TP3_Regression_Lineaire

March 14, 2023

1 TP3 : régression linéaire

[ ]:
```
import  numpy as np

import matplotlib.pyplot as plt
```

[ ]:
```
# un exemple simple x réel et y aussi

rng = np.random.RandomState(42) #pour générer les mêmes données

#constituer  un exmple de points aléatoires
x = 10 * rng.rand(50)
print('la taille de notre échantillon est :',x.shape)

y=2*x-1 + rng.randn(50) # définir une relation entre x et y + bruit

#afficher data y=f(x) [y en fonction de x]  comme un nuage de points
plt.scatter(x, y);
```

1.1 Y-a-t-il une relation entre x et y : trouver f tel que $y = f(x)$?

Pour répondre à cette question, nous allons supposer que f est une fonction affine de la forme :

$$f(x) = a * x + b$$

avec a et b sont des inconnus (réels) à déterminer.

On connaît (x_i, y_i) pour $i = 1 \dots 50$

Et la relation $y_i = a * x_i + b$, pour $i = 1 \dots 50$

qui forme un système linéaire facile à résoudre (plus de données que d'inconnus)

Pour résumer : le but est de trouver la droite "la plus proche" de l'ensemble (nuage) de points. Un bon critère pour vérifier "la plus proche" est minimiser l'erreur quadratique moyenne :

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - a * x_i - b)^2$$

Après calcul on trouve les valeurs optimales :

$\hat{a} = \frac{\sigma_{xy}}{\sigma_x^2}$ et $\hat{b} = \bar{y_n} - \bar{x_n} * \frac{\sigma_{xy}}{\sigma_x^2}$ avec :

- $\bar{y_n} = \frac{1}{n}\sum_{i=1}^{n} y_i$
- $\bar{x_n} = \frac{1}{n}\sum_{i=1}^{n} x_i$
- $\sigma_y = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_n})^2$
- $\sigma_x = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_n})^2$
- $\sigma_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_n})(y_i - \bar{y_n})$

Au lieu de faire le calcul à la main, nous allons utiliser des bibliothèques python pour trouver la solution.

1.1.1 Exemple1 : Formulation avec la biblio sklearn

[ ]:
```
# On peut résoudre ce problème de régression linéaire avec sklearn
# on choisit et on charge le modèle
from sklearn.linear_model import LinearRegression

X = x[:, np.newaxis]
print('la tailles des entrées est :',X.shape)

models = LinearRegression(fit_intercept=True)
models.fit(X, y)
```

[ ]:
```
a=models.coef_
print('-'*5,'la solution','-'*5)
print('la valeur trouvée de a est : ', a[0])

b=models.intercept_
print('la valeur trouvée de b est : ', b)
```

Si maintenat on a un nouveau $xnew = 2.5$ qui est différent de tous les x_i observés on peut trouver son image $ynew$ avec la relation :

$$ynew = a * xnew + b$$

[ ]:
```
#solution pour un seul point
xnew=np.array([2.50])
ynew = models.predict(xnew.reshape(-1, 1))
print(ynew)
```

On peut aussi appliquer la même méthode sur xnew comme tableau de valeurs au lieu d'un seul scalaire

[ ]:
```
#solution pour un tableau de points
xnew=np.linspace(-1,12,10)
#s'assurer d'avoir le bon format
```

```
xnew=xnew[:, np.newaxis]

ynew = models.predict(xnew)
print(ynew)
```

Vérification visuelle

[ ]:
```
plt.scatter(x, y,color='k');# données apprentissage en noir
plt.scatter(xnew, np.zeros(xnew.shape[0]),color='b');# x_i non observés en bleu
plt.scatter(xnew, ynew,color='r', marker='*');# y_i prédit ave la régression␣
 ↪linéaire (x_i,y_i) en rouge
```

[ ]:
```
#on peut aussi afficher la fonction f
plt.scatter(x, y,color='k');
#plt.scatter(xnew, ynew);
plt.plot(xnew, ynew,'r');
#l'erreur est donnée par la somme cumulée des distances
#entre les points en noir et la droite en rouge

ypred=models.predict(X)
print(ypred.shape)
print('Biais ou erreur en chaque point : \n')
plt.figure()
plt.plot(x, (y-ypred), 'g*')
plt.show()
print('L\'erreur globale peut être donnée l\'erreur quadratique moyenne : ',np.
 ↪mean((y-ypred)**2))
```

1.2 Nous venons de faire notre premier exemple pour le cas simple $ x$ réel et y réel

On peut généraliser ce résultat quelque soit la taille de $x : x \in R^d$ et pour toute dimension finie d

Exemple :

[ ]:
```
from mpl_toolkits.mplot3d import Axes3D
#constituer  un exmple de data
x = np.array(10 * rng.rand(100,2))
y=2*np.inner(np.array([-1,1]), x)+ 2*rng.randn(x.shape[0])

fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x[:,0], x[:,1],y,c='b', marker='o');
ax.set_xlabel('valeur de x[:,0]')
ax.set_ylabel('aleur de  x[:,1]')
ax.set_zlabel('valeur de y ')
```

```
plt.show()
```

[ ]:
```
model = LinearRegression(fit_intercept=True)
model.fit(x, y)
```

[ ]:
```
xnew = np.array(10 * rng.rand(1000,2))
ynew = model.predict(xnew)
```

[ ]:
```
fig=plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x[:,0], x[:,1],y,c='b', marker='o');
ax.scatter(x[:,0], x[:,1],-y,c='b', marker='o');
ax.set_xlabel('valeur de x[:,0]')
ax.set_ylabel('aleur de  x[:,1]')
ax.set_zlabel('valeur de y ')

ax.scatter(xnew[:,0], xnew[:,1],ynew,c='r', marker='*');

plt.show()
```

1.2.1 Exercice TD2 :

Reprendre l'exercice de la régression linéaire dans le TD2.

1- Faire un programme python qui répond aux questions.

2- Comparer les résultats théoriques vu en TD et les résultats par le code.

3- Prédire le taux de scolarisation pour des PIBs non observés.

4- Suivre les mêmes étapes que l'exemple 1 au dessus pour visualiser et inerpréter le biais.

5- Commenter la qualité du modèle : bon ou mauvais ? Pourquoi ?

1.2.2 Un test de la régression logistique sur des données réelles

1.3 C'est un cas particulier où $y \in \{0, \dots, k\}$

[ ]:
```
#importer les bibliothèques
#pour l'affichage (si déjà fait pour np, plt)
%matplotlib inline

#charger des datasest de sklearn
from sklearn import datasets
```

[ ]:
```
#charger la base iris
iris = datasets.load_iris()
#vérifier le type de la variable iris
print(type(iris))
```

```
#vérifier le type de données
print(type(iris.data))
#vérifier les dimensions
print(iris.data.shape)

#Sur wikipédia chercher la signification de ces données
```

[ ]:
```
X = iris.data[:, :2] # Utiliser les deux premières colonnes afin d'avoir un␣
 ↪problème de classification binaire.

print(np.unique(iris.target))
#on va garder deux classes seulement pour un test simple
y = (iris.target != 0) * 1 # re-étiquetage des fleurs
print(X.shape)
print(np.unique(y))
```

[ ]:
```
#visualisation des données
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='classe 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='classe 1')
plt.legend();
```

[ ]:
```
#charger le modèle pour y binaire
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(C=1e20) #  Régression logistique
# Entrainement du modèle avec toutes les données
model.fit(X, y)
```

[ ]:
```
Xnew = np.array([
    [5.5, 2.5],
    [7, 3],
    [3,2],
    [5,3]
])

model.predict(Xnew)
```

Analyse des résultats :

- La première observation [5.5, 2.5] pour $y = 1$
- La deuxième observation [7, 3] pour $y = 1$
- La troisième observation [3,2] pour $y = 0$
- La quatrième observation [5,3] pour $y = 0$

[ ]:
```
#vérification visuelle

#visualisation des données
```

```
plt.figure(figsize=(10, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='y= 0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='y= 1')

s = np.random.rand(*Xnew[:, 0].shape) * 800 + 500
print(s.shape)
Color='kygm' #noir jaune vert magneta
for i in range(Xnew.shape[0]):
    plt.scatter(Xnew[i, 0], Xnew[i, 1],s[i],␣
 ↪color=Color[i],marker=r'$\clubsuit$',)
plt.legend();
```

2 Rappel de votre cours de BD

2.1 Pandas : un moyen efficace pour lire et manipuler des données

[ ]:
```
# comme n'importe quelle librarire, il faut commencer par la charger à l'aide␣
 ↪de la commande import
import pandas
# maintenat que c'est fait on peut utiliser son contenu
# par exemple :vérifier la version installée sur votre machine
pandas.__version__
```

[ ]:
```
# et si on lui donne un nom pour faciliter les appels
import pandas as pd
pd.__version__
```

2.2 L'objet DataFrame

La DataFrame est un objet bi-dimentionnel avec des colonnes de types potentiellement différents. On peut voir la DataFrame comme une feuille Exce ou une table SQL.

[ ]:
```
# Lecture d'un fichier de données et le récupérer sous forme de dataframe sous␣
 ↪le nom df
df = pd.read_csv("Prix_Appartements.csv") # à partir d'un csv
df.head(5)
```

[ ]:
```
## On peut afficher les dimensions (nombre de lignes et de colonnes) ## avec␣
 ↪l'attibut shape (comme avec numpy)
print('la taille :',df.shape) ## (nb lignes, nb colonnes) print('*'*40)
print('Avec :',df.shape[0],' lignes') ## (nb lignes, nb colonnes) print('*'*40)
print('Avec :',df.shape[1],' colonnes') ## (nb lignes, nb colonnes)␣
 ↪print('*'*40)
```

[ ]:
```
## La commande df.head(n) permet d'afficher uniquement les n premiers éléments␣
 ↪# car la taille de la dataframe est grande avec 4622 lignes
df.head(6) # les 6 premières lignes de 0 à 5 = 6-1
```

[ ]:
```
## De même df.tail(n) affiche les n=3 derniers éléments
df.tail(3)
```

[ ]:
```
# La commande describe() est très utile. Elle permet d'obtenir, en une seule␣
↪commande,
# des statistiques des colonnes (UNIQUEMENT pour les colonnes de type numérique)

df.describe()
```

2.2.1 Exercice : Use case

2.3 dataset dans le fichier Prix_Appartements.csv

1- Avec la commande pd.read_csv ouvrir le fichier csv dans une dataframe df_1.

2- Afficher les 7 premières lignes.

3- Afficher les noms des colonnes.

4- Créer une nouvelle dataframe ne contenant que l'âge et le prix.

5- Affichier les lignes d'index impair.

6- En utilisant la colonne prix, calculer le prix moyen p_m, la médiane k_m et l'écart-type σ_m.

7- Afficher les lignes dont le prix est supérieur au prix moyen p_m.

8- Choisir deux colonnes qui représentront les variables X et Y. Par exemple la distance au métro le plus proche et le prix au mètre carré. Pouvez les récupérer dans deux tableaux numpy.

9- Appliquer une régression linéaire pour vérifier la corrélation entre X et Y à l'aide des graphiques comme dans l'exircice du TD2.

10- Quelle variable (colonne) est la plus corrélée avec le prix ?

11- Pour éviter à ce que les grandes valeurs dominent les petites, on peut normaliser en divisant chaque colonne par le maximum en valeur absolue. Reprendre la question 10 avec des valeurs normalisées.

11- Vérifier la corrélation entre le prix et deux voire plusieurs colonnes. Conclure.

[ ]:
```

```