TP introductif en Big Data

Saïda Guezoui, groupe DS

Octobre 2021

1 Introduction

Le Big Data prend aujourd'hui une place importante dans le marché des entreprises et il est de plus en plus utilisé.

Le bootstrap est un algorithme permettant l'obtention de plusieurs échantillons à partir de notre échantillon origine. Cet algorithme consiste à faire un réechantillonage avec remise d'un jeu de donnée pour avoir des échantillons statistiques qui ont la même taille que notre échantillon initial.

Il existe deux types de bootstrap, paramétrique et non paramétrique.

Dans ce TP, nous étudirons ces différents bosstrap à l'aide des bibliothèques et modules en Python suivant

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import seaborn as sns
```

Listing 1: Modules Python3

2 Etude du jeu de donnée "*student_score*"

2.1 Importation du jeu de données et calcul de la matrice de correlation

```
# Importation des donnees du fichier txt
score = np.genfromtxt(fname = "student_score.txt")

# Creation d'un DataFrame
score_student  = pd.DataFrame(score,columns = ['mech' , 'vecs', 'alg', 'analy', '
    stat'])

# On supprime la premiere ligne qui ne contient que des NaN
score_student = score_student.drop(data.index[0])
print("Donnees : \n",score_student)

# Calcul des correlations entre les variables
correlation = score_student.corr()
print("\n Matrice de corellation :")
sns.heatmap(data = correlation, annot=True)
```

2.2 Estimation du paramètre θ

```
# Calcul des valeurs et vecteurs propres de la matrice des correlations
v, vp = np.linalg.eig(correlation)

# Calcul de Theta
theta = max(v)/ sum(v)
print("\n Theta = ", theta)
```

La valeur de θ est d'environ 0.7.

2.3 Définition de la fonction boostrap

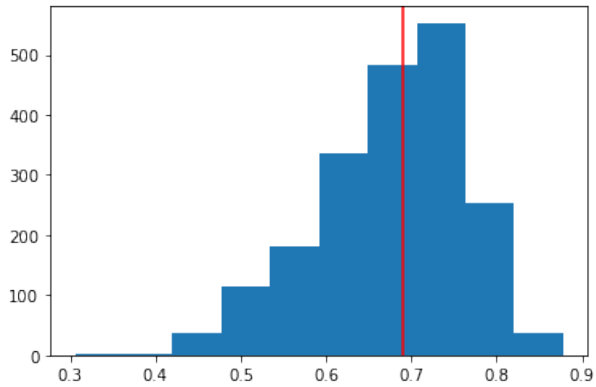
Nous définissons la fonction boostrap comme une fonction qui prend en paramètre un data donné et nombre B fixé à 2000 dans cette partie. On cherche à obtenir les valeurs de notre vecteur $\hat{\theta}$, nous faisons donc un réechantillage avec remise sur notre jeu de données, puis sur la matrice de correlation. Ensuite, nous appliquons la formule de θ suivante pour avoir notre liste des $\hat{\theta}$:

$$\theta = \frac{\lambda_{max}}{\sum(\lambda)}$$

Avec λ_{max} est la valeur propre maximale.En appliquant cette fonction au jeu de données "*score_student*", nous obtenons alors un vecteur de $\hat{\theta}$ de taille 200.

```
def boostrap(data,B):
    theta_list = []

    for _ in range(B):
        # Echantillon tire et replace = True signifie un tirage avec remise
        sample = data.sample(n = len(data.index), replace = True)

        correlation = sample.corr()

        eigenvalue, eigenvector = np.linalg.eig(correlation)

        theta = max(eigenvalue)/sum(eigenvalue)
        theta_list.append(theta)

    return theta_list

theta_list = boostrap(score_student,B = 2000)
```

2.4 Représentation sous forme d'histogramme



2.5 L'erreur quadratique moyenne

L'erreur quadratique moyenne est d'environ 0.086.

3 Etude du jeu de données "diabete"

Dans cette partie, on s'intéresse au jeu de données "diabete" contenant des informations sur diabète des indiennes Pima. Nous cherchons à prédire la survenue du diabète à l'aide de la régression logistique en utilisant les autres covariables.

3.1 Importation du jeu de données

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3.2 Création du Design, de la variable à prédire "Outcome"

3.3 Application du modèle de regression logistique

```
# Creation du modele
model_pima = LogisticRegression(max_iter = 300)

# Apprentissage
model_pima.fit(Design_pima, Y_pima)

# Test du modele (score)
print("Score : ", model_pima.score(Design_pima,Y_pima))

# Parametres du modele
intercept_pima = model_pima.intercept_
coefs_pima = model_pima.coef_
```

Définition de la fonction logit et du p.

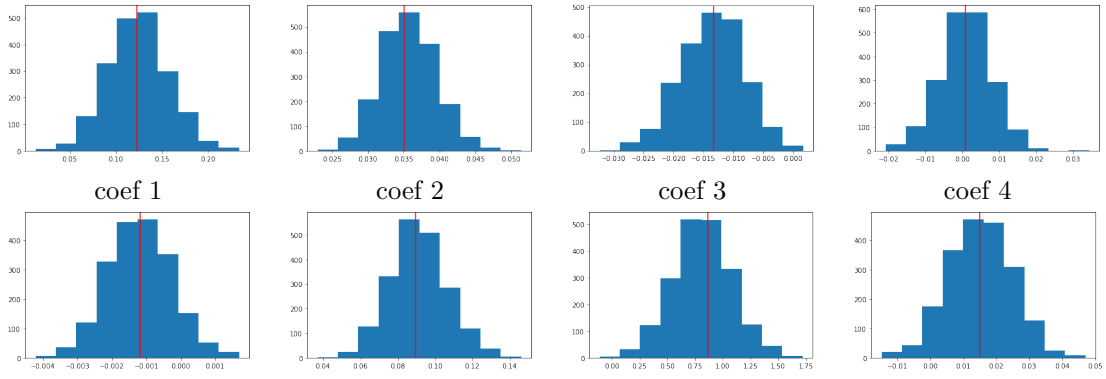
```
# Logit
def g(x):
    return np.exp(x) / (1 + np.exp(x))

# Calcul de p_hat
p_hat = g(intercept_pima + coefs_pima.dot(Design_pima.T))[0]
```

3.4 Application du bootstrap paramétrique

```
def bootstrap_para(B) :
    Coefs = []
    Intercepts = []

    for _ in range(B) :
        Y_pima_hat = np.random.binomial(n = 1, p = p_hat, size = len(p_hat))
        model_pima.fit(Design_pima, Y_pima_hat)
        Coefs.append(model_pima.coef_[0])
        Intercepts.append(float(model_pima.intercept_))

    return Coefs, Intercepts
```

```
12
13 vect_coef, liste_intercepts = bootstrap_para(2000)
```

Affichage des distributions des coefficients sous forme d'histogrammes



coef 1 coef 2 coef 3 coef 4

coef 5 coef 6 coef 7 coef 8

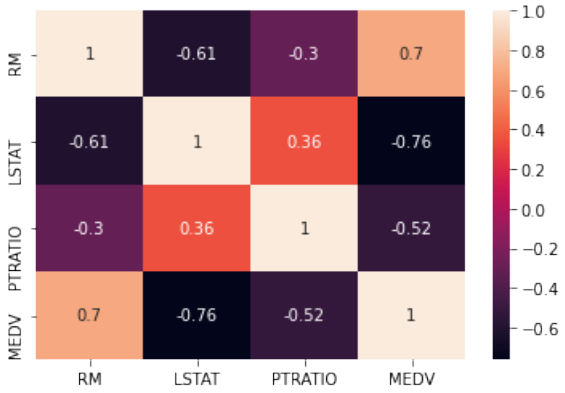
4 Etude du jeu de données "housing"

Nous intéressons cette fois-ci aux prix des logements dans différents quartiers de Boston. La variable en question est "MEDV". Nous cherchons à prédire ce prix en fonction des autres covariables lstat, ptratio et rm en utilisant un modèle de régression linéaire et un algorithme de bootstrap non paramétrique.

4.1 Importation du jeu de données housing et calcul de la matrice de corélation

```
1 housing = pd.read_csv("housing.csv")
2 housing  = pd.DataFrame(housing,columns = ['RM' , 'LSTAT', 'PTRATIO', 'MEDV'])
3 print(housing)
4
5 # Suppression de la premiere colonne d'indice
6 housing = housing.drop(housing.index[0])
7
8 # Etude de la correlation
9 correlation = housing.corr().round(2)
10 sns.heatmap(data = correlation, annot=True)
```

	RM	LSTAT	PTRATIO	MEDV
0	6.575	4.98	15.3	504000.0
1	6.421	9.14	17.8	453600.0
2	7.185	4.03	17.8	728700.0
3	6.998	2.94	18.7	701400.0
4	7.147	5.33	18.7	760200.0



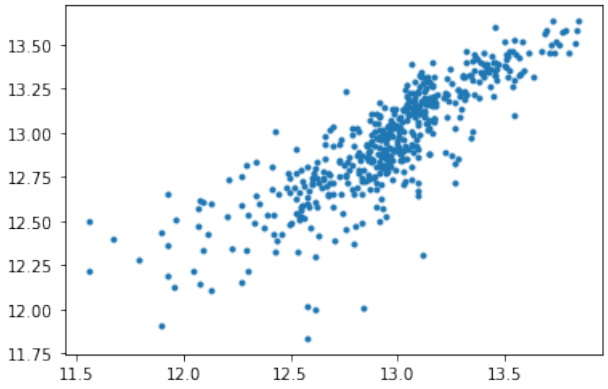
Nous constatons que la variable MEDV est corélée positivement avec la variable RM et négativement avec les variables LSTAT et PTRATIO. Nous pouvons donc appliquer un modèle de régression linéaire.

4.2 Création du Design et de la variable à prédire log(MEDV)

```
# Variable a predire Y = log(MEDV)
Y_housing = np.log(housing['MEDV'])
# Design
Design_housing = housing.drop('MEDV',1)
```

4.3 Application de la régression linéaire

```
REG_housing = LinearRegression()
# Entrainement du modele
Results = REG_housing.fit(Design_housing, Y_housing)

# Coefs et intercept
Coefs_housing = REG_housing.coef_
Intercept_housing = REG_housing.intercept_

# Affichage
print("Les coeficients de la regression lineaire : \n", Coefs_housing)
print("\nL'intercept de la regression lineaire : \n", Intercept_housing)

# Nuage de points de la regression
plt.plot(Y_housing, REG_housing.predict(Design_housing),'.')
plt.show()
```



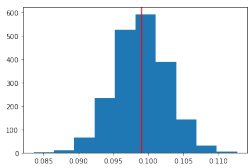
4.4 Calcul du σ^2 et ϵ

```
RMSE = np.sqrt(((Y_housing - REG_housing.predict(Design_housing))**2).sum()/len(
    Y_housing))
print(RMSE)

Epsilon = Y_housing - Intercept_housing - Coefs_housing.dot(Design_housing.T)
print(Epsilon)
```

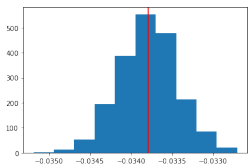
4.5 Application du bootstrap non paramétrique

```
def bootstrap_non_para(B) :
    Coefs = []
    Intercepts = []

    for _ in range(B) :
        Epsilon_star = np.random.normal(0, RMSE**2, len(Epsilon))
        Y_housing_hat = Intercept_housing + Coefs_housing.dot(Design_housing.T) +
    Epsilon_star
        REG_housing.fit(Design_housing, Y_housing_hat)
        Coefs.append(REG_housing.coef_)
        Intercepts.append(float(REG_housing.intercept_))

    return Coefs, Intercepts

Coefs_h, Intercepts_h = bootstrap_non_para(2000)
```

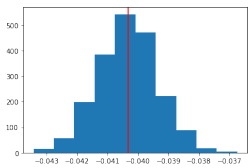
Représentation des distributions des intercepts et coefs sous forme d'histogrammes



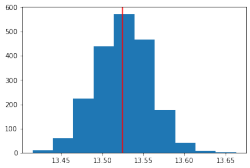
coef 1



coef 2



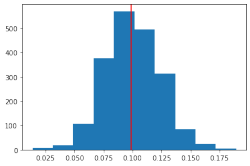
coef 3



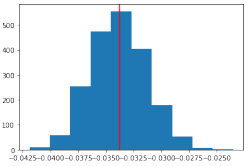
intercept

```
def boostrap(data,B):
    coefs = []
    intercepts = []

    REG_housing = LinearRegression()

    for _ in range(B):
        sample = data.sample(n = len(data.index), replace = True)

        X = sample.drop('MEDV',1)
        y = np.log(sample['MEDV'])

        REG_housing.fit(X, y)

        coefs.append(REG_housing.coef_)
        intercepts.append(REG_housing.intercept_)

    return coefs, intercepts

coefs_housing, intercepts_housing = boostrap(housing,B = 2000)
```

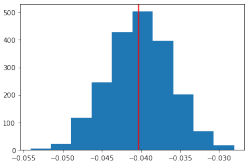
Représentation des distributions des intercepts et coefs sous forme d'histogrammes



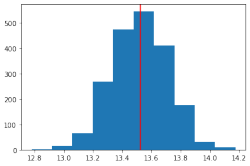
coef 1



coef 2



coef 3



intercept