):
        tab_alphaj = np.zeros(p)
        for j in range(0, p):
            if(j not in selected_variables):
                # Compute Alphaj and add it in tab_Alphaj
                Xj = X[:, j]
                # XXX: Get alpha_j value here
                valeur_alphaj = np.abs(Xj.dot(residual))
                tab_alphaj[j] = valeur_alphaj
        jmax = np.argmax(tab_alphaj)
        selected_variables.append(jmax)
        X_selected = X[:, selected_variables]
        # XXX: perform OLS over selected variables
        # Store coefficients
        coef_selected[selected_variables] = skl_linmod.coef_
        # XXX: Update residual
    return coef_selected, selected_variables
```

On codera cet algorithme comme résumé par l'Algorithme 1 : on ajoutera itérativement à l'ensemble des variables actives la variable dont le produit scalaire avec les résidus est le plus grand en valeur absolue. (on utilise la notation X_S pour la matrice créée en extrayant seulement les colonnes de X dont les indices sont dans S, idem pour $\boldsymbol{\theta}_S$: c'est un vecteur de taille $|S|$, ayant les même valeurs que $\boldsymbol{\theta}$ aux coordonnées indexées par S). L'Algorithme 1 se présente ainsi :

Algorithm 1: Algorithme *forward stage-wise selection* (ou *Orthogonal Matching Mursuit*)

Entrées : observations $\mathbf{y}$, matrice $X = [\mathbf{x}_1, \cdots, \mathbf{x}_p]$; nombre variables sélectionnées : M

Initialiser $\boldsymbol{\theta} = 0$; $r = \mathbf{y}$ et $S = \varnothing$

for $i = 1, \ldots, M$ **do**

  Calculer pour tout $j \in [\![1, p]\!] \cap S^c, \alpha_j = |\langle \mathbf{x}_j, r \rangle|$

  Trouver $j_{\max} = \underset{j \in [\![1,p]\!] \cap S^c}{\arg\max}\ \alpha_j$

  $S \leftarrow S \cup \{j_{\max}\}$ (rajout de $j_{\max}$ aux indices retenus)

  $\boldsymbol{\theta}_{\text{int}} \leftarrow \underset{\theta \in \mathbb{R}^{|S|}}{\arg\min} \|\mathbf{y} - X_S \theta\|^2$

  $r \leftarrow \mathbf{y} - X_S \boldsymbol{\theta}_{\text{int}}$

$\boldsymbol{\theta}_S = \boldsymbol{\theta}_{\text{int}}$

Output : $\boldsymbol{\theta} \in \mathbb{R}^p$, $S \subset [\![1, p]\!]$

11) Créer une classe `MYOMP` qui implémente `stpforward`, en partant de l'exemple :

```
class MYOMP(LinearModel, RegressorMixin):
    """Orthogonal Matching Pursuit model (OMP).

    Parameters
    ----------
    n_nonzero_coefs : int, optional
        Desired number of non-zero entries in the solution. If None (by
        default) this value is set to 10% of n_features.
    """

    def __init__(self, n_nonzero_coefs=None, fit_intercept=False,
                 normalize=False, precompute='auto'):
        self.fit_intercept = False
        self.normalize = normalize
        self.precompute = precompute
        self.n_nonzero_coefs = n_nonzero_coefs

    def fit(self, X, y):
        """Fit the model using X, y as training data.

        Parameters
        ----------
        X : array-like, shape (n_samples, n_features)
        Training data.
        y : array-like, shape (n_samples,) or (n_samples, n_targets)
        Target values.
        Returns
        -------
        self : object
        returns an instance of self.
        """
        self.coef_ = np.zeros([X.shape[1], ])
        # XXX: MODIFY HERE !!!
        self.intercept_ = 0.
        return self
```

12) Appliquer `MYOMP` au jeu de données, pour $M = 1, 2, 3, 4, 5$.

13) Comparer votre sortie avec celle de `OrthogonalMatchingPursuit` de `sklearn`.

14) Utiliser une validation croisée (avec 3 folds) pour choisir le nombre de variables à garder, en prenant comme critère de performance l'erreur quadratique (moyennée sur les folds). Faut-il en garder 1, 2, 3, 4 ou bien 5 variables sur cet exemple ? On veillera à utiliser la fonction `GridSearchCV`.

15) Afficher sur un graphique l'erreur quadratique (moyennée sur les folds) en fonction de M. Ajoutez-y la valeur de l'optimal sélectionnée ainsi qu'un un intervalle de confiance à + ou - un écart-type de cette erreur.