Objectifs du TP :
- Utiliser `scikit-learn` pour réaliser des régressions linéaires multivariées sur données réelles
- Comprendre en pratique les mesures de qualité d'un régresseur linéaire
- Comprendre le rôle de la standardisation en régression linéaire
- Implémenter et analyser l'algorithme de régression linéaire par la méthode des moindres carrés

1 La régression linéaire par moindres carrés sous `sklearn`

À partir des données $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, l'objectif est d'apprendre un vecteur $\mathbf{w} = w_0, w_1, \ldots, w_d$ tel que $y_i \approx w_0 + \langle w_{1:d}, \mathbf{x}_i \rangle$ pour tout $(\mathbf{x}_i, y_i) \in S$ où la notation $\langle \mathbf{u}, \mathbf{v} \rangle := \sum_{j=1}^d u_j v_j$ désigne le produit scalaire entre les deux vecteurs $\mathbf{u}$ et $\mathbf{v}$.
L'algorithme de régression linéaire par la méthode des moindres carrés cherche à minimiser le risque empirique de l'échantillon d'apprentissage, c'est-à-dire à résoudre le problème d'optimisation suivant :

$$\text{Trouver le vecteur } \mathbf{w} \in \mathbb{R}^{\mathbf{d+1}} \text{ qui minimise } \sum_{i=1}^{n} \big(y_i - w_0 - \langle \mathbf{w}_{1:d}, x_i \rangle\big)^2.$$

Une forme analytique de la solution de ce problème est donnée par la formule suivante :

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y},$$

où $\mathbf{X} \in \mathbb{R}^{\mathbf{n} \times (\mathbf{d+1})}$ est la matrice contenant les données $(x_i)_{i=1}^n$, complétée par une première colonne de 1, et où $\mathbf{y} \in \mathbb{R}^n$ est le vecteur $(y_1, \ldots, y_n)^\top$.

1.1 Etude d'un régresseur linéaire simple

La bibliothèque `sklearn` contient une fonction permettant de calculer la fonction de régression linéaire avec la méthodes des moindres carrés. Il s'agit de la fonction `LinearRegression()` du package `linear_model`.

1. Consulter la documentation en ligne `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression`
2. Utiliser cette fonction de régression sur toutes les données du tableau ci-après (x = prix de vente d'un produit, et y = nombre de produits vendus) pour calculer la droite de régression avec le paramétrage par défaut, et afficher l'erreur quadratique moyenne calculée sur l'échantillon d'apprentissage (via la fonction `mean_squared_error` du package `metrics`).

x	5.5	6.0	6.5	6.0	5.0	6.5	4.5	5
y	420	380	350	400	440	380	450	420

3. Que représentent les attributes `coef_` et `intercept_` ?
4. Afficher le score (coefficient de détermination) du prédicteur sur l'échantillon d'apprentissage : que représente ce score, est-il bon sur ce régresseur ?
5. Utiliser ces deux attributs pour afficher en rouge la droite apprise (avec la fonction `plot` de `matplotlib.pyplot`) ; sur le même graphique, afficher aussi la droite de régression en vert telle qu'apprise par l'algorithme, le tout avec le nuage des points de l'échantillon affichés aussi (pour un nuage de points, on utilise la fonction `scatter` de `pyplot`). Si tout va bien, les droites rouge et verte doivent coincider.
6. Refaire cet apprentissage + affichage des paramètres appris + affichage des résultats, mais avec l'option `fit_intercept` à `False`. Qu'observez-vous ? Comment expliquez-vous cela ?

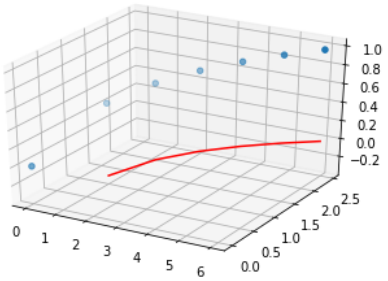
1.2 Jeu de données Eucalyptus

Le fichier `eucalypus.txt` contient des données sur des arbres appelés eucalyptus, où sont indiquées hauteurs et circonférences de centaines d'entre eux. L'objectif est d'étudier l'explication de la variable `hauteur` (h) par la variable `circonférence` (c).

1. Récupérer ces données dans deux tableaux numpy appelé, h et c. On utilisera pour se faire la fonction `numpy.loadtxt(nomfichier)` qui lit un tableau depuis le fichier texte en paramètre. Dans le fichier `eucalyptus.txt`, la première colonne correspond à la hauteur, et la seconde à la circonférence. La fonction `loadtxt` va ainsi retourner un tableau dont la première colonne sera extraite pour obtenir la cible (h), et la seconde colonne sera extraite pour obtenir l'explication c qui ne contient qu'une seule variable.
2. Afficher le nuage de points ces données sur un graphique : est-ce qu'une corrélation linéaire semble présente ?
3. Utiliser la fonction sklearn précédemment étudiée pour expliquer h par c via une droite, et afficher le résultat (la droite et les points d'apprentissage) sur un graphique. Indiquer les performances (coefficient de détermination) d'une telle régression linéaire estimée par validation croisée. Les résultats sont-ils satisfaisants ? Quelle est la valeur de hauteur prédite pour l'eucalyptus de circonférence 22.8 ?
4. Essayons un autre modèle de regression : au lieu de chercher w_0 et w_1 tels que $h = f(c) = w_0 + w_1 c$, nous allons chercher f telle que $f(c) = w_0 + w_1 c + w_2\sqrt{c}$, autrement dit nous ajoutons au jeu de données une variable qui est dépendante de c mais pas linéairement (la fct $\sqrt{-}$ est une transformation non linéaire de son argument). Compléter le jeu de données avec cette nouvelle colonne (variable) : on obtient un problème de régression multi-varié. Expliquer h par c et $\sqrt{c}$ via l'apprentissage d'un régresseur : reproduire les expérimentations précédentes (apprentissage, visualisation, estimation de la MSE par validation croisée), et conclure : obtient-on un meilleur modèle ?

   Pour afficher un plot ou un scatter (nuage de points) en 3D, l'extrait de code ci-après produit la figure en face.

```
from mpl_toolkits.mplot3d import Axes3D
a = np.array([0,1,2,3,4,5,6])
b=  np.sqrt(a)
c = np.log10(a+b+1)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(a,b,c)
ax.plot(a,b,np.log10(c),'r')
```



5. Nous pouvons tester beaucoup d'autres modèles de régression linéaire en suivant le même principe de transformation non-linéaire d'une variable pour en créer d'autres, par exemple : $h = w_0 + w_1 c + w_2\sqrt{c} + w_3 c^2$. Do it (sans visualisation).
6. In fine, quel est le meilleur modèle obtenu (selon les scores obtenus par validation croisée) ?

2 Programmation de la régression linéaire par moindres carrés (optionnel)

L'algorithme de régression par la méthode des moindres carrés, dans sa version standard, est un algorithme de régression *linéaire*. Les données d'apprentissage utilisées dans cette section s'écrivent sous la forme $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ avec $\mathbf{x}_i \in \mathbb{R}^d$ (d est le nombre d'attributs des données d'entrée) et $y_i \in \mathbb{R}$.

Algorithme Régression linéaire par moindres carrés

Entrée : une liste S de données d'apprentissage, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ où $\mathbf{x}_i \in \mathbb{R}^d$ et $y_i \in \mathbb{R}$,
Sortie : le vecteur de pondération w.

---

1. Ajouter le vecteur $\mathbf{1}$ à la matrice $\mathbf{X} \in \mathbb{R}^{\mathbf{n}\times\mathbf{d}}$ contenant les données $(\mathbf{x}_i)_{i=1}^n$,
2. Calculer $\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$, où $\mathbf{y} = (y_1, \ldots, y_n)^\top$
3. Retourner $\mathbf{w}$.

---

1. Implémenter l'algorithme de régression linéaire par moindres carrés décrit ci-dessus. Vous pourrez avoir à utiliser les commandes rappelées ci-dessous.

```
import numpy as np
import matplotlib.pyplot as plt

# commandes utiles
X = np.zeros([5,3]) # tableau de 0 de dimension 5x3
Y = np.ones([3,2]) # tableau de 1 de dimension 5x3
v = np.ones(3) # vecteur contenant trois 1
X[1:4,:2] = Y # remplacement d'une partie du tableau X
X.shape # dimensions de X
np.dot(X,Y) # produit matriciel
np.dot(X,v) # produit de la matrice X et du vecteur v
np.transpose(X) # transposée de X
np.linalg.inv(Z) # inverse de Z
```

2. Tester cette programmation sur les données jouet de la section 1.1. Obtient-on la même droite de régression ?
3. Tester cette programmation aux données Eucalyptus initiales : obtient-on les mêmes résultats qu'avec sklearn (équation de droite) ?
4. Indiquer le score et la MSE estimés par train_test_split sur les données eucalyptus avec le programme maison.