Data Science

Chap 1 : Data Manipulation

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 7, 2022

"Ceux qui abandonnent ne gagnent jamais, ceux qui gagnent n'abandonnent jamais." Napoléon Hill

Table de matières

Objectifs du chapitre

Dans ce premier chapitre, nous allons travailler sur la manipulation des données en Data Science.

À la fin de ce chapitre, vous serez en mesure :

- de manipuler les arrays Numpy.
- d'accéder à une partie des données ou de les subdiviser.
- d'éffectuer les opérations sur les données.

Definition

Definition

La Data Science est une discipline incontournable à l'ère du Big Data. Son **but ultime** est de favoriser la **prise de décision basée sur les données**.

La Data Science va permettre de trouver un meilleur algorithme de prédiction pour la météo, les prix d'un produit ou encore la création d'un système de recommandation sur Netflix ou Amazon par exemple.

Un domaine multidisciplinaire

Pour le faire, la Data Science comporte les étapes suivantes :

- collecter les données (**data mining**)
- nettoyer les données (**data engineering**)
- réaliser une analyse exploratoire des données (**data analysis**)
- construire et évaluer des modèles de Machine Learning (**Machine Learning**)
- partager les résultats des étapes précédentes (**data visualization**)

Pourquoi le langage de programmation Python ?

Dans ce cours, nous allons utiliser Python, car c'est le langage de programmation le plus populaire en Data Science. IL est facile à prendre en main, dispose d'une grande communauté et s'intègre bien à d'autres cadres (les applications Web).

Ce cours **se concentre** sur **l'analyse exploratoire des données** avec les trois librairies fondamentales de Python : Numpy, Pandas et Matplotlib. Nous allons également **couvrir** la librairie de Machine Learning **scikit-learn**.

Plan du cours

Dans les prochains chapitres, nous allons prédire le prix des maisons avec la régression linéaire, utiliser un algorithme de classification des fleurs (iris) et chercher des clusters parmi une liste de vins.

Attention !

Il existe d'autres langages de programmation en Data Science tels que R parmi les plus populaires.

Numerical Data 1

Les données proviennent de plusieurs sources avec des formats différents (vidéo, audio, textes ou un mélange de format). Par conséquent, le **premier travail à faire en Data Science** est **de transformer ces données en Numpy arrays ou arrays** afin de les manipuler.

Soit la liste des notes des élèves dans en anglais :
notes = [15,17,7,8,10, 13,18,5,11]
Pour savoir le nombre d'élèves ayant eu une note inférieur à 10, il suffit d'itérer sur la liste, en comparant chaque élément avec 10, et en incrémentant de 1 un compteur lorsque la condition est évaluée à vraie.

Numerical Data 2

Example

```
notes = [15,17,7,8,10, 13,18,5,11]
compt = 0
for note in notes :
    if note < 10:
        compt+=1
print(compt)
```

Numpy or Numerical Python 1

Le résultat du précédent exemple est plus facile à trouver avec Numpy.

Example

```
notes = [15,17,7,8,10, 13,18,5,11]
import numpy as np
notes_tab =np.array(notes)
print(notes_tab < 10).sum()
```

L'instruction **import** permet **d'accéder aux fonctions et aux modules** à l'intérieur de Numpy.

Attention !

Les modules Python peuvent accéder au code d'un autre module en important le fichier ou la fonction grâce à l'instruction **import**.

Exercice d'application 1

Compléter le programme suivant afin de créer un numpy array

```
import numpy as np

y = [ 4, 5, 6, 5, 9 ]

tab = ..... (y)
```

Size (taille d'un array)

Les tableaux de type Numpy sont appelé les **ndarray (n-dimensional array)**. L'attribut **size** permet de donner **la longueur** du ndarray.

notes_tab.size

Attention !

Lorsqu'un array est créé dans Numpy, il est impossible de changer sa longueur. **size** est **l'équivalent de len** pour les list, str, les dict, etc.

Shape (dimension d'un array)

l'attribut **shape** permet de savoir **la dimension du ndarray**.

notes_tab.shape

Le resultat est un **tuple** (type de données non modifiable) contenant une unique valeur, indiquant qu'il y a une seule dimension. C'est donc un 1d array.

Reshape 1

Nous avons également les notes en français des mêmes élèves.
notes_francais = [14, 5, 6, 15, 9, 12, 13, 7, 10]

Il serait donc judicieux de les combiner de façon à avoir une matrice 2 * 45 où la première ligne contient les notes en anglais et la deuxième ligne, les notes en français. L'attribut **reshape()** permet de réaliser cette opération (voir la slide suivante).

Pour rappel, les notes d'anglais sont :
notes = [15, 17, 7, 8, 10, 13, 18, 5,11]

Reshape 2

Example

notes_ang_fr = notes + notes_francais
Je convertir la liste notes_ang_fr en numpy array
notesnd_ang_fr = np.array(notes_ang_fr)
notesnd_ang_fr.reshape((2,9))

Attention !

Numpy peut calculer pour nous le shape (la dimension) si nous indiquons -1 comme dimension inconnue.

Par exemple, si nous disposons d'un array arr de 2d de shape(3,5), arr.reshape(-1) va donner un 1d de shape(15,), tandis que arr.reshape(-1,3) va donner un 2d de shape(5,3).

Exercice d'application 2

Reshape d'un numpy array

Soit arr un 1d array de shape(28,), écrire un programme permettant de reshape arr en 2d array de shape(4,7).

Quelle est le résultat de arr.reshape(2,-1) ?

Data type d'un ndarray

Les Numpy arrays sont homogènes, c'est-à-dire que tous les éléments qu'ils contiennent sont du même type.

l'attribut **dtype** permet de **vérifier le type d'un nd**
arr.dtype

Si le premier élément de la liste est un float 15.0 au lieu de 15 :
notes = [15.0, 17, 7, 8, 10, 13, 18, 5,11]
Alors après avoir converti la liste en nd array, tous les nombres de la liste deviennent des floats.
notes_arr = np.array(notes)
notes
notes.dtype

Exercice d'application 3

Type d'un numpy array

Quel est le type de l'array suivant:

notes = [15, 17, 7, 8, 10, 13, 18, 5,11]

- double
- float
- integer

Selection d'une donnée spécifique

L'index des tableaux en Numpy commence à zéro comme pour les listes. Pour selctionner le deuxième élément d'un 1d ndarray, notes par exemple :
notes[1].

Pour un 2d ndarray, il y a deux axes, l'axe 0 et l'axe 1, correspondant respectivement aux lignes et aux colonnes.
tab = [[123,345,231,456], [12,16,34,10]]
tab correspond à un tableau à 2 dimensions, 2 lignes et 4 colonnes.
Lorsqu'on ecrit **tab[1,1]**, cela correspond **à la deuxième ligne et à la deuxième colonne, c'est-à-dire le nombre 16**.

Exercice d'application 4

Selection d'une valeur

Écrire un code permettant de selectionner la valeur 34 du tableau suivant :

tab = [[123,345,231,456], [12,16,34,10]]

Selection d'une plage (slicing) de données

Soit le nd array suivant :
tab = [[123,345,231,456], [12,16,34,10]]

Pour selectioner les deux premiers éléments de la première ligne, nous pouvons procéder comme suit :
tab[0,0:2] ou encore tab[0, :2]
Puisque les index commencent à 0, on peut omettre 0.

Pour selectionner toute la quatrième colonne, nous peut faire :
tab[:,3]

Attention !

Comme pour les listes en Python, la syntaxe est la même :
arr[start:stop:step]
Par défaut, le start=0, stop=size et le step=1.

Exercice d'application 5

Selection d'une plage de valeurs

Choisir parmi les propositions suivantes, laquelle permet de selectionner entièrement la 3éme colonne :

tab = [[123,345,231,456] [12,16,34,10]]

- tab[:,3]
- tab[3:,]
- tab[:,2]

Affectation ou modification d'une ou de plusieurs éléments d'un nd array

tab = [123,345,231,456]
Pour affecter la valeur 200 au deuxième élémentde tab :
tab[1] = 200
tab = [[123,345,231,456], [12,16,34,10]]
Pour modifier le 3éme élément de la première ligne :
tab[0,2] = 150
Nous pouvons également changer l'ensemble d'une ligne (la deuxième ligne) comme suit :
tab[1,:] = 90
Nous pouvons également modifier les valeurs de deux plages de données comme suit :
tab[:2,:2] = 40
Les deux premières valeurs de la première et de la deuxième ligne deviennent 40.

Exercice d'application 6

Affectation d'une valeur à l'ensemble d'une colonne

Écrire un code permettant d'affecter la valeur 10 à toute la deuxième colonne de tab :

tab = [[123,345,231,456], [12,16,34,10]]

Affectation d'un array à un array

Un **array** peut-être **affecté** à un **autre array** à partir du moment où l**eurs dimensions correspondent**.
tab = [[123,345,231,456], [12,16,34,10]]
tab[:,0] = [11, 23]

Nous pouvons également mettre à jour les données d'une plage comme suit :
new_tab = np.array([6,8],[32,21])
tab[:, 2:] = new_tab
Seules les deux dernières colonnes de chaque ligne se mettent à jour.

Exercice d'application 7

Affectation de plusieurs valeurs

Écrire un code permettant d'affecter :
new_tab = np.array([6,8],[32,21])
aux deux premières colonnes de tab :

tab = [[123,345,231,456], [12,16,34,10]]

Combinaison de deux arrays 1

arr1=np.array([123,345,231,456]) arr2= np.array([12,16,34,10])
Vous pouvez transposer les nd array de manière à avoir les lgnes en colonnes et les colonnes en lignes.
Après avoir redimensionner tab en (4,2), nous pouvons les **empiler horizontalement (par colonne)** grâce à la fonction **hstack(())** :

```
arr1 = arr1.reshape((4,1))
arr2 = arr2.reshape((4,1))
arr1 = np.hstack((arr1, arr2))
print(arr1.shape)
print(arr1)
```

Attention !

Les fonctions **reshape(( ))** et **hstack(( ))** ont deux parenthèses ouvrantes.

Combinaison de deux arrays 2

Pour combiner plus de deux arrays horizontalement, il vous suffit d'ajouter l'array supplémentaire dans la fonction hstack(()) comme suit :

arr1 = np.hstack((arr1, arr2, arr3))

Attention !

Vous ne pouvez utiliser la fonction hstack(()) que si les nd arrays ont le même nombre de lignes.

Exercice d'application 8

Merger verticalement deux nd arrays

arr1 = np.hstack((arr1, arr2, arr3))

Écrivez l'instruction permettant de **combiner verticalement** les arrays : arr1, arr2, arr3.

La fonction concatenate()

Vous utiliserez la fonction **concatenate()** pour **merger deux nd array en ligne ou verticalement** (axis=0). C'est **l'équivalent de la fonction vstack(())**.
Pour m**erger en colonne ou horizontalement**, il suffit de spécifier **axis = 1**. C'est **l'équivalent de la fonction hstack(())**.

```
arr1=np.array([123,345,231,456])
arr2= np.array([12,16,34,10])
arr1 = arr1.reshape((4,1))
arr2 = arr2.reshape((4,1))
Merger en colonne :
arr1 = np.concatenate((arr1, arr2), axis=1)
Merger en ligne :
arr1 = np.concatenate((arr1, arr2), axis=0)
print(arr1.shape)
print(arr1)
```

Exercice d'application 9

Merger horizontalement trois nd arrays

Complétez l'instruction suivante de manière à **merger horizontalement les 3 nd arrays**.

np.concatenate((arr1, arr2, arr3), axis = ...)

Opérations sur le nd arrays

arr1 = np.concatenate((arr1, arr2), axis=1)
Nous pouvons multiplier par deux les éléments de la première colonne comme suit :
arr1[:,0]*2

Attention !

D'autres opérations telles que l'addition, la soustraction, la division, l'exposant fonctionne de la même façon.

Exercice d'application 10

Division

Écrire une instruction permettant de diviser toute la première colonne par 2.

Les méthodes 1

arr1=np.array([123,345,231,456])
arr2= np.array([12,16,34,10])
L'instruction suivante permet de **calculer la somme de tous les éléments** d'un nd array :

```
arr1 = arr1.reshape((4,1))
arr2 = arr2.reshape((4,1))
arr1 = np.hstack((arr1, arr2))
arr1.sum()
```

Les méthodes 2

Pour calculer la somme de chaque ligne, qui n'est rien d'autres que la somme de chaque colonne, on peut faire :
arr1.sum(axis=0)
Le résultat (array([1155, 72])) est un array contenant la somme de chaque colonne.
Pour calculer somme de chaque colonne, c'est-à-dire la somme de chaque ligne, on utilise l'instruction :
arr1.sum(axis=1)
Le résultat est un array contenant la somme de chaque ligne :
array([135, 361, 265, 466])

Attention !

D'autres opérations telles que .min(), .max(), .mean() fonctionnent de la même façon.

Exercice d'application 11

Calculer la somme en ligne

Écrire une instruction permettant de calculer la somme de chaque ligne du na array arr1.

Comparaisons

Voici quelques opérateurs de comparaison : $<$, $>$, $<=$, $>=$, $==$, etc.
Pour filtrer uniquement les valeurs inférieures à 300 :
arr1=np.array([123,345,231,456])
arr1[:] ¡ 300
Le résultat(array([True, False, True, False])) est un tableau de booléen indiquant True pour les valeurs qui remplissent la condition et False dans le cas contraire.

Pour savoir combien éléements remplissent l a condition ,il suffit d'ajouter la fonction .sum() au tableau de booléen comme suit :
(arr1[:] $<$ 300).sum()

Exercice d'application 11

Calculer la somme en ligne

arr1=np.array([123,345,231,456,345])

Écrire une instruction permettant de calculer le nombre d'éléments égal à 345.

Masque (Mask) et sous-masque (subsetting)

Pour extraire un sous-ensemble de données remplissant une condition spécifique, on peut procéder comme suit :

arr1=np.array([123,345,231,456], [41, 55,12,15])

temp = arr1[:] < 300

temp.sum()

On a ainsi créé un array de booléen.

valinf = arr1[temp, 0]

On peut ainsi passer en premier argument temp, voir le shape du nouveau array valinf.

valinf.shape

print(valinf)

Le résultat est : [123 41]

On a ainsi récupérer le sous-ensemble tel que la première valeur est inférieure à 300.

Exercice d'application 12

Récupération d'un sous-ensemble

arr1=np.array([123,345,231,456], [41, 55,12,15])

Écrire un programme permettant de récupérer un array dont les éléments de la deuxième ligne de arr1 sont inférieurs à 50.

Sous-ensemble avec plusieurs conditions

Il est possible de créer un sous-ensemble satisfaisant plusieurs critères :

```
arr1=np.array([[123,345,231,456], [41, 55,12,15]])
temp = (arr1[0,:] ¡ 300) & (arr1[1,:] ¡ 17)
temp.sum()
valinf = arr1[:,temp]
valinf.shape
print(valinf)
```

Attention !

Les opérations de comparaison se font de façon simultanée (les deux en même temps).

Quiz 1

1) Créer un numpy array à partir de la liste y

y = [2, 3, 4]

2) Quelle est la shape du array suivant ?

y = np.array([2, 2, 2])

1. (1, 3)
2. (3, 1)
3. (3,)

Quiz 2

3) Quelle instruction permet de calculer la somme de chaque colonne de arr ?

1. arr.sum()
2. arr.sum(axis = 1)
3. arr.sum(axis = 0)

4) Écrire l'instruction permettant de calculer le minimum des colonnes de arr

arr = np.array([[123,345,231,456], [41, 55,12,15]])

Attention !

Utilisez la fonction min() et l'attribut axis.

Quiz 3

5) Laquelle des propositions permet de multiplier la deuxième ligne par 2

1. arr[1] * 2
2. arr[:, 1]*2
3. arr[1, :]*2

Data Science

Chap 2 : Data Analysis ≈ chap 2 : Pandas - Python for Data Science

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 8, 2022

"La motivation est ce qui vous fait démarrer. L'habitude est ce qui vous permet de continuer." Jim Rohn

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons travailler sur la structure des données en Data Science.

À la fin de ce chapitre, vous serez en mesure :

- de manipuler les DataFrames.
- d'accéder à une partie des données ou de les subdiviser.
- de claculer les statistiques descriptives.
- d'utiliser la fonction agg() avec le groupby

Numpy Vs Pandas

Contrairement à Numpy où les arrays sont homogènes, Pandas permet de manipuler des données de type différents.

Series

Les series Pandas sont des tableaux à une dimension avec un label, pouvant contenir n'importe quel type (entie, float, string, etc.).

Vous pouvez créer une série comme suit :

```
import pandas as pd
pd.Series('a':3, 'b':10)
```

DataFrames 1

Nous allons travailler sur les le fichiers mtcars (caractéristiques de modèles de voitures) de R que vous pouvez télécharger à cette adresse :
https://gist.github.com/seankross/a412dfbd88b3db70b74b.
Lorsqu'on importe un csv, on peut décider qu'une colonne (model) sera l'index comme suit :
dt = pd.read_csv("mtcars.csv", index_col ="model")
La dimension de notre DataFrame est :
dt.shape
Pour savoir le nombre de lignes :
dt.shape[0] = 32
Pour savoir le nombre de colonnes :
dt.shape[1] = 11
Pour le size (le nombre d'éléments de dt) :
32*11 = 352

DataFrames 2

head(n=3) = head(3)
Pour accéder à tous les noms des colonnes :
dt.columns
Pour accéder à une colonne spécifique :
dt['cyl']
dt['cyl'].shape
Le résultat est une série.
Pour accéder à plusieurs colonnes en même temps :
dt[['cyl','mpg']].head(5)
On a les 5 premiers éléments des colonnes cyl et mpg. Le résultat est un DataFrame.
On peut égalemetn accéder à plusieurs colonnes grâce à la fonction **loc()** en spécifiant la colonne de départ et la colonne de fin comme suit :
print(dt.loc[:, 'cyl':'qsec'].head(5))

Quantile(Quartile)

dt.quantile([0.25,0.5,0.75,1])

La médiane = Quartile(0.5)

Summary statistics = describe(). Cette fonction ignore les "NaN".

S'applique aussi sur les variables catégorielles.

la fonction agg()

Nous pouvons réaliser de nombreuses opérations avec le groupby et la fonction agg().

Calculons le minimum, la moyenne et le maximun de la colonne mpg (miles/gallon) regroupé par cyl (nombre de cylindres) :
print(dt.groupby('cyl')['mpg'].agg(['min', np.median, max]))

Nous pouvons également obtenir le min, mean et max de mpg et la médiane, mean de la cylindrée disp :
print(dt.groupby('cyl').agg('mpg': [min, np.mean, max],'disp':[np.median, np.mean]))

Data Science

Chap 3 : Data visualization ≈ chap 3 : Visualization des données - Python for Data Science

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 11, 2022

"Le seul endroit où le succès vient avant le travail, c'est dans le dictionnaire." Albert Einstein

Table de matières

Objectifs du chapitre

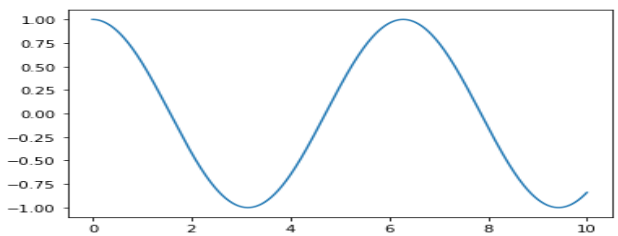
Dans ce chapitre, nous allons travailler sur la visualisation des données en Data Science.

À la fin de ce chapitre, vous serez en mesure :

- de construire des graphiques et de les customiser.

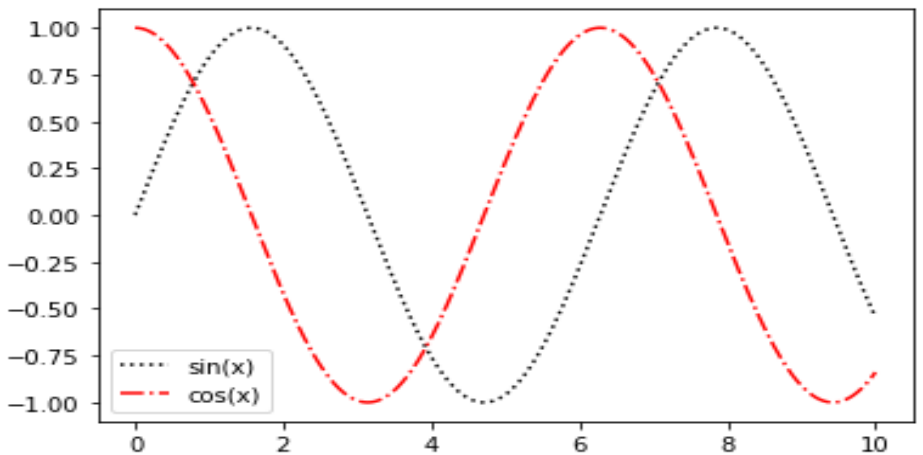
line 1

Pour créer un graphique, il faut commencer par créer une figure et des axes :
import matplotlib.pyplot as plt
x = np.linspace(0,10,1000) : mille nombres de 0 à 10
y = np.cos(x)
fig = plt.figure()
ax = plt.axes()
Pour voir le graphique :
ax.plot(x, y)
plt.show()
Le graphique est une courbe permet de voir les évolutions à court terme ou à long terme.



line 2

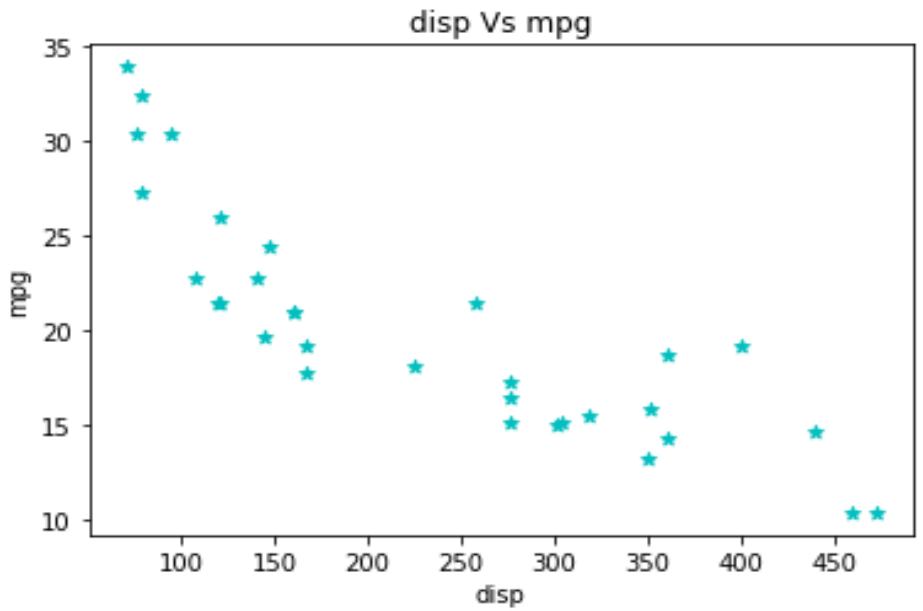
Nous pouvons customiser le graphique comme suit :
x = np.linspace(0,10,1000) : mille nombres de 0 à 10
plt.plot(x, np.sin(x), color='k', label='sin(x)', linestyle =':')
plt.plot(x, np.cos(x), color='r', label='cos(x)', linestyle ='-.')
plt.show()



k = couleur black confer
https://matplotlib.org/2.0.2/api/colors_api.html

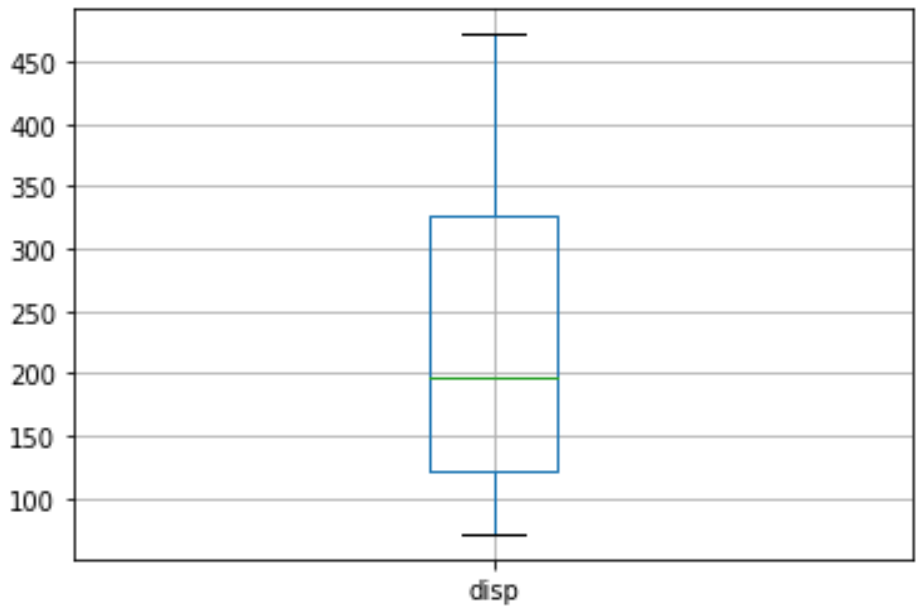
Scatter

```
plt.scatter(dt['disp'], dt['mpg'], marker ='*', color='c')
plt.xlabel('disp')
plt.ylabel('mpg')
plt.title('cyl Vs mpg')
plt.show()
```



Boxplot

dt.boxplot(column='disp')



La ligne verte représente le médiane.

Data Science

Chap 4 : Régression Linéaire

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 11, 2022

"Un problème sans solution est un problème mal posé." Albert Einstein

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons travailler sur la manipulation de la librairie Scikit-learn dédié à la Data Science en Python.

À la fin de ce chapitre, vous serez en mesure :

- de manipuler la librairie Scikit-learn
- de réaliser une analyse exploratoire des données
- d'analyser la matrice des corrélations
- d'entraîner un modèle de régression linéaire simple ou multivarié
- d'évaluer et de comparer les performances des modèles

Definition

Definition

La Data Science est une discipline incontournable à l'ère du Big Data. Son **but ultime** est de favoriser la **prise de décision basée sur les données**.

La Data Science va permettre de trouver un meilleur algorithme de prédiction pour la météo, les prix d'un produit ou encore la création d'un système de recommandation sur Netflix ou Amazon par exemple.

Apprentissage Supervisé et Apprentissage non Supervisé

Definition

On parle **d'apprentissage supervisé** lorsqu'on connait **les données cibles (également appelées étiquettes)** par exemple prédire le prix de vente du baril de pétrole.

En **apprentissage non supervisé**, nous ne disposons **pas de réponse antérieure connue**, par exemple déterminer la composition du sol de la planète Mars.

La régression et la classification sont des algorithmes d'apprentissage supervisé.

Scikit-learn

Scikit-Learn est l'une des librairies de Machine learning les plus connues en Python, permettant d'implementer de nombreux algorithmes.

La syntaxe est la suivante :

import $\Longrightarrow$ **instantiate** $\Longrightarrow$ **fit** $\Longrightarrow$ **predict**

Pour la suite, nous allons travailler avec le jeu de données portant sur 442 patients atteints de diabete (https://scikit-learn.org/stable/datasets/toy_dataset.html).

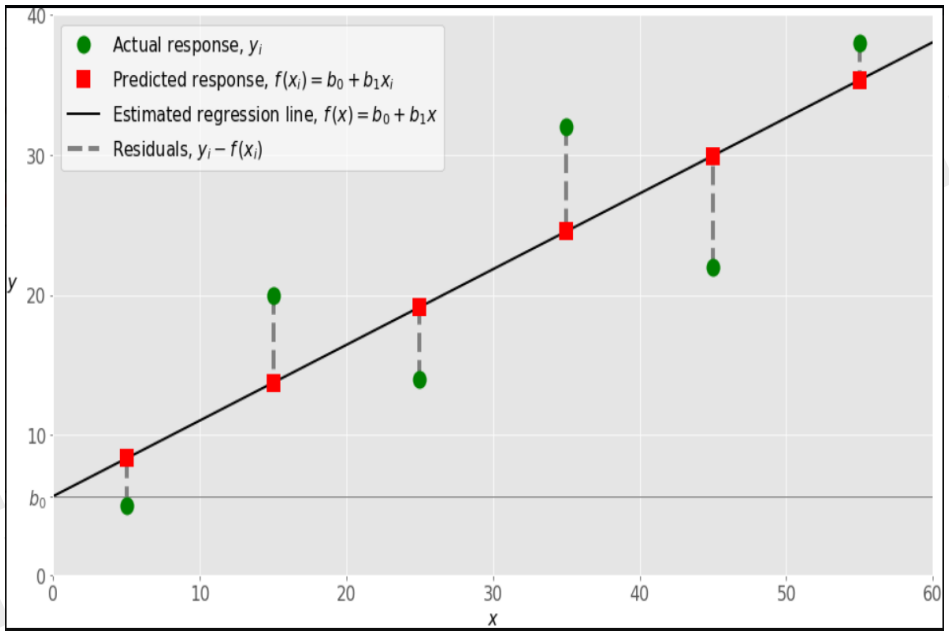
La régression linéaire 1

La régression linéaire permet **d'ajuster une droite aux données**. Elle cherche une relation entre les données. Cela se traduit mathématiquement par l'équation :

y = a + b*x, qui est l'équation d'une droite.
où a désigne l'ordonnée à l'origine (intercept), b la pente, x la variable explicative et y, la variable à prédire.

Notre mission consiste à trouver a et b de façon à minimiser les erreurs, qui est la distance entre la droite et les données.

La régression linéaire 2



Le régression linéaire cherche la droite qui minimises la somme des carrées des résidus. Elle est très utilisée pour prédire des outputs continus, car facile à implémenter.

La régression linéaire 2

```
import numpy as np
import pandas as pd

Importation et préparation de la base de données
from sklearn.datasets import load_diabetes
diabete_dataset = load_diabetes()
diabete = pd.DataFrame(diabete_dataset.data, columns =
diabete_dataset.feature_names)
diabete.head()
diabete.info()
diabete['prog'] = diabete_dataset.target
```

Exercice d'application 1

Nombre de lignes et de colonnes

Écrire un programme qui permet d'afficher le nombre de lignes et de colonnes du DataFrame diabete.

Vérifications sommaires

Une fois que le DataFrame est prêt, il est important de vérifier le type de chaque colonne grâce aux instructions telles que :

```
diabete.info()
diabete.head()
diabete. tail()
```

Statistiques descriptives 1

Nous pouvons également calculer les statistiques descriptives grâce à la fonction describe().

diabete.describe().round(2)

round(2) permet d'arrondir à deux décimales après la virgule les résultats pour une meilleure visibilité.

```
In [26]: diabete.describe().round(2)
Out[26]:
            age     sex     bmi      bp  ...      s4      s5      s6    prog
count  442.00  442.00  442.00  442.00  ...  442.00  442.00  442.00  442.00
mean    -0.00    0.00   -0.00    0.00  ...    0.00   -0.00   -0.00  152.13
std      0.05    0.05    0.05    0.05  ...    0.05    0.05    0.05   77.09
min     -0.11   -0.04   -0.09   -0.11  ...   -0.08   -0.13   -0.14   25.00
25%     -0.04   -0.04   -0.03   -0.04  ...   -0.04   -0.03   -0.03   87.00
50%      0.01   -0.04   -0.01   -0.01  ...   -0.00   -0.00   -0.00  140.50
75%      0.04    0.05    0.03    0.04  ...    0.03    0.03    0.03  211.50
max      0.11    0.05    0.17    0.13  ...    0.19    0.13    0.14  346.00
```

Statistiques descriptives 2

Dans le tableau précédent, toutes les colonnes ne sont pas affichées.
Pour remédier à cela, nous pouvons selectionner une plage de colonnes ou les colonnes qui nous intéressent comme suit :
diabete.loc[:,'age':'s2'].describe().round(2)

```
In [28]: diabete.loc[:,'age':'s2'].describe().round(2)
Out[28]:
```

	age	sex	bmi	bp	s1	s2
count	442.00	442.00	442.00	442.00	442.00	442.00
mean	-0.00	0.00	-0.00	0.00	-0.00	0.00
std	0.05	0.05	0.05	0.05	0.05	0.05
min	-0.11	-0.04	-0.09	-0.11	-0.13	-0.12
25%	-0.04	-0.04	-0.03	-0.04	-0.03	-0.03
50%	0.01	-0.04	-0.01	-0.01	-0.00	-0.00
75%	0.04	0.05	0.03	0.04	0.03	0.03
max	0.11	0.05	0.17	0.13	0.15	0.20

Nous avons selectionné les 6 premières colonnes.

Statistiques descriptives 3

Nous pouvons sélectionner le reste des colonnes comme suit :
diabete.loc[:,'s3':].describe().round(2)
diabete.loc[:,'s3':'prog'].describe().round(2)

```
In [30]: diabete.loc[:,'s3':].describe().round(2)
Out[30]:
             s3      s4      s5      s6    prog
count  442.00  442.00  442.00  442.00  442.00
mean    -0.00    0.00   -0.00   -0.00  152.13
std      0.05    0.05    0.05    0.05   77.09
min     -0.10   -0.08   -0.13   -0.14   25.00
25%     -0.04   -0.04   -0.03   -0.03   87.00
50%     -0.01   -0.00   -0.00   -0.00  140.50
75%      0.03    0.03    0.03    0.03  211.50
max      0.18    0.19    0.13    0.14  346.00
```

Exercice d'application 2

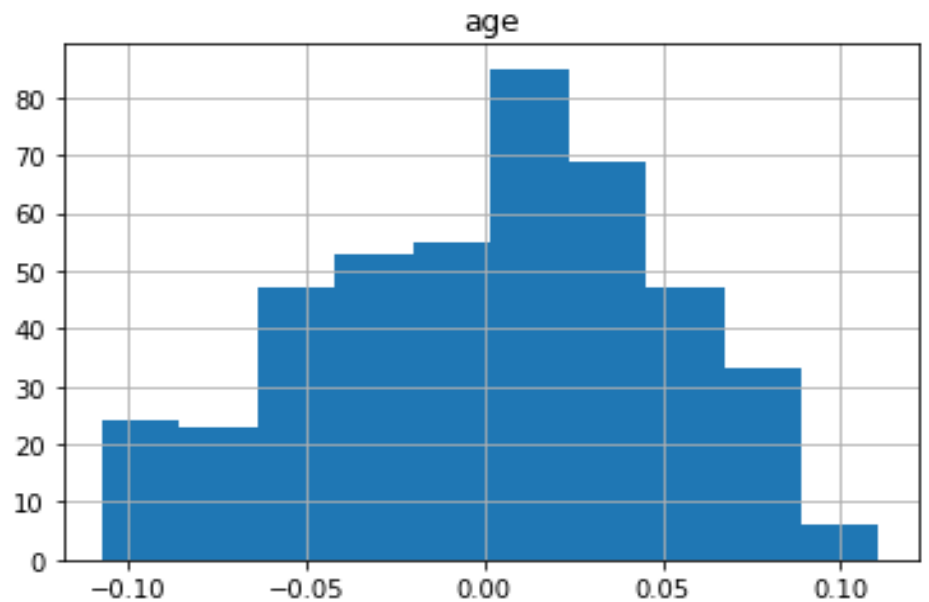
Interprétation des statistiques descriptives

Que constatez-vous à la lecture des statistiques descriptives de diabete (mean, std, la colonne prog).

Au-delà des statisques descriptives : Les visuels 1

Les visuels permettent de voir clairement (symétrie, asymétrie,etc) ce que revèlent les statistiques descriptives.

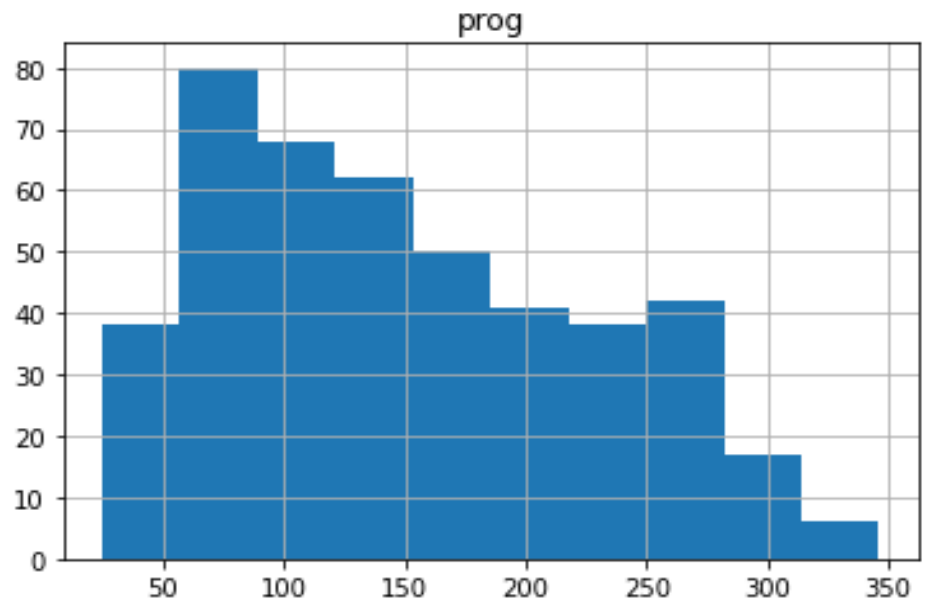
diabete.hist(column='age')
la variable age est plutôt symétrie autour d'une moyenne centrée à 0.



Au-delà des statisques descriptives : Les visuels 2

diabete.hist(column='prog')

La colonne prog présente une asymétrie à gauche.



Ces visuels sont également des puissants outils de communication.

Exercice d'application 3

Histogramme avec bins

Écrire un programme permettant de construire l'histogramme de la variable age avec 6 bins.

Corrélation 1

Afin de bien comprendre les relations entre les variables en analyse exploratoire, nous avons recours à la matrice des corrélations :
corrmat = diabete.corr().round(2)

```
In [34]: corrmat
Out[34]:
       age   sex   bmi    bp    s1    s2    s3    s4    s5    s6  prog
age   1.00  0.17  0.19  0.34  0.26  0.22 -0.08  0.20  0.27  0.30  0.19
sex   0.17  1.00  0.09  0.24  0.04  0.14 -0.38  0.33  0.15  0.21  0.04
bmi   0.19  0.09  1.00  0.40  0.25  0.26 -0.37  0.41  0.45  0.39  0.59
bp    0.34  0.24  0.40  1.00  0.24  0.19 -0.18  0.26  0.39  0.39  0.44
s1    0.26  0.04  0.25  0.24  1.00  0.90  0.05  0.54  0.52  0.33  0.21
s2    0.22  0.14  0.26  0.19  0.90  1.00 -0.20  0.66  0.32  0.29  0.17
s3   -0.08 -0.38 -0.37 -0.18  0.05 -0.20  1.00 -0.74 -0.40 -0.27 -0.39
s4    0.20  0.33  0.41  0.26  0.54  0.66 -0.74  1.00  0.62  0.42  0.43
s5    0.27  0.15  0.45  0.39  0.52  0.32 -0.40  0.62  1.00  0.46  0.57
s6    0.30  0.21  0.39  0.39  0.33  0.29 -0.27  0.42  0.46  1.00  0.38
prog  0.19  0.04  0.59  0.44  0.21  0.17 -0.39  0.43  0.57  0.38  1.00
```

Corrélation 2

D'après cette matrice, les variables bmi et prog sont très corrélées (0.59), ce qui signifie que plus on est âgé, plus la progression est rapide.

Les variables s3 et s4 sont négativement corrélées (-0.74).

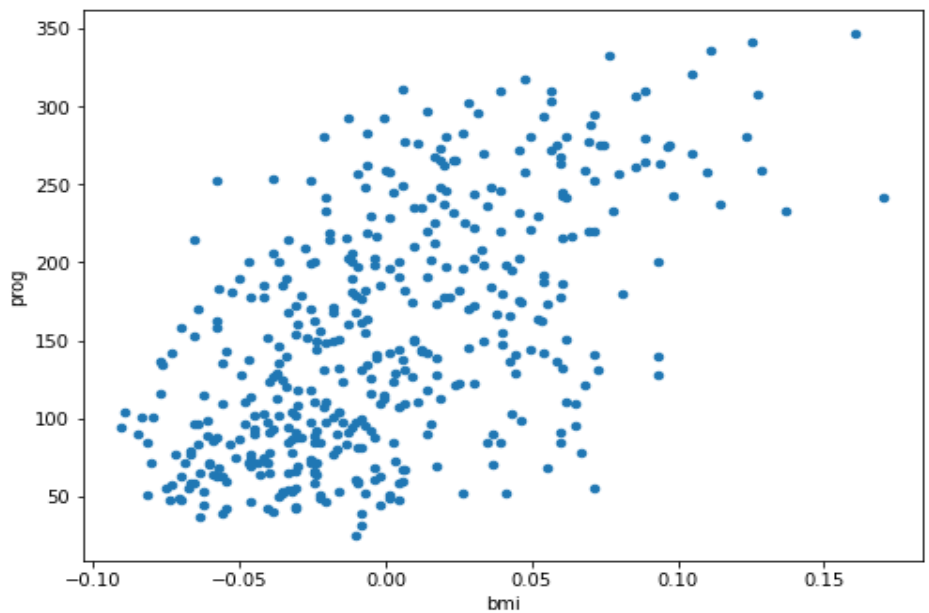
Attention !

L'étape d'analyse exploratoire des données est importante pour la sélection des variables et des algorithmes de prédiction.

Scatter 1

La progression du diabète semble augmenter avec le poids.

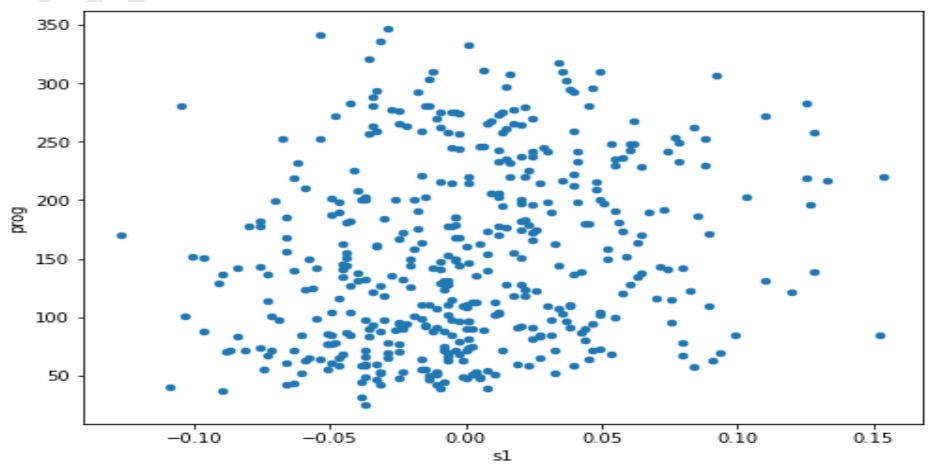
```
diabete.plot(
        kind="scatter",
        x='bmi',
        y='prog',
        figsize=(8,6))
plt.show()
```



Scatter 2

A l'opposé, la relation entre la progression et la variable s1 ne semble pas très claire.

```
diabete.plot(
        kind="scatter",
        x='s1',
        y='prog',
        figsize=(8,6))
plt.show()
```



Quel prédicteur choisir ?

Le prédicteur bmi semble mieux décrire l'évolution du diabete. Il sera donc préférer à s1.
Scikit-learn manipule les **données sous forme matricielle**, c'est-à-dire en **2 dimensions pour les prédicteurs**. On les obtient à l'aide des doubles brackets (pour rappel, les simples brackets donnent des Series). Tandis que la **variable à prédire est à une dimension**, une Serie.

```
X=diabete[['bmi']]
X.shape

Y = diabete['prog']
Y.shape
```

Importance de la sélection des variables

Attention !

La sélection des variables est importante.

Elle facilite l'interprétation des résultats de certains modèles, la réduction du temps de calcul ainsi que les problèmes de surapprentissage.

Instanciation du modèle

Dans Scikit-learn, **chaque modèle** est construit sous la forme d'une **classe à importer et à instancier** le cas échéant :

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
print(model)
```

Échantillon de test/échantillon d'entraînement 1

Nous divisons l'échantillon en deux parties, une partie des données vont servir pour **entraîner le modèle (échantillon d'entraînement)** et l'autre **(échatillon de test) pour évaluer les performances du modèle**.

La règle est de dédier **70% à 80% des données** à **l'échantillon d'entraînement** et le reste à l'échantillon de test.

La **fonction train_test_split** de Scikit-learn **permet de diviser les données en deux sous-ensembles d'entraînement et de test de façon aléatoire**.

Échantillon de test/échantillon d'entraînement 2

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size =
0.3, random_state=1)

print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

Attention !

Il est important de ne pas utiliser les données de test pour entraîner le modèle.

Fitting ou calibrage du modèle 1

Le fitting d'un modèle n'est rien d'autre que l'entraînement du modèle. On va se servir du jeu de données d'entraînement pour chercher les coefficients de la régression linéaire (l'intercept et la pente), on parle également de calibration/ajustement du modèle.

Une fois entraîné, le modèle peut maintenant servir pour réaliser des prédictions. En effet, les performances du modèle seront mesurées sur les données de test dont nous ne nous sommes pas servies pour entraîner le modèle.

La syntaxe pour entraîner le modèle est :

model.fit(X_train, Y_train)

La fonction **fit()** déclenche les calculs et les **résultats sont stockés** dans l'objet appelé **model**.

Fitting ou calibrage du modèle 1

L'ajustement ou le fitting du modèle signifie que les deux paramètres (intercept et le slope) ont été appris.

Par convention, dans Scikit-learn, tous les paramètres ont un underscore (tiret de 8 = _) à la fin.

Par exemple, pour lire la valeur de l'intercept avec deux chiffres après la virgule :

model.intercept_.round(2) $\Longrightarrow$ 151.41

De même que pour le coefficient de pente (slope) :

model.coef_.round(2) $\Longrightarrow$ [1030.62]

Exercice d'application 3

Fitting d'un modèle

Quels sont les paramètres (intercept et slope) de la régression linéaire entre prog et la variable s3 ?.

Prédiction 1

Une fois l'apprentissge terminé, le modèle va être évalué sur les données de test (le modèle va prédire l'outcome ou la variable à expliquer grâce à ses paramètres précédemment estimés dans les données d'entraînement). La **fonction predict()** permet de **réaliser cette opération de prédiction**

predicted = model.intercept_ + model.coef_*bmi.

predicted = model.predict(X_test)

Prédiction 2

On peut vérifier la dimension de predicted comme suit :

print(predicted.shape) $\Longrightarrow$ *(133,)*

Qui doit forcément est la même dimension que Y_test, car nous allons les comparer ultérieurement pour estimer la précision ou l'erreur du modèle.

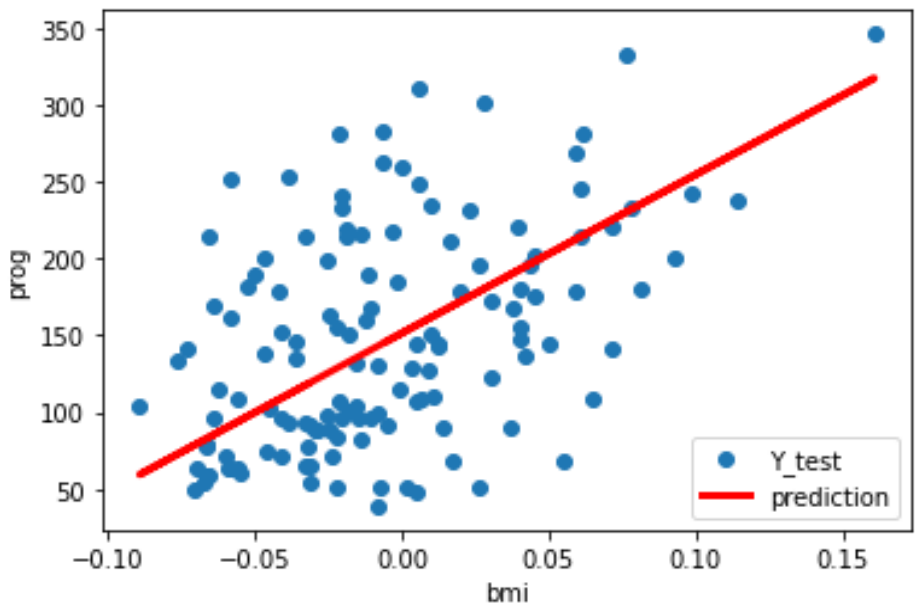
print(type(predicted))

predicted est également un numpy array.

Performance du modèle 1

Évaluer les performances du modèle (ici de regression linéaire simple), consiste à **comparer les prédictions faites par le modèle avec les vraies observations**, autrement, **nous allons comparer Y_test et predicted**.

Commençons par une comparaison visuelle :



Performance du modèle 2

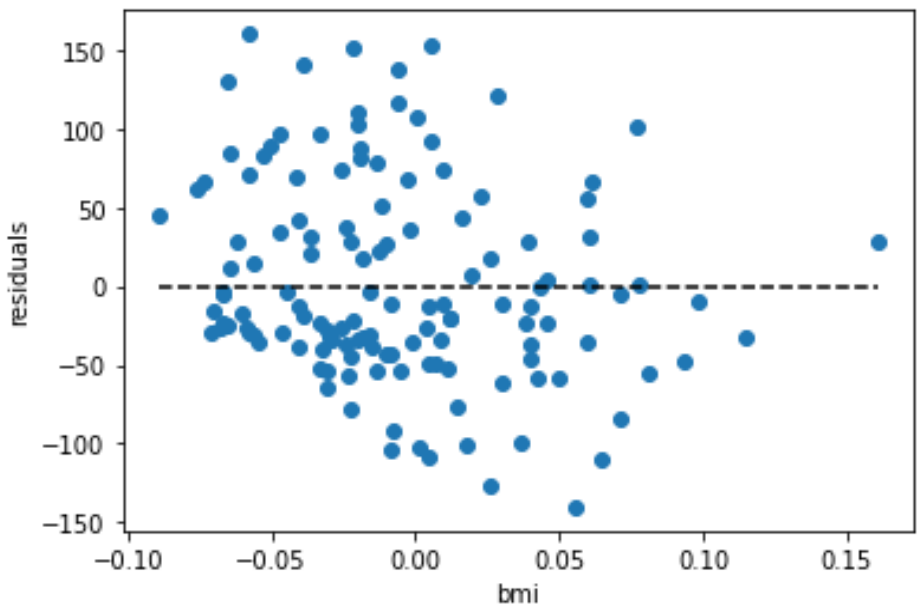
Nous constatons que certains points se trouvent sur la droite de régression (prediction = vraie observation) tandis que d'autres sont plutôt éloignés (prediction différent de la vraie observation). Nous pouvons mesurer la **distance entre ces points et la droite de regression de façon verticale**. Cette distance s'appelle **le résidu ou l'erreur**.

Le résidu ou l'erreur est la différence entre la vraie observation de l'outcome ou de la cible (target ou Y, ou la variable à expliquer) et la valeur prédite par le modèle.

Plus le résidu est proche de 0, plus l'erreur de prédiction est faible, et plus notre modèle est meilleur (la prédiction se rapproche de la réalité).

Performance du modèle 3

Nous pouvons calculer et représenter sur un scatter plot les résidus comme suit :



Les résidus sont dispersés autour de la ligne horizontale y=0, sans aucune forme particulière.

Performance du modèle 4

Cette distribution apparemment aléatoire est un signe que le modèle fonctionne.

Idéalement, les résidus devraient être symétriques et aléatoires autour de l'axe horizontal.

Si le graphique des résidus montre une forme particulière linéaire ou non linéaire, cela signifie que le modèle peut être amélioré.

Attention !

Le graphe des résidus peut indiquer un biais du modèle et les indicateurs statistiques revèlent la qualité de l'ajustement.

Exercice d'application 4

Formes non linéaires

Quels sont les formes non linéaires que vous connaissez ?

Mean Squared Error (MSE) 1

Les résidus étant une Serie, il convient de trouver une metrique (une valeur) pour aggréger tous ces points. **La moyenne** semble une bonne façon **d'aggréger les résidus**.

residuals.mean() $\Longrightarrow$ 2.396

2.396 est pas très loin de 0 mais il y a un **problème**. **Les résidus peuvent être positifs ou négatifs, donc prendre leur moyenne les annule**. Ce n'est donc pas une métrique précise.

Pour **résoudre ce problème**, nous prennons **le carré de chaque résidu, puis nous calculons la moyenne des carrés des résidus**. C'est le **Mean Squared Error(MSE)**, moyenne des carrés des erreurs.

*(residuals**2).mean()* $\Longrightarrow$ 4095.914

Mean Squared Error (MSE) 2

Nous pouvons directement calculer le MSE via la méthode **mean_squared_error()** de Scikit-learn.
from sklearn.metrics import mean_squared_error
print(mean_squared_error(Y_test, predicted).round(3)) $\Longrightarrow$
4095.914

De façon générale, plus le MSE est faible, mieux c'est (le modèle est meilleur). A ce niveau, il n' y a pas de bon ou de mauvais seuil. Nous pouvons définir un seuil en nous basant sur la variance de l'outcome dans l'ensemble de test :

Y_test.var().round(3) $\Longrightarrow$ *5072.609*

On peut constater qu'un MSE de 4095.914 n'est pas si mauvais comparativement à la variance calculée précédemment.

Root Mean Squared Error (RMSE)

Pour que l'échelle des erreurs soit la même que celle des l'outcome (Y_test), on utilise souvent le Root Mean Squared Error (RMSE ou l'erreur quadratique moyenne). Il s'agit de la racine carrée de MSE.

print(np.sqrt(mean_squared_error(Y_test, predicted)).round(3))
$\Longrightarrow$ 63.999

versus

print(np.sqrt(Y_test.var()).round(3)) $\Longrightarrow$ *71.222*

Exercice d'application 5

MSE

Pourquoi la moyenne ne semble pas une bonne métrique pour évaluer la performance d'un modèle ?

R-squared (R^2) 1

Une autre façon **d'évaluer la performance d'un modèle** est de calculer le R^2 grâce à la fonction **model.score()**.

print(model.score(X_test, Y_test).round(3)) $\Longrightarrow$ *0.186*

C'est **la proportion de la variabilité totale** expliquée par le modèle. Environ 18% de variabilité dans l'échantillon de test est expliquée par le modèle.

La variabilité totale est calculée comme la somme au carré des différences entre
l'outcome et la moyenne de l'outcome dans l'ensemble de test.

*print(((Y_test-Y_test.mean())**2).sum().round(3))* $\Longrightarrow$
669584.391

R-squared (R^2) 2

Tandis que **la variabilité non expliquée par le modèle est la somme au carré des résidus** :

*(residuals**2).sum().round(3)* $\Longrightarrow$ 544756.612

Par conséquent, **la proportion de variabilité expliquée** est :

1 - (544756.612/669584.391) $\Longrightarrow$ 0.186

R-squared (R^2) 3

Un modèle parfait explique toute la variablité dans les données. Le R^2 est compris entre 0% et 100%.

0% signifie que le modèle explique aucune variablité dans les données de l'outcome autour de sa moyenne tandis que 100% indique que le modèle explique toute la variablité présente dans les données.

Attention !

L'évaluation obtenue avec le R^2 ainsi que le graphe des résidus fournissent une information sur les performances du modèle.

Régression linéaire multiple 1

Contrairement à la régression linéaire simple (où l'on utilise un seul prédicteur), le nombre de prédicteurs de la régression linéaire multiple est plus élévé (au moins deux).

Comme la variable s3 est négativement corrélée avec prog, nous allons l'ajouter en tant que 2ème prédicteur, de manière à construire une régression linéaire multivariée. En définitive, prog va maintenant dépendre linéairement de s3 et de bmi :

prog = a + b*bmi + c*s3

Comme dans le cas de la régression simple, la recherche des valeurs des coefficients se déroulent de la même façon excepté la partie préparation des données, où nous allons ajouter le variable s3.

Régression linéaire multiple 2

Regression linéaire multiple
Préparation des données
X2 = diabete[['bmi', 's3']]
Y = diabete['prog']
échantillon de test et d'entraînement
Subdivision aléatoire pour s'assurer des mêmes tirages

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state=1)
model2 = LinearRegression()
model2.fit(X2_train, Y_train))
print(model2.intercept_) $\Longrightarrow$ 151.892
print(model2.coef_) $\Longrightarrow$ [903.484 -330.12]
prog = 151.89 + 903.48*bmi - 330.12*s3

Régression linéaire multiple 3

La prédiction est donnée par :

predicted2 = model2.predict(X2_test)

Exercice d'application 6

Entraîner un modèle

Entraîner un modèle de régression linéaire multivarié avec les 3 variables suivantes : bmi, s3 et s5.

Affichez les coefficients.

Comparaison des deux modèles (simple et multivarié) 1

Le meilleur modèle est celui avec un MSE plsu faible.

Dans le premier modèle de régression simple, le MSE = 4095.917

Dans second modèle, le MSE est :

print(mean_squared_error(Y_test, predicted2).round(2)) $\Longrightarrow$
3815.76

Avec un MSE plus faible, le second modèle est meilleur que le premier dans la prédiction de prog.

Comparaison des deux modèles (simple et multivarié) 2

Attention !

En général, plus on inclut les prédicteurs, plus le MSE sera plus faible. en revanche, attention d'inclure certains prédicteurs se comportant comme des bruits aléatoires ou résidus aléatoires (espérance des résidus non nulle ou une forme particulière se dégage des résidus dans un scatter plot).

Après une analyse exploratoire des données, on entraîne le modèle avec les variables qui expliquent mieux l'outcome, autrement dit, les variables fortement corrélées avec l'outcome. Idéalement les résidus doivent suivre une distribution normale, ou s'en rapprocher.

Exercice d'application 7

Types de dsitribution

Quelles sont les types de distributions ou lois de probabilités que vous connaissez ?

Quiz 1

1) Équation de droite

Écrire l'équation d'une droite

2) régression linéaire

Quelle instruction permet d'importer le modèle de régression linéaire dans Scikit-learn ?

3)Performance

Quel est l'échantillon de données qui est utilisé pour évaluer les performances du modèle ?

Quiz 2

4) fitting d'une régression linéaire

Écrire un programme qui permet de fitter un modèle de regréssion linéaire multivarié avec trois régresseurs de votre choix.

5) Résidus

Expliquer le choix de vos régresseurs à l'aide d'une analyse des résidus de chaque régresseur (moyenne, MSE et scatter plot).

6) R^2

Calculer le R^2 grâce à la fonction r2_score que vous pouvez importer comme suit :

from sklearn.metrics import r2_score

Quiz 3

7) Décrire la base de données diabete

Décrire la base de données 'diabete' sur laquelle nous avons travaillée (nom du créateur, année de mise à disposition du public, description des Series ou colonnes), etc.

Data Science

Chap 5 : Classification

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 21, 2022

"Les personnes ayant des objectifs clairs et écrits accomplissent beaucoup plus dans une période de temps plus courte que les personnes n'ayant pas d'objectifs écrits ne pourrait jamais imaginer."
Brian Tracy

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons travailler sur les différentes étapes de la Data Science en Python.

À la fin de ce chapitre, vous serez en mesure :

- de réaliser une analyse exploratoire
- de réaliser une modélisation et une prédiction
- de calculer une précision et la matrice de confusion
- d'effectuer la validation croisée, le tuning des paramètres et de réaliser les prédictions sur les nouvelles données

Introduction 1

La classification consiste à regrouper les données en classes. La **classification** est utilisée dans des cas où **l'outcome** (Y ou variable à expliquer) est **une variable discréte** (variable contenant un nombre fini de valeurs isolées, séparées, distinctes versus variable continue ou quantitative).

Par exemple, prédire si un patient est infecté ou pas, classer les images ou encore prédire si une transaction financière est un financement du terrorisme ou pas.

Introduction 2

Dans chaque exemple, **l'outcome** est une **variable catégorielle**, c'est-à-dire **représentée par une étiquette ou un label** en un nombre fini de classes.

Les **valeurs discrètes** peuvent être **numériques** comme par exemple de **voyageurs dans un bus ou catégorielle, comme les couleurs**.

Classification binaire/Classification multiclasse

Lorsqu'il y a **deux classes à prédire**, on parle de c**lassification binaire**. Par exemple, vivant ou mort dans le Titanic en fonction du type de billets.

On parle de **classification multiclasse** lorsque le **nombre de classes à prédire dépasse deux**. Par exemple classer les différentes espéces de fleurs (versicolor, virqinica, ou setosa) dans le jeu de données iris en fonction de la longueur de leurs pétales et de leurs sépales.

Algorithmes de classification

Il existe plusieurs **algorithmes de classification** tels que la **regréssion logistique, les knn (K Nearest Neighbors), les decisions trees, naive bayes, SVM (Support Vector Machines), les réseaux de neurones (neural networks)**, etc.

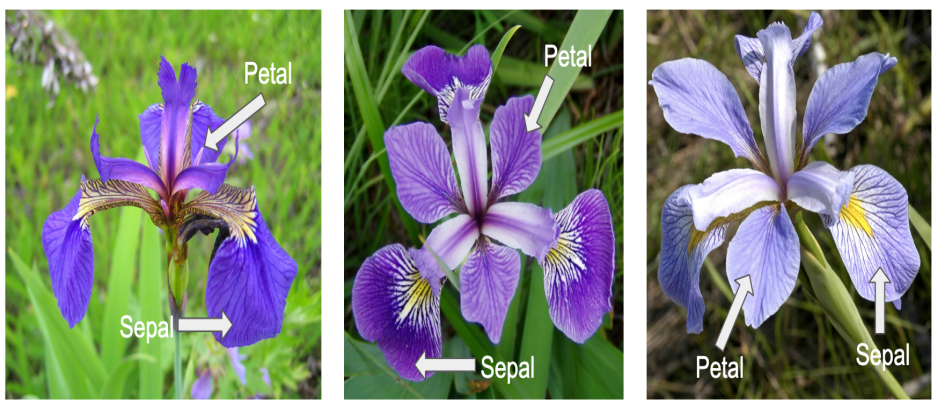
Dans ce chapitre, nous allons nous pencher sur **l'algorithme knn** pour **classer les différentes espèces de fleurs des données iris**.

La base de données iris

Les données sur les iris sont les populaires dans la littérature des algorithmes de reconnaissance. Cette base de données a été créée par R.A Fisher et mise à la disposition du public en Juillet 1988 par Michael Marshall (`https://scikit-learn.org/stable/datasets/toy_dataset.html#iris-plants-dataset`).

Elles comportent **150 fleurs (les iris)** et **quatre (4) attributs numériques** : **sepal length** en cm, **sepal width** en cm, **petal length** en cm, **petal width** en cm.

Iris setosa *Iris versicolor* *Iris virginica*



Exercice d'application 1

Importatation et préparation des données iris

1)Comme pour les données sur le diabete manipulées au chapitre 4 (régression linèaire), importer les données d'iris et construisez le DataFrame correspondant.

2) Vous vous rendez compter que le target est une liste d'integers. Servez-vous de l'attribut target_name pour voir les noms associés à ces entiers. Puis transformer la colonne target que vous avez nommée précédemment species, de manière à avoir les noms des iris correspondants à chaque entier. Pour info, 0 correspond à l'espèce *setosa*, 1 à l'espèce *versicolor* et 2 à l'espèce *virginica*. Vous trouverez quelques astuces sur `https://datatofish.com/replace-values-pandas-dataframe/`

Exercice d'application 1

Instruction de suppression d'une Serie ou colonne

1) Quelle est l'instruction qui permet de supprimer une Serie ou une colonne ?

2) Commentez la.

Exercice d'application 2

Modification des noms de colonne

Modifiez le nom des colonnes comme suit : 'sepal_len', 'sepal_wd', 'petal_len','petal_wd', 'species'

Summary Statistics 1

iris.describe()

	sepal_len	sepal_wd	petal_len	petal_wd
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Nous constatons que les quatres prédicteurs sont numériques (quantitatifs) avec des amplitudes différentes. Il n'y a pas de valeurs manquantes, **c'est donc un jeu de données propre, prêt à emploi**.

Summary Statistics 2

Comme les amplitudes des variables explicatives (préditeurs) sont à peu près similaires, nous n'allons pas effecteur de **normalisation**. **La normalisation consiste à centrer (moyenne = 0) et réduire (écart-type = 1) une variable**. Elle peut être une étape importante dans le prétraitement des données pour des algorithmes d'apprentissage automatique. On l'appelle également **feature scaling** (mise à l'echelle).

Consulter la page suivante pour savoir l'importance de la normalisation `https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html`

Consulter la page suivante pour voir comment normaliser une donnée
`https://fr.softoban.com/normalization-data-python`

Exercice d'application 2

Summary statistics de deux variables

Écrire une instruction (en une ligne) permettant de calculer les summary statistics de sepal_len, sepal_wd.

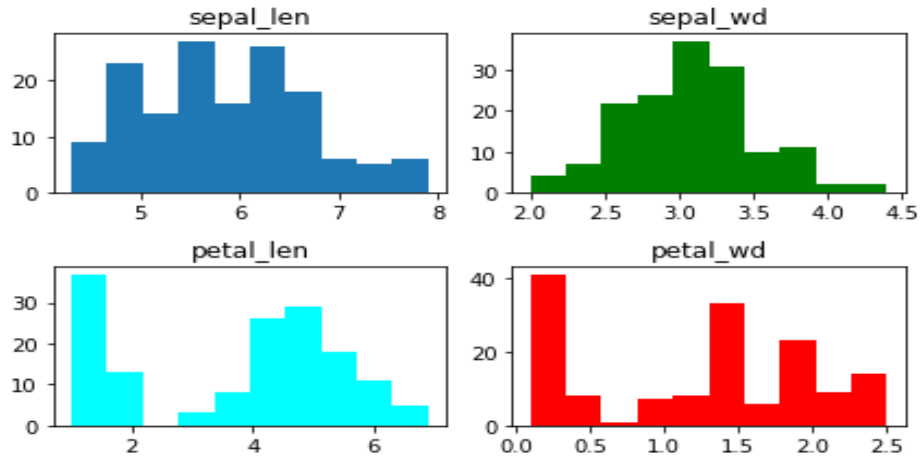
Répartition des classes

L'ensemble de données contient 3 classes de 50 instances chacune :
iris.groupby('species').size()
iris['species'].value_counts()
La fonction **value_counts()** est très utile pour rapidement comprendre la distribution des données. Elle compte le nombre d'entrées unique dans une colonne spécifique.

Le jeu de données iris est équilibré (**balanced dataset**). Il existe des cas où les données ne sont pas équitablement réparties entre classes : On parle de jeu de données déséquilibrés (**unbalanced dataset**). Exemple, les données de fraude. Dans ce dernier cas, une analyse légèrement différente est utilisée, voir le lien suivant :
`https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learn`

Univariate plot

Une meilleure compréhension des attributs (prédicteurs) commencent par une analyse visuelle univariée (histograms, box-plots, density plots).



Les variables **sepal_len, sepal_wd** semblent suivre une **loi normale** (distribution Gaussienne) qui se caractérise par **une forme en cloche symétrique** des données, tandis que **petal_len et petal_wd ne sont pas distribuées normalement**.

Exercice d'application 3

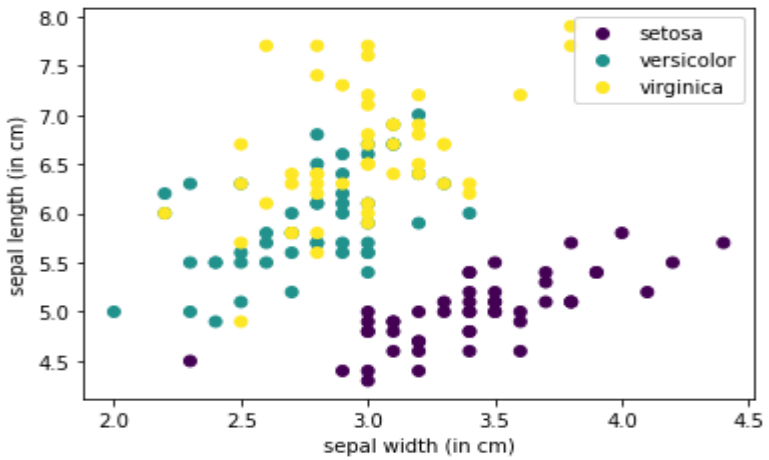
Histogrammes groupés

Réproduire dans un seul graphe les quatre histogrammes du slide précédent.

Vous pouvez vous inspirer des conseils qui se trouvent à `https://pythonguides.com/matplotlib-multiple-plots/`

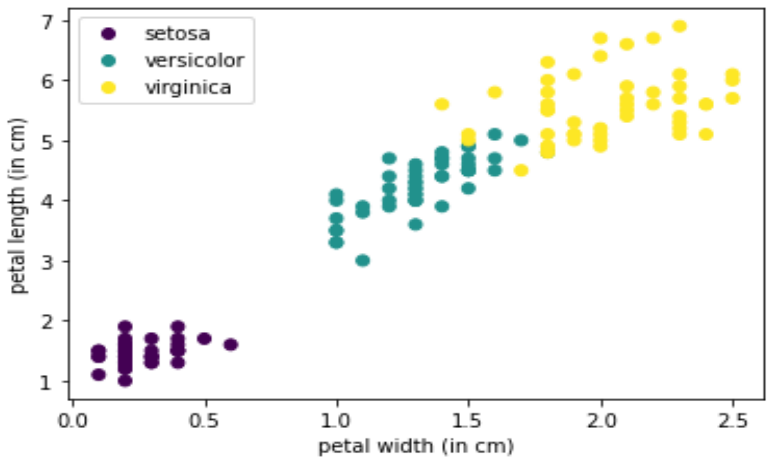
Multivariate plot 1

Pour voir les relations entre les variables, on utilise les scatter plots. Nous allons colorier chaque point en fonction de l'espèce à laquelle il appartient.



En croisant sepal_len et sepal_wd, nous pouvons distinguer facilement les iris de l'espèce setosa des autres espèces.
Séparer les espèces versicolor et virginica semble plus difficile en raison du chevauchement des poins verts et jaunes.

Multivariate plot 2



La longueur et la largeur des petals sont très correlées. Ces deux variables sont donc très utiles pour identifier les différentes espèces. Il faut noter que la frontière entre les espèces versicolor et virginica semble un peu floue, ce qui indique les difficultés de classification. Celle-ci est importante pour savoir quelle variable sera utiliser pour entraîner le modèle.

Exercice d'application 4

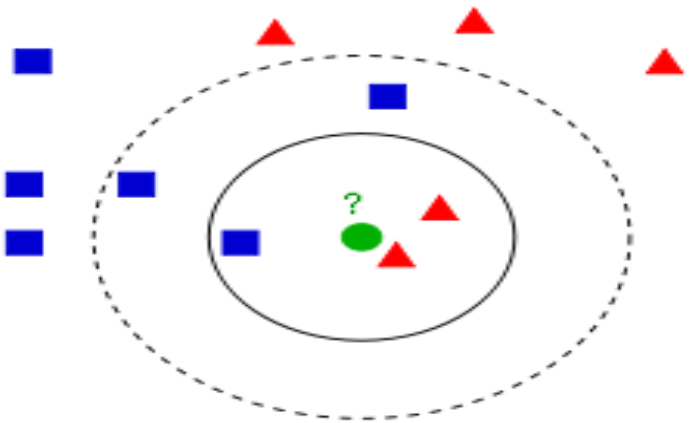
Multiple scatter-plots

Exécutez et commentez la fonction ci-dessous :

pd.plotting.scatter_matrix(iris, c = colors)

K nearest neighbors 1

K nearest neighbors (knn) est un algorithme de Machine Learning supervisé qui prend un point, regarde ses k plus proches voisins et lui associe une classe par vote majoritaire (la classe la plus présente parmi les k proches voisins).



Dans l'exemple ci-dessus, il y a 2 classes, les traingles rouges et les carrés bleus. Á quelle classe appartiendrait le point vert en se basant sur un **3nn**, c'est-à-dire **k = 3 (les plus proches voisins)** ?

K nearest neighbors 2

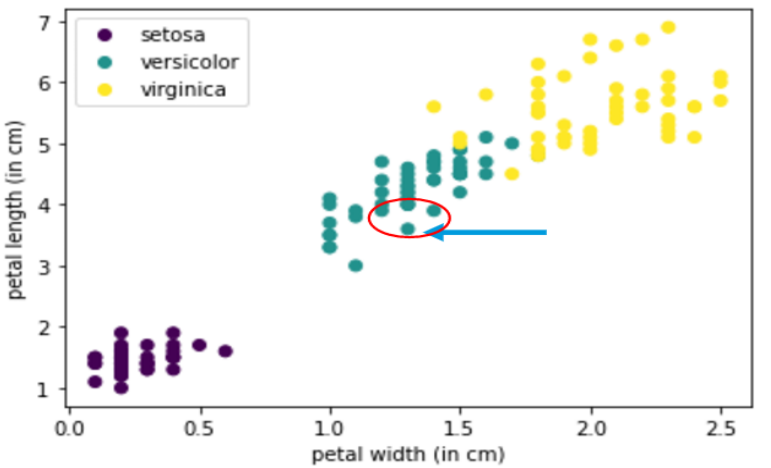
Des **3 plus proches voisins** du point vert, 2 sont des triangles rouges et 1 est un carré bleu, par conséquent, **le point vert est prédit comme un triangle rouge** (car ils sont majoritaires).

Si **k = 5**, c'est le cas du cercle en trait interrompu court fin, alors le **point vert est prédit comme étant un carré bleu**, car ils sont majoriataires.

Le changement de la valeur de k peut affecter l'output du modèle de prédiction. **k** est donc considéré comme **un hyperparamètre (hyperparameter)**, c'est-à-dire un paramètre dont la valeur, définie avant le processus d'apprentissage, est utilisée pour contrôler ce dernier.

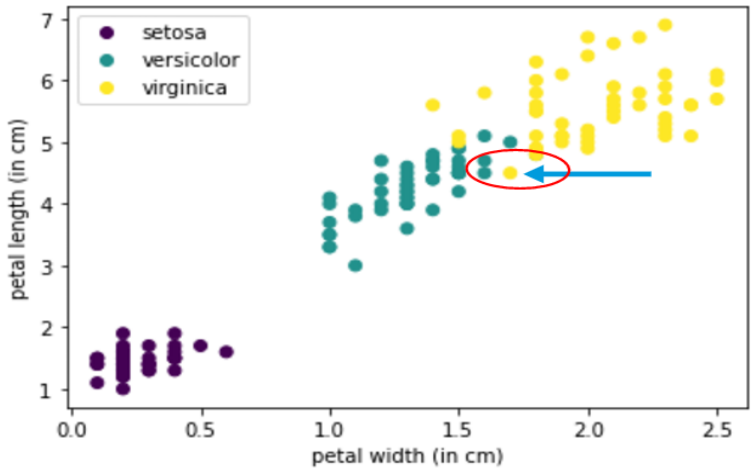
K nearest neighbors 3

L'algorithme knn est implementé dans le module sklearn.neighbors de Scikit-learn.
from sklearn.neighbors import KNeighborsClassifier



Soit le **point (versicolor)** indiqué par la **flèche bleue**, un **3nn** va **prédire assez facilement la classe versicolor**, car ses 3 plus proches voisins (à l'intérieur du cercle rouge) sont les versicolor.

K nearest neighbors 4



Ici le 3nn du point (virginica) indiqué par la flèche bleue va prédire un versicolor, car c'est la classe majoritaire (à l'intérieur du cercle rouge). L'algorithme se trompe à ce niveau en raison du chevauchement (absence de frontière claire) entre les versicolor et les virginica dans cette zone.

K nearest neighbors 5

Attention !

L'algorithme knn peut être également utilisé pour des problèmes de régression à la seule différence qu'au niveau des prédictions, la décision va reposer sur la moyenne des k plus proches voisins au lieu du vote majoritaire.

Exercice d'application 5

Explication de l'algorithme knn

1) Un 5nn et un 7nn conduisent-ils aux mêmes résultats ?

2) Expliquez

3) Que représente k ?

Data Preparation 1

Plus haut, nous avons vu que les variables petan_len et petal_wd étaient les plus utiles dans la classification des espèces d'iris.
Nous définissons de nouvelles variables X et y comme suit :

X = iris[['petal_len', 'petal_wd']]
y = iris['species']
Nous divisons nos données en deux échantillons : l'échantillon d'apprentissage et l'échantillon de test :

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=1, stratify=y)
```

Data Preparation 2

test_size=0.30 signifie que l'ensemble de test contient 30% de nos données et le reste dans l'ensemble d'apprentissage.

stratify=y signifie que la répartition 30/70 des étiquettes (noms des espèces) demeure similaire dans l'ensemble de test et l'ensemble d'apprentissage.

print(y_train.value_counts())
print(y_test.value_counts())

Attention !

Dans les problèmes de classification, l'échantillonnage stratifié est souvent utilisé pour s'assurer que les ensembles d'entraînement et de test contiennent approximativement le même pourcentage d'échantillons de chaque classe dans la variable cible (y).

Exercice d'application 6

Explication d'un échantillonnage non stratifié

1) Exécuter et commenter le programme suivant :

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=1)
print(y_train.value_counts())
print(y_test.value_counts())
```

2) Quelles sont les conséquences d'un échantillonnage non stratifié ?

Modélisation 1

Nous sommes maintenant prêts à construire et à entraîner notre modèle knn. Commençons par importer la classe du modèle :

from sklearn.neighbors import KNeighborsClassifier
Créons une instance knn de la classe KNeighborsClassifier :
knn = KNeighborsClassifier(n_neighbors=5)

Nous avons juste à définir le paramètre k=5 (de façon aléatoire), c'est-à-dire le nombre de k plus proches voisins à prendre en compte dans la classification.
Notre modèle peut maintenant être entraîné :

print(knn.fit(X_train, y_train))

Modélisation 2

Nous utilisons la plupart des paramètres par défaut tels que : **la métrique de minkowski et le p = 2 qui ensemble, imposent que la distance à utiliser est une distance euclidienne**.

Pour plus de details sur les paramètres, veuillez vous reporter à cette page : `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

Prédiction 1

La méthode **predict()** de Scikit-learn permet de prédire l'espèce d'iris en se basant sur la matrice des prédicteurs X.

pred = knn.predict(X_test)
Pour rappel, les prédictions se font toujours sur l'ensemble de test.
Les 5 premières prédictions sont :
pred[:5]
['virginica', 'setosa', 'setosa', 'versicolor', 'versicolor']
Chaque prédiction est stockée dans un tableau à une dimension (1darray).

Attention !

la fonction predict() renvoie un tableau de labels de classes prédites.

Exercice d'application 7

Étapes jusqu'à la prédiction d'un modèle knn

1) Quelles sont les différentes étapes dans l'ordre, de la modèlisation jusqu'à la prédiction d'un modèle knn ?

2) Écrivez le code correspondant, prendre k = 6.

Probability prediction 1

Dans tous les algorithmes de classification implementés dans Scikit-learn, il existe une méthode supplémentaire appelée *predict_prob()*. Cette méthode renvoie **les probabilités prédictes de la variable cible sous forme de tableau**.
Par exemple, les probabilités prédictes de la 24ème et de la 25ème espèce d'iris :

```
pred_prob = knn.predict_proba(X_test)
print(pred_prob[24:26])
[[0. 1. 0. ]
 [0. 0.8 0.2]]
```

La probabilité que **la 24ème fleur** prédite soit 'setosa' ou 'virginica' est nulle. La **probabilité qu'elle soit 'versicolor' est 1**.
Pour **la 25ème fleur**, la probabilité d'être une 'setosa' est nulle, **une 'versicolor' est de 0.8** et une 'virginica' est 0.2.

Probability prediction 2

Ces probabilités signifient que parmi les 5 proches voisins de la 25ème fleur dans l'ensemble de test par exemple, 1 est une 'virginica' et 4 sont des 'versicolor'.
Les prédictions correspondantes sont donc :

pred[24:26] $\Longrightarrow$ ['versicolor', 'versicolor']

Attention !

En classification, la prédiction douce (soft prediction) renvoie les probabilités prédites d'appartenance des données à chacune des classes, or la prédiction dure (hard prediction) ne renvoie que les étiquettes/labels.

Exercice d'application 8

Probabilités de prédiction

Soient les probabilités prédictes suivantes : [0.4, 0.3, 0.3] (pour rappel, l'ordre des labels est setosa, versicolor et virginica').

Quelle fleur sera prédicte ?

Accuracy (précision) 1

Definition

En classification, **la métrique** la plus regardée est la **précision**. **Elle représente la proportion des données dont les étiquettes/labels prédictes correspondent aux étiquettes/labels observées** dans l'ensemble de test.

print((pred==y_test.values).sum()) $\Longrightarrow$ 44
print(y_test.size) $\Longrightarrow$ 45

Le modèle **se trompe donc une fois** et la **précision** est de :
print((pred==y_test.values).sum()/y_test.size) $\Longrightarrow$ $44/45 = 0.9777$

Accuracy (précision) 2

Les **méthodes** suivantes permettent de **calculer directement la précision** :

1) *knn.score(X_test, y_test)* $\Longrightarrow$ 0.9777

2) *from sklearn.metrics import accuracy_score*
accuracy_score(y_test, pred) $\Longrightarrow$ 0.9777

Exercice d'application 9

Calcul de la précision

Commentez les instructions suivantes :

1) *print((pred==y_test.values).sum())*

2) *print(y_test.size)*

La matrice de confusion (Matrix Confusion) 1

La précision de la classification seule peut être trompeuse s'il y a un nombre inégal d'observations dans chaque classe ou s'il y a plus de deux classes dans les données. Le calcul d'une matrice de confusion donne une meilleure idée de ce que la classification réussit à faire et des types d'erreurs qu'elle commet.

Definition

La matrice de confusion est un tableau récapitulatif du nombre de prédictions correctes et incorrectes par classe.

La matrice de confusion (Matrix Confusion) 2

Le module **sklearn.metrics** permet d'importer et utiliser la fonction **confusion_matrix()**.

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))

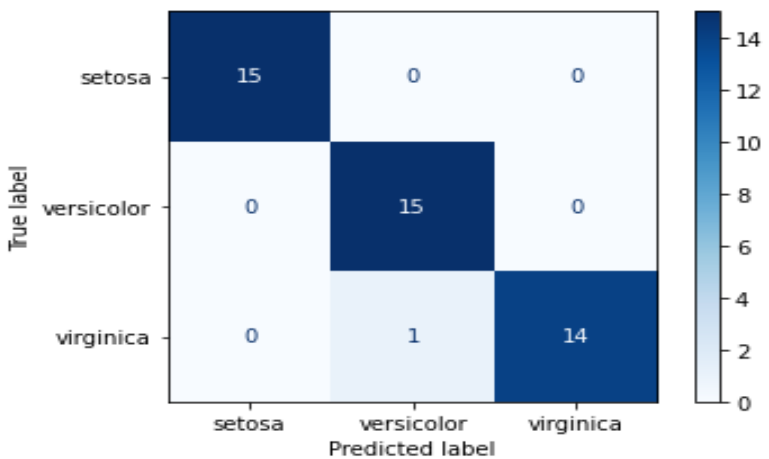
```
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
```

La matrice de confusion (Matrix Confusion) 3

la fonction **plot_confusion_matrix()** de **sklearn.metrics** permet de visualiser cette matrice :

from sklearn.metrics import plot_confusion_matrix

plot_confusion_matrix(knn, X_test, y_test, cmap=plt.cm.Blues)



La matrice de confusion (Matrix Confusion) 4

Nous avons spécifié les étiquettes/labels dans l'ordre. Chaque colonne de la matrice correspond à une classe prédite, et chaque ligne correspond à une classe réelle (vraie observation). La somme des lignes correspond donc au nombre total d'instances de la classe.

La première ligne correspond à l'espèce setosa observée ou réelle : [15, 0, 0] indique que 15 iris-setosa sont correctement prédites et qu'aucun n'est mal prédite.

La matrice de confusion (Matrix Confusion) 5

La dernière ligne [0, 1, 14] suggère que sur 15 iris-virginica réelles ou observées, 0 a été prédit comme iris-setosa, 1 a été prédit comme iris-versicolor et les 14 autres ont été correctement identifiés comme iris-virginica. Cela correspond à notre observation lors de l'analyse exploratoire des données, à savoir qu'il y avait un certain chevauchement entre les deux espèces sur le nuage de points et qu'il est plus difficile de distinguer l'iris-versicolor de l'iris-virginica que d'identifier l'iris-setosa.

Attention !

Une matrice de confusion est un tableau souvent utilisé pour décrire les performances d'un modèle de classification sur un ensemble de données de test dont les valeurs réelles sont connues.

Exercice d'application 10

Matrice de confusion

import numpy as np
y_true = np.array(['lune', 'soleil', 'soleil', 'lune', 'soleil'])
pred = np.array(['lune', 'soleil', 'lune', 'lune', 'soleil'])
confusion_matrix(y_true, pred, labels =['soleil', 'lune'])
Compléter la matrice de confusion suivante :

.....	.....
0	.....

k-fold cross Validation 1

Précédemment, nous avons réalisé un fractionnement entre des données en jeu de test et en jeu d'entraînement, avant d'entraîner le modèle et de pouvoir mesurer les performances de ce dernier sur les données de test. Il s'agit d'une technique simple de validation croisée, égalment connue sous le nom de (holdout method).

Pour rappel, **nous avons procédé à une division aléatoire**, par conséquent, **la performance du modèle peut être sensible à la façon dont les données sont divisées**. Pour **surmonter ce problème**, nous introduisons **la validation croisée k-fold** (k-fold cross Validation).

k-fold Cross Validation 2

En validation croisée, **les données sont divisées entre k sous-ensembles**. Ensuite, la méthode holdout est repetée k fois, de manière à ce que chaque fois, un des k sous-ensembles est utilisé comme ensemble de test et le reste (k - 1), comme ensemble d'entraînement.

On fait ensuite **la moyenne de la précision sur les k essais pour obtenir l'efficacité totale du modèle**. De cette façon, tous les points de données sont utilisés ; et il y a plus de métriques de sorte que nous ne dépendons pas d'une seule donnée de test pour l'évaluation de la performance du modèle.

k-fold Cross Validation 3

Dans Scikit-learn, la fonction cross_val_score() permet d'utiliser la validation croisée sur le modèle et les données.

from sklearn.model_selection import cross_val_score
knn_cv = KNeighborsClassifier(n_neighbors=3)

Modèle avec 5 validations croisées :

cv_scores = cross_val_score(knn_cv, X, y, cv=5)

Chacun des k sous-ensembles contient 20% des données brutes.

k-fold Cross Validation 4

print(cv_scores.round(3)) $\Longrightarrow$ [0.967 0.967 0.9 0.933 1.]

On constate qu'en raison des affectations aléatoires, la précision des jeux de données fluctue entre 0,9 et 1.

cv_scores.mean().round(3) $\Longrightarrow$ 0.953

Nous ne pouvons pas nous fier à une seule répartition entre données d'entraînement et données de test, mais nous signalons que le modèle 3nn a une précision de 95,33 % sur la base d'une validation croisée à 5 volets.

k-fold Cross Validation 5

Attention !

En règle générale, la validation croisée 5 ou 10 fois est préférable, mais il n'existe pas de règle absolue. Plus k est grand, plus la différence de taille entre l'ensemble d'entraînement et les sous-ensembles de rééchantillonnage est faible. Plus cette différence diminue, plus le biais de la technique devient faible.

Exercice d'application 11

Validation croisée

1) Calculer la précision du modèle 3nn lorsque cv=10.

2) Que constatez-vous ?

Tuning parameters 1

Lorsque nous avons construit notre premier modèle knn, nous avons fixé l'hyperparamètre k à 5, puis à 3 plus tard dans la validation croisée k-fold ; des choix vraiment aléatoires. Quel est le meilleur k ? **Trouver le k optimal** s'appelle régler l'hyperparamètre **(tuning hyperparameter)**. Un outil pratique est grid search.

Dans scikit-learn, nous utilisons la méthode **GridSearchCV()**, qui entraîne notre modèle plusieurs fois sur un ensemble de valeurs spécifiées avec le paramètre **param_grid** et calcule le score de validation croisée, afin que nous puissions vérifier laquelle de nos valeurs pour l'hyperparamètre testé est la plus optimale.

Tuning parameters 2

from sklearn.model_selection import GridSearchCV

créons notre modèle knn
knn2 = KNeighborsClassifier()

Créons un dictionnaire des valeurs que nous voulons tester comme n voisins :
param_grid = 'n_neighbors': np.arange(2, 10)

Utilisons la méthode gridsearch() pour tester tous les valeurs des k voisins
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)

Entraînons notre modèle
knn_gscv.fit(X, y)

Tuning parameters 3

Le n paramètre optimal est :
print(knn_gscv.best_params_) $\Longrightarrow$ *4*

La précision du modèle est :
print(knn_gscv.best_score_) $\Longrightarrow$ *0.966666*

En utilisant **la méthode gridsearch() pour trouver l'hyperparamètre optimal de notre modèle, la précision de ce dernier s'amèliore de plus de 1%.**

Tuning parameters 4

Nous sommes maintenant prêts pour construire notre modèle :

```
knn_final =
KNeighborsClassifier(n_neighbors=knn_gscv.best_params_['n_neighbors'])
knn_final.fit(X, y)
y_pred = knn_final.predict(X)
print(knn_final.score(X, y)) ⟹ 0.973333
```

Notre 4nn final a une précision de 97,3% dans la prédiction des espèces d'iris.

Attention !

Les techniques de validation croisée k-fold et de tuning parameters par la méthode gridsearch() sont applicables aux problèmes de classification et de régression.

Exercice d'application 12

7-fold cross validation and tuning parameters

1) Reproduire les étapes illustrées sur les 3 dernières slides, cette fois avec un cv=7 (de la création du modèle jusqu'au calcul de la précision, en passant par le calcul du paramètre optimal).

2) Que constatez-vous ?

Prédiction avec les nouvelles données 1

Nous sommes désormais prêts à deployer notre modèle knn final. Soit le nouvel enregistrement suivant : **sepal length et sepal width** sont **respectivement de 5.33cm et 2.95 cm**, **petal length et petal width** sont **de 3.76cm et 1.2cm respectivement**. Quel est l'espèce d'iris qui sera prédite en utulisant notre modèle final entraîné avec des paramètres optimaux ?

Nous allons donc utiliser la méthode **model.predict()** vu précédement dans la partie prédiction. Cependant, comme nous avons entraîné notre modèle uniquement sur **deux variables**, **les plus représentatives de la réalite (petal length et petal width)**. Nous utiliserons donc uniquement ces deux dernières variables dans un numpy array pour éffectuer les prédictions.

new_data = np.array([3.76, 1.2])

Prédiction avec les nouvelles données 2

Si on alimente le modèle comme suit :

knn_final.predict(np.array(new_data))

Nous obtenons une erreur : *ValueError: Expected 2D array, got 1D array instead:*

Souvenez-vous, nous avons avons entraîné le modèle sur un 2D DataFrame, donc le modèle attend un 2D array, qui pourrait être un numpy array ou un pandas DataFrame. Or new_data est un 1D, nous devons le transformer en 2D comme indiqué dans le message d'erreur.

new_data = new_data.reshape(1, -1)

Prédiction avec les nouvelles données 3

Le modèle prédit que **l'espèce** en question est **une ['versicolor']**.

Attention !

La méthode **model.predict()** peut également prendre comme pramètre une liste à 2D. Par exemple, *knn_final.predict([[3.76, 1.2]])* prédit le même résultat.

Exercice d'application 13

Prédiction sur de nouvelles données

1) Quelle est la prédiction pour une fleur d'iris ayant les caractéristiques suivantes :

petal_len = 5.03 cm et petal_wd = 2.25 cm

Probabilités de prédiction avec les nouvelles données 1

Trois fleurs d'iris ont la même petal width (1.2cm), mais différente petal length : 5.25cm, 3.76cm et 1.58cm.

new_data = np.array([[3.76, 1.2], [5.25, 1.2], [1.58, 1.2]])

knn_final.predict(np.array(new_data))

Les prédictions correspondantes sont : ['versicolor', 'virginica', 'setosa']

Les probabilités de prédiction sont données par :

knn_final.predict_proba(new_data)

Probabilités de prédiction avec les nouvelles données 2

Les probabilités de prédiction sont :

```
array([[0. , 1. , 0. ],
       [0. , 0.4, 0.6],
       [1. , 0. , 0. ]])
```

La somme de chaque ligne est égale à 1.

Pour la deuxième fleur, notre modèle prédit qu'il y a une probabilité de 40% qu'elle soit une versicolor et 60% qu'elle soit une virginica. Ce qui est en règle avec les prédictions obtenues.

Exercice d'application 14

Probabilités de prédiction sur de nouvelles données

1) Quelles sont les probabilités de prédiction pour des fleur d'iris ayant les caractéristiques suivantes :

new_data = np.array([[3.76, 1.2], [1.58, 1.2]])

Quiz 1

1) Définition

Définir une variable catégorielle.

2) Exemples

Citer 5 exemples de variables catégorielles.

3) la méthode count_values()

Que fait la méthode count_values() ?

Quiz 2

4) Scatter plot

Construire un scatter plot entre petal length et sepal width (utilisez les données d'iris).

5) Entraînement du modèle

Dans le module sklearn.neighbors, quelle est la classe qui permet de réaliser la classification ?

Quiz 3

6) Prédiction

import numpy as np
from sklearn.metrics import confusion_matrix
y_true = np.array(['cat', 'dog', 'dog', 'cat', 'fish', 'dog', 'fish'])
y_pred = np.array(['cat', 'cat', 'cat', 'cat', 'fish', 'dog', 'fish'])
confusion_matrix(y_true, y_pred, labels=['cat', 'dog', 'fish'])
Quelle est l'output de la matrice de prédiction ?

....
....
0 0 2

Data Science

Chap 6 : Clustering (regroupement)

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

March 28, 2022

"Le point de départ de toute réalisation est le désir. Gardez cela toujours à l'esprit. Un désir faible donne de maigres résultats, tout comme un feu chancelant réchauffe peu." Napoléon Hills

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons travailler sur les différentes étapes de la Data Science en Python.

À la fin de ce chapitre, vous serez en mesure :

- de réaliser une analyse exploratoire
- de réaliser une modélisation et une prédiction
- de calculer une précision et la matrice de confusion
- d'effectuer la validation croisée, le tuning des paramètres et de réaliser les prédictions sur les nouvelles données

Introduction 1

Definition

Le **clustering** est un **algorithme d'apprentissage non supervisé** permettant de regourper des données en sous-groupes (clusters) ayant les mêmes caractéristiques intra-groupes, tout en se distinguant d'autres sous-groupes.

On l'utilise souvent lorsque **nous ne connaissons pas les vrais valeurs (valeurs observées, les labels/étiquettes) de la variable cible**.

Introduction 2

Quelques exemples d'utilisation de cet algorithme dans notre vie quotidienne sont : regrouper les documents, les musiques, les films en se basant sur leur contenu ou encore de trouver un segment de marché en se basant sur les comportements d'achat des clients.

Attention !

Le but du clustering est de séparer les données en groupes, ou clusters, ayant plus de traits similaires entre eux par rapport aux données des autres clusters.

Différents types d'algorithmes de clustering

Il existe plus d'une centaine d'algorithmes de clustering. 12 d'entre eux ont été implementés dans Scikit-learn, mais peu ont gagné en popularité.
De façon générale, il y a 4 types de modèles :
- **Les modèles basés sur les centroïdes** : chaque cluster est représenté par un vecteur moyen unique (par exemple, k-means).
- **Les modèles basés sur la connectivité** : ils sont construits sur la base de la connectivité de la distance (par exemple, clustering hiérarchique).
- **Les modèles basés sur la distribution** : ils sont construits en utilisant des distributions statistiques (par exemple, les mélanges gaussiens).
- **Les modèles basés sur la densité** : les clusters sont définis comme des zones denses (par exemple, DBSCAN).
Veuillez cliquer ici pour plus de details.

L'algorithme K-means 1

K-means est l'un des algorithmes de clustering les plus populaires. L'algorithme fonctionne comme suit :
soit n données :
Étape 1 : L'initialisation : l'algorithme effectue un choix aléatoire de donnée k dans n et les définissent comme centre des clusters appélé centroïde.
Étape 2 : Assignation dans les clusters : Chaque donnée est assigné à son centroïde le plus proche en fonction de sa distance par rapoort à chaque centroïde, ce qui forme k clusters.
Étape 3 : Mise à jour des centroïdes : pour chaque nouveau cluster, on calcule son centroïde en prennant la moyenne de tous les points assignés à ce cluster.
Étape 4 : Répétition des étapes 2 et 3 : On repéte l'étape 2 et 3 jusqu'à ce qu'aucune des affectations dans les clusters changent ou atteignent le maximun d'itérations.

L'algorithme K-means 2

L'algorithme K-means est implementé dans Scikit-learn dans le module sklearn.cluster :

from sklearn.cluster import KMeans

Attention !

Cet algorithme est très populaire, car il est facile à mettre en oeuvre et s'adpate bien à des gros volumes de données. Cependant, il est difficile de prédire le nombre de clusters, car il peut rester bloquer sur des optimums locaux. De plus, ses performances peuvent être médiocres lorsque les clusters sont de taille et de densité variables.

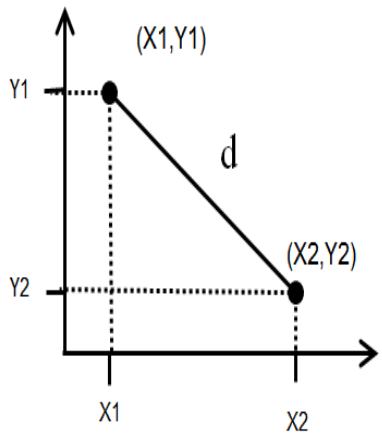
Exercice d'application 1

Étapes de K-means

Décrire comment fonctionne l'algorithme de clustering K-means.

La métrique distance 1

La distance calculée dans le k-means peut-être **la distance euclidienne**, qui est une ligne droite entre deux points.



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

La métrique distance 2

Soit deux points X1(0, 1), X2(2, 0), le calcul de la distance euclidienne entre ces deux points sous Numpy est :

$$\sqrt{(0-2)^2+(1-0)^2} = \sqrt{(2)^2+(1)^2} = \sqrt{5}$$

```
import numpy as np
X1 = np.array([0, 1])
X2 = np.array([2, 0])
print(np.sqrt(((X1-X2)**2).sum()))
```

2.23606797749979

```
print(np.sqrt(5))
```

2.23606797749979

La métrique distance 3

Soient deux points $P(p_1, p_2, ..., p_n)$ et $Q(q_1, q_2, ..., q_n)$ dans un espace à n dimensions le calcul de la distance euclidienne est donnée par la formule de Pythagore :

$$d(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

Attention !

Il existe d'autres métriques tels que la distance de Manhattan, la distance cosinus, etc. Le choix de la distance à utiliser depend des données.

Exercice d'application 2

Distance euclidienne

Calculez la distance euclidienne entre X(2,4) et Y(3,2).

Les données sur les vins du sud de l'Italie

Dans ce module, nous analysons le résultat d'une analyse chimique de vins cultivés dans une région d'Italie. Le but est d'essayer de regrouper les observations similaires et de déterminer le nombre de clusters possibles. Cela va permettre de réaliser des prédictions et à réduire la dimension. Il y a 13 variables pour chaque vin, et si nous pouvions regrouper tous les vins dans 3 groupes, alors cela réduirait l'espace à 13 dimensions à un espace à 3 dimensions.

Les données sur les vins

L'analyse a rapporté les quantités de 13 composants de 178 vins : alcool, acide malique, cendres, alcalinité des cendres, magnésium, phénols totaux, flavanoïdes, phénols non flavanoïdes, proanthocyanines, intensité de la couleur, teinte, od280/od315 des vins dilués et proline.

Chargement des données

```
import numpy as numpy
import pandas as pd
from sklearn.datasets import load_wine
data = load_wine()
wine = pd.DataFrame(data.data, columns=data.feature_names)
print(wine.shape)
print(wine.columns)
```

Statistiques descriptives

wine.iloc[:,:3].describe()

	alcohol	malic_acid	ash
count	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517
std	0.811827	1.117146	0.274344
min	11.030000	0.740000	1.360000
25%	12.362500	1.602500	2.210000
50%	13.050000	1.865000	2.360000
75%	13.677500	3.082500	2.557500
max	14.830000	5.800000	3.230000

Nous constatons que les caractéristiques ne sont pas toute à la même échelle. Nous les mettrons à l'échelle plus loin.

Exercice d'application 3

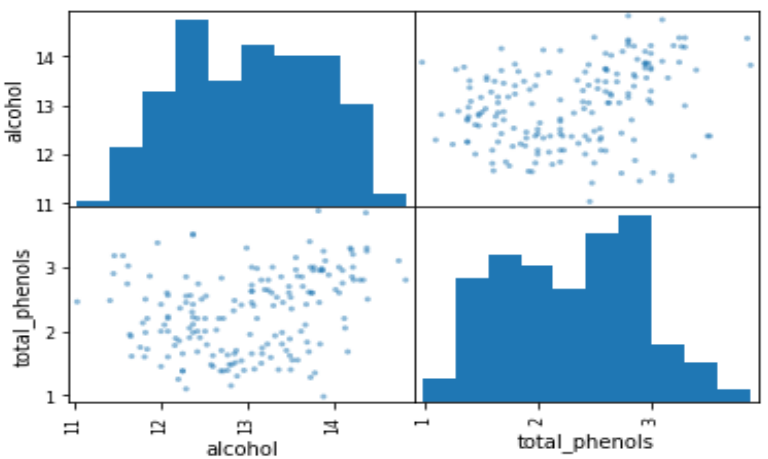
Summary statistics

Calculez les summary statistics des 5 premières variables.

Analyse visuelle des données 1

Nous utilisons une fonction de traçage pour construire les histogrammes en diagonales et les cross-plots dans les zones restantes :

scatter_matrix(wine.iloc[:,[0,5]])



Analyse visuelle des données 2

Comme nous ne connaissons pas les vraies valeurs de la variable cible, nous examinons les diagrammes croisés pour trouver un candidat raisonnable pour k, le nombre de clusters. Il semble y avoir environ trois sous-groupes.
Rappelez-vous qu'il n'y a pas de bonnes ou de mauvaises réponses pour le nombre de sous-groupes. Dans les données du monde réel, il est rare que nous trouvions des groupes clairs, mais nous faisons notre meilleure estimation. Par exemple, dans le nuage de points ci-dessus, il semble y avoir trois sous-groupes.

Analyse visuelle des données 3

Attention !

Qu'il s'agisse d'un problème d'apprentissage supervisé ou non supervisé, l'analyse exploratoire des données est essentielle et fortement recommandée avant de se lancer dans la modélisation.

Pré-traitement des données 1

Après l'analyse visuelle des cross-plots, nous choisissons deux variables pour mieux illustrer l'algorithme : alcoohol et total_phenols dont l'analyse bivariée suggère trois(3) sous-groupes.
$X = wine[['alcohol', 'total\_phenols']]$
Contrairement aux modèles d'apprentissage supervisé, les modèles d'apprentissage automatique non supervisé ne nécessitent généralement pas de diviser les données en ensembles d'apprentissage et de test puisqu'il n'y a pas de variabl cible observée pour valider le modèle.

Pré-traitement des données 2

Cependant, les algorithmes basés sur les centroïdes nécessitent une étape de prétraitement car les k-means fonctionnent mieux sur des données où chaque attribut est d'une échelle similaire. Une façon d'y parvenir est de normaliser les données :

z = (x - mean) / std

z est une donnée standardisée, centrée à 0 et de variance unitaire. Le m**odule sklearn.preprocessing** qui contient la classe **StandardScaler** permet de réaliser cette opération de **standardisation**.

Pré-traitement des données 3

X = wine[['alcohol', 'total_phenols']]
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scale.fit(X)
Tableau des moyennes
print(scale.mean_) $\Longrightarrow$ [13.00061798 2.29511236]
Tableau des écart-types :
print(scale.scale_) $\Longrightarrow$ [0.80954291 0.62409056]
La moyenne de la variable alcohol est 13.0 et son écart-type est 0.809.

Pré-traitement des données 4

Une fois le travail de normalisation terminé, nous pouvons maintenant calibrer et transformer notre modèle :
X_scaled = scale.transform(X)
print(X_scaled.mean(axis=0)) $\Longrightarrow$ [7.84141790e-15
-1.95536471e-16]
print(X_scaled.std(axis=0)) $\Longrightarrow$ [1. 1.]

Une bonne pratique consiste à mettre à l'échelle les variables avant d'entraîner le modèle si les algorithmes sont basés sur la distance.
Veuillez cliquer ici pour plus de detais.

Modélisation 1

Comme la régression linéaire ou les knn, tout comme n'importe quel algorithme d'apprentissage dans Scikit-learn, nous devons suivre le workflow suivant : **instantiate** ==> **fit** ==> **predict**. Nous allons définir le nombre de clusters n_clusters à 3 et laisser les autres paramètres (la méthode pour initialiser les centroïde, le critère d'arrêt de l'algorithme, etc.) à leur valeur par défaut.

from sklearn.cluster import KMeans
instantiate the model
kmeans = KMeans(n_clusters=3)
fit the model
kmeans.fit(X_scaled)
make predictions
y_pred = kmeans.predict(X_scaled)
print(y_pred)

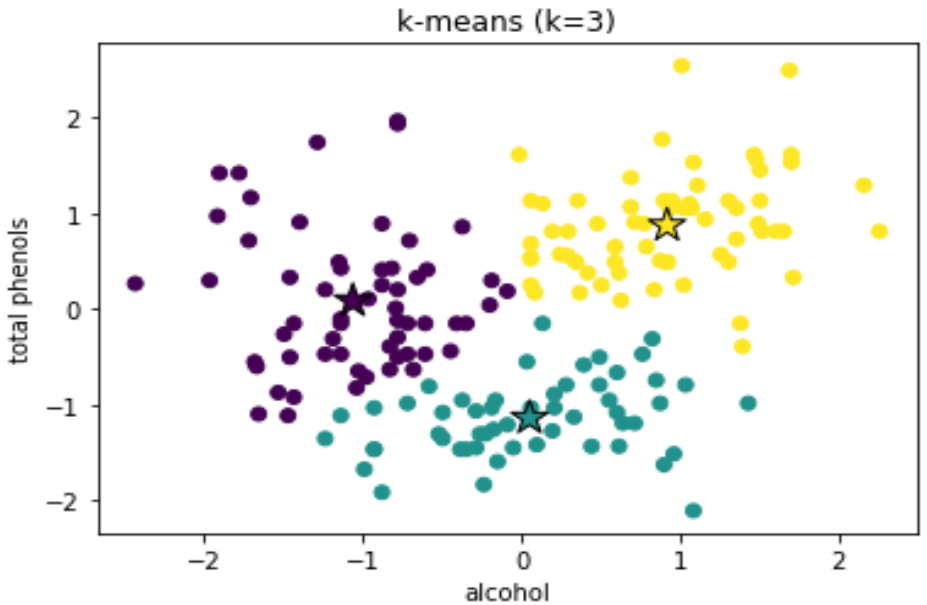
Modélisation 2

Il y a 59 vins dans le cluster 0, 54 dans le cluster 1 et 65 dans le cluster 3.

Pour voir les coordonnées des 3 centroïdes :
*kmeans.cluster_centers_*

import matplotlib.pyplot as plt
plot the scaled data
plt.scatter(X_scaled[:,0],X_scaled[:,1],c= y_pred)
identify the centroids
plt.scatter(kmeans.cluster_centers_[:, 0],kmeans.cluster_centers_[:, 1],marker="",s = 250,c = [0,1,2],edgecolors='k')*
plt.xlabel('alcohol'); plt.ylabel('total phenols')
plt.title('k-means (k=3)')
plt.show()

Modélisation 3



Les étoiles sont les centroïdes. Les K-means divisent les vins en trois groupes : faible taux d'alcool mais phénols totaux élevés (en haut à droite en jaune), taux d'alcool élevé et phénols totaux élevés (en haut à gauche en violet), et phénols totaux faibles (en bas en vert).

Modélisation 4

Pour tout vin nouveau présentant le rapport chimique sur l'alcool et les phénols totaux, nous pouvons maintenant le classer en fonction de sa distance à chacun des centroïdes. Supposons qu'il y ait un vin nouveau avec un taux d'alcool de 13 et un taux de phénols totaux de 2.5, prédisons à quel cluster le modèle va attribuer le vin nouveau.

Premièrement, mettons les nouveaux données dans un 2D array :
X_new = np.array([[13, 2.5]])
Ensuite, standardisons les nouvelles données :
X_new_scaled = scale.transform(X_new)
print(X_new_scaled)

Modélisation 5

Nous pouvons maintenant procéder à la prédiction du cluster auquel appartiendra le nouveau point.
print(kmeans.predict(X_new_scaled)) $\Longrightarrow$ [2]
Le nouveau vin est donc plus proche du controïde du cluster 2 dont le centroïde est (0.92017418, 0.87087204)

Une des limites de kmean standard est l'initialisation aléatoire des centroïdes. L'algorithme k-means++ résout cet obstacle en spécifiant une procédure d'initialisation des centroïdes avant de procéder à l'algorithme k-means standard. Dans scikit-learn, le mécanisme d'initialisation est défini sur k-means++, par défaut.

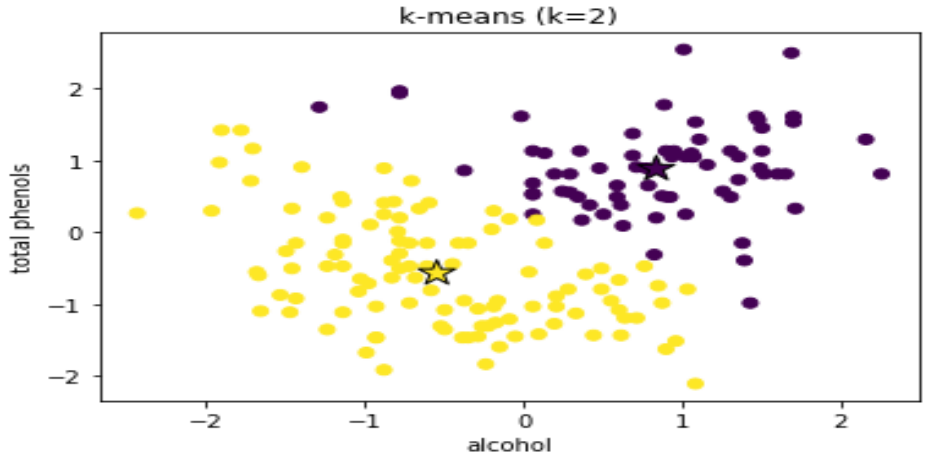
Exercice d'application 4

Complétez le programme suivant de recherche des centroïdes

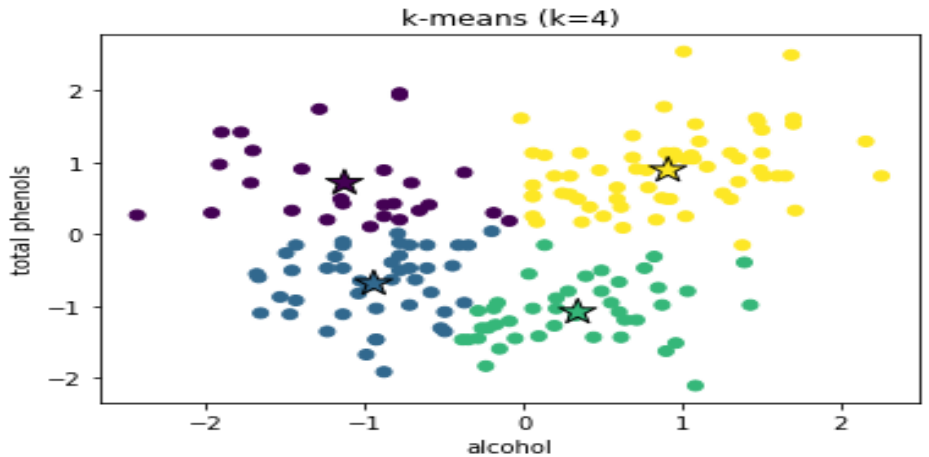
from sklearn. ..... import KMeans
..... = KMeans(n_clusters=3)
kmeans.fit(X_scaled)
*kmeans.cluster_centers_*

Recherche du k optimal : la méthode du coude 1

$k = 2$

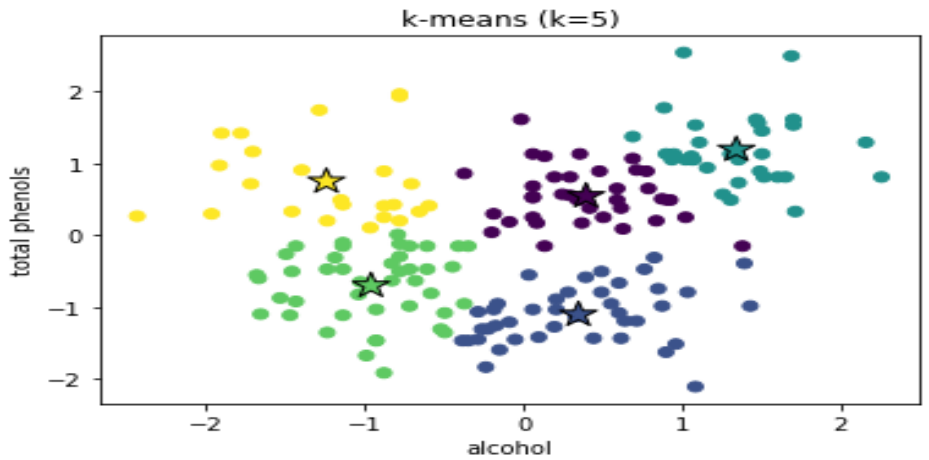


$k = 4$



Recherche du k optimal : la méthode du coude 2

$k = 5$



Nous pouvons ainsi diviser les données en un nombre entier quelconque de clusters allant de 1, un cas extrême où toutes les données appartiennent à un grand cluster, à n, un autre cas extrême où chaque donnée est son propre cluster.

Quel k (nombre de clusters) devrons-nous choisir entre 2, 3, 4, 5 ?

Exercice d'application 5

Construire le graphique correspondant à k = 5

Contruisez le graphe correspondant à k = 5.

Recherche du k optimal : la méthode du coude 3

K-means répartit les données en k clusters tels que les données d'un cluster sont plus proches que les données des autres clusters.

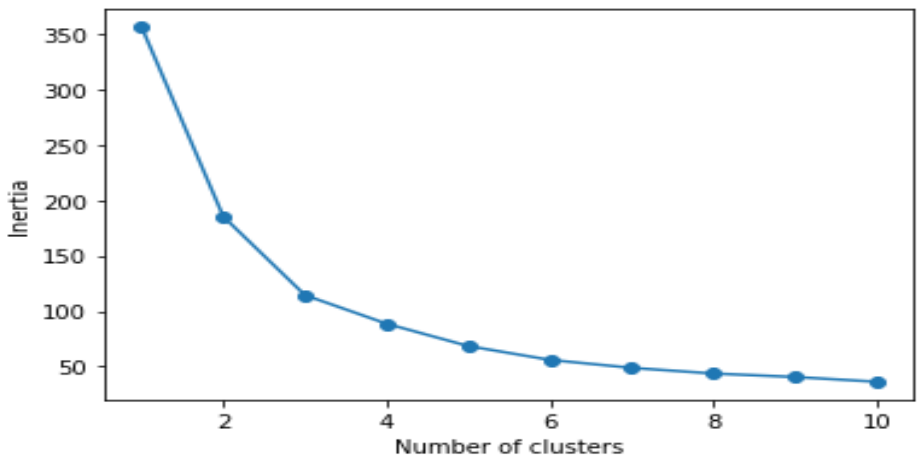
Definition

La densité/proximité/inertie/distorsion des données autour du centre du cluster est mesurée comme la somme des carrés de la distance d'un point et son centroïde le plus proche.

Dans Scikit-learn, l'attribut correspondant à l'inertie est **inertia_**.
Par exemple, lorsque k = 2, la distorsion est de 185.

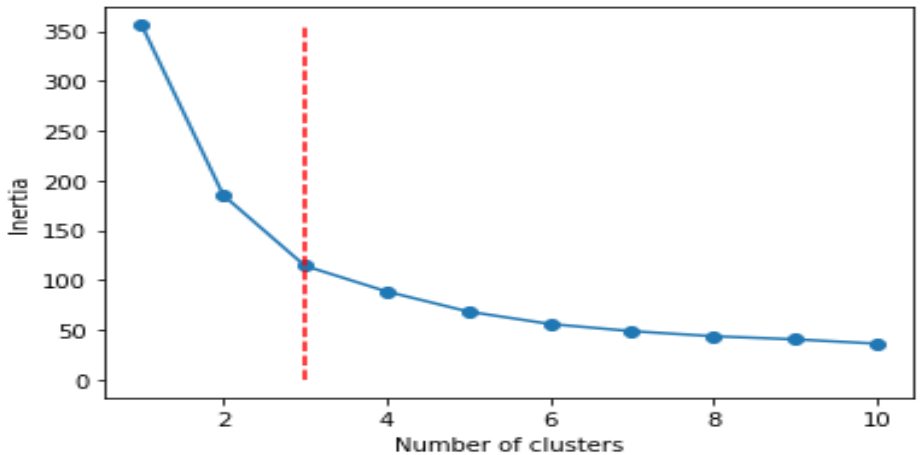
```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X_scaled)
kmeans.inertia_
```

Mesure de l'inertie pour différente valeur de k de 1 à 11



Comme le montre le graphique, l'inertie diminue lorsque le nombre de clusters augmente. **Le k optimal devrait être celui où l'inertie ne diminue plus aussi rapidement**.

Choix du k optimal 1



Par exemple, **k=3 semble être optimal, au fur et à mesure que l'on augmente le nombre de clusters de 3 à 4, la diminution de l'inertie ralentit significativement, par rapport à celle de 2 à 3**. Cette approche est appelée **méthode du coude** (Elbow method). Il s'agit d'un outil graphique utile pour estimer le k optimal dans les k-means.

Choix du k optimal 2

Attention !

Une seule valeur de l'inertie ne permet pas de déterminer le k optimal, car plus k est grand, plus l'inertie sera faible.

Exercice d'application 6

Choisir la bonne réponse

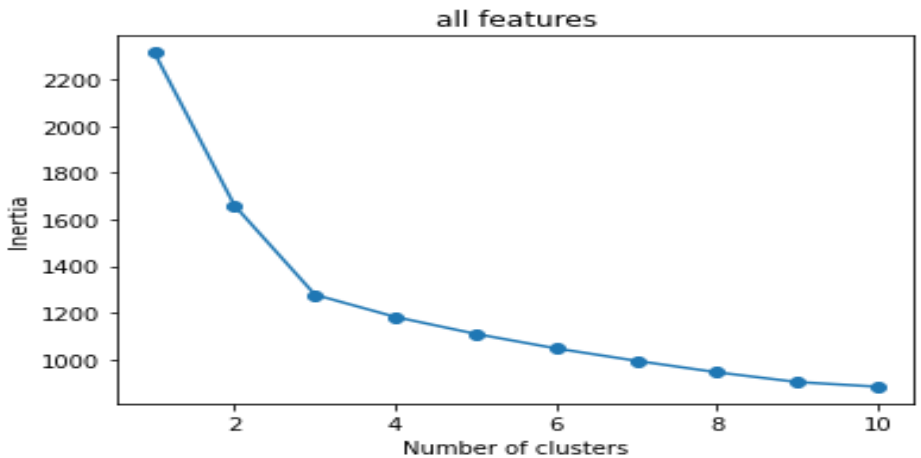
1. L'inertie dimunie lorsque k dimunie
2. L'inertie dimunie lorsque k augmente
3. k ne peut pas être plus grand que 5

Modèlisation avec plus de variables

Plus haut, nous avons construit le modèle de kmeans en utilisant juste deux(02) variables parmi treize (13) variables au total. Ce choix s'est fait de façon arbitraire et c'est facile de construire les graphiques correspondants aux résultats. Et si nous utilisons toutes les variables ?

X= *wine*
Ne pas oublier de standardiser chaque variable :
scale = *StandardScaler()*
scale.fit(X)
X_scaled = *scale.transform(X)*

Modèlisation avec plus de variables



De même, nous constatons que l'inertie ne diminue plus aussi rapidement après k = 3. Nous finalisons ensuite le modèle en fixant n_clusters = 3

Prédictions avec le k optimal

```
k_opt = 3
kmeans = KMeans(k_opt)
kmeans.fit(X_scaled)
y_pred = kmeans.predict(X_scaled)
print(y_pred)
```

Comparaison des prédictions des deux modèles 1

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2]

contre **le premier modèle à 2 variables (X = wine[['alcohol', 'total_phenols']]) et k = 3**

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 2 0 0 0 1 0
1 0 2 1 2 1 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 0 0 2
0 0 0 2 2 2 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 1 0 0 0 0 0 0 2 2 2 2 0 2
2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 1 1 0 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 1]

Comparaison des prédictions des deux modèles 2

Par rapport aux prédictions utilisant seulement deux variables, les deux modèles produisent des résultats très similaires. Par exemple, les deux modèles prédisent que les 21 premiers vins appartiennent au même groupe, tout comme les 19 derniers vins. En fait, seuls 13 des 178 vins ont été classés différemment par les deux modèles.

Il est naturel de se demander quel modèle est le meilleur. Rappelons que le clustering est une méthode d'apprentissage non supervisée, ce qui signifie que nous ne connaissons pas la vérité de base des étiquettes/labels. Il est donc difficile, voire impossible, de déterminer que le modèle avec 2 variables est plus précis pour regrouper les vins que celui avec les 13 caractéristiques, ou vice versa.

Exercice d'application 7

Affichage des valeurs prédictes

Affichez les 22 premières prédictions de chaque modèle (à deux variables et avec toutes les variables), faîtes de même pour les 19 dernières prédictions.

Que constatez-vous ?

Comparaison des prédictions des deux modèles 3

Le choix du modèle, c'est-à-dire des variables, est souvent déterminé par des informations externes. Par exemple, le service marketing veut savoir si une stratégie spécifique à un continent est nécessaire pour vendre ces vins. Nous avons maintenant accès aux informations démographiques des consommateurs et les trois clusters identifiés à partir du modèle A correspondent mieux aux clients d'Europe, d'Asie et d'Amérique du Nord respectivement que le modèle B ; alors le modèle A est le gagnant. Il s'agit d'un exemple trop simplifié, mais vous avez compris l'essentiel.

Attention !

Dans la pratique, les variables sont souvent choisies par la collaboration entre les data scientists et les experts du domaine.

Exercice d'application 8

Choisir la bonne réponse

Il existe une méthode universelle pour choisir le nombre optimal de variables à utiliser dans les problèmes de clustering.

1. Vrai
2. Faux

Quiz 1

1) Définition

La segmentation des clients basée sur le comportement d'achat en est un exemple :

1. de régression linéaire
2. de clustering
3. de classification

2) Apprentissage non supervisé

les modèles d'apprentissage automatique non supervisés ne nécessitent généralement pas de diviser les données en ensembles d'entraînement et de test :

1. Vrai
2. Faux

Quiz 2

3) Définition

Compétez le programme suivant permettant de standardiser les données :

..... sklearn.preprocessing
import .....
X = wine[['alcohol', 'total_phenols']]
..... = StandardScaler()
scale. ..... (X)
X_scaled = scale.transform(.....)

Quiz 3

3) Calcul des centroïdes

Lequel permet de determiner les centroïdes :

1. kmeans.centroids_
2. kmeans.cluster_centroids_
3. kmeans.cluster_centers_

4) K-means versus knn

K-means est un algorithme d'apprentissage supervisé comme knn :

1. Vrai
2. Faux