# Machine Learning (ML)

## Chap 1 : Les concepts de base

David TCHOUTA

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

May 6, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Introduction
3. Machine Learning
4. Statistics
5. Manipulation
6. Scatter plot
7. Quiz

# Objectifs du chapitre

Dans ce premier chapitre, nous allons apprendre les concepts de base du Machine Learning (ML).

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

- de définir le Machine Learning
- de calculer les statistiques descriptives
- d'importer, de lire et de manipiler les données avec Pandas
- de faire des calculs avec Numpy
- de construire les graphiques avec Matplotlib

# Definition

## Definition

Le Machine Learning (ML) est la façon dont nous prenons les données et les transformons en information. Nous utilisons la puissance de calcul des ordinateurs pour analyser les données du passé et prédire les résultats des nouvelles données.

Le Machine Learning est de plus en plus répandu dans nos usages quotidiens. Par exemple, lorsque l'algorithme de Netflix récommande un film, il se base sur les films que les autres utilisateurs ont regardés (y compris nous-même) pour faire cette recommandation. De même que les prix pratiqués par Amazon se basent sur la manière dont les articles similaires ont été vendus par le passé. Il en va de même pour la surveillance des financements illicites.

# Les prérequis

Dans ce chapitre, nous utilisons bien évidemmnet le langage de programmation Python, ce qui suppose que l'apprenant a un bon niveau en Python.

Les packages tels que Pandas (lecture des données), Numpy (calcul des données numériques), Matplotlib (construire les graphiques) et Scikit-learn (manipulation des modèles de ML) seront utilisés.

**Attention !**

Nous allons couvrir la théorie et la pratique des modèles de ML, en mettant l'accent sur la manière dont ils sont utilisés dans le monde réel.

# Contenu du cours 1

En ML, on distingue l'apprentissage supervisé de l'apprentissage non supervisé.

On parle **d'apprentissage supervisé lorsque nous connaissons la valeur réelle de la variable cible** (par exemple, prédire à quel prix une maison sera vendu) tandis que **l'apprentissage non supervisé** est determiné par le fait que **la réponse passée n'est pas connu** (par exemple, déterminer les sujets de discussion dans une revue d'économie).

# Contenu du cours 2

Nous allons nous concentrer sur les algorithmes d'apprentissage supervisé tels que la régression (prédire une valeur numérique, par exemple la croissance économique d'une région) et la classification (prédire à quelle classe appartient une donnée, par exemple prédire si une transaction est une fraude ou pas). Nous allons nous concentrer sur les problèmes de classification.

Nous allons prédire quel passager survivra au crash du Titaninc, déterminer un chiffre manuscrit à partir d'une image ou encore utiliser les données d'une biopsie pour déterminer si une cellule est cancéreuse ou pas.

# Contenu du cours 3

Nous verrons en detail les algorithmes suivants :

1. la regréssion logistique (**Logistic Regression**)
2. les arbres de décision (**decision Trees**)
3. les forêts aléatoires (**Random Forests**)
4. les réseaux de neurones (**Neural Networks**)

# Les statistiques de base

**La moyenne (mean)** est la somme des valeurs de la distribution divisée par le nombre d'elements de la distribution.

**La médiane (median)** est la valeur centrale d'une distribution.

Soit la distribution d'âge suivante : 15, 16, 18, 19, 22, 24, 29, 30, 34

moyenne = (15 + 16 + 18 + 19 + 22 + 24 + 29 + 30 + 34) / 9 = 207/9 = 23

La médiane est la cinquième valeur (22) puisqu'il y a 9 valeurs au total.

# Exercice d'application 1

## Moyenne et médiane

Calculer la moyenne est la médiane de la distribution suivante : 0,1,1,2,6

# Les percentiles 1

la **médiane** correspond au **50e percentile (deuxième quartile)**, ce qui signifie que 50% des données sont inférieures à la médiane et 50% des données sont supérieures à la médiane. Elle nous renseigne sur l'endroit où se situe la valeur centrale. Nous nous intéressons également au **25e percentile (premier quartile)** et au **75e percentile (troixième quartile)**.

Le 25e percentile est la valeur qui se trouve à un quart des données. C'est la valeur pour laquelle 25% des données sont inférieures à cette valeur (et 75% des données sont supérieures à cette valeur)

# Les percentiles 2

De même, le 75e percentile correspond aux trois quarts des données. Il s'agit de la valeur pour laquelle 75% des données sont inférieures à cette valeur (et 25% des données sont supérieures à cette valeur).

Reprenons notre distribution de l'âge : 15, 16, 18, 19, 22, 24, 29, 30, 34

Nous avons 9 valeurs, donc 25% des données correspondraient à la 2e valeur environ. La 3e valeur est supérieure à 25% des données, donc le 25e percentile est 18.

De même, 75% des données correspondent à la 6e valeur environ. La 7e valeur est donc supérieur à 75% des données. Ainsi, le 75e percentile est de 29 (la 7e valeur des données).

# Les percentiles 3

**Attention !**

S'il y a un nombre pair de données, pour trouver la médiane (ou le 50e percentile), vous prenez la moyenne des deux valeurs du milieu.

# Exercice d'application 2

## Percentiles

Soit la distribution suivante : 0, 1, 1, 1, 1, 1, 2, 2, 2, 3, 6

1) le 25e percentile est : .....

2) le 50e percentile est : .....

3) le 75e percentile est : .....

# Écart-type & Variance 1

Pour avoir une compréhension plus poussée de la distribution, nous pouvons calculer **l'écart-type et la variance**, qui répresentent **les mesures de dispersion des données**.
**Nous mesurons de combien un point se situe par rapport à la moyenne**.

Soit la distribution suivante : **15, 16, 18, 19, 22, 24, 29, 30, 34**
Sa moyenne est de 23. Si nous calculons la distance de chaque point par rapport à la moyenne, (23-15 = 8 pour la première valeur), on a le tableau suivant :

8, 7, 5, 4, 1, 1, 6, 7, 11

On élève ces valeurs au carré et on les additionne :
$8^2 + 7^2 + 5^2 + 4^2 + 1^2 + 1^2 + 6^2 + 7^2 + 11^2 = 362$

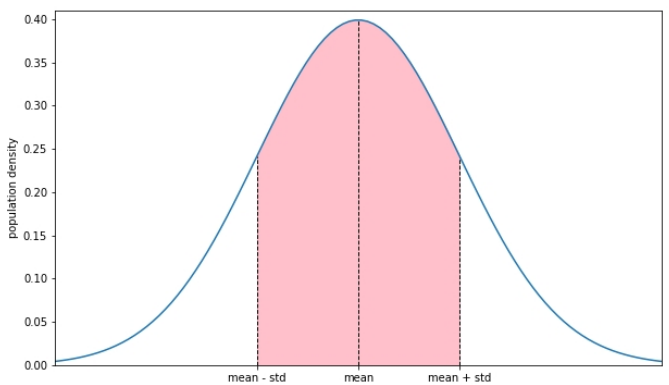
# Écart-type & Variance 2

Si nous **divisons cette valeur par le nombre total de valeurs** et cela nous donne **la variance** : $362 / 9 = 40.22$.

En prenant **la racine carré de la variance**, nous obtenons **l'écart-type** : $\sqrt{40.22} = 6.34$

# Écart-type & Variance 3

Si nos données sont normalement distribuées comme dans le graphique ci-dessous, 68% de la population se situe dans la fourchette d'un écart-type de la moyenne. Dans le graphique, nous avons mis en évidence la zone située à l'intérieur d'un écart-type de la moyenne. Vous pouvez voir que la zone rose représente environ deux tiers (plus précisément 68%) de la surface totale sous la courbe. Si nous supposons que nos données sont normalement distribuées, nous pouvons dire que 68% des données se situent dans un écart-type de la moyenne.



# Écart-type & Variance 4

Dans notre exemple, bien que les âges ne soient probablement pas exactement distribués normalement, nous supposons qu'ils le sont et disons qu'environ 68% de la population a un âge compris dans un écart-type de la moyenne. Comme la moyenne est de 23 et l'écart-type de 6,34, nous pouvons dire qu'environ 68% des âges de notre population se situent entre 16,66 (23 - 6,34) et 29,34 (23 + 6,34).

**Attention !**

Même si les données ne présentent jamais une distribution normale parfaite, nous pouvons tout de même utiliser l'écart-type pour avoir une idée de la façon dont les données sont distribuées.

# Exercice d'application 3

## Écart-type & Variance

1) Qu'est-ce que la variance ?

2) Qu'est-ce que l'écart-type ?

3) Comment sont-ils calculés ?

4) Que traduit un écart-type élévé ou faible ?

# Numpy : Calcul Statistique 1

Nous allons utiliser les **fonctions de Numpy** pour le calcul des statistique tel que la **moyenne (mean), la médiane (median), les percentiles, std, var**.

Importons Numpy et initialisons la variable data pour avoir la liste des âges :

```
import numpy as np
data = [15, 16, 18, 19, 22, 24, 29, 30, 34]
print("mean:", np.mean(data))
print("median:", np.median(data))
print("50th percentile (median):", np.percentile(data, 50))
print("25th percentile:", np.percentile(data, 25))
print("75th percentile:", np.percentile(data, 75))
print("standard deviation:", np.std(data))
print("variance:", np.var(data))
```

# Exercice d'application 4

## Calculez les percentiles

*import numpy as np*

*print(np. ..... (data, 70))*

# Pandas : Lire et Manipuler les données 1

Pandas est un module Python qui nous aide à lire et à manipuler des données. Ce qu'il y a de bien avec Pandas, c'est que vous pouvez prendre des données et les afficher sous la forme d'un tableau lisible par l'homme, mais elles peuvent aussi être interprétées numériquement, ce qui vous permet d'effectuer de nombreux calculs.

Nous appelons le tableau de données un DataFrame.

# Exercice d'application 5

## L'objet Pandas

Comment appelle-t-on l'objet Pandas (Pandas data object) ?

# Pandas : Lire et Manipuler les données 2

Pour importer Pandas : *import pandas as pd*

Nous allons travailler avec la base de données des passagers du Titanic. Pour chaque passager, nous aurons des données sur eux et nous saurons s'ils ont survécu ou non à l'accident.

Nos données sont stockées dans un fichier CSV (comma-separated values). Le fichier titanic.csv se trouve à `https://gist.github.com/fyyying/4aa5b471860321d7b47fd881898162b7`.

# Pandas : Lire et Manipuler les données 3

La première ligne est l'en-tête, puis chaque ligne suivante est constituée des données d'un seul passager.

Chargons les données dans Pandas de manière à obtenir un Dataframe :

*import pandas as pd*
*data = pd.read_csv(titanic.csv)*
*print(data.head())*

```
In [7]: print(titanic.head())
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S
```

# Exercice d'application 6



## Lire les données

Écrire un programme qui permet d'afficher les 5 premières lignes et les 5 premières colonnes (de Passengerd jusqu'à Sex) de la base titanic.

# Summary Statistics 1

Dans Pandas , **la méthode describe()** permet d'afficher **les statistiques descriptives**.
Pour forcer Python d'afficher un certain nombre de columns :
*pd.options.display.max_columns = 4*
*print(data.describe())*

```
In [76]: print(titanic.describe())
       PassengerId    Survived  ...       Parch        Fare
count   891.000000  891.000000  ...  891.000000  891.000000
mean    446.000000    0.383838  ...    0.381594   32.204208
std     257.353842    0.486592  ...    0.806057   49.693429
min       1.000000    0.000000  ...    0.000000    0.000000
25%     223.500000    0.000000  ...    0.000000    7.910400
50%     446.000000    0.000000  ...    0.000000   14.454200
75%     668.500000    1.000000  ...    0.000000   31.000000
max     891.000000    1.000000  ...    6.000000  512.329200
```

NB : Seules les Statistiques de colonnes numériques sont affichées.

# Summary Statistics 2

Signification des indicateurs affichés :
**count :** C'est le nombre de lignes ayant une valeur. Dans notre cas, chaque passager a une valeur pour chaque colonne, donc la valeur est 891 (le nombre total de passagers).
**mean :** c'est la moyenne
**std :** c'est l'écart-type
**min :** c'est la plus petite valeur
**25% :** c'est le 25e percentile
**50% :** c'est le 50e percentile, ou encore la médiane
**75% :** c'est le 75e percentile
**max :** c'est la valeur la plus élévée
La méthode **describe()** permet d'avoir **quelques intuitions par rapport aux données (forme, dispersion, valeurs abérrantes, etc.)**.

# Exercice d'application 7

## Summary Statistics

Quelles sont les valeurs du maximum de Pclass et la médiane de age ?

# Selection d'une seule colonne (Serie)

Pour selectionner une **seule colonne**, on utilise les square brackets([ ]) et le nom de la colonne qu'on veut selectionner.
*col = data['age']*
*print(col)*

Le **résultat** est ce qu'on appelle une **Pandas Series**. **Une Serie est comme un DataFrame, avec juste une seule colonne**.

# Exercice d'application 8

## Type

col = titanic['Survived']

Quel est le type de col ?

# Selection de plusieurs colonnes

Pour selectionner plusieurs colonnes, nous mettons le nom des colonnes dans une liste :
['Survived', 'Age', 'Sex']. Puis nous insérons cette liste dans les brackets comme suit :

*subset = data[['Survived', 'Age', 'Sex']]*
*print(subset.head())*

**Attention !**

Lorsqu'on veut selectionner **une seule colonne**, on utilise les **single square brackets [ ]**. Lorsqu'on veut **récupérer plusieurs colonnes**, on utilise les **double square brackets [[ ]]**.

# Exercice d'application 9

## Selection des plusieurs colonnes

Écrire un programme permettant de selectionner les colonnes Fare, Age et SibSp.

# Création d'une colonne

Nous allons créer une nouvelle colonne **genre** de type boolean (True pour les male et False pour les female : *data['Sex']=='male'*) en se basant sur la colonne Sex : *data['genre'] = data['Sex'] == 'male'*

```
In [103]: print(titanic.head())
   PassengerId  Survived  ...  Embarked  genre
0            1         0  ...         S   True
1            2         1  ...         C  False
2            3         1  ...         S  False
3            4         1  ...         S  False
4            5         0  ...         S   True
```

**Attention !**

Souvent, nos données ne sont pas dans le format idéal. Heureusement, Pandas nous permet de créer facilement de nouvelles colonnes basées sur nos données afin de les formater de manière appropriée.

# Exercice d'application 10



## Création d'une nouvelle colonne

Écrire un programme pour créer la colonne "First Class", qui est True si le passager est dans la Pclasse 1 et False sinon.

# Numpy

Numpy est un module de Python qui permet de manipuler les listes (table data = numpy array) et les tableaux.

**Bien qu'on peut faire certaines opérations sous Pandas, Il est préférable d'utiliser Numpy pour les calculs sur les tableaux. Pandas a en fait été construit en utilisant Numpy comme base.**

# Conversion de Pandas à Numpy

**L'attribut values** permet **de convertir une Pandas Serie** en **numpy array** :

*data['Fare'].values*
*array([ 7.25 , 71.2833, 7.925, 53.1, 8.05, 8.4583, . . .*

Le résultat est un tableau à une dimension. Vous pouvez le constater puisqu'il n'y a qu'un seul jeu de parenthèses et qu'il ne s'étend que sur la page (et non vers le bas).

# Exercice d'application 11

## Conversion d'une Serie en Numpy array

Écrivez le code pour obtenir un Numpy array des valeurs de la colonne 'Age' à partir du DataFrame titanic.

# Conversion d'un Pandas DataFrame en Numpy array

Si nous avons **un Pandas DataFrame** (au lieu d'une Pandas Serie comme précédemment), nous pouvons toujours utiliser **l'attribut values**, mais **il renvoie un numpy array à 2 dimensions** :

*data[['Pclass', 'Fare', 'Age']].values*

```
array([[ 3.    ,  7.25  , 22.    ],
       [ 1.    , 71.2833, 38.    ],
       [ 3.    ,  7.925 , 26.    ],
       ...,
       [ 3.    , 23.45  ,     nan],
       [ 1.    , 30.    , 26.    ],
       [ 3.    ,  7.75  , 32.    ]])
```

C'est **un numpy** array à **deux dimensions**, car il y a **deux square brackets**.

# Exercice d'application 12

## Conversion d'un Pandas DataFrame en Numpy array

Quelle est la dimension du numpy array renvoyé par l'instruction suivante : *titanic[['Sex', 'Survived', 'Age']].values* ?

# L'attribut shape

Nous utilisons **l'attribut shape** pour déterminer la **taille de notre numpy array**. **La taille** nous indique le **nombre de lignes et de colonnes de nos données**.

*arr = data[['Pclass', 'Fare', 'Age']].values*
*print(arr.shape)* $\Longrightarrow$ (891, 3)

Ce résultat signifie que nous avons 891 lignes et 3 colonnes.

Vous pouvez également utiliser **l'attribut shape** sur un **Pandas DataFrame** (titanic.shape).

# Exercice d'application 13



## Taille d'un Numpy array

Quel est l'output des instructions suivantes ?

*arr = df[['Survived', 'Pclass']].values*

*print(arr.shape)*

# Selection des valeurs dans un Numpy Array

Soit le numpy array suivant :
*arr = df[['Pclass', 'Fare', 'Age']].values*
*print(arr[0, 1])*
Ce sera **la 2ème colonne de la 1ère ligne** (rappelez-vous que nous commençons à compter à 0). Ce sera donc le tarif du 1er passager, soit 7,25.

Nous pouvons également sélectionner **une seule ligne**, par exemple, la ligne entière du premier passager :
*print(arr[0])*

Pour sélectionner **une seule colonne** (la colonne Age par exemple) :
*print(arr[:,2])* $\Longrightarrow$ array([ 3. , 7.25, 22. ])

# Exercice d'application 14

## Selection

Soit le tabaleau suivant :

*arr = data[['Pclass', 'Fare', 'Age']].values*

Écrire un programme permettant d'afficher/selectionner les frais de tous les passagers.

# Selection des données en fonction critère

Créons un sous-ensemble de données des passagers ayant moins de 18 ans (mineurs) :

*mask = arr[:, 2] ¡ 18*
*print(arr[mask])*

Une autre façon d'obtenir le même résultat :
*print(arr[arr[:, 2] ¡ 18])*

**Attention !**

Un masque (mask) est un tableau booléen (True/False) qui nous indique les valeurs du tableau qui nous intéressent.

# Exercice d'application 15

## Selection

Soit le tableau suivant :

*arr = data[['Pclass', 'Fare', 'Age']].values*

Complétez le programme suivant de manière à afficher/selectionner uniquement les passagers de la première classe (Pclass=1):

*arr[ ..... == 1 ]*

# Somme des valeurs boolean

*arr = df[['Pclass', 'Fare', 'Age']].values*
*mask = arr[:, 2] ¡ 18*
Rappelez-vous que les **valeurs vraies sont interprétées comme 1** et que les **valeurs fausses sont interprétées comme 0**. Nous pouvons donc simplement additionner le tableau et cela équivaut à compter le nombre de valeurs vraies.
*print(mask.sum())*

Une autre façon d'obtenir le même résultat :
*print((arr[:, 2] ¡ 18).sum())*

**Attention !**

Sommer un tableau de valeurs booléennes donne le nombre de valeurs vraies.

# Exercice d'application 16

## Sommer un tableau de boolean

Soit le tableau suivant :

*arr = data[['Pclass', 'Fare', 'Age']].values*

Laquelle des propositions suivantes est correcte pour compter le nombre de passagers dans la classe 1 ?

1. (arr[0] == 1).sum()
2. (arr[:, 0] == 1).sum
3. (arr[:, 0] == 1).sum()
4. (arr[0] == 1).sum

# Scatter plot 1

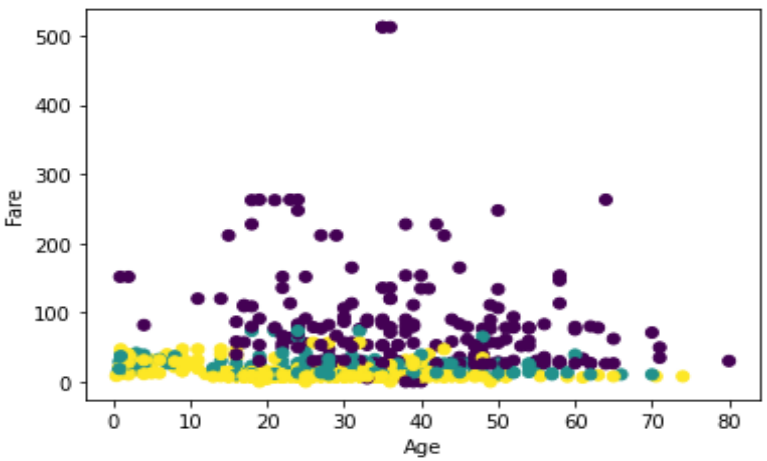
Nous pouvons utiliser la bibliothèque **matplotlib** pour tracer des graphiques. Le tracé des graphiques peut souvent nous aider à avoir un premier aperçu des données.

Nous utilisons la fonction **scatter()** pour tracer un **cross-plot**. Le premier argument est l'axe des x (direction horizontale) et le second argument est l'axe des y (direction verticale).

Nous pouvons également utiliser nos données **pour coder en couleur notre nuage de points**. Cela donnera à chacune des 3 classes une couleur différente. Nous ajoutons le **paramètre c** et lui donnons une Pandas Series. Dans ce cas, notre Pandas Series a 3 valeurs possibles (1ère, 2ème et 3ème classe), nous verrons donc nos points de données recevoir chacun une des trois couleurs.

# Scatter plot 2

*import matplotlib.pyplot as plt plt.scatter(titanic['Age'], titanic['Fare'], c=titanic['Pclass'])*
*plt.xlabel('Age')*
*plt.ylabel('Fare')*
*plt.show()*




Les points violets sont de première classe, les points verts de deuxième classe et les points jaunes de troisième classe.

# Exercice d'application 17

## Construire un cross-plot

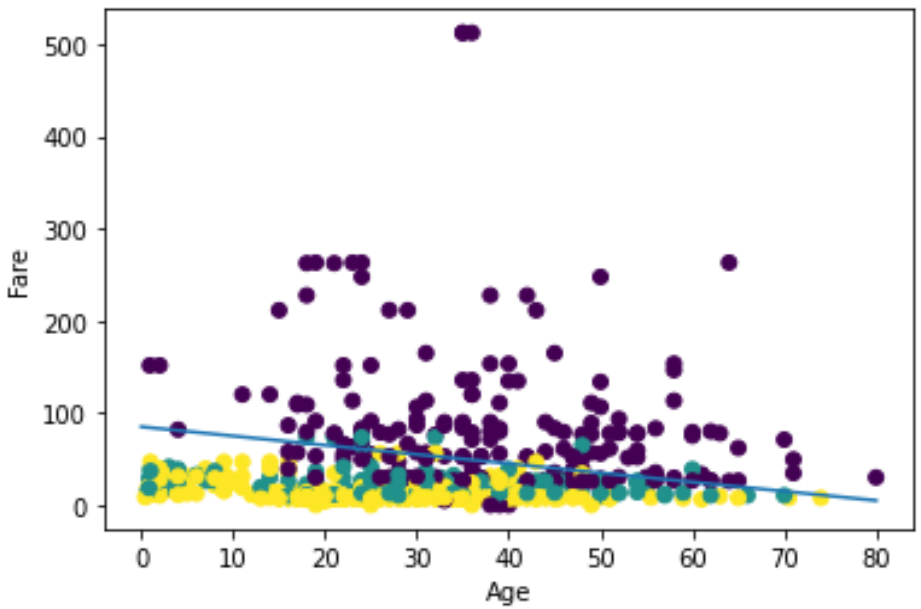
Écrivez un programme pour créer un cross-plot avec Pclass sur l'axe des y et Fare sur l'axe des x. Attribuez un code de couleur selon qu'ils ont survécu ou non. Ajoutez les étiquettes "Fare" et "Pclass" sur les axes x et y respectivement.

# Tracé d'une droite 1

**La fonction plot()** permet de tracer **une droite à partir de ces coordonnées**.L'exemple suivant trace une ligne qui sépare approximativement la première classe de la deuxième et de la troisième classe. Á vue d'oeil, nous placerons la droite de (0, 85) à (80, 5). Notre syntaxe ci-dessous comporte une liste de valeurs x et une liste de valeurs y :

```
plt.scatter(titanic['Age'], titanic['Fare'], c=titanic['Pclass'])
plt.xlabel('Age')
plt.ylabel('Fare')
plt.plot([0, 80], [85, 5])
plt.show()
```

# Tracé d'une droite 2




Vous pouvez voir que les points jaunes (3ème classe) et verts (2ème classe) sont principalement en dessous de la ligne et que les points violets (1ère classe) sont principalement au-dessus. Nous avons fait cela manuellement, mais dans le prochain module, nous apprendrons à le faire de manière algorithmique.

# Exercice d'application 18

## Construire une droite

Reprendre le graphique précédent en traçant cette fois-ci une droite qui va de (0, 15) à (100, 15).

# Quiz 1

**1) Quel module permet de manipuler et lire les données avec un main data object de DataFrame ?**

1. Matplotlib
2. Pandas
3. Numpy

**2) Quel module permet de réaliser des calculs et des analyses numériques avec un main data object d' array ?**

1. Matplotlib
2. Pandas
3. Numpy

# Quiz 2

## 3) Quelle est la médiane et la moyenne de la Serie suivante ?

5,10, 15, 20, 25

1. mean = 20 / median = 10
2. mean = 15 / median = 10
3. mean = 15 / median = 15

## 4) Quelle est la mesure de la dispersion des données ?

1. Mean
2. Median
3. Standard deviation
4. Variance

# Quiz 3

## 5) Chargez et affichez les données

Soit un fichier csv appelé eleves.csv avec trois colonnes : nom, prenom et age.

Écrire un programme qui permet de charger le fichier csv comme un Pandas DataFrame et afficher uniquementla Serie appelée 'nom'.

## 6) Quelle est l'instruction qui permet de récupérer les colonnes age et prenom comme un Numpy array ?

1. data['prenom', 'age']
2. data['prenom', 'age'].values
3. data[['prenom', 'age']]
4. data[['prenom', 'age']].values

# Quiz 4

## 7) Construction d'un cross-plot

Soit un fichier csv appelé eleves.csv avec trois colonnes : nom, taille et age.

Complétez le programme suivant permettant de construire un cross-plot avec la taille sur l'axe des abscisses et l'age sur l'axe des ordonnées :

*import matplotlib as plt*
*..... ( data['taille'], data['age'])*

# Machine Learning (ML)

## Chap 2 : La Classification

David TCHOUTA

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

April 5, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Intro
3. Classification lin.
4. Reg. logistique
5. Scikit-learn
6. Cancer du sein
7. Quiz

# Objectifs du chapitre

Dans ce chapitre, nous allons dans un premier temps aborder les notions de classification linéaire et de régression logistique. La deuxième partie sera consacré au module Scikit-learn avant de terminer sur un exemple de classification consistant à prédire le cancer du sein.

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

- de definir le Machine Learning
- de comprendre et d'effecteur une classification linéaire
- de comprendre et d'effecteur une régression logistique
- de manipuler le module Scikit-learn, incontournable en ML
- de réaliser des prédictions sur un jeu de données

# Definition

## Definition

**Le Machine Learning (ML)** se décompose en deux catégories : **l'apprentissage supervisé et l'apprentissage non supervisé**. **L'apprentissage supervisé** signifie **qu'on dispose/connaît des étiquettes/labels des données passées (les vraies valeurs/valeurs observées) de la variable cible ou de la réponse**. Tandis qu'en **apprentissage non supervisé**, **la variable cible ou la réponse n'est pas connue**. **En apprentissage supervisé, on distingue la Classifiction et la régression**. **Les problèmes de classification** sont ceux dont **la réponse est une variable catégorielle** (True or False ou plusieurs catégories). **Les problèmes de régression** sont ceux dont **la réponse est une variable quantitative ou numérique**.

# Exemples

Par exemple **prédire le prix des maisons** est un **problème de régression** puisque la **variable réponse** (prix de la maisosn) est une **variable quantitative**.

Tandis que **prédire si une personne fera défaut ou non** est un **problème de classification**.

**Attention !**

La **régression logistique**, bien qu'il y a le mot régression dans son nom est un **algorithme permettant de résoudre les problèmes de classification** et non de régression.

# Terminologie utilisée 1

Reprenons notre exemple du Titanic :

```
In [7]: print(titanic.head())
   PassengerId  Survived  Pclass  ...     Fare Cabin  Embarked
0            1         0       3  ...   7.2500   NaN         S
1            2         1       1  ...  71.2833   C85         C
2            3         1       3  ...   7.9250   NaN         S
3            4         1       1  ...  53.1000  C123         S
4            5         0       3  ...   8.0500   NaN         S
```

La colonne **'Survived'** est celle **qu'on va prédire**, on l'appelle encore la **variable cible** ou la **réponse (target) ou la variable endogène ou la variable dépendante**. C'est une liste de 0 et 1. 0 signifie que le passager est décédé et 1 signifie qu'elle a survécu. **Le reste des colonnes** sont des informations sur les passagers qui nous permettront de prédire la variable cible. On appelle chacun de ces colonnes **des prédicteurs ou des variables explicatives ou variables exogènes (features or predictors)**.

# Terminologie utilisée 2

Bien que nous sachions si chaque passager de l'ensemble de données a survécu, nous aimerions être en mesure de faire des prédictions sur les autres passagers pour lesquels nous n'avons pas pu collecter ces données.
Nous allons construire un **modèle d'apprentissage automatique** pour nous aider à le faire.

# Exercice d'application 1

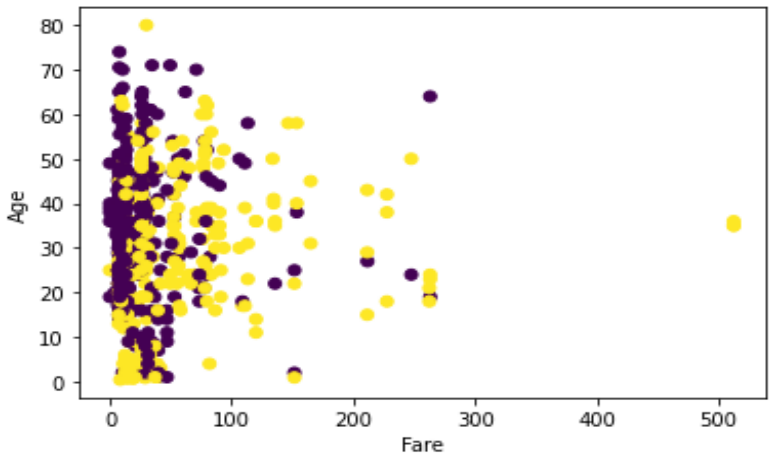
## Terminologie

Un prédicteur est ce que nous essayons de prédire et une cible est un élément de données que nous pouvons utiliser pour faire notre prédiction :

1. True
2. False

# Analyse graphique de la classification

Pour des raisons de simplicité, nous allons utiliser seulement deux prédicteurs (Age et Fare) :




Sur l'axe des abscisses, nous avons le prix du billet du passager (Fare) et sur l'axe des ordonnées, son âge (age). Les **points jaunes** correspondent **aux passagers qui ont survécu** et les **points violets** aux passagers **qui n'ont pas survécu**.

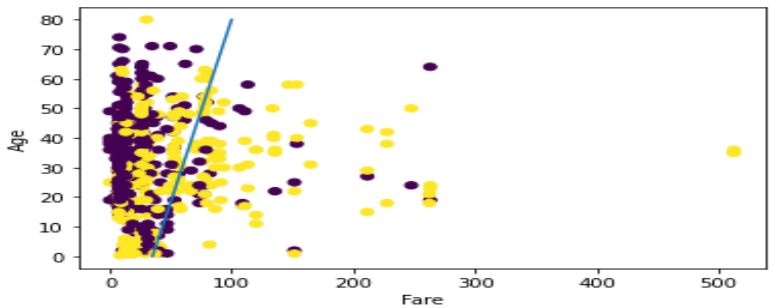
# Objectif

Vous pouvez voir **qu'il y a plus de points jaunes en bas du graphique qu'en haut**. Cela s'explique par le fait que **les enfants avaient plus de chances de survivre que les adultes**, ce qui correspond à notre intuition. De même, il y a p**lus de points jaunes sur la droite du graphique**, ce qui signifie que **les personnes qui ont payé plus cher avaient plus de chances de survivre**.

**La tâche d'un modèle linéaire est de trouver la ligne qui sépare le mieux les deux classes, de manière à ce que les points jaunes soient d'un côté et les points violets de l'autre**.

# Tracé de la frontière

Ci-dessous un exemple d'une bonne ligne. La ligne est utilisée pour faire des prédictions sur les nouveaux passagers. Si les données d'un passager se trouve sur le côté droit de la ligne, on peut prédire qu'il a survécu. S'il est à gauche, on peut prédire qu'il n'a pas survécu.




**Attention !**

Le challenge de la construction du modèle sera de déterminer quelle est la meilleure ligne possible

# Exercice d'application 2

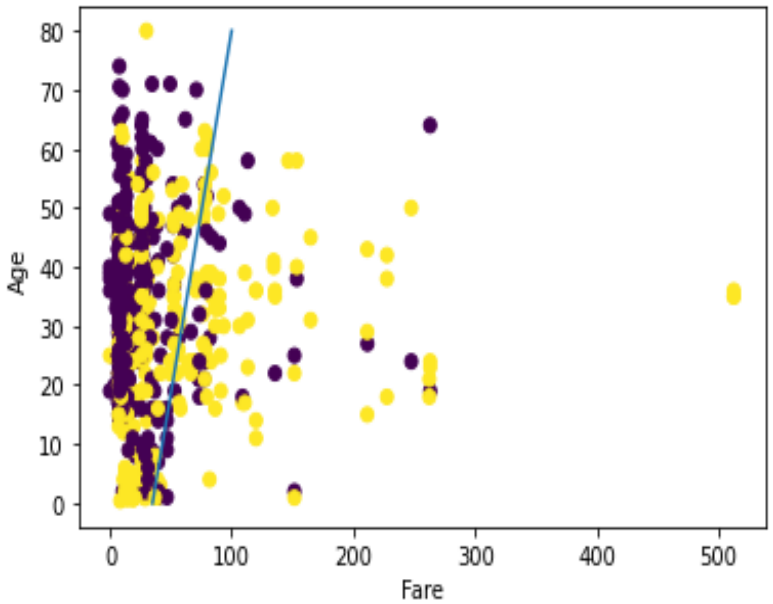
## Objectif et challenge

Quel est l'objectif et le challenge d'un problème de classification ?

# Équation d'une droite 1

L'équation d'une droite est donnée sous la forme suivante :
$0 = ax + by + c$
a, b et c sont des coefficients. Supposons que a = 1, b=-1 et c=-30 :
$0 = (1)x + (-1)y + (-30)$
Pour rappel, la variable Fare est en abscisses et l'Age en ordonnées. Pour tracer une droite à partir d'une équation, nous avons besoin de deux points qui se trouvent sur la droite. Nous pouvons voir, par exemple, que le point (30, 0) se trouve sur la droite (Fare 30, Age 0). Si nous l'intégrons dans l'équation, on a :
$30 - 0 - 30 = 0$
On voit aussi que le point (50, 20) est sur la droite (Fare 50, Age 20) :
$50 - 20 - 30 = 0$

# Équation d'une droite 2




**Attention !**

Les coefficients de la ligne sont ce qui contrôle où se trouve la ligne.

# Exercice d'application 2

## Équation d'une droite

Soit l'équation de droite suivante :

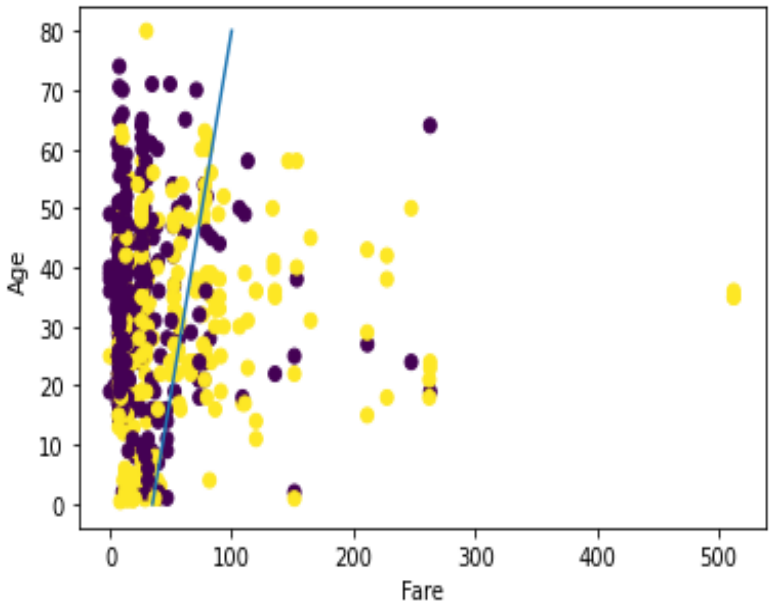
$2x + y - 5 = 0$

Quels sont les points qui se trouvent sur le droite ?

1. (4, 2)
2. (2, 1)
3. (3, 3)
4. (0, 5)

# Prédiction en fonction de la frontière 1

Soit l'équation de droite suivante : $0 = (1)x + (-1)y - 30$



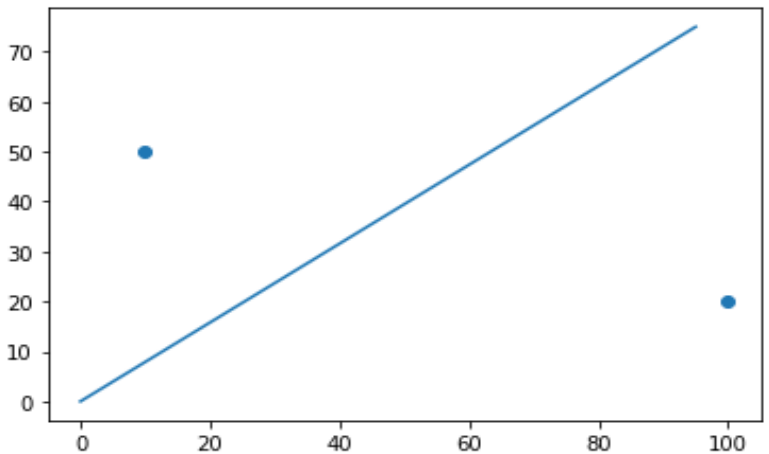

Si nous prenons les données d'un passager, nous pouvons utiliser cette équation pour déterminer de quel côté de la ligne il se situe. Par exemple, disons que nous avons un passager dont le tarif (Fare) est de 100 et l'âge (Age) de 20.

# Prédiction en fonction de la frontière 2

Si on remplace x et y par leur valeur dans léquation : (1)100 + (-1)20 - 30 = 100 - 20 - 30 = **50**
Comme le **résultat est positif**, le point se situe **à droite de la frontière**, nous prédisons donc que le **passager a survécu**.
Si nous prenons un passager ayant payé un tarif de 10 et âgé de 50 ans : (1)10 + (-1)50 - 30 = **-70**
Comme le **résultat est négatif**, le point se situe **à gauche de la frontière**, nous prédisons donc que le **passager n'a pas survécu**.

# Prédiction en fonction de la frontière 3

Nous pouvons voir les deux points (100, 20) et (10, 50) ci-dessous :




**Attention !**

L'endroit où se situe le point détermine si le passager a survécu ou non.

# Exercice d'application 3

## Prédiction

Soit l'équation de droite suivante :

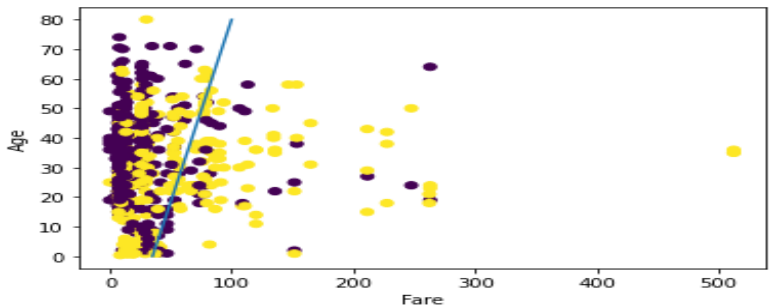
$2x + y - 5 = 0$

Quels sont les points qui auront une prédiction positive ?

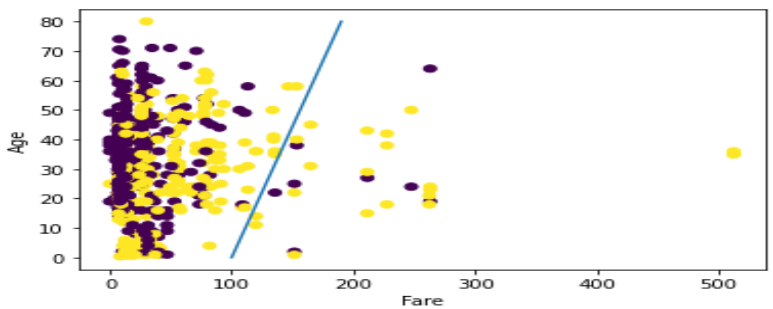
1. (-1, 8)
2. (4, -4)
3. (2, 0)
4. (3, 0)

# Choix de la frontière 1

Soit l'équation de droite suivante (frontière 1) : $0 = (1)x + (-1)y - 30$




Soit l'équation de droite suivante (frontière 2) : $0 = (4)x + (5)y - 400$



# Choix de la frontière 2

Si nous regardons les deux lignes, nous constatons que la frontière 1 a plus de points jaunes sur la droite et plus de points violets sur la gauche.
La frontière 2 n'a pas beaucoup de points à sa droite ; la plupart des points jaunes et violets se trouvent à gauche. La frontière 1 est donc la frontière préférée, car elle réussit mieux à séparer les points jaunes et violets.

Nous devons définir mathématiquement cette idée afin de pouvoir trouver algorithmiquement la meilleure ligne.

**Attention !**

La régression logistique est un moyen de trouver mathématiquement la meilleure frontière.

# Exercice d'application 4

## Qu'est-ce qu'une bonne frontière ?

Quel est le but de tracer une frontière dans un modèle linéaire pour la classification ?

1. pour passer le plus grand nombre de points possible
2. pour séparer deux classes
3. pour avoir tous les points d'un côté de la frontière
4. pour éviter le plus de points possible

# Probabilités de survie 1

Afin de tracer la meilleure frontière possible, nous devons avoir un moyen de noter la frontière. Commençons par examiner les données individuellement.

Idéalement, **les coordonnées d'un passager qui a survécu** devraient être **à droite et bien éloigné de la frontière**. **Les coordonnées d'un passager n'ayant pas survécu** devraient **être à gauche et bien éloigné de la frontière**. Plus la donnée est éloignée de la ligne, plus nous sommes sûrs qu'elle se trouve du bon côté de la frontière.

Pour **chaque coordonnée**, nous aurons **un score**, qui est une valeur comprise **entre 0 et 1**. Nous pouvons l'imaginer comme **la probabilité qu'un passager survive**.

# Probabilités de survie 2

Si la valeur est **proche de 0**, le point se trouve **loin à gauche de la frontière**, ce qui signifie que nous sommes sûrs que **le passager n'a pas survécu**. Si la valeur est **proche de 1**, le point se trouve **loin à droite de la frontière**, ce qui signifie que nous sommes sûrs que **le passager a survécu**. Une **valeur de 0,5** signifie que le point **se trouve directement sur la ligne** et que **nous ne savons pas si le passager a survécu**.

**L'équation permettant de calculer ce score** est présentée ci-dessous, bien que l'intuition qui la sous-tend soit bien plus importante que l'équation elle-même.

# Probabilités de survie 3

$$\frac{1}{1+e^{-(ax+by+c)}}$$

Cette fonction s'appelle la **sigmoïde (sigmoid)**.

**Attention !**

La régression logistique ne donne pas seulement une prédiction (a survécu ou non), mais une probabilité (80% de chances que cette personne ait survécu).

# Exercice d'application 5

## Probabilité de survie

Dans la régression logistique, nous calculons une probabilité. Pour les données du Titanic, nous prédisons que le passager survit si la probabilité est de :

1. -1
2. 1
3. 0.75
4. 0.25
5. 0

# Équation de vraisemblance 1

Pour savoir si la frontière est au bon endroit, nous devons noter si nos prédictions sont correctes ou pas. Idéalement, si nous prédisons avec une probabilité élevée que le passager survive (ce qui signifie que ce point se trouve très à droite de la frontière), alors le passager a effectivement survécu. Nous serons donc **récompensés lorsque nous prédisons quelque chose de correct et pénalisés si nous prédisons quelque chose d'incorrect**.

Ci-dessous **l'équation de vraisemblance** :

$$likelihood = \begin{cases} p & \text{si le passager a survécu} \\ 1-p & \text{si le passager n'a pas survécu} \end{cases}$$

# Équation de vraisemblance 2

p est la probabilité de survie (calculée par la fonction sigmoïde) vu précédemment. **La vraisemblance** sera **une valeur comprise entre 0 et 1**. **Plus la valeur est élevée, plus la frontière est meilleure**.

Voici quelques possibilités :

- Si la probabilité prédite p est de 0,25 et que le passager n'a pas survécu, nous obtenons un score de 0,75 (bon).
- Si la probabilité prédite p est de 0,25 et que le passager a survécu, nous obtenons un score de 0,25 (mauvais).

**Nous multiplions tous les scores individuels pour chaque passager afin d'obtenir un score total pour notre frontière**. Ainsi, nous pouvons **comparer différentes frontières pour déterminer la meilleure**.

# Équation de vraisemblance 3

Supposons 4 coordonnées suivantes :

| **a survécu ou non ?** | **probabilités prédictes** | **score** |
|---|---|---|
| a survécu | 0.2 | 0.2 |
| n'a pas survécu | 0.25 | 0.75 |
| a survécu | 0.7 | 0.7 |
| n'a pas survécu | 0.2 | 0.8 |

On obtient le **score total** en **multipliant** les **quatre scores** :
0.2*0.75 *0.7* 0.8 = **0.084**

# Équation de vraisemblance 4

Cette valeur sera toujours très faible puisqu'il s'agit de la probabilité que notre modèle prédise tout parfaitement. Un modèle parfait aurait une probabilité prédite de 1 pour tous les cas positifs et de 0 pour tous les cas négatifs.

**Attention !**

La vraisemblance est la façon dont nous notons et comparons les choix possibles d'une frontière avec le meilleur ajustement possible.

# Exercice d'application 6

## Vraisemblance

Quel est le score de vraisemblance si la probabilité prédicte est de 0.75 et que le passager n'a pas survécu ?

1. 0.65
2. 0.2
3. 1
4. 0.25
5. 0

# Scikit-learn

**Scikit-learn** (en abregé sklearn) est le module de Python dans lequel l**a plupart des algorithmes de Machine Learning** est implementé.

Notez que scikit-learn est continuellement mis à jour. Si vous avez une version légèrement différente du module installé sur votre ordinateur, tout fonctionnera encore correctement, mais vous pourriez voir des valeurs légèrement différentes.

**Attention !**

Scikit-learn est l'un des modules Python les mieux documentés qui soient. Vous pouvez trouver de nombreux exemples de code sur scikit-learn.org.

# Exercice d'application 7

## Scikit-learn

Scikit-learn permet de :

1. faire de la Data Visualization
2. Data analysis
3. manipuler les modèles de ML

# Préparation des données avec Pandas 1

La préparation des données (chargement, lecture et modification) se fait avec Pandas :

| | Survived | Pclass | Sex | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | male | 22.0 | 1 | 0 | 7.2500 |
| 1 | 1 | 1 | female | 38.0 | 1 | 0 | 71.2833 |
| 2 | 1 | 3 | female | 26.0 | 0 | 0 | 7.9250 |
| 3 | 1 | 1 | female | 35.0 | 1 | 0 | 53.1000 |
| 4 | 0 | 3 | male | 35.0 | 0 | 0 | 8.0500 |

Tout d'abord, nous devons rendre toutes nos colonnes numériques. Rappelez-vous comment créer une colonne booléenne pour le sexe : *data['male'] = data['Sex'] == 'male'*
Maintenant, prenons toutes les prédicteurs et créons un tableau numpy appelé X. Nous sélectionnons d'abord toutes les colonnes qui nous intéressent, puis nous utilisons la méthode values pour les convertir en un tableau numpy.

# Préparation des données avec Pandas 2

$X = data[['Pclass', 'male', 'Age', 'SibSp', 'Parch', 'Fare']].values$

Maintenant, prenons la cible (la colonne Survived) et stockons-la dans une variable y :
$y = data['Survived'].values$

**Attention !**

Il est courant d'appeler notre tableau 2d de prédicteurs X et notre tableau 1d de valeurs cibles y.

# Exercice d'application 8

## Préparation des données

Écrire un programme permettant de créer et de stocker les variables dépendantes (Fare et Age) dans une variable X de type numpy array et la cible Survived dans une variable y de type numpy array.

# La régression logistique sous Scikit-learn 1

L'instruction suivante permet d'importer le modèle de régression logistique :
*from sklearn.linear_model import LogisticRegression*
Tous les modèles sklearn sont construits comme des classes Python. Nous commençons par instancier la classe.
*model = LogisticRegression()*
Nous pouvons maintenant utiliser les données que nous avons préparées précédemment pour entraîner le modèle. La méthode fit est utilisée pour construire le modèle. Elle prend deux arguments : X (les prédicteurs sous forme de tableau numpy 2d) et y (la cible sous forme de tableau numpy 1d).

# La régression logistique sous Scikit-learn 2

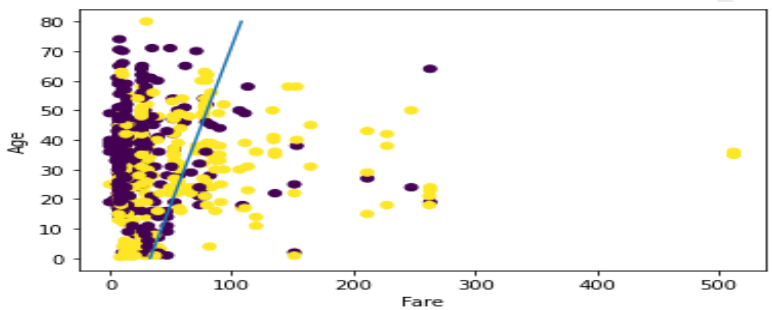
Pour simplifier, supposons que nous construisons un modèle de régression logistique en utilisant uniquement les colonnes Fare et Age. Tout d'abord, nous définissons X comme étant la matrice des prédicteurs et y comme étant le tableau de la variable cible. $X =$ *data[['Fare', 'Age']].values*
$y =$ *data['Survived'].values*
Now we use the fit method to build the model :
*model.fit(X, y)*
Ajuster le modèle signifie utiliser les données pour choisir une frontière avec le meilleur ajustement. Nous pouvons voir les coefficients avec les attributs coef_ et intercept_.
*print(model.coef_, model.intercept_)*

# La régression logistique sous Scikit-learn 3

data = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
*from sklearn.linear_model import LogisticRegression*
*model = LogisticRegression()*
*X = data[['Fare', 'Age']].values*
*y = data['Survived'].values*
*model.fit(X, y)*
*print(model.coef_, model.intercept_)*
[[ 0.01615949 -0.01549065]] [-0.51037152]
Ces valeurs signifient que l'équation de droite est la suivante : 0 = 0.0161594x + -0.01549065y + -0.51037152.

# La régression logistique sous Scikit-learn 4

Ci-dessous la frontière issue de cette équation de droite :




Vous pouvez voir qu'elle fait un travail convenable (mais pas génial) pour séparer les points jaunes et violets. Nous nous sommes un peu handicapés en n'utilisant que 2 de nos prédicteurs, donc dans les prochaines parties nous utiliserons toutes les variables dépendantes

# Exercice d'application 9

## Préparation des données

Complétez le code pour construire un modèle de régression logistique. Supposons que nous avons un tableau numpy 2d X des prédicteurs et un tableau numpy 1d y de la cible.

*from ..... .linear_model import LogisticRegression*
*model = LogisticRegression()*
*model. .....(X, y)*

# Prédiction 1

Maintenant, construisons le modèle avec un plus grand nombre de prédicteurs (06) comme ci-dessous :

```
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
model = LogisticRegression()
model.fit(X, y)
```

Nous pouvons utiliser la méthode **predict()** :

```
model.predict(X)
```

# Prédiction 2

Le premier passanger dans la base est :
[3, True, 22.0, 1, 0, 7.25]
Ce qui signifie Cela signifie que le passager fait partie de la classe 3, qu'il est un homme, qu'il a 22 ans, qu'il a 1 frère/conjoint à bord, 0 parent/enfant à bord et qu'il a payé 7,25 $. Voyons ce que le modèle prédit pour ce passager. **Notez que même avec un seul point de données, la méthode predict prend un tableau numpy à 2 dimensions et renvoie un tableau numpy à 1 dimension**.
print(model.predict([[3, True, 22.0, 1, 0, 7.25]])) $\Longrightarrow$ [0]
Le résultat est 0, ce qui signifie que le modèle prédit que ce passager n'a pas survécu.

# Prédiction 3

Voyons ce que le modèle prédit pour les 5 premières lignes de données et comparons-le au tableau de la variable cible. Nous obtenons les 5 premières lignes de données avec X[:5] et les 5 premières valeurs de la cible avec y[:5].

print(model.predict(X[:5])) $\Longrightarrow$ [0 1 1 1 0]
print(y[:5]) $\Longrightarrow$ [0 1 1 1 0]

**Nous constatons que le modèle prédit correctement le cinq (05) premières réponses**.

**Attention !**

La méthode predict renvoie un tableau de 1 et 0, où 1 signifie que le modèle prédit que le passager a survécu et 0 signifie que le modèle prédit que le passager n'a pas survécu.

# Prédiction 4

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
data =
pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
data['male'] = data['Sex'] == 'male'
X = data[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = data['Survived'].values
model = LogisticRegression()
model.fit(X, y)
print(model.predict([[3, True, 22.0, 1, 0, 7.25]]))
print(model.predict(X[:5]))
print(y[:5])
```

# Exercice d'application 10

## Prédiction

Soit X la matrice des prédicteurs et y la variable cible de type boolean (True or False). Supposons le programme suivant :

```
model = LogisticRegression()
model.fit(X, y)
print(model.predict(X[:5]))
```

Quels sont les résultats possibles ?

1. ([-1, -1, 0, 0, 0])
2. ([0, 1, 0, 1, 1, 0])
3. ([0, 1, 2, 1, 2, 0])
4. ([0, 0, 0, 0, 0, 0])
5. ([1, 1, 1, 1, 1, 1])

# Score de prédiction 1

Nous pouvons nous faire une idée de la **qualité de notre modèle** en **comptant le nombre de points de données qu'il prédit correctement**. C'est ce qu'on appelle le **score de précision (accuracy)**.

Créons un array contenant les valeurs y prédites :

y_pred = model.predict(X)

Nous créons maintenant un tableau de valeurs booléennes indiquant si notre modèle a prédit correctement ou non chaque passager :

y_pred == y

Pour obtenir le nombre de prédictions qui sont vraies, nous pouvons utiliser la méthode sum() de Numpy :

print((y == y_pred).sum()) $\Longrightarrow$ 714

# Score de prédiction 2

Cela signifie que sur les 887 points de données, le modèle fait une prédiction correcte pour 714 d'entre eux.
Pour obtenir le pourcentage d'exactitude, nous le divisons par le nombre total de passagers. Nous obtenons le nombre total de passagers en utilisant l'attribut shape :
y.shape[0]
Ainsi, notre score de précision est calculé comme suit :
print((y == y_pred).sum() / y.shape[0]) $\Longrightarrow$ 0.804960541149436
**La précision du modèle est donc de 80%**. En d'autres termes, **le modèle effectue une prédiction correcte sur 80% des points de données**.

# Score de prédiction 3

Nous pouvons donc obtenir le même résultat en utilisant la méthode **score()**.

.

La méthode score() utilise le modèle pour faire une prédiction pour X et compte le pourcentage d'entre eux qui correspondent à y :
print(model.score(X,y)) $\Longrightarrow$ 0.804960541149943 6

# Exercice d'application 11

## Accuracy

Supposons que y = [0, 0, 0, 1, 1] et que le résultat de model.predict(X) est [0, 0, 1, 1, 0]
Quel est l'output de l'instruction ci-dessous :

1. 0.4
2. 0.8
3. 0.0
4. 0.6
5. 0.8

# Introduction au Breast Cancer Dataset

Maintenant que nous avons mis en place les outils nécessaires pour construire un modèle de régression logistique pour un jeu de données de classification, nous allons introduire un nouveau jeu de données.

Dans l'ensemble de données sur le cancer du sein, chaque point de données contient **les mesures d'une image d'une masse mammaire et indique si elle est cancéreuse ou non**. **L'objectif est d'utiliser ces mesures pour prédire si la masse est cancéreuse**.

Ce jeu de données est intégré à scikit-learn, nous n'aurons donc pas besoin de lire un csv.
Commençons par charger le jeu de données et jetons un coup d'œil aux données et à la façon dont elles sont formatées.

# Chargement des données

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
cancer_data = load_breast_cancer()
```

L'objet retourné (que nous avons stocké dans la variable cancer_data) est un objet similaire à un dictionnaire Python. Nous pouvons voir les clés disponibles avec la méthode keys :

```
print(cancer_data.keys())
```

Nous commencerons par examiner avec l'attribut **DESCR**, qui donne une **description détaillée de l'ensemble de données** :

```
print(cancer_data['DESCR'])
```

# Description de la base de données

Nous pouvons voir qu'il y a **30 prédicteurs**, **569 points de données** et que **la cible est soit maligne (cancéreuse) soit bénigne (non cancéreuse)**. Pour chacun des points de données, nous avons des mesures de **la masse mammaire (rayon, texture, périmètre, etc.)**. Pour chacune **des 10 mesures**, plusieurs valeurs ont été calculées, de sorte que nous avons **la moyenne, l'erreur standard et la pire valeur**. On obtient ainsi **10 * 3 ou 30 prédicteurs au total**.

**Attention !**

Dans l'ensemble de données sur le cancer du sein, plusieurs prédicteurs (variables dépendantes) sont calculées à partir d'autres colonnes. Le processus de détermination des variables supplémentaires à calculer est appelé **feature engineering**.

# Exercice d'application 12

## Chargement des données

Complétez le programme permettant de charger les données du cancer du sein depuis scikit-learn.

```
from sklearn. ..... import load_breast_cancer
cancer_data = ..... _breast_cancer()
```

# Chargement des données dans Pandas 1

Nous allons maintenant charger les données en dehors de l'objet cancer_data.
Les données des prédicteurs sont stockées avec la clé "data".
Lorsque nous l'examinons, nous constatons qu'il s'agit d'un tableau numpy comportant 569 lignes et 30 colonnes. C'est parce que nous avons 569 points de données et 30 prédicteurs.
Le tableau suivant est un tableau numpy des données :
cancer_data['data']
Nous utilisons l'attribut shape pour voir qu'il s'agit d'un tableau de 569 lignes et 30 colonnes :
cancer_data['data'].shape
Pour mettre cela dans un DataFrame Pandas et le rendre plus lisible, nous voulons les noms des colonnes. Ceux-ci sont stockés avec la clé 'feature_names' :
cancer_data['features_names']

# Chargement des données dans Pandas 2

Nous pouvons maintenant créer un Pandas DataFrame avec toutes nos données.

df = pd.DataFrame(cancer_data['data'],
columns=cancer_data['feature_names'])
print(df.head())

| | mean radius | mean texture | mean perimeter | ... | worst concave points |
|---|---|---|---|---|---|
| 0 | 17.99 | 10.38 | 122.80 | ... | 0.2654 |
| 1 | 20.57 | 17.77 | 132.90 | ... | 0.1860 |
| 2 | 19.69 | 21.25 | 130.00 | ... | 0.2430 |
| 3 | 11.42 | 20.38 | 77.58 | ... | 0.2575 |
| 4 | 20.29 | 14.34 | 135.10 | ... | 0.1625 |

Nous pouvons voir que nous avons 30 colonnes dans le DataFrame, puisque nous avons 30 variables dépendantes. Le résultat est tronqué pour qu'il tienne sur l'écran. Nous avons utilisé la méthode **head()**, donc notre résultat ne comporte que 5 points de données.

# Chargement des données dans Pandas 3

Nous devons charger les données de la variable cible dans notre DataFrame, qui peuvent être trouvées avec la clé 'target'. Nous pouvons voir que la cible est un tableau numpy 1d de 1 et de 0 :
cancer_data['target']
Si nous regardons la forme du tableau, nous voyons qu'il s'agit d'un tableau à une dimension avec 569 valeurs (ce qui correspond au nombre de points de données dont nous disposions) :
cancer_data['target'].shape
Afin d'interpréter ces 1 et 0, nous devons savoir si 1 ou 0 est bénin ou malin. Ceci est donné par l'attribut target_names :
cancer_data['target_names']
Ce qui donne le tableau ['malin' 'bénin'] qui nous dit que **0 signifie maligne et 1 signifie bénigne**.

# Ajout de la colonne target (cible) au Dataframe

```
df = pd.DataFrame(cancer_data['data'],
columns=cancer_data['feature_names'])
df['target'] = cancer_data['target']
print(df.head())
```

```
   mean radius  mean texture  ...  worst fractal dimension  target
0        17.99         10.38  ...                  0.11890       0
1        20.57         17.77  ...                  0.08902       0
2        19.69         21.25  ...                  0.08758       0
3        11.42         20.38  ...                  0.17300       0
4        20.29         14.34  ...                  0.07678       0

[5 rows x 31 columns]
```

**Attention !**

Il est important de vérifier que vous interprétez correctement les colonnes booléennes. Dans notre cas, une variable cible de 0 signifie maligne et de 1 signifie bénigne.

# Exercice d'application 13

## Interprétation de la réponse

Si la variable cible est de 0, cela signifie que la tumeur est :

1. bénigne
2. maligne

# Construction du modèle de régression logistique 1

Maintenant que nous avons jeté un coup d'œil à nos données et que nous les avons mises dans un format confortable, nous pouvons construire notre matrice de prédicteurs X et notre tableau de cibles y afin de pouvoir construire un modèle de régression logistique :

X = df[cancer_data.feature_names].values

y = df['target'].values

Créons maintenant l'objet regression logistique et utilisons la méthode **fit()** pour **construire le modèle** :

model = LogisticRegression()

model.fit(X, y)

Ne pas exécuter les lignes de code ci-dessus

# Construction du modèle de régression logistique 2

Lorsque nous exécutons ce code, nous obtenons un avertissement de convergence. Cela signifie que le modèle a besoin de plus de temps pour trouver la solution optimale. Une option consiste à augmenter le nombre d'itérations. Vous pouvez également passer à un solveur différent, ce que nous allons faire. Le solveur est l'algorithme que le modèle utilise pour trouver l'équation de la ligne.

Vous pouvez voir les solveurs possibles dans la documentation sur la régression logistique.

# Construction du modèle de régression logistique 3

model = LogisticRegression(solver='liblinear')
model.fit(X, y)
Voyons ce que le modèle prédit pour le premier point de données de notre ensemble de données. Rappelez-vous que la méthode predict prend un tableau à 2 dimensions et que nous devons donc placer le point de données dans une liste :
model.predict([X[0]]) $\Longrightarrow$ [0]
Le **modèle prédit** donc que le **premier point de données** est une **tumeur maligne**.
Pour voir comment le modèle fonctionne sur l'ensemble des données, nous utilisons la méthode score() pour voir **la précision du modèle** :
model.score(X, y) $\Longrightarrow$ 0.959578207381370 8
Nous constatons que le modèle obtient **96%** de prédictions correctes.

# Programme

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
cancer_data = load_breast_cancer()
df = pd.DataFrame(cancer_data['data'],
columns=cancer_data['feature_names'])
df['target'] = cancer_data['target']
X = df[cancer_data.feature_names].values
y = df['target'].values
model = LogisticRegression(solver='liblinear')
model.fit(X, y)
print("prediction for datapoint 0:", model.predict([X[0]]))
print(model.score(X, y))
```

# Exercice d'application 14

## Accuracy

Une précision (Accuracy) de 96% signifie que :

1. le modèle a trouvé 96% de tumeur maligne
2. le modèle a trouvé 96% de tumeur benigne
3. le modele a 96% des points de données du bon côté de la frontière
4. le modèle a réalisé 96% de prédictions correctes

# Quiz 1

## 1) la variable cible

Si la variable cible d'un problème de classification a une valeur catégorielle, cela signifie qu'elle a combien de valeurs possibles ?

1. continues
2. infini
3. fini

# Quiz 2

## 2) Le jeu de données Titanic

Sélectionnez toutes les réponses qui sont vraies pour la façon dont nous avons construit des modèles pour l'ensemble de données du Titanic :

1. la colonne Survived est un prédicteur
2. Pclass est une variable exogène
3. la colonne Survived est la réponse
4. la colonne Survived est la variable endogène

# Quiz 3

## 3) Arrangez la séquence des lignes de code

Réorganisez ces lignes de code pour construire un modèle de régression logistique avec X et y et affichez le pourcentage de valeurs prédites correctement.

1. print(model.score(X,y))
2. model = LogisticRegression()
3. model.fit(X,y)
4. from sklearn.linear_model import LogisticRegression

# Quiz 4

## 4) Frontière

Nous avons utilisé la régression logistique pour trouver une frontière pour l'ensemble de données du Titanic, dans laquelle des valeurs suivantes de la probabilité prédite, on prédit que le passager survivra et le point de données est le plus éloigné de la ligne de séparation :

1. 0.8
2. 0.77
3. 0.93
4. 0.55

# Quiz 5

## 5) Vraisemblance

Si nous prédisons qu'un passager a 0,8 chance de survie et que le passager a survécu, la probabilité est de 0,8. Si nous prédisons qu'un passager a 0,6 chance de survie et que le passager n'a pas survécu, quelle est la probabilité ?

1. 0.3
2. 0.6
3. 0.8
4. 0.4

# Quiz 6

## 6) Quelle frontière choisir ?

Un modèle de régression logistique trouvera la frontière qui a le plus haut niveau de :

1. vraisemblance (likelihood)
2. précision (accuracy)
3. coefficients

# Quiz 7

## 7) Quelle précision ?

Soit le code suivant, quelle est la précision ?

```
print(model.predict(X,y)
[1 0 0 0 1]
print(y)
[1 1 0 0 0]
```

1. 80%
2. 40%
3. 60%
4. 50%

# Machine Learning (ML)

## Chapitre 3 : Évaluation des modèles
## Partie I

David TCHOUTA

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

April 9, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Intro
3. Precision/Recall
4. Metrics
5. Training/Testing
6. ROC

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue les métriques (mesures de performance des modèles) les plus importantes en ML.

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

- de manipuler la précision et le rappel (recall)
- de manipuler le F1 score et la matrice de confusion
- de comprendre le problème de surapprentissage (overfitting)
- de manipuler la courbe ROC (Receiver Operating Characteristic) et AUC (Area Under the Curve)
- de réaliser le compromis entre la precision et la recall

# Autres métriques 1

Dans le chapitre précédent, nous avons mesuré la qualité du modèle grâce à la précision (accuracy). Par exemple, si nous disposons de 100 points et que le modèle prédit 70 correctement et 30 de manière incorrect, alors on dit que la précision est de 70%.
La précision est intuitif et facile à comprendre mais ce n'est pas toujours la meilleure métrique à observer.
Par exemple, supposons que nous disposons d'un modèle permettant de prédire si les frais de carte de crédit est frauduleux.
Sur 10000 cartes de crédit, nous avons 9900 frais légitimes et 100 frais frauduleux.
Nous pourrions construire un modèle qui prédit simplement que chaque débit est légitime et il obtiendrait 9900/10000 (99%) des prédictions correctes.

# Autres métriques 2

**Attention !**

**La précision** est une bonne mesure si nos classes sont réparties de manière égale, mais **elle est très trompeuse** si **nos classes sont déséquilibrées**.

**Attention !**

Il faut toujours être prudent avec la précision. Vous devez connaître la distribution des classes pour savoir comment interpréter sa valeur.

# Exercice d'application 1

## Accuracy

Supposons que l'on vous demande de construire un modèle pour prédire les spams. Votre ensemble d'apprentissage comprend 1000 e-mails, 950 sont des e-mails légitimes et 50 sont des spams. Vous construisez un modèle qui prédit simplement que tous les e-mails sont légitimes. Quelle est la précision du modèle ?

1. 99%
2. 100%
3. 50%
4. 95%

# Matrice de confusion (Confusion Matrix) 1

Comme nous l'avons remarqué dans la partie précédente, nous ne nous intéressons pas seulement au nombre de points de données pour lesquels nous prédisons la bonne classe, mais aussi au nombre de points de données positifs pour lesquels nous prédisons correctement, ainsi qu'au nombre de points de données négatifs pour lesquels nous prédisons correctement.

Nous pouvons voir toutes les notions importantes dans ce que l'on appelle la **confusion matrix**(or Error Matrix or Table of Confusion).

# Confusion Matrix 2

La matrice de confusion est un tableau montrant quatre valeurs :

1. Les points de données que nous avions prédits positifs et qui sont en fait positifs
2. Les points de données que nous avions prédits positifs et qui sont en fait négatifs
3. Les points de données que nous avons prédits négatifs et qui sont en fait positifs
4. Les points de données que nous avons prédits négatifs et qui sont en fait négatifs

Le **1er et le 4e** sont les points de données que nous avons **prédits correctement**, le **2e et le 3e** point sont les points de données que nous avons **prédits incorrectement**.

# Confusion Matrix 3

Dans notre jeu de données Titanic, nous avons 887 passagers, 342 ont survécu (positif) et 545 n'ont pas survécu (négatif). Le modèle que nous avons construit dans le module précédent a la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 233 | 65 |
| **Predicted negative** | 109 | 480 |

Les nombres en bleu représentent le nombre de prédictions correctes. Ainsi, sur les **342 passagers qui ont survécu**, nous en avons prédit **233 correctement** (et **109 incorrectement**). Sur les **545 passagers qui n'ont pas survécu**, nous en avons prédit **480 correctement (et 65 incorrectement)**.

# Confusion Matrix 3

Nous pouvons utiliser la matrice de confusion pour calculer la précision. Pour rappel, la précision est le nombre de points de données correctement prédits divisé par le nombre total de points de données :
$(233+480)/(233+65+109+480) = 713/887 = \mathbf{80.38\%}$
Il s'agit en effet de la même valeur que celle obtenue au chapitre précédent.

**Attention !**

La matrice de confusion décrit pleinement les performances d'un modèle sur un ensemble de données, mais elle est difficile à utiliser pour comparer les modèles.

# Exercice d'application 2

## Accuracy

Soit la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 22 | 24 |
| **Predicted negative** | 10 | 44 |

Quelle est la précision ?

1. 60%
2. 80%
3. 66%
4. 32%

# True Positives, True Negatives, False Positives, False Negatives 1

Chaque cellule de la matrice de confusion dispose d'un nom :

1. Un **vrai positif (TP or True Positive)** est un point de données pour lequel nous avons prédit un résultat positif et pour lequel nous avons eu raison (1ere cellule).
2. Un **vrai négatif (True Negative or TN)** est un point de données pour lequel nous avons prédit un résultat négatif et pour lequel nous avons eu raison (4e cellule).
3. Un **faux positif (FP or False Positive)** est un point de données que nous avons prédit positivement et pour lequel nous nous sommes trompés (2e cellule).
4. Un **faux négatif (FN or False Negative)** est un point de données pour lequel nous avons prédit un résultat négatif mais pour lequel nous nous sommes trompés (3e cellule).

# True Positives, True Negatives, False Positives, False Negatives 2

Il peut être difficile de s'y retrouver dans ces termes. La façon de s'en souvenir est la suivante : le deuxième mot est la nature de notre prédiction (positive ou négative) et le premier mot indique si cette prédiction est correcte (vraie ou fausse).
Vous verrez souvent la matrice de confusion comme suit :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | TP | FP |
| **Predicted negative** | FN | TN |

**Attention !**

Les quatre valeurs de la matrice de confusion (TP, TN, FP, FN) sont utilisées pour calculer plusieurs métriques différentes que nous utiliserons par la suite.

# Exercice d'application 3

## Confusion matrix

Soit la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 233 | 65 |
| **Predicted negative** | 109 | 480 |

Complétez par TN, TP, FP, FN

1. Il y a 233 .....
2. Il y a 65 .....
3. Il y a 109 .....
4. Il y a 480 .....

# Precision 1

Il y a **deux métriques** généralement observées en **classification** : la **précision** (**precision**, à ne pas confondre avec l'accuracy) et le **recall** (rappel).
En théorie, la **précision fait référence au pourcentage de résultats positifs qui sont pertinents** et le **rappel au pourcentage de cas positifs correctement classés**.

# Precision 2

La **précision est le pourcentage de prédictions positives du modèle qui sont correctes**. Nous la définissons comme suit :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | TP | FP |
| **Predicted negative** | FN | TN |

$$precision = \frac{nombre.de.predictions.positives.qui.sont.correctes}{nombre.de.predictions.positives} = \frac{TP}{TP+FP}$$

# Precision 3

Si nous reprenons notre exemple de la matrice de confusion du Titanic :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 233 | 65 |
| **Predicted negative** | 109 | 480 |

La précision est : precision = 233 / (233 + 65) = 0.7819

# Exercice d'application 4

## Calcul de la précision

Soit la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 30 | 20 |
| **Predicted negative** | 10 | 40 |

Quelle est la précision ?

1. 0.5
2. 0.6
3. 0.8
4. 0.65

# Recall (Rappel) 1

Le **rappel (recall)** est le **pourcentage de cas positifs que le modèle prédit correctement**. Encore une fois, nous utiliserons la matrice de confusion pour calculer notre résultat :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | TP | FP |
| **Predicted negative** | FN | TN |

$$recall = \frac{nombre.de.predictions.positives.correctes}{nombre.de.cas.positifs} = \frac{TP}{TP+FN}$$

recall = 233 / (233 + 109) = 0.6813

# Recall (Rappel) 2

**Attention !**

Le **rappel** est une mesure du nombre de cas positifs que le modèle peut rappeler. On l'appelle également **sensitivity** ou **true positive rate**.

# Exercice d'application 5

## Calcul du recall

Soit la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 30 | 20 |
| **Predicted negative** | 10 | 40 |

Quelle est le rappel ?

1. 0.5
2. 0.6
3. 0.75
4. 0.65

# Quel compromis entre la precision et le recall ? 1

Nous serons souvent dans une situation où nous devrons choisir entre augmenter le rappel (tout en diminuant la précision) ou augmenter la précision (et diminuer le rappel). Cela dépendra de la situation que nous voudrons maximiser.

Par exemple, supposons que nous construisons un modèle pour prédire si les frais de carte de crédit sont frauduleux. Les cas positifs pour notre modèle sont les frais frauduleux et les cas négatifs sont les frais légitimes.

Considérons les deux scénarios suivants :
1. Si nous prédisons que les frais sont frauduleux, nous la rejetterons. 2. Si nous prédisons que les frais sont frauduleux, nous appelons le client pour confirmer le débit.

# Quel compromis entre la precision et le recall ? 2

Dans le cas 1, le client subit un énorme désagrément lorsque le modèle prédit une fraude de manière incorrecte (un faux positif). Dans le cas 2, un faux positif est un inconvénient mineur pour le client.

Plus le nombre de faux positifs est élevé, plus la précision est faible. En raison du coût élevé des faux positifs dans le premier cas, il vaudrait la peine d'avoir un faible rappel afin d'avoir une précision très élevée. Dans le deuxième cas, vous souhaitez un meilleur équilibre entre la précision et le rappel.

**Attention !**

Il n'y a pas de règle absolue concernant les valeurs de précision et de rappel à atteindre. Cela dépend toujours des données et de l'application.

# Exercice d'application 6

## Compromis entre la precision et le recall

Supposons un modèle pour prédire le spam. Les cas positifs sont des spams et les cas négatifs sont des mails légitimes. Si nous devons supprimer les e-mails que nous prédisons être des spams, qu'est-ce qui est le plus important à maximiser ?

1. recall
2. precision

# Autre métrique : F1 score 1

L'accuracy était une mesure attrayante car il s'agissait d'un seul chiffre. La précision et le rappel étant deux nombres, il n'est pas toujours évident de choisir entre deux modèles si l'un a une meilleure précision et l'autre un meilleur rappel. **Le score F1 est une moyenne de la précision et du rappel**, ce qui nous permet d'obtenir un score unique pour notre modèle.
Voici la formule mathématique pour le score F1 :

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

# Autre métrique : F1 score 2

Calculons le score F1 de notre modèle pour le jeu de données Titanic.
Nous allons utiliser les valeurs de précision (0.7819) et de rappel (0.6813) que nous avons calculés précédemment.
Le **score F1** est : 2 * (0.7819) * (0.6813) / (0.7819 + 0.6813) = **0.7281**

**Attention !**

Le score F1 est la moyenne harmonique des valeurs de précision et de rappel.

# Exercice d'application 7

## F1 score

Vous avez construit un modèle qui a une précision de 0,8 et un rappel de 0,5. Quelle est l'équation pour le score F1 ?

1. 2 * (0.8 + 0.5) / (0.8 * 0.5)
2. (0.8 + 0.5) / (0.8 * 0.5)
3. 2 * (0.8 * 0.5) / (0.8 + 0.5)
4. (0.8 * 0.5) / (0.8 + 0.5)

# Metrics in sklearn 1

Commençons par rappeler notre code du module précédent pour construire un modèle de régression logistique. Le code charge les données du Titanic à partir du fichier csv et le place dans un DataFrame Pandas. Nous créons ensuite une matrice des prédicteurs X et de la variable cible y. Nous créons un modèle de régression logistique et l'ajustons à nos données. Enfin, nous créons une variable y_pred de nos prédictions.

```
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
model = LogisticRegression()
model.fit(X, y)
y_pred = model.predict(X)
```

# Metrics in sklearn 2

Maintenant, nous sommes prêts à utiliser nos fonctions de calcul des métriques. Importons-les depuis scikit-learn :

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
```

Chaque fonction prend deux tableaux numpy à une dimension : les valeurs réelles de la cible et les valeurs prédites de la cible. Nous avons les valeurs réelles de la cible et les valeurs prédites de la cible. Nous pouvons donc utiliser les fonctions métriques comme suit :

```
print("accuracy:", accuracy_score(y, y_pred)) ⟹ 0.804960
print("precision:", precision_score(y, y_pred)) ⟹ 0.773462
print("recall:", recall_score(y, y_pred)) ⟹ 0.698830
print("f1 score:", f1_score(y, y_pred)) ⟹ 0.734254
```

# Metrics in sklearn 3

Nous constatons que l'accuracy est de 80%, ce qui signifie que 80% des prédictions du modèle sont correctes. La précision est de 78%, ce qui correspond au pourcentage de prédictions positives du modèle qui sont correctes. Le rappel est de 68%, ce qui correspond au pourcentage de cas positifs que le modèle a prédit correctement. Le score F1 est de 73%, qui est une moyenne de la précision et du rappel.

**Attention !**

Avec un seul modèle, les métriques ne nous disent pas grand-chose. Pour certains problèmes, une valeur de 60% est bonne, et pour d'autres, une valeur de 90% est bonne, selon la difficulté du problème. Nous utiliserons les métriques pour comparer différents modèles et choisir le meilleur.

# Exercice d'application 8

## Accuracy, precision, recall and F1 score

Supposons que nous ayons un tableau numpy 2d de X variables dépendantes et un tableau numpy 1D y de la variable cible. Réorganisez le code afin de construire un modèle sur les données et d'afficher la précision, le rappel et le score F1 dans cet ordre.

1. print("recall:", recall_score(y, ypred))
2. print("F1 score:", f1_score(y, ypred))
3. ypred = model.predict(X)
4. model = LogisticRegression()
5. print("precision:", precision_score(y, ypred))
6. model.fit(X,y)

# Confusion Matrix in Sklearn 1

Scikit-learn possède une fonction permettant de calculer la matrice de confusion que nous pouvons utiliser pour obtenir les quatre valeurs de la matrice de confusion (vrais positifs, faux positifs, faux négatifs et vrais négatifs). En supposant que y représente nos vraies valeurs cibles et que y_pred représente les valeurs prédites, nous pouvons utiliser la fonction confusion_matrix() comme suit :

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y, y_pred))
```

```
[[475  70]
 [103 239]]
```

**Attention !**

Ne pas oublier d'importer la métrique avant de l'utiliser.

# Confusion Matrix in Sklearn 2

**Scikit-learn inverse la matrice de confusion pour montrer les négatifs (Actual et Predicted) en premier :**

| | **Predicted negative** | **Predicted positive** |
|---|---|---|
| **Actual negative** | 475 | 70 |
| **Actual positive** | 103 | 239 |

Au lieu de :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 239 | 70 |
| **Predicted negative** | 103 | 475 |

# Confusion Matrix in Sklearn 3

**Attention !**

Comme les valeurs cibles négatives correspondent à 0 et les positives à 1, scikit-learn les a classées dans cet ordre. Assurez-vous de bien vérifier que vous interprétez les valeurs correctement.

# Exercice d'application 9

## Confusion matrix

Soit la matrice de confusion suivante :

| | |
|---|---|
| 4 | 1 |
| 3 | 2 |

Complétez :

1. True positives : .....
2. False positives : .....
3. False negatives : .....
4. True negatives : .....

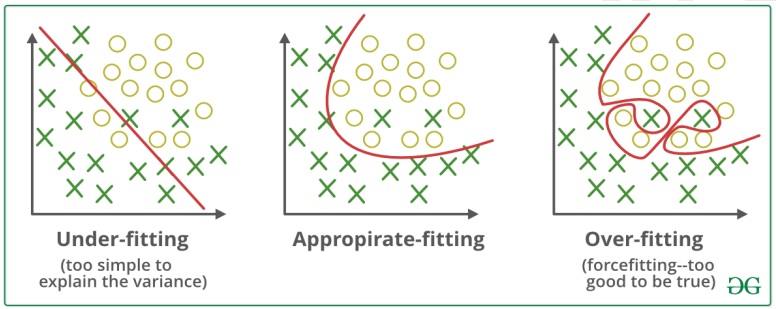
# Overfitting ou surapprentissage 1

## Definition

Jusqu'à présent, nous avons construit un modèle avec toutes nos données, puis nous avons observé ses performances sur ces mêmes données. Cela gonfle artificiellement nos chiffres puisque notre modèle a pu voir les réponses au quiz avant que nous lui donnions le quiz. Cela peut conduire à ce que nous appelons **l'overfitting (le surapprentissage)**. On parle **surapprentissage** (d'ajustement excessif) **lorsque nous obtenons de bons résultats sur les données que le modèle a déjà vues, mais que nous n'obtenons pas de bons résultats sur les nouvelles données**.

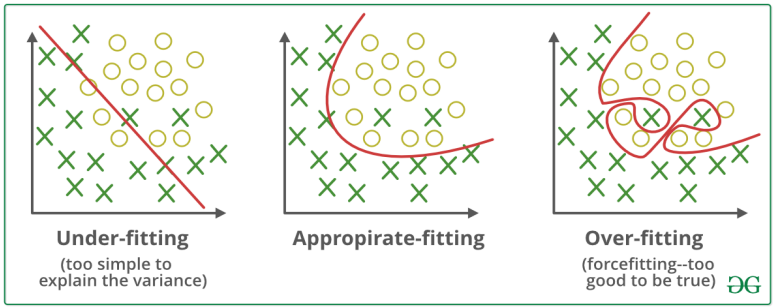
# Overfitting ou surapprentissage 2

On peut voir un modèle surajusté ou surentraîné (overfit) comme suit:




Dans le graphique 3, la ligne est trop serrée et tente de placer chaque point de données du bon côté de la ligne, mais elle ne tient pas compte de l'essence des données.

# Overfitting ou surapprentissage 3




Dans le graphique 3, vous pouvez voir que nous avons fait un assez bon travail pour obtenir les points jaunes à droite et les points verts à gauche, mais cela ne reflète pas ce qui se passe. Un seul point aberrant pourrait totalement perturber l'emplacement de la ligne. Alors que le modèle obtiendrait un excellent score sur les données qu'il a déjà vues, il est peu probable qu'il soit performant sur les nouvelles données.

# Overfitting ou surapprentissage 4

**Attention !**

Plus nos données comporteront des variables dépendantes, plus nous serons enclins au surapprentissage (overfitting).

# Exercice d'application 10

## Overfitting : choisir les bonnes réponses

On parle d'overfitting lorsque :

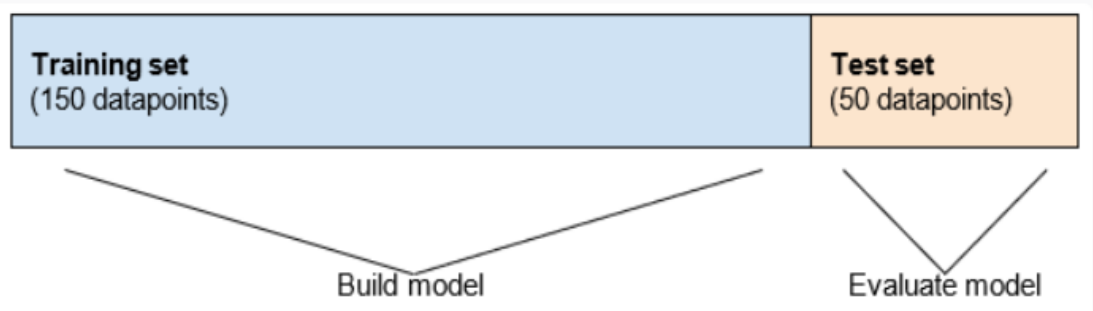
1. nous obtenons de meilleures performances (lorsqu'on réalise des prédictions) sur les nouvelles données .
2. nous avons de piètres performances sur les données que le modèle avait déjà vues.
3. nous avons de meilleures performances sur les données que le modèle avait déjà vues.
4. nous n'avons de pas de meilleures performances sur les nouvelles données.

# Ensemble d'entraînement et ensemble de test 1

Pour donner à un modèle une évaluation non biaisée, nous aimerions savoir comment notre modèle se comporteraient sur des données qu'elles n'ont pas encore vues.
En effet, notre modèle fera des prédictions sur des données dont nous ne connaissons pas la réponse. Nous aimerions donc évaluer la performance de notre modèle sur de nouvelles données, et pas seulement sur les données qu'il a déjà vues. Pour simuler la réalisation de prédictions sur de nouvelles données non vues, nous pouvons **diviser notre ensemble de données en un ensemble d'apprentissage et un ensemble de test**. **L'ensemble d'apprentissage est utilisé pour construire les modèles. L'ensemble de test est utilisé pour évaluer les modèles**. Nous divisons nos données avant de construire le modèle, ainsi le modèle n'a aucune connaissance de l'ensemble de test et nous lui donnerons une évaluation bien meilleure.

# Ensemble d'entraînement et ensemble de test 2

Si notre ensemble de données contient 200 points de données, la répartition en un **ensemble d'apprentissage (training set) et un ensemble de test (test set)** peut se présenter comme suit :



# Répartition entre ensemble d'entraînement et ensemble de test



**Attention !**

**Une répartition standard consiste à placer 70 à 80% de nos données dans l'ensemble d'apprentissage et 20 à 30% dans l'ensemble de test**. En utilisant moins de données dans l'ensemble d'apprentissage, notre modèle n'aura pas autant de données pour apprendre, nous voulons donc lui en donner le plus possible tout en en laissant suffisamment pour l'évaluation.

# Exercice d'application 11

## Training set and test set

Lequel permet d'évaluer le modèle ?

1. training set
2. test set
3. l'ensemble des données
4. la base de données

# Training set and test set in Sklearn 1

Scikit-learn dispose d'une fonction intégrée pour **diviser les données en un ensemble d'entraînement et un ensemble de test**.

En supposant que nous ayons un tableau numpy bidimensionnel X de nos prédicteurs et un tableau numpy unidimensionnel y de la cible, nous pouvons utiliser **la fonction train_test_split()**. Elle placera **aléatoirement chaque point de données** soit dans l'ensemble **d'entraînement**, soit dans l'ensemble **de test**. **Par défaut, l'ensemble d'apprentissage représente 75% des données et l'ensemble de test représente les 25% restants des données**.

Nous pouvons utiliser l'attribut shape pour voir les tailles de nos ensembles de données.

# Training set and test set in Sklearn 2

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
X_train, X_test, y_train, y_test = train_test_split(X, y)
print("whole dataset:", X.shape, y.shape) ⟹ (887, 6) (887,)
print("training set:", X_train.shape, y_train.shape) ⟹ (665, 6)
(665,)
print("test set:", X_test.shape, y_test.shape) ⟹ (222, 6) (222,)
```

# Training set and test set in Sklearn 3

Nous pouvons voir que sur les 887 points de données de notre ensemble de données, 665 sont dans notre ensemble d'apprentissage et 222 dans l'ensemble de test. Chaque point de données de notre ensemble de données est utilisé exactement une fois, soit dans l'ensemble d'apprentissage, soit dans l'ensemble de test. Notez que nous avons 6 prédicteurs dans notre ensemble de données, donc nous avons toujours 6 prédicteurs dans notre ensemble d'entraînement et notre ensemble de test.

**Attention !**

Nous pouvons modifier la taille de notre ensemble d'apprentissage en utilisant le paramètre **train_size**. Par exemple, **train_test_split(X, y, train_size=0.6)** place **60% des données dans l'ensemble d'apprentissage et 40% dans l'ensemble de test**.

# Exercice d'application 12

## Training set and test set in Sklearn

Nous avons un tableau numpy 2d X de 100 points de données et 4 prédicteurss et un tableau 1d y de 100 valeurs cibles. Quel est le résultat du programme ci-dessous ?

```
X_train, X_test, y_train, y_test = train_test_split(X, y)

print(X_train.shape, y_train.shape)
```

( ..... , .....) ( ..... ,)

# Construction d'un modèle dans Sklearn 1

Maintenant que nous savons comment diviser nos données en un ensemble d'apprentissage et un ensemble de test, nous devons modifier la façon dont nous construisons et évaluons le modèle. Toute la construction du modèle se fait avec l'ensemble d'apprentissage et toute l'évaluation se fait avec l'ensemble de test.

Dans le chapitre précédent, nous avons construit un modèle et l'avons évalué sur le même ensemble de données. Maintenant, nous construisons le modèle en utilisant l'ensemble d'entraînement:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

# Construction d'un modèle dans Sklearn 2

Évaluons le modèle sur l'ensemble de test :
print(model.score(X_test, y_test))
En fait, toutes les métriques que nous calculons dans les parties précédentes doivent être calculées sur l'ensemble de test :
evaluating the model
y_pred = model.predict(X_test)
print("accuracy:", accuracy_score(y_test, y_pred)) $\Longrightarrow$ 0.833333
print("precision:", precision_score(y_test, y_pred)) $\Longrightarrow$ 0.793650
print("recall:", recall_score(y_test, y_pred)) $\Longrightarrow$ 0.675675
print("f1 score:", f1_score(y_test, y_pred)) $\Longrightarrow$ 0.729927

**Attention !**

Les valeurs des métriques peuvent changer d'une exécution à une autre.

# Construction d'un modèle dans Sklearn 3

Nos valeurs d'accuracy, de précision, de rappel et de score F1 sont en fait très similaires aux valeurs obtenues lorsque nous avons utilisé l'ensemble des données. C'est un signe que notre modèle n'est pas surajusté (n'est pas overfitting).

**Attention !**

Si vous exécutez le code, vous remarquerez que vous obtenez des scores différents à chaque fois. Cela s'explique par le fait que la répartition entraînement-test est effectuée de manière aléatoire et que les scores seront différents selon les points qui se trouvent dans l'ensemble de formation et dans l'ensemble de test. Nous verrons, lors de la leçon sur la validation croisée, que nous disposons de moyens plus précis pour mesurer ces scores.

# Exercice d'application 13

## Training set and test set

Quelle est l'instruction correcte ?

1. model.fit(X_test, y_test) – print(model.score(X_train, y_train))
2. model.fit(X_train, y_train) – print(model.score(X_train, y_train))
3. model.fit(X, y) – print(model.score(X_test, y_test))
4. model.fit(X_train, y_train) – print(model.score(X_test, y_test))
5. model.fit(X_train, y_test) print(model.score(X_train, y_test))

# Utilisation d'un état aléatoire (random state or seed) 1

Comme nous l'avons remarqué dans la partie précédente, lorsque nous divisons aléatoirement les données en un ensemble d'apprentissage et un ensemble de test, nous nous retrouvons avec des points de données différents dans chaque ensemble à chaque fois que nous exécutons le code. C'est le résultat du caractère aléatoire, et nous avons besoin qu'il soit aléatoire pour qu'il soit efficace, mais cela peut parfois rendre difficile le test du code.

# Random state or seed 2

Par exemple, chaque fois que nous exécutons le code suivant, nous obtenons des résultats différents :

```
from sklearn.model_selection import train_test_split
X = [[1, 1], [2, 2], [3, 3], [4, 4]]
y = [0, 0, 1, 1]
X_train, X_test, y_train, y_test = train_test_split(X, y)
print('X_train', X_train)
print('X_test', X_test)
```

# Random state or seed 3

Pour obtenir **la même répartition à chaque fois**, nous pouvons utiliser l'attribut **random_state**. Nous choisissons un **nombre arbitraire à lui donner**, et **à chaque fois que nous exécutons le code, nous obtenons la même répartition**.

```
from sklearn.model_selection import train_test_split
X = [[1, 1], [2, 2], [3, 3], [4, 4]]
y = [0, 0, 1, 1]
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=2)
print('X_train', X_train)
print('X_test', X_test)
```

# Exercice d'application 14

## Random state or seed

Nous utilisons un paramètre random_state pour nous assurer que nous obtenons la même répartition aléatoire chaque fois que le même code est exécuté :

1. True
2. False

# Seuil de régression logistique 1

Si vous vous souvenez de la section 2, nous avons parlé du **compromis entre la précision et le rappel**. Avec un **modèle de régression logistique**, nous disposons d'un moyen facile de passer de la précision au recall. Le modèle de régression logistique ne **renvoie** pas seulement une **prédiction**, mais aussi une **valeur de probabilité** comprise entre 0 et 1. En général, si la valeur est $>=$**0.5**, nous prédisons que l**e passager a survécu**, et si la valeur est $<$**0.5**, le passager **n'a pas survécu**. Cependant, nous pourrions choisir n'importe quel seuil entre 0 et 1.

# Seuil de régression logistique 2

Si nous **augmentons le seuil**, nous aurons **moins de prédictions positives**, mais ces dernières auront plus de chances d'être correctes. Cela s**ignifie que la précision sera plus élevée et le rappel plus faible**. D'un autre côté, si **nous abaissons le seuil**, nous aurons **plus de prédictions positives**, et nous aurons donc plus de chances d'attraper tous les cas positifs. Cela signifie que **le rappel sera plus élevé et la précision plus faible**.

**Attention !**

Chaque choix de seuil est un modèle différent. **Une courbe ROC (Receiver operating characteristic) est un graphique montrant tous les modèles possibles et leurs performances**.

# Exercice d'application 15

## Seuil de regression logistique

Si nous choisissons un seuil de 0,75, laquelle des affirmations suivantes est vraie concernant nos prédictions et notre précision ?

1. plus de prédictions négatives / la précision sera plus faible
2. les prédictions plus négatives / la précision sera plus élevée
3. plus de prédictions positives / la précision sera plus élevée

# Sensitivity and specificity 1

**Definition**

Une courbe ROC est un graphique de la sensibilité par rapport à la spécificité. Ces valeurs présentent le même compromis que la précision et le rappel.

Revenons sur la matrice de confusion, car nous l'utiliserons pour définir la sensibilité et la spécificité :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | TP | FP |
| **Predicted negative** | FN | TN |

La sensibilité,encore appelé le rappel (recall), c'est le taux de vrais positifs. Rappelons qu'elle est calculée comme suit :

$$recall = sensitivity = \frac{nombre.de.predictions.positives.correctes}{nombre.de.cas.positifs} = \frac{TP}{TP+FN}$$

# Sensitivity and specificity 2

La spécificité (specificity) est le taux de vrais négatifs (true negative rate) :

$$specificity = \frac{nombre.de.predictions.negatives.correctes}{nombre.de.cas.negatifs} = \frac{TN}{TN+FP}$$

Nous avons effectué un train test split (diviser nos données entre train et test) sur notre jeu de données Titanic et obtenu la matrice de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 61 | 21 |
| **Predicted negative** | 35 | 105 |

Nous avons 96 cas positifs et 126 cas négatifs dans notre ensemble de test.

# Sensitivity and specificity 3

Calculons la sensibilité et la spécificité :
sensibility $= 61/96 = 0.6354$
specificity $= 105/126 = 0.8333$
L'objectif est de maximiser ces deux valeurs, bien qu'en général, le fait d'augmenter l'une dimunie l'autre. Selon la situation, on mettra davantage l'accent sur la sensibilité ou la spécificité.

**Attention !**

Alors que nous examinons généralement les valeurs de précision et de rappel, pour **la représentation graphique**, la norme est d'utiliser **la sensibilité et la spécificité**. Il est possible de construire une courbe précision-rappel, mais ce n'est pas une pratique courante.

# Exercice d'application 16

## Sensitivity and specificity

Soit la matrice de confusion suivante:

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 30 | 20 |
| **Predicted negative** | 10 | 40 |

Choisir la bonne réponse de la sensibilité et de la spécificité :

1. 0.22 / 0.40
2. 0.3 / 0.2
3. 0.75 / 0.666

# Sensitivity and specificity in Sklearn 1

Scikit-learn n'a pas défini de fonctions pour la sensibilité et la spécificité, mais nous pouvons le faire nous-mêmes. La sensibilité est la même que le rappel, elle est donc facile à définir :

```
from sklearn.metrics import recall_score
sensitivity_score = recall_score
print(sensitivity_score(y_test, y_pred))
```

Maintenant, pour définir la spécificité, si nous réalisons que c'est aussi le rappel de la classe négative, nous pouvons obtenir la valeur de la fonction **sklearn precision_recall_fscore_support()**.

```
from sklearn.metrics import precision_recall_fscore_support
print(precision_recall_fscore_support(y_test, y_pred))
(array([0.81118881, 0.70886076]), array([0.83453237, 0.6746988
]), array([0.82269504, 0.69135802]), array([139, 83], dtype=int64))
```

# Sensitivity and specificity in Sklearn 2

Le deuxième tableau est le rappel, nous pouvons donc ignorer les trois autres tableaux. Il y a deux valeurs. La première est le rappel de la classe négative et la seconde est le rappel de la classe positive. La deuxième valeur est le rappel standard ou la valeur de sensibilité, et vous pouvez voir que la valeur correspond à ce que nous avons obtenu ci-dessus. La première valeur est la spécificité. Écrivons donc une fonction pour obtenir cette valeur.

```
def specificity_score(y_true, y_pred):
    p, r, f, s = precision_recall_fscore_support(y_true, y_pred)
    return r[0]
print(specificity_score(y_test, y_pred)) ⟹ 0.83453237
```

**Attention !**

La sensibilité est identique au rappel (ou rappel de la classe positive) et la spécificité est le rappel de la classe négative.

# Sensitivity and specificity in Sklearn 3

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score,
precision_recall_fscore_support
sensitivity_score = recall_score
def specificity_score(y_true, y_pred):
    p, r, f, s = precision_recall_fscore_support(y_true, y_pred)
    return r[0]
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

# Sensitivity and specificity in Sklearn 4

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=5)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("sensitivity:", sensitivity_score(y_test, y_pred))
print("specificity:", specificity_score(y_test, y_pred))
```

# Exercice d'application 17

## Sensitivity and specificity in Sklearn

Quelle valeur obtenons-nous avec le code suivant ? p, r, f, s = precision_recall_fscore_support(y_test, y_pred)
print(r[1])

1. precision
2. specificity
3. sensibility

# Seuil de régression logistique dans Sklearn 1

Lorsqu'on utilise la méthode de **predict()** de Scikit-learn, les valeurs **0 et 1 de la prédiction vous sont données**. Cependant, en coulisses, le modèle de régression logistique obtient une valeur de probabilité entre 0 et 1 pour chaque point de données, puis arrondit à 0 ou 1. Si nous voulons choisir un seuil différent de 0,5, nous aurons besoin de ces **valeurs de probabilité**.

# Seuil de régression logistique dans Sklearn 2

Nous pouvons utiliser **la fonction predict_proba** pour les obtenir :

```
model.predict_proba(X_test)
```

Le résultat est un tableau numpy avec 2 valeurs pour chaque point de données (par exemple [0.78, 0.22]). Vous remarquerez que la somme des deux valeurs est égale à 1. La première valeur est la probabilité que le point de données soit dans la classe 0 (n'a pas survécu) et la seconde est la probabilité que le point de données soit dans la classe 1 (a survécu).

# Seuil de régression logistique dans Sklearn 3

Nous n'avons besoin que de la deuxième colonne de ce résultat, que nous pouvons extraire grâce à la syntaxe suivante :
model.predict_proba(X_test)[:, 1]
Maintenant, nous voulons simplement comparer ces valeurs de probabilité avec notre seuil. Supposons que nous voulons un seuil de 0.75. Nous comparons le tableau ci-dessus à 0.75. Cela nous donnera un tableau de valeurs Vrai/Faux qui sera notre tableau de valeurs cibles prédites.
$y\_pred = model.predict\_proba(X\_test)[:, 1] > 0.75$
Un seuil de 0,75 signifie que nous devons être plus confiants pour faire une prédiction positive. Cela se traduit par moins de prédictions positives et plus de prédictions négatives.

# Seuil de régression logistique dans Sklearn 4

Nous pouvons maintenant utiliser n'importe quelle métrique scikit-learn en utilisant y_test comme valeurs réelles et y_pred comme valeurs prédites :

```
print("precision:", precision_score(y_test, y_pred))
print("recall:", recall_score(y_test, y_pred))
```

# Seuil de régression logistique dans Sklearn 5

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score
from sklearn.model_selection import train_test_split
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

# Seuil de régression logistique dans Sklearn 6

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1] > 0.75
print("precision:", precision_score(y_test, y_pred))
print("recall:", recall_score(y_test, y_pred))
```

**Attention !**

En fixant le seuil à 0,5, on obtient le modèle de régression logistique original. Toute autre valeur de seuil donne un autre modèle.

# Exercice d'application 18

## Seuil de régression logistique dans Sklearn

Laquelle des propositions suivantes nous donne un tableau des probabilités prédites (chaque valeur sera la probabilité que le point de données appartienne à la classe positive) ?

1. model.predict_proba(X_test)[1]
2. model.predict_proba(X_test)
3. model.predict_proba(X_test)[:, 0]
4. model.predict_proba(X_test)[:, 1]
5. model.predict_proba(X_test)[0]

# Construction du ROC 1

La courbe ROC est un graphique de la spécificité par rapport à la sensibilité. Soit un modèle de régression logistique, calculons la spécificité et la sensibilité pour chaque seuil possible. Chaque probabilité prédite est un seuil. Si nous avons 5 points de données avec les probabilités prédites suivantes : 0.3, 0.4, 0.6, 0.7, 0.8, nous utiliserons chacune de ces 5 valeurs comme seuil.

Notez que nous traçons la sensibilité par rapport à la 1-spécificité (uniquement la deuxième colonne). Il n'y a pas de raison particulière de procéder de cette façon, si ce n'est que c'est la norme.
Commençons par examiner le code permettant de **construire la courbe ROC**. Scikit-learn possède une fonction **roc_curve()** que nous pouvons utiliser. Cette fonction prend les vraies valeurs de la variable cible et les probabilités prédites par notre modèle.

# Construction du ROC 2

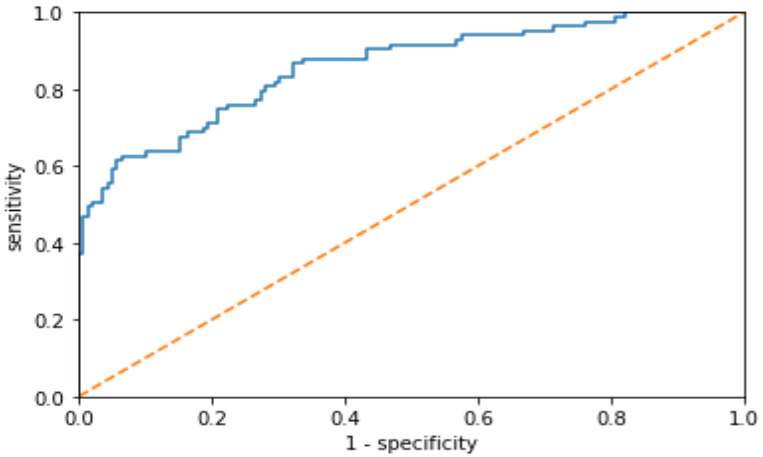
Nous utilisons d'abord la méthode predict_proba() sur le modèle pour obtenir les probabilités. Ensuite, nous appelons la fonction roc_curve. La **fonction roc_curve() renvoie un tableau des taux de faux positifs, un tableau des taux de vrais positifs et les seuils**. Le taux de faux positifs est égal à 1-spécificité (axe des x) et le taux de vrais positifs est un autre terme pour la sensibilité (axe des y). Les valeurs de seuil ne seront pas nécessaires dans le graphique.

Ci-dessous le code pour tracer la courbe ROC dans matplotlib. Notez que nous avons également du code pour tracer une ligne diagonale. Cela peut nous aider à voir visuellement à quelle distance se trouve notre modèle par rapport à un modèle qui prédit de manière aléatoire.

# Construction du ROC 3

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_curve
import matplotlib.pyplot as plt
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred_proba = model.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba[:,1])
plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], linestyle='–')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('1 - specificity') v plt.ylabel('sensitivity')
plt.show()
```

# Construction du ROC 4




**Attention !**

Comme nous n'utilisons pas les valeurs seuils pour construire le graphique, celui-ci ne nous indique pas quel seuil donnerait chacun des modèles possibles.

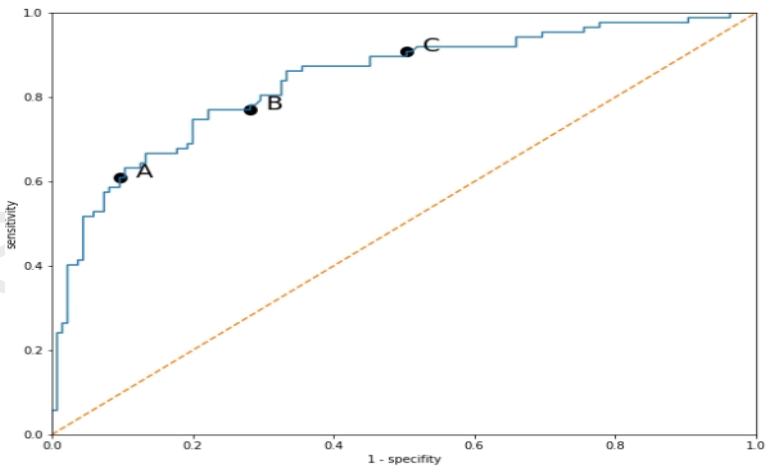
# Exercice d'application 19

## Construction du ROC

Écrire un programme qui permet de construire la courbe ROC avec scikit-learn.

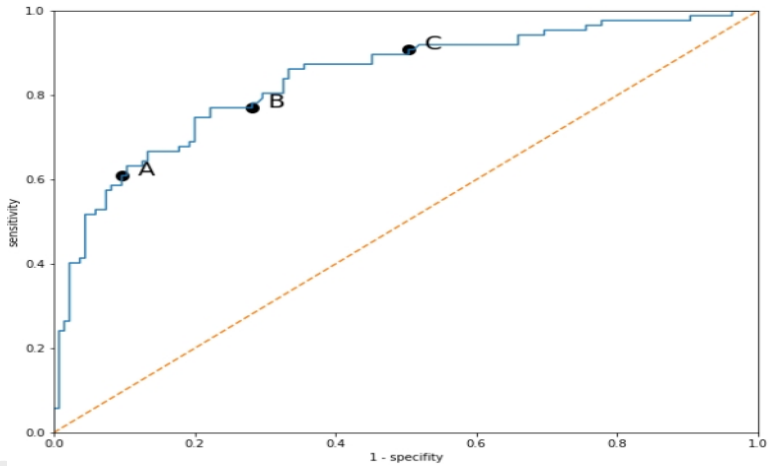
# Interprétation de la courbe ROC 1

La courbe ROC montre la performance, non pas d'un seul modèle, mais de plusieurs modèles. Chaque choix de seuil est un modèle différent.
Soit courbe ROC avec les points mis en évidence ci-dessous :




Chaque point A, B et C correspond à un modèle avec un seuil différent.

# Interprétation de la courbe ROC 2




Le modèle A a une sensibilité de 0,6 et une spécificité de 0,9 (rappelons que le graphique indique une spécificité de 1).
Le modèle B a une sensibilité de 0,8 et une spécificité de 0,7.
Le modèle C a une sensibilité de 0,9 et une spécificité de 0,5.

# Interprétation de la courbe ROC 3

**La façon de choisir entre ces modèles dépendra des spécificités de notre situation.**

**Attention !**

Plus la courbe se rapproche du coin supérieur gauche, meilleures sont les performances. La courbe ne doit jamais tomber en dessous de la ligne diagonale, car cela signifierait que les performances sont inférieures à celles d'un modèle aléatoire.

# Exercice d'application 19

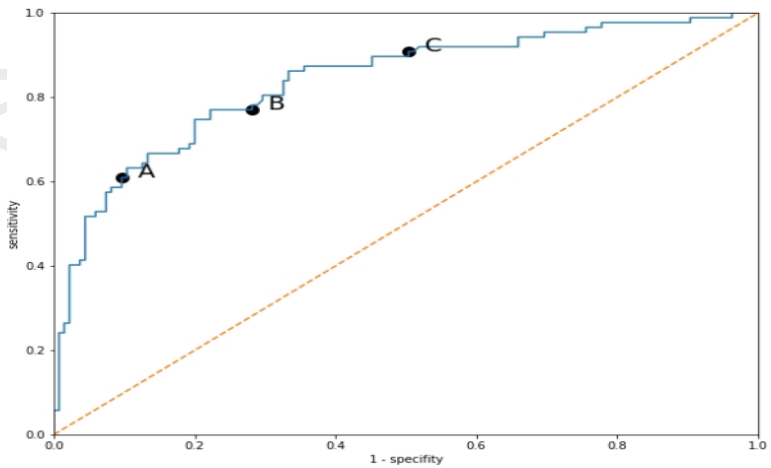
## Interprétation de la courbe ROC

Dans quel coin du tracé de la courbe ROC se situerait le modèle idéal ?

1. Nord-Ouest
2. Nord-Est
3. Sud-Ouest
4. Nord-Ouest

# Choix de la courbe ROC 1

Lorsque nous sommes prêts à finaliser notre modèle, nous devons choisir un seuil unique que nous utiliserons pour faire nos prédictions. La courbe ROC est un moyen de nous aider à choisir le seuil idéal pour notre problème.
Regardons à nouveau notre courbe ROC avec trois points mis en évidence :



# Choix de la courbe ROC 2

Si nous sommes dans une situation où **il est plus important que toutes nos prédictions positives soient correctes que nous attrapions tous les cas positifs (ce qui signifie que nous prédisons correctement la plupart des cas négatifs), nous devrions choisir le modèle avec une spécificité plus élevée (modèle A)**.

Si nous sommes dans une situation où **il est important d'attraper le plus grand nombre possible de cas positifs, nous devrions choisir le modèle ayant la plus grande sensibilité (modèle C)**.

# Choix de la courbe ROC 3

Si nous voulons un équilibre entre la sensibilité et la spécificité, nous devons choisir le modèle B.

**Attention !**

Il peut être difficile de garder la trace de tous ces termes. Même les experts doivent les consulter à nouveau pour s'assurer qu'ils interprètent correctement les valeurs.

# Exercice d'application 20

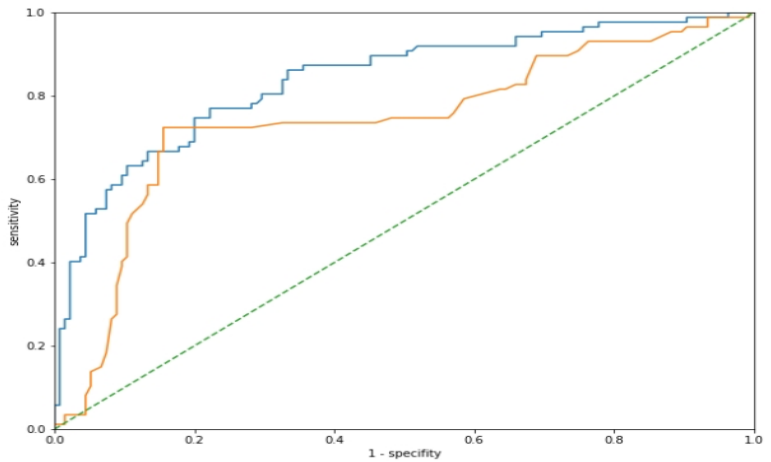
## Choix du modèle grâce à la courbe ROC

Supposons que nous ayons un modèle permettant de prédire les fraudes à la carte de crédit. Si nous détectons des frais frauduleux sur le compte d'une personne, nous allons désactiver sa carte de crédit. Nous voulons donc nous assurer que notre prédiction est exacte lorsqu'elle est positive. Lequel des trois modèles du tracé ROC est préféré dans ce cas ?

1. modèle B
2. modèle A
3. modèle C

# Aire sous la courbe 1

Nous utiliserons parfois la courbe ROC pour comparer deux modèles différents. Voici une comparaison des courbes ROC de deux modèles.




Vous pouvez constater que **la courbe bleue est plus performante que la courbe orange** puisque la **ligne bleue est presque toujours supérieure à la ligne orange**.

# Aire sous la courbe 2

Pour obtenir une mesure empirique de ce résultat, nous calculons l'aire sous la courbe, également appelée **AUC (Area Under the Curve)**. Il s'agit de l'aire sous la courbe ROC. Il s'agit d'une **valeur comprise entre 0 et 1**, **plus elle est élevée, mieux c'est**.

**Comme le ROC est un graphique de tous les différents modèles de régression logistique avec différents seuils, l'AUC ne mesure pas la performance d'un seul modèle. Elle donne un sens général de la performance du modèle de régression logistique. Pour obtenir un modèle unique, vous devez encore trouver le seuil optimal pour votre problème.**

# Aire sous la courbe 3

Scikit-learn grâce à la **fonction roc_auc_score()** permet de **calculer l'aire sous la courbe** :

(roc_auc_score(y_test, y_pred_proba[:,1])

Voici les valeurs des deux courbes (bleue et orange) :

Blue AUC: 0.8379
Orange AUC: 0.7385

Nous constatons empiriquement ($0.8379 > 0.7385$) que la courbe bleue est la plus performante.

# Aire sous la courbe 4

Nous pouvons utiliser la fonction roc_auc_score() pour calculer le score AUC d'un modèle de régression logistique sur les données du Titanic. Nous construisons deux modèles de régression logistique, le modèle 1 avec 6 prédicteurs et le modèle 2 avec seulement les prédicteurs Pclass et male (variable booléenne que nous avons créée). **Nous constatons que le score AUC du modèle 1 est plus élevé à chaque exécution**.

# Aire sous la courbe 5

```
from sklearn.metrics import roc_auc_score
model1 = LogisticRegression()
model1.fit(X_train, y_train)
y_pred_proba1 = model1.predict_proba(X_test)
print("model 1 AUC score:", roc_auc_score(y_test, y_pred_proba1[:,
1]))
model2 = LogisticRegression()
model2.fit(X_train[:, 0:2], y_train)
y_pred_proba2 = model2.predict_proba(X_test[:, 0:2])
print("model 1 AUC score:", roc_auc_score(y_test, y_pred_proba2[:,
1]))
```

# Aire sous la courbe 6

**Attention !**

Il est important de noter que cette métrique nous indique la performance générale d'un modèle de régression logistique sur nos données. **Comme une courbe ROC montre la performance de plusieurs modèles, l'AUC ne mesure pas la performance d'un seul modèle**.

# Exercice d'application 21

## AUC

Complétez le programme ci-dessous pour calculer le score AUC d'un modèle de régression logistique. Supposez que X_train, X_test, y_train, y_test ont déjà été créés.

```
model = ..... ()
model. ..... (X_train, y_train)
y_pred_proba = model.predict.proba( ..... )
print(roc_auc_score( ..... , y_pred_proba[:, 1]))
```

# Machine Learning (ML)

## Chapitre 3 : Évaluation des modèles
## Partie II

David TCHOUTA

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

April 9, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Cross Validation
3. Cross Validation in Sklearn
4. Comparaison des modèles
5. Quiz

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue la notion de validation croisée et nous verrons comment s'effectue la comparaison des modèles.

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

- de comprendre et d'utiliser la validation croisée dans Scikit-klearn
- d'effectuer la comparaison des modèles

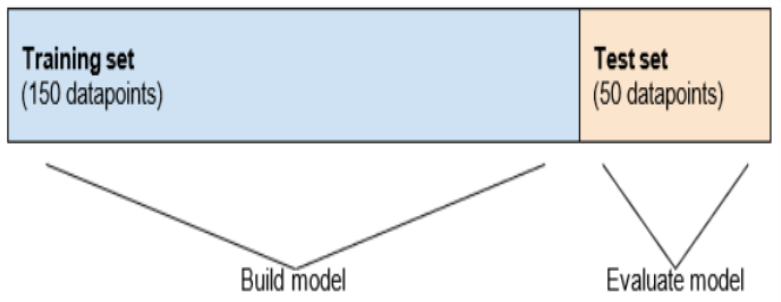
# k-fold Cross Validation 1

Nous procédons à une évaluation parce que nous voulons obtenir une mesure précise de la performance du modèle. Si notre ensemble de données est petit, notre ensemble de test le sera aussi. Il se peut donc qu'il ne soit pas un bon assortiment aléatoire de points de données et que, par hasard, nous nous retrouvions avec des points de données faciles ou difficiles dans notre ensemble d'évaluation (de test).

Puisque notre objectif est d'obtenir la meilleure mesure possible de nos métriques (accuracy, précision, rappel et score F1), nous pouvons faire un peu mieux qu'un seul ensemble de formation et de test.

# k-fold Cross Validation 2

Pour rappel la répartition entre ensemble d'entraînement et ensemble de test est la suivante :




Comme nous pouvons le voir, toutes les valeurs de l'ensemble d'entraînement ne sont jamais utilisées pour l'évaluation. Il serait injuste de construire le modèle avec l'ensemble d'entraînement puis de l'évaluer avec cet même ensemble, mais nous n'obtenons pas une image aussi complète que possible des performances du modèle.

# k-fold Cross Validation 3

Pour le constater de manière empirique, exécutons le code de la section 3 qui effectue une séparation train/test. Nous allons le ré-exécuter plusieurs fois et voir les résultats. Chaque ligne est le résultat d'une répartition aléatoire différente entre le train et le test.

| | **Accuracy** | **Precision** | **Recall** | **F1 Score** |
|---|---|---|---|---|
| 1 | 0.83 | 0.77 | 0.78 | 0.77 |
| 2 | 0.79 | 0.78 | 0.62 | 0.69 |
| 3 | 0.81 | 0.77 | 0.74 | 0.75 |
| 4 | 0.81 | 0.75 | 0.70 | 0.72 |
| 5 | 0.81 | 0.77 | 0.70 | 0.73 |

# k-fold Cross Validation 4

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
from sklearn.model_selection import train_test_split
import numpy as np
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

# k-fold Cross Validation 5

```
building the model
model = LogisticRegression()
model.fit(X_train, y_train)
evaluating the model
y_pred = model.predict(X_test)
print(" accuracy: 0:.3f".format(accuracy_score(y_test, y_pred)))
print("precision: 0:.3f".format(precision_score(y_test, y_pred)))
print(" recall: 0:.3f".format(recall_score(y_test, y_pred)))
print(" f1 score: 0:.3f".format(f1_score(y_test, y_pred)))
```

# k-fold Cross Validation 6

Vous pouvez voir que chaque fois que nous l'exécutons, nous obtenons des valeurs différentes pour les métriques. L'accuracy varie de 0.79 à 0.83, la précision de 0.75 à 0,78 et le rappel de 0,62 à 0,78. Il s'agit de larges fourchettes qui dépendent simplement de la chance ou de la malchance que nous avons eue quant aux points de données qui se sont retrouvés dans l'ensemble de test.

**Attention !**

Au lieu d'effectuer une seule division formation/test, nous allons diviser nos données en un ensemble de formation et un ensemble de test plusieurs fois.

# Exercice d'application 1

## Division du Training set et test set

La division de l'ensemble de données en un seul ensemble de train et un seul ensemble de test à des fins d'évaluation pourrait donner une mesure inexacte des paramètres d'évaluation :

1. si l'ensemble des données est petit
2. si l'ensemble des données est large

# Mutiples ensembles d'entraînement et de test 1

Nous avons appris dans la partie précédente qu'en fonction de notre ensemble de tests, nous pouvons obtenir différentes valeurs pour les métriques d'évaluation. Nous voulons obtenir une mesure de l'efficacité de notre modèle en général, et pas seulement une mesure de son efficacité sur un ensemble de tests spécifique.

Au lieu de prendre une partie des données comme ensemble de test, nous allons diviser notre ensemble de données en 5 parties. Supposons que nous ayons 200 points de données dans notre ensemble de données :

| chunk 1 | chunk 2 | chunk 3 | chunk 4 | chunk 5 |
|---|---|---|---|---|

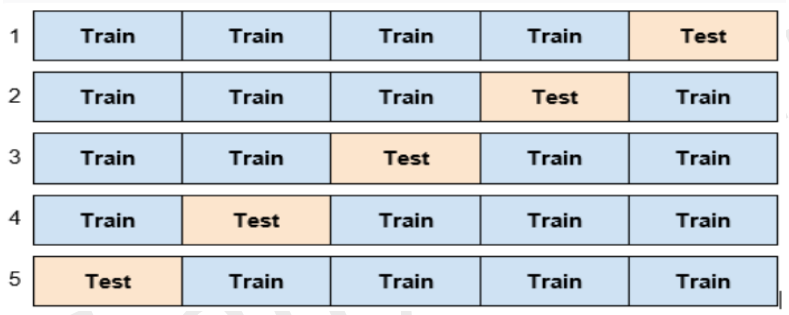
# Mutiples ensembles d'entraînement et de test 2

Chacun de ces 5 chunks servira d'ensemble de test. Lorsque le chunk 1 est l'ensemble de test, nous utilisons les 4 chunks restants comme ensemble d'entraînement. Nous avons donc 5 ensembles d'entraînement et de test comme suit :

| | | | | | |
|---|---|---|---|---|---|
| 1 | Train | Train | Train | Train | Test |
| 2 | Train | Train | Train | Test | Train |
| 3 | Train | Train | Test | Train | Train |
| 4 | Train | Test | Train | Train | Train |
| 5 | Test | Train | Train | Train | Train |

À chaque fois, nous disposons d'un ensemble de test de 20% (40 points de données) et d'un ensemble d'apprentissage de 80% (160 points de données).

# Mutiples ensembles d'entraînement et de test 3




**Attention !**

Chaque point de données se trouve dans exactement un ensemble de test.

# Exercice d'application 2

## Mutiples ensembles d'entraînement et de test

Si nous disposons d'un ensemble de données de 100 points de données et que nous le divisons en 5 parties pour créer 5 ensembles d'entraînement et de test, combien de points de données y aura-t-il dans chaque ensemble d'entraînement et de test ?

1. 90 points de données de l'ensemble d'apprentissage / 10 points de données de l'ensemble de test
2. 80 points de données de l'ensemble d'apprentissage / 20 points de données de l'ensemble de test
3. 60 points de données de l'ensemble d'apprentissage / 20 points de données de l'ensemble de test

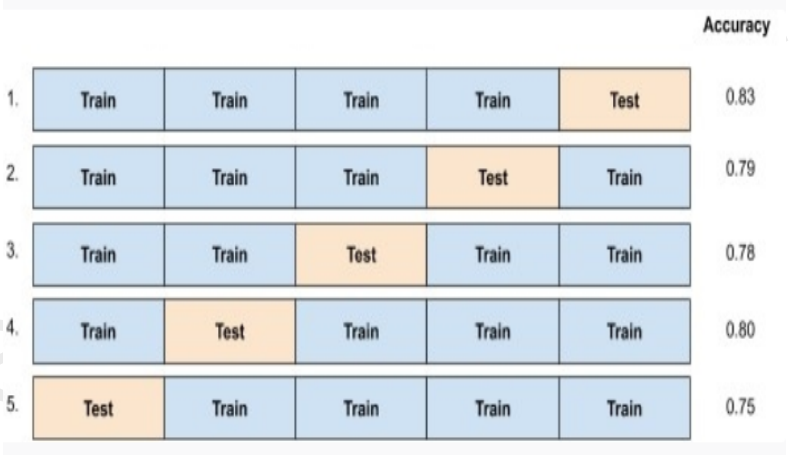
# Construction et évaluation d'un modèle sur de multiples splits 1

Dans la partie précédente, nous avons vu comment nous pouvions créer 5 ensembles de test, chacun avec un ensemble d'entraînement différent.

Maintenant, pour chaque ensemble d'apprentissage, nous construisons un modèle et l'évaluons en utilisant l'ensemble de test associé. Ainsi, nous construisons 5 modèles et calculons 5 scores.

# Construction et évaluation d'un modèle sur de multiples splits 2

Calculons le score de précision de notre modèle :




La précision est la moyenne des 5 valeurs :

$(0.83+0.79+0.78+0.80+0.75)/5 = \mathbf{0.79}$

# Construction et évaluation d'un modèle sur de multiples splits 3

Si nous n'avions utilisé qu'un seul ensemble de formation et de test et que nous avions obtenu le premier au hasard, nous aurions obtenu une précision de 0.83. Si nous avions obtenu le dernier au hasard, nous aurions obtenu une précision de 0.75. La moyenne de toutes ces valeurs possibles permet d'éliminer l'impact de l'ensemble de test dans lequel un point de données atterrit.

Vous ne verrez des valeurs aussi différentes que si vous disposez d'un petit ensemble de données. Avec les grands ensembles de données, nous nous contentons souvent d'un ensemble d'entraînement et de test pour plus de simplicité.

# Construction et évaluation d'un modèle sur de multiples splits 4

Ce processus de **création de plusieurs ensembles d'entraînement et de test** est appelé **validation croisée k-fold (k-fold cross validation)**. Le nombre k est le nombre de morceaux en lesquels nous divisons notre ensemble de données. Le **nombre standard est 5**, comme nous l'avons fait dans notre exemple ci-dessus.

**Attention !**

L'objectif de la validation croisée est d'obtenir des mesures précises pour nos paramètres (accuracy, précision, rappel). Nous construisons des modèles supplémentaires afin d'avoir confiance dans les chiffres que nous calculons et rapportons.

# Exercice d'application 3

## Construction et évaluation d'un modèle sur de multiples splits

Quelle est la valeur de la précision de notre modèle si nous effectuons une validation croisée 5-fold et obtenons les 5 valeurs suivantes pour la précision ?

0.2, 0.5, 0.6, 0.9, 0.8

1. 0.9
2. 0.3
3. 0.5
4. 0.6

# Choix final du modèle suite à la validation croisée 1

Nous avons construit 5 modèles au lieu d'un seul. Comment décider du modèle à utiliser ?

Ces 5 modèles ont été construits uniquement à des fins d'évaluation, afin que nous puissions rapporter les métriques. Nous n'avons pas réellement besoin de ces modèles et nous voulons construire le meilleur modèle possible. **Le meilleur modèle possible sera un modèle qui utilise toutes les données**. Nous gardons donc la trace de nos valeurs calculées pour nos métriques d'évaluation, puis nous construisons un modèle utilisant toutes les données.

# Choix final du modèle suite à la validation croisée 2

Cela peut sembler être un gaspillage incroyable, mais les ordinateurs ont une grande puissance de calcul, il vaut donc les uutiliser un peu plus pour **s'assurer que nous rapportons les bonnes valeurs pour nos métriques d'évaluation**. **Nous utiliserons ces valeurs pour prendre des décisions, il est donc très important de les calculer correctement**.

**Attention !**

La puissance de calcul pour la construction d'un modèle peut être un problème lorsque l'ensemble de données est important. Dans ces cas-là, nous nous contentons de faire une répartition train-test.

# Exercice d'application 4

## Choix final du modèle suite à la validation croisée

Après avoir effectué une validation croisée à 5-fold, comment choisir un modèle unique final ?

1. Créer un modèle en fusionnant les 5 modèles
2. Prenez le modèle avec le meilleur score de précision
3. Construire un nouveau modèle avec toutes les données

# La classe KFold 1

Scikit-learn a déjà implémenté le code permettant de diviser l'ensemble de données en k morceaux (chunks) et de créer k ensembles d'entraînement et de test.

Pour simplifier, prenons un ensemble de données avec seulement 6 points de données et 2 prédicteurs et une validation croisée 3-fold sur l'ensemble de données. Nous prenons les 6 premières lignes de l'ensemble de données Titanic et utilisons uniquement les colonnes Age et Fare :

```
X = df[['Age', 'Fare']].values[:6]
y = df['Survived'].values[:6]
```

# La classe KFold 2

Nous commençons **par instancier un objet de la classe KFold**. Il prend deux paramètres : **n_splits (c'est k, le nombre de morceaux à créer) et shuffle (si oui ou non il faut randomiser l'ordre des données)**. C'est généralement une bonne pratique de mélanger les données puisque vous obtenez souvent un ensemble de données triées :

kf = KFold(n_splits=3, shuffle=True)
La classe KFold possède une méthode de fractionnement qui crée les 3 fractionnements de nos données.
La **méthode split()** renvoie un **générateur**, nous utilisons donc **la fonction list pour le transformer en liste** :
list(kf.split(X))

# La classe KFold 3

```
from sklearn.model_selection import KFold
import pandas as pd
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
X = df[['Age', 'Fare']].values[:6]
y = df['Survived'].values[:6]
kf = KFold(n_splits=3, shuffle=True)
print(list(kf.split(X)))
```

# La classe KFold 4

Nous avons, comme prévu, 3 ensembles d'entraînement et de test. Le premier ensemble d'apprentissage est composé des points de données 0, 2, 3, 5 et l'ensemble de test est composé des points de données 1, 4 (vous pouvez ne pas avoir les mêmes résultats compte tenu du fait de shuffle = True).

**Attention !**

Le fractionnement est effectué de manière aléatoire. Attendez-vous donc à voir des points de données différents dans les ensembles à chaque fois que vous exécutez le code.

# Exercice d'application 5

## La classe KFold

Soit un tableau numpy à 2 dimensions X des prédicteurs et un tableau numpy à 1 dimension y des valeurs cibles. Terminez le code ci-dessous pour construire l'objet k-fold avec k=5 et générer les 5 chunks :

```
from sklearn.model_selection import KFold
kf = KFold(n_splits= ..... , shuffle=True)
chunks = kf. ..... (X)
```

# Création des ensembles d'apprentissage et de test avec les fold 1

**Nous avons utilisé la classe KFold class et la méthode split() pour obtenir les indices qui se trouvent dans chacun des splits**. Maintenant, utilisons ce résultat pour obtenir notre premier (des 3) fractionnement train/test :
Tout d'abord, extrayons le premier fractionnement :

```
splits = list(kf.split(X))
first_split = splits[0]
print(first_split)
(array([0, 2, 4, 5]), array([1, 3]))
```

# Création des ensembles d'apprentissage et de test avec les fold 2

Le premier tableau correspond aux indices de l'ensemble d'apprentissage et le second aux indices de l'ensemble de test. Créons ces variables :

```
train_indices, test_indices = first_split
print("training set indices:", train_indices)
print("test set indices:", test_indices)
training set indices: [0, 2, 4, 5]
test set indices: [1, 3]
```

# Création des ensembles d'apprentissage et de test avec les fold 3

Maintenant nous pouvons créer un X_train, un y_train, un X_test et un y_test basés sur ces indices :

X_train = X[train_indices]
X_test = X[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]

# Création des ensembles d'apprentissage et de test avec les fold 4

Si nous affichons chacun d'entre eux, nous verrons que quatre des points de données se trouvent dans X_train et que leurs valeurs cibles se trouvent dans y_train. Les deux points de données restants sont dans X_test et leurs valeurs cibles dans y_test.

```
print("X_train")
print(X_train)
print("y_train", y_train)
print("X_test")
print(X_test)
print("y_test", y_test)
```

# Exercice d'application 6

## Création des ensembles d'apprentissage et de test

Lequel des éléments suivants pourrait être la sortie de ce code en supposant que X a 3 points de données ?

```
kf = KFold(n_splits=3, shuffle=True)
splits = list(kf.split(X))
print(splits[0])
```

1. ([1], [0, 0])
2. ([0, 2], [1])
3. ([1, 2], [2])
4. ([0, 1], [2])
5. ([0], [1, 2])

# Construction du modèle 1

Nous pouvons maintenant utiliser les ensembles d'apprentissage et de test pour construire un modèle et faire une prédiction comme auparavant. Revenons à l'utilisation de l'ensemble des données (puisque 4 points de données ne sont pas suffisants pour construire un modèle décent).

Ci-dessous (la prochaine slide) l'intégralité du code permettant de construire et de noter le modèle sur le premier fractionnement d'une validation croisée à 5 fold. Notez que le code d'ajustement et de notation du modèle est exactement le même que lorsque nous avons utilisé la fonction train_test_split.

# Construction du modèle 2



**Attention !**

Jusqu'à présent, nous avons essentiellement fait une seule répartition train/test. Afin de faire une validation croisée k-fold, nous devons utiliser chacun des 4 autres fractionnements pour construire un modèle et le noter.

# Construction du modèle 3

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
import pandas as pd
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
kf = KFold(n_splits=5, shuffle=True)
splits = list(kf.split(X))
```

# Construction du modèle 4

```
train_indices, test_indices = splits[0]
X_train = X[train_indices]
X_test = X[test_indices]
y_train = y[train_indices]
y_test = y[test_indices]
model = LogisticRegression()
model.fit(X_train, y_train)
print(model.score(X_test, y_test))
```

# Exercice d'application 7

## Construction du modèle

Supposons une matrice de prédicteurs X. Complétez afin de définir correctement X_train et X_test pour le deuxième fold.

```
kf = KFold(n_splits=5, shuffle=True)
splits = list(kf.split(X))
a, b = splits[ ..... ]
X_train = X[ ..... ]
X_test = X[ ..... ]
```

# Bouclez sur tous les folds 1

Nous avons fait un pli (fold) à la fois, mais nous voulons vraiment boucler sur tous les plis pour obtenir toutes les valeurs. Nous allons mettre le code de la partie précédente dans une boucle for :

```
scores = []
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    model = LogisticRegression()
    model.fit(X_train, y_train)
    scores.append(model.score(X_test, y_test))
```

print(scores) $\Longrightarrow$ [0.8539, 0.7191, 0.8418, 0.7683, 0.8192]

# Bouclez sur tous les folds 2

Comme nous avons 5 plis, nous obtenons 5 valeurs de précision. Rappelons que pour obtenir une seule valeur finale, nous devons prendre la moyenne de ces valeurs.
print(np.mean(scores)) $\Longrightarrow$ 0.8004

Maintenant que nous avons calculé la précision, nous n'avons plus besoin des 5 modèles différents que nous avons construits. Pour une utilisation future, nous ne voulons qu'un seul modèle. Pour obtenir **le meilleur modèle possible, nous construisons un modèle sur l'ensemble des données**. **Si l'on nous demande la précision de ce modèle, nous utilisons la précision calculée par validation croisée (0.8004)**, même si nous n'avons pas réellement testé ce modèle particulier avec un ensemble de tests.

# Bouclez sur tous les folds 3

final_model = LogisticRegression() final_model.fit(X, y)

**Attention !**

Attendez-vous à obtenir des valeurs légèrement différentes à chaque fois que vous exécutez le code. La classe KFold divise les données de façon aléatoire à chaque fois, donc une division différente donnera des résultats différents, bien que vous devriez vous attendre à ce que la moyenne des 5 résultats soit généralement la même.

# Exercice d'application 8

## Création des ensembles d'apprentissage et de test

Supposons que nous avons une liste precision_scores de toutes les 5 valeurs de précision pour chacun des plis. Laquelle des valeurs suivantes est la valeur de précision unique finale ?

1. precision_scores[3]
2. np.median(precision_scores)
3. np.sum(precision_scores)/len(precision_scores)
4. precision_scores[0]
5. np.mean(precision_scores)

# Comparaison des modèles 1

Jusqu'à présent, nous avons utilisé nos techniques d'évaluation pour obtenir des scores pour un seul modèle. Ces techniques deviendront incroyablement utiles lorsque nous introduirons d'autres modèles et que nous voudrons déterminer lequel est le plus performant pour un problème spécifique.

Utilisons nos techniques pour comparer trois modèles :
- Un modèle de régression logistique utilisant toutes les prédicteurs de notre ensemble de données.
- Un modèle de régression logistique utilisant uniquement les colonnes Pclass, Age et Sexe.
- un modèle de régression logistique utilisant uniquement les colonnes Fare et Age.

# Comparaison des modèles 2

Nous ne nous attendons pas à ce que le deuxième ou le troisième modèle soit plus performant puisqu'il contient moins d'informations, mais nous pourrions déterminer que l'utilisation de ces deux ou trois colonnes seulement donne des résultats comparables à l'utilisation de toutes les colonnes.

**Attention !**

Les techniques d'évaluation sont essentielles pour décider entre plusieurs modèles.

# Exercice d'application 9

## Comparaison des modèles

Lequel des points suivants est un objectif de l'utilisation des mesures d'évaluation ?

1. Comparer deux modèles différents pour l'ensemble de données Titanic.
2. Comparer un modèle pour l'ensemble de données Titanic avec un modèle pour l'ensemble de données sur le cancer du sein.

# Construction de deux modèles avec Sklearn 1

Écrivons le code pour construire les deux modèles dans scikit-learn. Ensuite, nous utiliserons la validation croisée k-fold pour calculer l'accuracy, la précision, le rappel et le score F1 des deux modèles afin de pouvoir les comparer.

Tout d'abord, nous importons les modules nécessaires et préparons les données comme nous l'avons fait précédemment.

# Construction de deux modèles avec Sklearn 2

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import pandas as pd
import numpy as np
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
```

# Construction de deux modèles avec Sklearn 3

Maintenant nous pouvons construire l'objet KFold. Nous allons utiliser 5 splits, car c'est la norme. Notez que nous voulons créer un seul objet KFold que tous les modèles utiliseront. Il serait injuste que des modèles différents obtiennent une division différente des données.

```
kf = KFold(n_splits=5, shuffle=True)
```

Nous allons maintenant créer trois matrices de prédicteurs différentes X1, X2 et X3. Toutes auront la même cible y.

```
X1 = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
X2 = df[['Pclass', 'male', 'Age']].values
X3 = df[['Fare', 'Age']].values
y = df['Survived'].values
```

# Construction de deux modèles avec Sklearn 4

Comme nous allons le faire plusieurs fois, écrivons une fonction pour évaluer le modèle. Cette fonction utilise l'objet KFold pour calculer l'accuracy, la précision, le rappel et le score F1 pour un modèle de régression logistique avec la matrice de prédicteurs X et le tableau cible y donnés.
Nous appelons ensuite notre fonction trois fois pour chacune de nos trois matrices de prédicteurs et voyons les résultats.

**Attention !**

Attendez-vous à obtenir des résultats légèrement différents à chaque fois que vous exécutez le code. Les k-fold splits sont choisis aléatoirement, il y aura donc une petite variation en fonction du split dans lequel chaque point de données se retrouve.

# Construction de deux modèles avec Sklearn 5

```
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score
import pandas as pd
import numpy as np
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
kf = KFold(n_splits=5, shuffle=True)
X1 = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
X2 = df[['Pclass', 'male', 'Age']].values
X3 = df[['Fare', 'Age']].values
y = df['Survived'].values
```

# Construction de deux modèles avec Sklearn 6

```
def score_model(X, y, kf):
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model = LogisticRegression()
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy_scores.append(accuracy_score(y_test, y_pred))
        precision_scores.append(precision_score(y_test, y_pred))
        recall_scores.append(recall_score(y_test, y_pred))
        f1_scores.append(f1_score(y_test, y_pred))
```

# Construction de deux modèles avec Sklearn 7

```
    print("accuracy:", np.mean(accuracy_scores))
    print("precision:", np.mean(precision_scores))
    print("recall:", np.mean(recall_scores))
    print("f1 score:", np.mean(f1_scores))

print("Logistic Regression with all features")
print(score_model(X1, y, kf))
print("Logistic Regression with Pclass, Sex & Age features")
print(score_model(X2, y, kf))
print("Logistic Regression with Fare & Age features")
print(score_model(X3, y, kf))
```

# Exercice d'application 9

## Construction de deux modèles avec Sklearn

Complétez le code pour créer une quatrième matrice de prédicteurs qui ne comporte que les prédicteurs Pclass et Sex et utilise la fonction score_model pour afficher les scores. Supposons que nous ayons défini y comme étant les valeurs cibles et kf comme étant l'objet KFold.

X4 = df[['Pclass', 'male']]. .....
score_model(X4, y, kf)

# Choix du meilleur modèle 1

Regardons les résultats trouvés ci-dessus :
Logistic Regression with all features
accuracy: 0.802653462832476 3
precision: 0.772827797518754
recall: 0.699814740296489 2
f1 score: 0.731491553596476 8
Logistic Regression with Pclass, Sex & Age features
accuracy: 0.798228908779280 2
precision: 0.754400887474657 9
recall: 0.713568147620179 1
f1 score: 0.732391705598148 8
Logistic Regression with Fare & Age features
accuracy: 0.653907192280835 3
precision: 0.648793692958452 8
recall: 0.231219678007478 14
f1 score: 0.338153909443858 76

# Choix du meilleur modèle 2

Si l'on compare **les deux premiers modèles, ils ont des scores presque identiques. Le troisième modèle a des scores inférieurs pour les quatre paramètres**. **Les deux premiers modèles sont donc de bien meilleures options que le troisième**. Cela correspond **à l'intuition puisque le troisième modèle n'a pas accès au sexe du passager**. Nous nous attendons à ce que **les femmes aient plus de chances de survivre**, donc le fait d'avoir le sexe serait un prédicteur très précieux.

Puisque les deux premiers modèles ont des résultats équivalents, **il est logique de choisir le modèle le plus simple, celui qui utilise les caractéristiques Pclass, Sex** & **Age**.

# Choix du meilleur modèle 3

Maintenant que nous avons fait le choix du meilleur modèle, nous construisons un seul modèle final en utilisant toutes les données.
model = LogisticRegression() model.fit(X1, y)
Nous pouvons maintenant faire une prédiction avec notre modèle.
print(model.predict([[3, False, 25, 0, 1, 2]])) $\Longrightarrow$ [1]

**Attention !**

Nous n'avons essayé que trois combinaisons différentes de prédicteurs. Il est possible qu'une autre combinaison fonctionne également.

# Exercice d'application 10

## Choix du meilleur modèle

Le modèle 1 a une précision de 0.77 et un rappel de 0.68. Le modèle 2 a une précision de 0.70 et un rappel de 0.72. Quel modèle est le meilleur ?

1. ça depend
2. modèle 1
3. modèle 2

# Quiz 1

## 1) Matrice de confusion

Nous avons construit un modèle qui a la matrice de confusion suivante. Sur cette base, combien de points de données totaux y a-t-il, combien de points de données positifs y a-t-il et combien de points de données négatifs y a-t-il ?

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 4 | 4 |
| **Predicted negative** | 1 | 11 |

1. 15 total / 10 positif / 15 négatif
2. 15 total / 20 positif / 5 négatif
3. 20 total / 5 positif / 15 négatif

# Quiz 2

## 2) Calcul de l'accuracy

Soit la matrie de confusion suivante :

| | **Actual positive** | **Actual negative** |
|---|---|---|
| **Predicted positive** | 4 | 4 |
| **Predicted negative** | 1 | 11 |

Quelle est l'accuracy ?

1. 0.58
2. 0.8
3. 0.75

# Quiz 3

## 3) Qualité de l'accuracy

L'accuracy est une mauvaise mesure de la performance du modèle lorsque :

1. il y a un nombre égal de cas positifs et de cas négatifs
2. il y a plus de cas positifs que de cas négatifs
3. il y a plus de cas négatifs que de cas positifs

# Quiz 4

## 4) Évaluation d'un modèle

Lorsque nous évaluons un modèle, nous :

1. Sélectionnons les points de données les plus difficiles à intégrer dans notre ensemble de test afin de nous assurer que nous évaluons le modèle sur les points de données les plus difficiles.
2. Utilisons chaque point de données pour construire le modèle afin d'obtenir le meilleur modèle possible et voir ensuite comment il se comporte sur le même ensemble de données.
3. Divisons les données en un ensemble d'apprentissage et un ensemble de test afin de construire le modèle sur l'ensemble d'apprentissage et de tester le modèle sur des données non vues.

# Quiz 5

## 5) k-fold

Supposons que nous avons un ensemble de données de 150 points de données et que nous effectuons une validation croisée k-fold avec k=5.
Combien d'ensembles de formation avons-nous et de quelle taille ?

1. 15 séries et 150 points de données
2. 5 séries et 120 points de données
3. 10 séries et 80 points de données

# Quiz 6

## 6) k-fold

Complétez le code pour effectuer une validation croisée k-fold où k=5 et calculez l'accuracy. X est la matrice des prédicteurs et y est le tableau cible.

```
scores = [ ]
kf = KFold(n_splits=5, shuffle=True)
for train_index, test_index in kf.split(X):
X_train, X_test = X[train_index], X[test_index]
y_train, y_test = y[train_index], y[test_index]
model = LogisticRegression()
model. ..... (X_train, y_train)
scores. ..... (model.score(X_test, y_test))
print(np. ..... (scores))
```

# Machine Learning (ML)

## Chap 4 : Decision Trees

## Partie I

David TCHOUTA

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

April 11, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Introduction
3. Construction
4. Prédiction
5. Dec. Tree in Sklearn

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue les arbres de décision (construction et prédction).

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

- de construire des modèles basés sur les arbres de décision
- de réaliser les prédictions basées sur les arbres de décision
- de manipuler la classe DecisionTreeClassifier de Scikit-learn

# Un algorithme de Machine Learning paramètrique

Jusqu'à présent, nous avons traité la régression logistique. Dans la régression logistique, nous examinons les données sous forme graphique et nous traçons une frontière pour les séparer. Le modèle est défini par les coefficients qui définissent la frontière. Ces **coefficients** sont appelés **paramètres**. Comme le modèle est défini par ces **paramètres**, la **régression logistique** est un **algorithme d'apprentissage automatique paramétrique**.

# Un algorithme de Machine Learning non paramètrique

Dans ce module, nous allons présenter **les arbres de décision**, qui sont un exemple **d'algorithme d'apprentissage automatique non paramétrique**. Les arbres de décision ne seront pas définis par une liste de paramètres, comme nous le verrons dans les prochaines sections.

**Attention !**

Chaque algorithme d'apprentissage automatique est soit paramétrique, soit non paramétrique.

# Exercice d'application 1

## paramètrique Vs non paramètrique

La régression logistique est un algorithme d'apprentissage automatique paramétrique et l'arbre décisionnel est un algorithme d'apprentissage automatique non paramétrique.

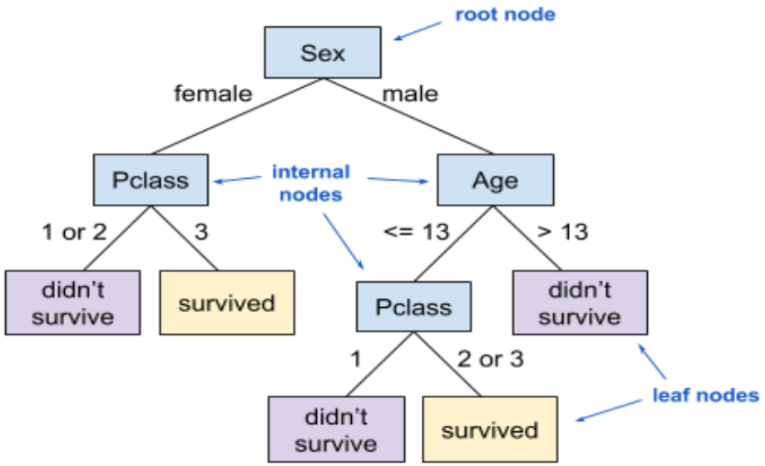
1. Vrai
2. Faux

# Terminologie 1

La raison pour laquelle de nombreuses personnes aiment les arbres de décision est qu'ils sont très faciles à interpréter. Il s'agit essentiellement d'une séquence de questions auxquelles vous répondez à propos d'un point de données jusqu'à ce que vous arriviez à une prédiction.

Ci-dessous un exemple d'arbre de décision pour le jeu de données Titanic. Nous verrons dans la prochaine section comment cet arbre est construit.

# Terminologie 2




Chacun des rectangles est appelé un **nœud (node)**. Les nœuds qui ont un prédicteur à partager sont appelés **noeuds internes (internal nodes)**. Le **tout premier nœud interne en haut** est appelé **nœud racine (root node)**. Les **nœuds finaux** où nous faisons les prédictions de survécu/non survécu sont appelés **nœuds finaux (leaf nodes)**. Les **nœuds internes** ont tous deux noeuds en dessous d'eux, que nous appelons les e**nfants du noeud (node's children)**.

# Terminologie 3

**Attention !**

Les termes désignant les arbres (racine, feuille) proviennent d'un arbre réel, bien qu'il soit à l'envers puisque nous dessinons généralement la racine au sommet. Nous utilisons également des termes qui considèrent l'arbre comme un arbre familial (nœud enfant et nœud parent).

# Exercice d'application 2

## paramètrique Vs non paramètrique

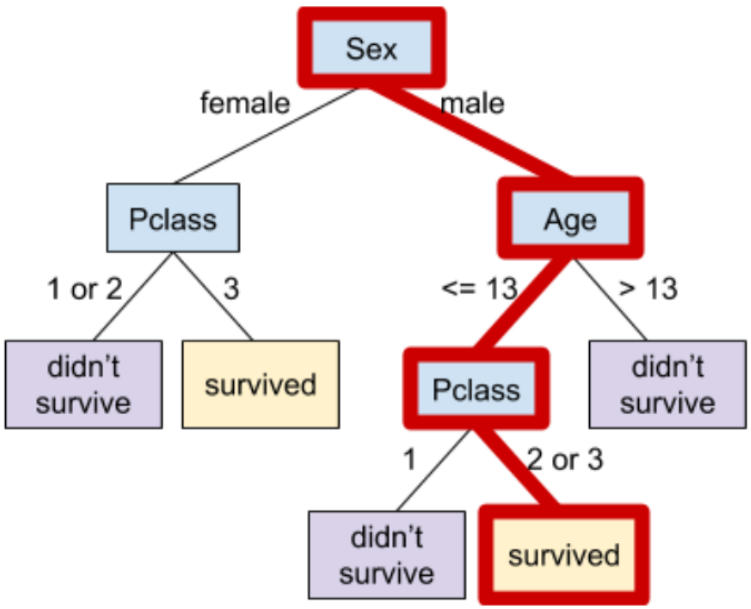
D'après l'arbre de la page précédente, lesquelles des affirmations suivantes sont correctes ?

1. il y a un nœud racine (root node)
2. il y a 4 nœuds finaux (leaf nodes)
3. la racine a 3 nœuds enfants (children nodes)
4. il y a 4 nœuds internes (internal nodes)

# Interprétation d'un arbre de décision 1

Pour interpréter cet arbre de décision, prenons un exemple. Supposons que nous voulons connaître la prédiction pour un passager masculin de 10 ans dans la classe P 2. Au premier nœud, puisque le sexe du passager est masculin, nous allons vers l'enfant de droite. Ensuite, puisque son âge est de 10 ans, ce qui est $<=13$, nous allons vers l'enfant de gauche, et au troisième nœud, nous allons vers l'enfant de droite puisque la classe P est 2. Dans le diagramme suivant, nous mettons en évidence le chemin pour ce passager.

# Interprétation d'un arbre de décision 2



# Interprétation d'un arbre de décision 3

Notez qu'il n'y a aucune règle concernant l'utilisation de chaque prédicteur, l'ordre d'utilisation des prédicteurs ou, pour une valeur continue (comme l'âge), l'endroit où l'on effectue la division. Dans un arbre de décision, il est courant que chaque division ne comporte que deux options.

**Attention !**

Les arbres de décision sont souvent privilégiés si vous avez un public non technique, car ils peuvent facilement interpréter le modèle.

# Exercice d'application 3

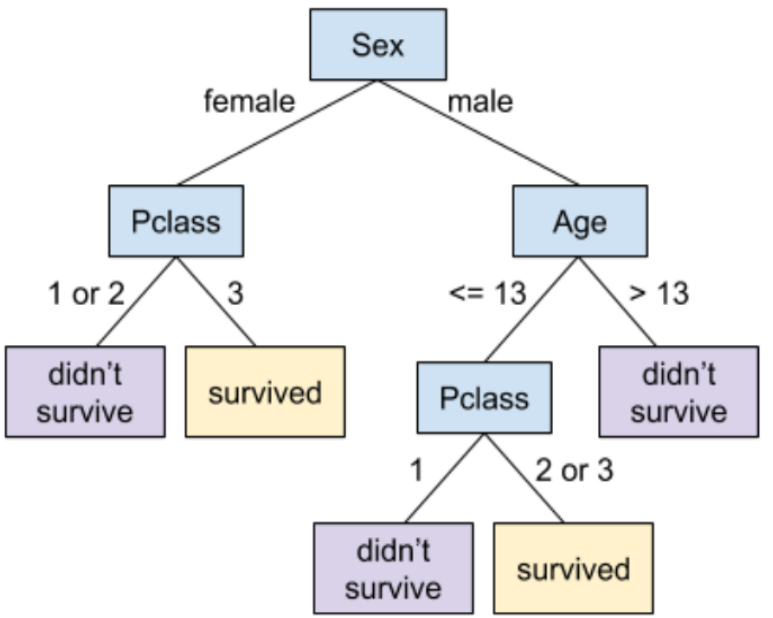
## Interprétation d'un arbre de décision

En utilisant l'arbre de décision présenté à la page précédente, quelle est la prédiction pour une passagère âgée de 20 ans et appartenant à la classe P 2 ?

1. a survécu
2. n'a pas survécu

# Construction d'un arbre de décision 1

Lors de la construction de l'arbre de décision, nous ne choisissons pas au hasard le prédicteur à répartir en premier. Nous voulons commencer par choisir le prédicteur ayant le plus grand pouvoir prédictif. Regardons à nouveau notre arbre de décision :



# Construction d'un arbre de décision 2

Intuitivement, pour notre ensemble de données sur le Titanic, étant donné que **les femmes étaient souvent prioritaires sur les canots de sauvetage**, nous nous attendons à ce que **le sexe soit un prédicteur très important**. Il est donc logique d'utiliser cette variable en premier. De chaque côté de l'arbre de décision, nous déterminons indépendamment la variable sur laquelle nous devons nous baser. Dans l'exemple ci-dessus, la deuxième répartition pour les femmes porte sur la classe sociale. Pour les hommes, la deuxième répartition se fait sur l'âge. Nous notons également que dans certains cas, nous effectuons trois divisions et dans d'autres seulement deux.

# Construction d'un arbre de décision 3



**Attention !**

Pour un ensemble de données, il existe un grand nombre d'arbres décisionnels différents qui peuvent être créés en fonction de l'ordre dans lequel vous utilisez les variables exogènes. Dans les prochaines sections, nous verrons comment choisir mathématiquement le meilleur arbre de décision.

# Exercice d'application 3

## Construction d'un arbre de décision

Sélectionnez tout ce qui s'applique à la version des arbres de décision que nous utilisons.

1. chaque internal node a exactement 2 children
2. chaque leaf node a une prédiction pour la variable cible
3. chaque chemin jusqu'à un leaf node est de même length
4. chaque prédicteur doit être utilisé
5. un prédicteur peut-être utilisé seulement une fois

# Qu'est-ce qu'un bon split ? 1

Afin de déterminer la variable sur laquelle nous devons diviser les données en premier, nous devons **évaluer chaque division possible** afin de choisir la **division ayant le score le plus élevé**. Notre objectif est de diviser parfaitement les données. Si, par exemple, toutes les femmes ont survécu à l'accident et que tous les hommes n'ont pas survécu, la répartition sur le sexe serait parfaite. Cela se produira rarement dans un ensemble de données réel, mais nous voulons nous en rapprocher le plus possible.

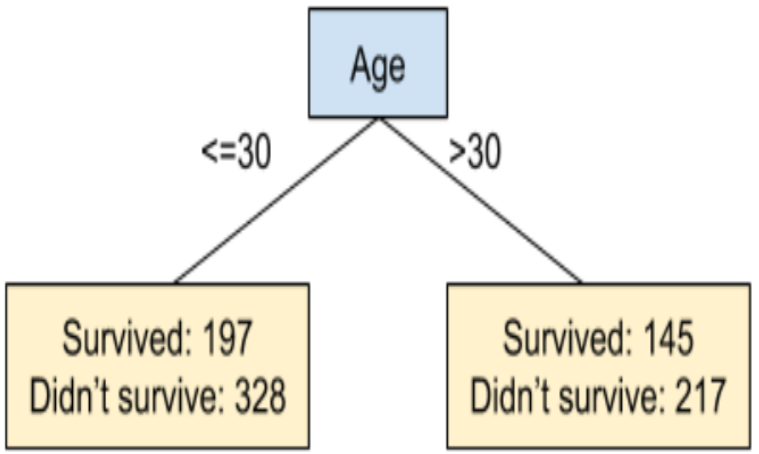
# Qu'est-ce qu'un bon split ? 2

Le terme mathématique que nous allons mesurer est appelé **gain d'information (information gain)**. Il s'agira d'une valeur de 0 à 1, où **0 est le gain d'information d'un fractionnement inutile et 1 le gain d'information d'un fractionnement parfait**. Dans les deux prochaines parties, nous définirons **l'impureté de Gini (gini impurity)** et **l'entropie (entropy)** que nous utiliserons pour **définir le gain d'information**. Tout d'abord, nous allons discuter de l'intuition de ce qui constitue un bon split.

# Qu'est-ce qu'un bon split ? 3

Considérons quelques divisions possibles pour l'ensemble de données du Titanic. Nous verrons comment les données sont réparties et pourquoi l'une est meilleure que l'autre.
Tout d'abord, essayons de diviser les données en fonction de l'âge. Puisque l'âge est une variable numérique, nous devons choisir un seuil pour le fractionnement. Supposons que nous séparons sur Age<=30 et Age>30. Voyons combien de passagers nous avons de chaque côté, et combien d'entre eux ont survécu et combien n'ont pas survécu.

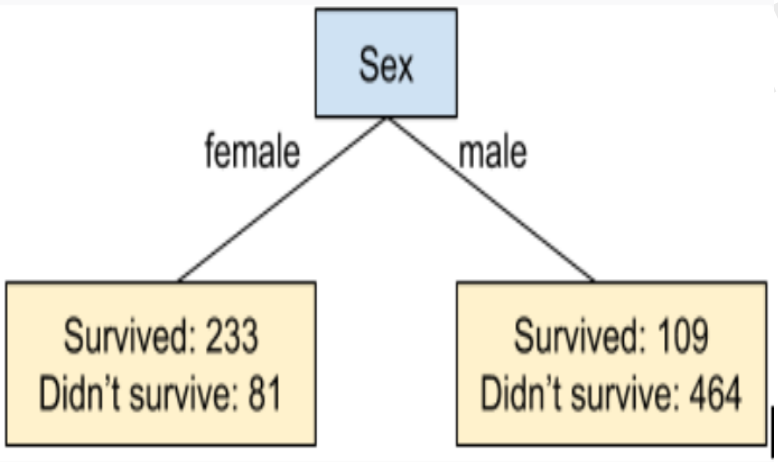
# Qu'est-ce qu'un bon split ? 4




Des deux côtés, **nous avons environ 40% des passagers qui survivent. Nous n'avons donc rien gagné à diviser les données de cette façon**.

# Qu'est-ce qu'un bon split ? 5

Maintenant, essayons de diviser sur le sexe :




On peut voir que du côté des femmes, **la grande majorité a survécu**. Du côté des hommes, la **grande majorité n'a pas survécu**. **C'est une bonne division**.

# Qu'est-ce qu'un bon split ? 6

Ce que nous recherchons, **c'est l'homogénéité (ou la pureté) (homogeneity or purity) de chaque côté**. **Idéalement, nous devrions envoyer tous les passagers qui ont survécu d'un côté et ceux qui n'ont pas survécu de l'autre côté**. Nous allons examiner deux mesures mathématiques différentes de la pureté. Nous utiliserons les valeurs de pureté pour calculer le gain d'information.

**Attention !**

Si l'on **choisit bien la variable** sur laquelle se fait la division, **chaque côté de la division est pur**. **Un ensemble est pur si tous les points de données appartiennent à la même classe (soit ils ont survécu, soit ils n'ont pas survécu)**.

# Exercice d'application 4

## Qu'est-ce qu'un bon split ?

Supposons que nous commençons avec 100 points de données, 50 qui ont survécu et 50 qui n'ont pas survécu. Quelle est la meilleure répartition des données ?

1. Côté gauche : 25 ont survécu, 25 n'ont pas survécu Côté droit : 25 ont survécu, 25 n'ont pas survécu
2. Côté gauche : 5 ont survécu, 5 n'ont pas survécu Côté droit 45 ont survécu, 45 n'ont pas survécu
3. Côté gauche : 10 ont survécu, 40 n'ont pas survécu Côté droit : 40 ont survécu, 10 n'ont pas survécu

# Gini Impurity 1

**Definition**

L'impureté de Gini (Gini impurity) est une mesure de la pureté d'un ensemble. Nous verrons plus tard comment nous pouvons utiliser l'impureté de Gini pour calculer le gain d'information.

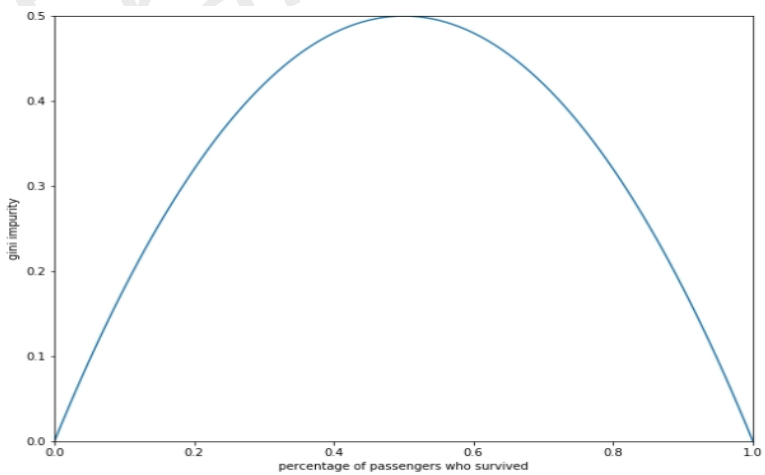
Nous calculons l'impureté de Gini sur un sous-ensemble de nos données en fonction du nombre de points de données de l'ensemble qui sont des passagers ayant survécu et du nombre de passagers n'ayant pas survécu. **Il s'agira d'une valeur comprise entre 0 et 0.5, où 0.5 correspond à une impureté totale (50% ont survécu et 50% n'ont pas survécu) et 0 à une pureté totale (100% dans la même classe)**.

# Gini Impurity 2

La **formule de gini** est donnée ci-dessous où p est le pourcentage de passagers qui ont survécu, et (1-p) est le pourcentage de passagers qui n'ont pas survécu.

$$\mathbf{gini = 2 * p * (1\text{-}p)}$$

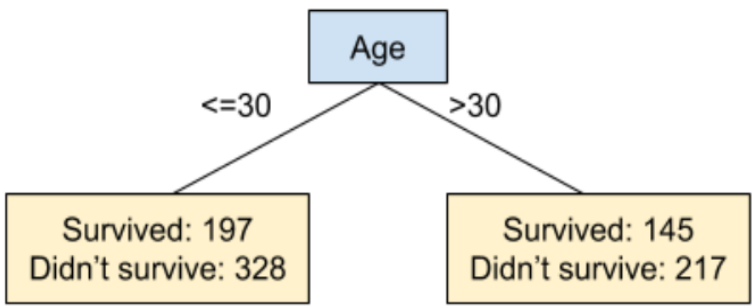
Ci-dessous le graphe de l'impureté de Gini :



# Gini Impurity 3

Nous pouvons constater que la valeur maximale est de 0.5 lorsque exactement 50% des passagers de l'ensemble ont survécu. Si tous les passagers ont survécu ou n'ont pas survécu (le pourcentage est de 0 ou 1), alors la valeur est 0.

Calculons l'impureté de Gini pour nos exemples de la partie précédente. Tout d'abord, nous avions une répartition sur Age<=30 et Age>30. Calculons les impuretés de Gini des deux ensembles que nous avons créés.



# Gini Impurity 4

À gauche, pour les passagers ayant un âge<=30, calculons d'abord le pourcentage de passagers ayant survécu :

Pourcentage de passagers ayant survécu = 197/(197+328) = 0.3752
Pourcentage de passagers qui n'ont pas survécu = 1 - 0.375 = 0.6248

Utilisons maintenant ce chiffre pour calculer l'indice de Gini :
2 * 0.3752 * 0.6248 = 0.4689
2 * 145/(145+217) * 217/(145+217) = 0.4802
Nous pouvons voir que **cette valeur est proche de 0,5, la valeur maximale pour l'impureté gini. Cela signifie que l'ensemble est impur**.

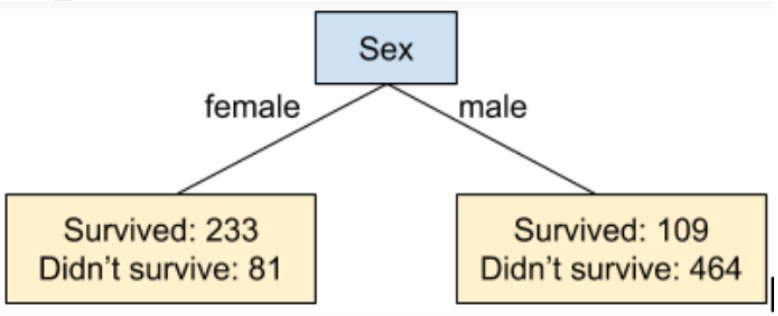
# Gini Impurity 5

Calculons maintenant l'impureté de gini pour le côté droit, les passagers avec un âge¿30 :

**2 * 145/(145+217) * 217/(145+217) = 0.4802**

Cette **valeur est également proche de 0,5, nous avons donc à nouveau un ensemble impur**.

Regardons maintenant les valeurs de Gini pour **l'autre répartition** que nous avons essayée, la **répartition sur le sexe**.



# Gini Impurity 6

Du côté gauche, pour les passagers féminins, nous calculons la valeur suivante pour l'impureté de Gini.
2 * 233/(233+81) * 81/(233+81) = 0.3828
Du côté droit, pour les passagers masculins, nous obtenons la valeur suivante.
2 * 109/(109+464) * 464/(109+464) = 0.3081
**Ces deux valeurs sont plus petites que les valeurs de gini pour le fractionnement sur l'âge, nous déterminons donc que le fractionnement sur la variable sexe est un meilleur choix**.

**Attention !**

Actuellement, nous avons deux valeurs pour chaque fractionnement potentiel. Le gain d'information sera un moyen de les combiner en une seule valeur.

# Exercice d'application 5

## Gini Impurity

Supposons que l'on compare deux splits. Split A : Côté gauche : 10 ont survécu, 40 n'ont pas survécu. Côté droit : 40 ont survécu, 10 n'ont pas survécu. Split B : Côté gauche : 5 ont survécu, 5 n'ont pas survécu. Côté droit : 45 ont survécu, 45 n'ont pas survécu. Lesquels des affirmations suivantes sont vraies :

1. pour le split A, l'impureté de Gini du côté gauche est égale à l'impureté de Gini du côté droit
2. l'impureté de Gini du côté gauche du split A est supérieure à l'impureté de Gini du côté gauche du split B
3. pour le split B, l'impureté de Gini du côté gauche est égale à l'impureté de Gini du côté droit
4. l'impureté de Gini du côté droit du split A est supérieure à l'impureté de Gini du côté droit du split B

# Entropy 1

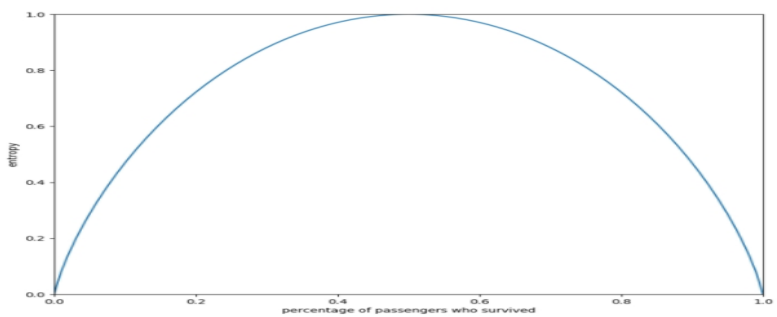
## Definition

**L'entropie** est une autre mesure de la pureté. Il s'agira d'une valeur comprise entre 0 et 1 où **1 est complètement impur (50% ont survécu et 50% n'ont pas survécu) et 0 est complètement pur (100% de la même classe)**.

La formule de **l'entropie** vient de la **physique**. **p** est à nouveau le **pourcentage** de **passagers qui ont survécu**.

$$\textbf{entropy} = -\ [p log_2 p + (1-p) log_2 (1-p)]$$

**Ne pas oublier : import math pour manipuler math.log2()**

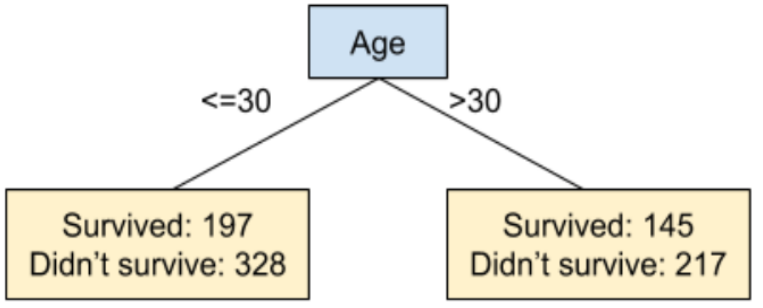


# Entropy 2

Vous pouvez voir qu'elle a une forme similaire à la fonction de Gini. Comme pour l'impureté de Gini, la valeur maximale correspond au cas où 50% des passagers de notre ensemble ont survécu, et la valeur minimale au cas où tous les passagers ou aucun n'ont survécu. La forme des graphiques est un peu différente. Vous pouvez voir que le graphique de l'entropie est un peu plus gros.

Calculons maintenant les valeurs d'entropie pour les deux mêmes splits potentielles.

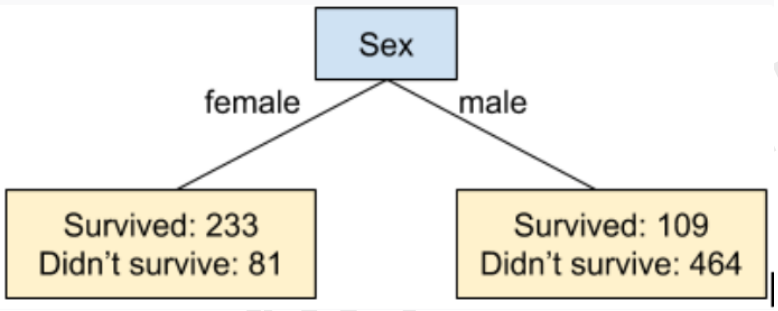
# Entropy 3




Sur la gauche (Age<=30):
p = 197/(197+328) = 0.3752
Entropy = -(0.375 * log(0.375) + (1-0.375) * log(1-0.375)) = 0.9546
Sur la droite (Age>30):
p = 145/(145+217) = 0.4006
Entropy = -(0.401 * log(0.401) + (1-0.401) * log(1-0.401)) = 0.9713
Ces valeurs sont **toutes deux proches de 1, ce qui signifie que les ensembles sont impurs**.

# Entropy 4

Maintenant, faisons le même calcul pour le split de la variable sexe :




Sur la gauche (female):
p = 233/(233+81) = 0.7420
Entropy = -(p * math.log2(p) + (1-p) * math.log2(1-p)) = 0.8237
(import math)
Sur la droite (male):
p = 109/(109+464) = 0.1902
Entropy = -(p * math.log2(p) + (1-p) * math.log2(1-p)) = 0.7019
(import math)

# Entropy 5

Vous pouvez voir que **ces valeurs d'entropie sont plus petites que les valeurs d'entropie trouvé en fonction du split sur l'âge, il s'agit donc d'une meilleure split**.

**Attention !**

Il n'est pas évident de savoir si gini ou l'entropie est un meilleur choix. Souvent, cela ne fait pas de différence, mais vous pouvez toujours effectuer une validation croisée pour comparer un arbre décisionnel avec entropie et un arbre décisionnel avec gini pour voir lequel est le plus performant.

# Exercice d'application 6

## Entropy

Quelles sont les entropies correctes du côté gauche et du côté droit du split suivant : Côté gauche : 20 ont survécu, 0 n'a pas survécu. Côté droit : 40 ont survécu, 40 n'ont pas survécu.

1. entropie du côté gauche $= 1$ / entropie du côté droit $= 0$
2. entropie du côté gauche $= 1$ / entropie du côté droit $= 1$
3. entropie du côté gauche $= 0$ / entropie du côté droit $= 1$
4. entropie du côté gauche $= 0$ / entropie du côté droit $= 0$

# Information Gain 1

Maintenant que nous avons un moyen de calculer une valeur numérique pour l'impureté, nous pouvons définir le **gain d'information**.

$$\textit{Information Gain} = H(S) - \frac{|A|}{|S|}H(A) - \frac{|B|}{|S|}H(B)$$

**H est notre mesure d'impureté (soit l'impureté de Gini, soit l'entropie)**. S **est l'ensemble de données original** et **A et B sont les deux ensembles en lesquels nous divisons l'ensemble de données S**. Dans le premier exemple ci-dessus, **A représente les passagers dont l'âge est inférieur ou égal à 30 ans** et **B les passagers dont l'âge est supérieur à 30 ans**. Dans le deuxième exemple, **A représente les passagers de sexe féminin** et **B les passagers de sexe masculin**. **|A| signifie la taille de A**.

# Information Gain 2

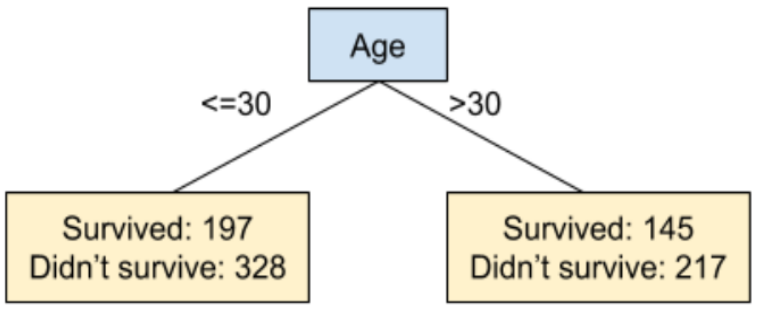
Calculons cette valeur pour nos deux exemples. Utilisons l'impureté Gini comme mesure de l'impureté.

Nous avons déjà calculé la plupart des valeurs d'impureté de Gini, mais nous devons calculer l'impureté de Gini de l'ensemble. Il y a 342 passagers qui ont survécu et 545 passagers qui n'ont pas survécu, sur un total de 887 passagers, donc l'impureté de Gini est la suivante :

**Gini = 2 * 342/887 * 545/887 = 0.4738**

# Information Gain 3

Ci-dessous le premier split potentiel :



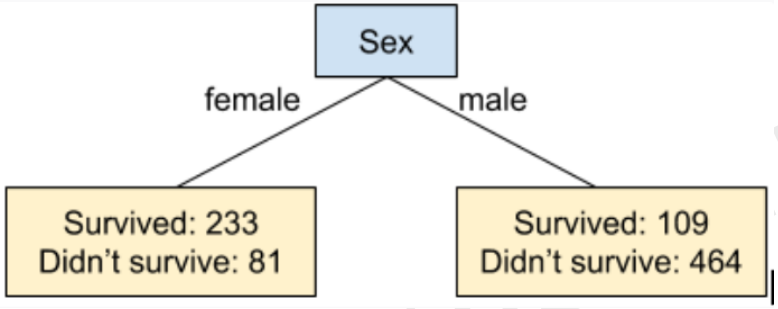

Notez que nous avons 197+328=525 passagers à gauche (âge¡=30) et 145+217=362 passagers à droite (âge¿30). Ainsi, en intégrant les valeurs d'impureté de Gini que nous avons calculées précédemment, nous obtenons le gain d'information suivant :

Information gain = 0.4738 - 525/887 * 0.4689 - 362/887 * 0.4802 = 0.0003

Cette **valeur est très faible**, ce qui signifie que **nous ne gagnons que très peu à cette scission**.

# Information Gain 4

Calculons le gain d'information pour le split par sexe :




Nous avons 233+81=314 passagers à gauche (femmes) et 109+464=573 passagers à droite (hommes). Le gain d'information est :

Information gain = 0.4738 - 314/887 * 0.3828 - 573/887 * 0.3081 = 0.1393

Nous pouvons constater que **le gain d'information est bien meilleur pour ce split**. Par conséquent, **le split sur le sexe est un bien meilleur choix pour construire notre arbre de décision que le split sur l'âge avec le seuil 30**.

# Information Gain 5

**Attention !**

Le travail que nous avons effectué ne visait qu'à comparer deux répartitions possibles. Nous devrons effectuer les mêmes calculs pour chaque répartition possible afin de trouver la meilleure. Heureusement, nous n'avons pas à faire ces calculs à la main.

# Exercice d'application 7

## Information Gain

L'ensemble de données initial comprend 50 survivants et 50 non-survivants et a une impureté de gini de 0.5.
La division place 25 points de données d'un côté avec une impureté de Gini de 0.4 et 75 points de données de l'autre côté avec une impureté de Gini de 0.2.
Sur la base de ces informations, quelle est l'équation correcte pour calculer le gain d'information ?

1. 0.2 - 0.25 * 0.4 - 0.75 * 0.5
2. 0.5 - 0.25 * 0.4 - 0.75 * 0.2
3. 0.5 - 0.25 * 0.4 + 0.75 - 0.2

# Construction d'un arbre de décision 1

Nous avons jeté les bases nécessaires à la construction de l'arbre de décision. Ci-dessous le processus que nous suivons :
Pour déterminer comment effectuer le premier split, nous passons en revue tous les splits possibles et calculons le gain d'information si nous utilisons ce fractionnement. Pour les prédicteurs numériques tels que l'Age, la PClass et le Fare, nous essayons tous les seuils possibles. Le split sur le seuil d'âge de 50 signifie que les points de données avec un âge $<= 50$ sont un groupe et ceux avec un âge $> 50$ sont l'autre. Ainsi, puisqu'il y a 89 âges différents dans notre ensemble de données, nous avons 88 répartitions différentes à essayer pour la variable âge.

# Construction d'un arbre de décision 2

Nous essayerons tous les potentiels splits suivants :

1. Sex (male | female)
2. Pclass (1 or 2 | 3)
3. Pclass (1 | 2 or 3)
4. Age (0 | >0)
5. Age (<= 1| > 1)
6. Age (<= 2| > 2)
7. etc. . . .

# Construction d'un arbre de décision 3

Il existe 1 fractionnement potentiel pour le sexe, 2 fractionnements potentiels pour la classe P et 88 fractionnements potentiels pour l'âge. Il existe 248 valeurs différentes pour Fare, ce qui signifie qu'il y a 247 répartitions possibles pour cette variable. Si nous ne prenons en compte que ces quatre prédicteurs, nous avons 338 splits potentiels à réaliser.

Pour chacun de ces fractionnements, **nous calculons le gain d'information et nous choisissons le fractionnement ayant la valeur la plus élevée**.

# Construction d'un arbre de décision 4

Maintenant, on fait la même chose pour le niveau suivant. Supposons que nous avons fait la première répartition sur le sexe. Maintenant, pour tous les passagers de sexe féminin, nous essayons tous les fractionnements possibles pour chacune des variables et nous choisissons celle qui apporte le plus d'informations. Nous pouvons effectuer deux fois le même fractionnement sur la même variable si celle-ci a plusieurs seuils possibles. Le sexe ne peut être divisé qu'une seule fois, mais les variables Fare et Age peuvent être divisées plusieurs fois.

# Construction d'un arbre de décision 5

Indépendamment, nous effectuons un calcul similaire pour les passagers masculins et choisissons la répartition qui présente le gain d'information le plus élevé. Ainsi, nous pouvons avoir une deuxième répartition différente pour les passagers masculins et les passagers féminins.

Nous continuons ce processus jusqu'à ce que nous n'ayons plus de variables à splitter.

# Construction d'un arbre de décision 6



**Attention !**

Cela fait beaucoup de choses à essayer, mais il suffit d'y affecter de la puissance de calcul des ordinateurs. Les arbres de décision sont un peu lents à construire, mais une fois l'arbre construit, il est très rapide de faire une prédiction.

# Exercice d'application 8

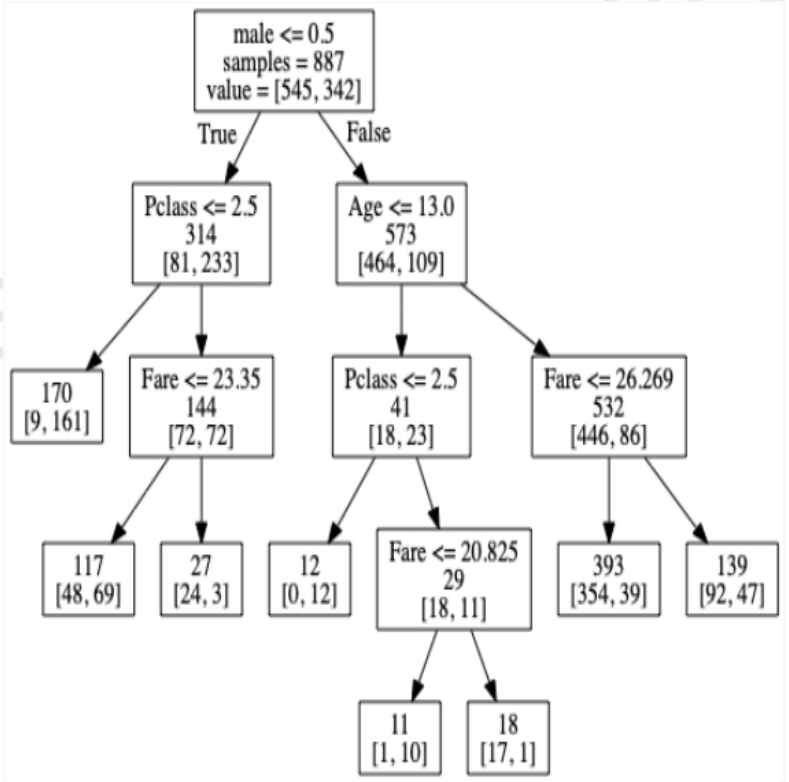
## Nombre de splits à réaliser

Supposons que nous n'utilisons que trois des variables de notre jeu de données : Sexe, PClass, Age. Il y a deux valeurs pour le Sex, trois pour la PClass (1, 2 ou 3), et cinq pour l'Age (5, 10, 20, 25, 30). Combien y a-t-il de splits possibles ?

1. ..... split(s) pour la variable Sex
2. ..... split(s) pour la variable PClass
3. ..... split(s) pour la variable Age

# Schéma de l'arbre de décision 1

Examinons un exemple d'arbre de décision pour l'ensemble de données Titanic. À l'intérieur de chaque nœud interne, nous avons la variable et le seuil de séparation, le nombre d'échantillons et la distribution des mêmes (de non survivants Vs de survivants) :



# Schéma de l'arbre de décision 2

Pour interpréter cet arbre de décisosn, commençons par reagrder le nœud racine :
male <= 0.5
samples = 887
value = [545, 342]

Cela signifie que le premier split se fera sur la colonne des hommes. Si la valeur est <=0.5 (ce qui signifie que le passager est de sexe féminin), nous allons vers le nœud enfant de gauche et si la valeur est >0.5 (ce qui signifie que le passager est de sexe masculin), nous allons vers le nœud enfant de droite.

Il y a 887 points de données pour commencer et 545 sont des cas négatifs (n'ont pas survécu) et 342 sont positifs (ont survécu).

# Schéma de l'arbre de décision 3

Si l'on regarde les deux enfants du nœud racine, on peut voir combien de points de données ont été envoyés dans chaque sens en fonction de la répartition par sexe. Il y a 314 passagers féminins dans notre jeu de données et 573 passagers masculins.

Vous pouvez voir que la deuxième répartition pour les passagers féminins est différente de la deuxième répartition pour les passagers masculins.

**Attention !**

Ce diagramme a été créé avec Graphviz, que nous apprendrons à utiliser dans une prochaine leçon.

# Exercice d'application 9

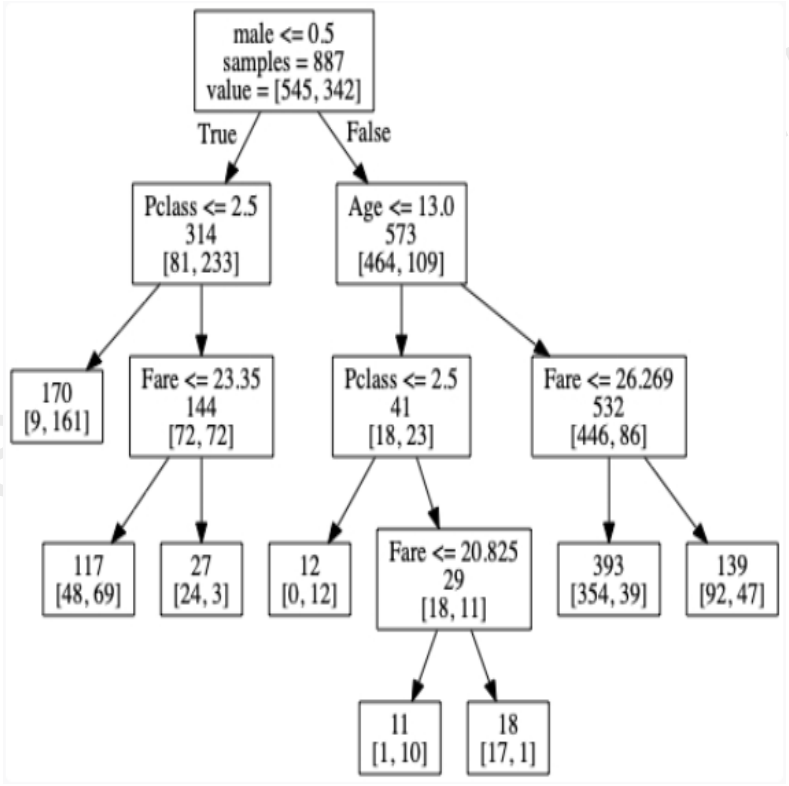
## Interprétation

Regardons l'enfant gauche du nœud racine (confer l'arbre de décision pour l'ensemble de données Titanic). Le texte du nœud indique Pclass $<= 2.5$ et le nœud est divisé sur la variable Pclass avec un seuil de 2.5. Combien de points de données vont à la division de gauche ?

1. 144
2. 532
3. 170

# Comment réaliser une prédiction ? 1

Soit l'arbre de décision pour l'ensemble de données Titanic :



# Comment réaliser une prédiction ? 2

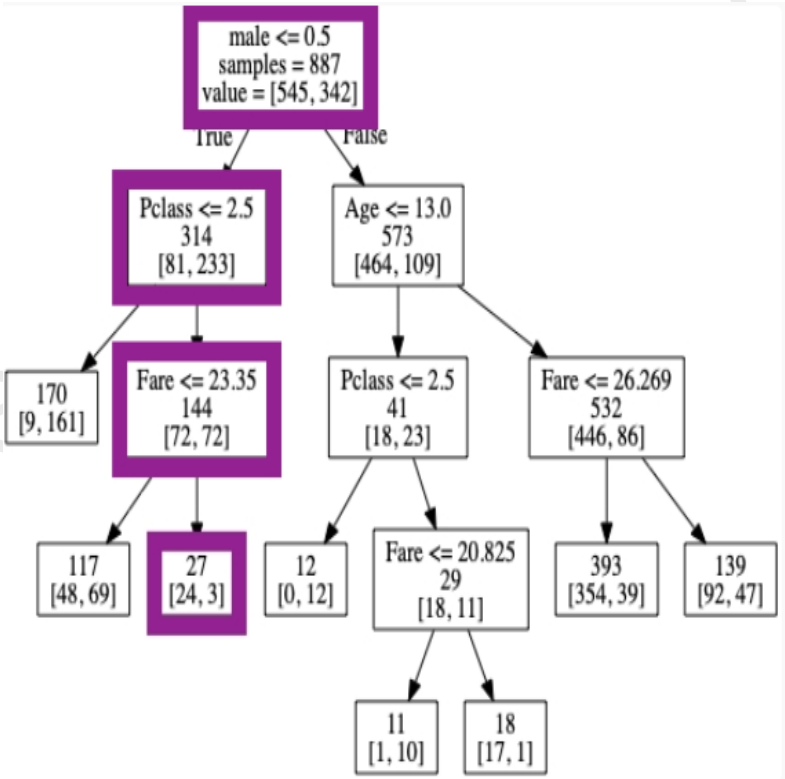
Supposons que nous voulions utiliser cet arbre de décision pour faire une prédiction pour un passager ayant les valeurs suivantes :
Sex: female
Pclass: 3
Fare: 25
Age: 30

# Comment réaliser une prédiction ? 3

Nous posons la question à chaque nœud et **allons vers l'enfant de gauche si la réponse est ou et vers la droite si la réponse est non**.
Nous commençons par **le nœud racine** :
La valeur de la variable masculine est-elle $<= 0.5$ ? (Cette question pourrait également être posée sous la forme "Le passager est-il une femme ?"). Puisque la réponse est oui, on passe à l'enfant de gauche.
La PClass est-elle $<= 2.5$ ? Puisque la réponse est non, nous passons à l'enfant de droite.
Est-ce que le tarif est $<= 23.35$ ? Puisque la réponse est non, nous passons à l'enfant de droite.

# Comment réaliser une prédiction ? 4

Maintenant nous sommes à un nœud terminal. Ci-dessous le chemin que nous avons pris en surbrillance :



# Comment réaliser une prédiction ? 5

Le nœud terminal auquel nous aboutissons a le texte suivant :
27
[24, 3]

Cela signifie qu'il y a 27 points de données dans notre ensemble de données qui atterrissent également à ce nœud terminal. 24 d'entre eux n'ont pas survécu et 3 d'entre eux ont survécu. **Cela signifie que notre prédiction est que le passager n'a pas survécu**.

**Attention !**

Comme il n'y a pas de règles quant à la façon dont l'arbre est construit, l'arbre de décision pose des questions complètement différentes à un passager féminin et à un passager masculin.

# Exercice d'application 10

## Interprétation

Supposons que nous avons un passager avec ces valeurs :
Sexe : homme
PClass : 3
Fare : 35
Age : 20 ans
Mettez dans l'ordre les variables sur lesquelles nous nous sommes séparés (l'arbre de décision pour l'ensemble de données Titanic).

1. Age
2. Fare
3. Sex

# La classe DecisionTreeClassifier() 1

Tout comme pour la régression logistique, Scikit-learn dispose d'une classe d'arbre décisionnel. Le code pour construire un modèle d'arbre de décision est très similaire à celui d'un modèle de régression logistique. Scikit-learn a fait cela intentionnellement pour qu'il soit facile de construire et de comparer différents modèles pour le même ensemble de données.

Ci-dessous l'instruction d'importation :
from sklearn.tree import DecisionTreeClassifier

# La classe DecisionTreeClassifier() 2

Nous pouvons maintenant appliquer les mêmes méthodes que celles utilisées avec la classe LogisticRegression : **fit (pour entraîner le modèle)**, **score (pour calculer le score de précision)** et **predict (pour faire des prédictions)**.
Nous commençons par créer un objet DecisionTreeClassifier :
model = DecisionTreeClassifier()

# La classe DecisionTreeClassifier() 3

Nous effectuons une répartition train/test en utilisant un random_state afin que chaque fois que nous exécutons le code, nous obtenions la même répartition :
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=3)

Ensuite, nous utilisons la méthode fit() pour entraîner le modèle :
model.fit(X_train, y_train)

# La classe DecisionTreeClassifier() 4

Nous pouvons utiliser la méthode de prédiction pour voir ce que le modèle prédit. Ici, nous pouvons voir la prédiction pour un passager masculin de la Pclass 3, qui a 22 ans, 1 frère/conjoint (sibling/spouse) à bord, 0 parent/enfant (parents/enfants) à bord, et a payé un tarif (Fare) de 7.25 :

print(model.predict([[3, True, 22, 1, 0, 7.25]])) $\Longrightarrow$ [0]

Nous voyons que le modèle prédit que le passager n'a pas survécu. C'est la même prédiction que le modèle de régression logistique a donné.

# La classe DecisionTreeClassifier() 5

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=3)
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
print(model.predict([[3, True, 22, 1, 0, 7.25]]))
```

# Exercice d'application 11

## La classe DecisionTreeClassifier()

Complétez le programme ci-dessous pour faire une prédiction pour une passagère de la PClass 2, âgée de 10 ans, qui n'a pas de frères et sœurs/conjoints/parents/enfants à bord, et qui a payé un tarif (Fare) de 9.

```
print(model.predict([[ ..... , ..... , 10, 0, 0, ..... ]]))
```

# Calcul des métriques 1

Nous pouvons utiliser les méthodes de score et de prédiction pour obtenir les scores d'accuracy, de précision et de rappel :

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.metrics import precision_score, recall_score
import numpy as np
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

# Calcul des métriques 2

```
kf = KFold(n_splits=5, shuffle=True, random_state=3)
dt_accuracy_scores = [ ]
dt_precision_scores = [ ]
dt_recall_scores = [ ]
lr_accuracy_scores = [ ]
lr_precision_scores = [ ]
lr_recall_scores = [ ]
```

# Calcul des métriques 3

```
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    dt = DecisionTreeClassifier()
    dt.fit(X_train, y_train)
    dt_accuracy_scores.append(dt.score(X_test, y_test))
    dt_y_pred = dt.predict(X_test)
    dt_precision_scores.append(precision_score(y_test, dt_y_pred))
    dt_recall_scores.append(recall_score(y_test, dt_y_pred))
    lr = LogisticRegression()
    lr.fit(X_train, y_train)
    lr_accuracy_scores.append(lr.score(X_test, y_test))
    lr_y_pred = lr.predict(X_test)
    lr_precision_scores.append(precision_score(y_test, lr_y_pred))
    lr_recall_scores.append(recall_score(y_test, lr_y_pred))
```

# Calcul des métriques 4

```
print("Decision Tree")
print(" accuracy:", np.mean(dt_accuracy_scores))⟹ 0.7564
print(" precision:", np.mean(dt_precision_scores)) ⟹ 0.6935
print(" recall:", np.mean(dt_recall_scores)) ⟹ 0.6656
print("Logistic Regression")
print(" accuracy:", np.mean(lr_accuracy_scores)) ⟹ 0.7948
print(" precision:", np.mean(lr_precision_scores)) ⟹ 0.7551
print(" recall:", np.mean(lr_recall_scores)) ⟹ 0.6979
```

# Calcul des métriques 5

Nous pouvons utiliser la validation croisée k-fold pour obtenir une mesure précise des métriques et comparer les valeurs avec un modèle de régression logistique. Nous utilisons un random_state lors de la création de l'objet KFold afin d'obtenir les mêmes résultats à chaque fois.

Vous pouvez voir que l'accuracy et la précision du modèle de régression logistique sont plus élevées, et que les recall des deux modèles sont à peu près les mêmes.

**Attention !**

Le modèle de régression logistique est plus performant, bien que l'on puisse toujours utiliser un arbre de décision pour sa facilité d'interprétation.

# Exercice d'application 12

## Calcul des métriques

Le programme suivant est valide si le modèle est un objet de quel type (plusieurs réponses) ?

```
model.fit(X, y)
print(model.score(X, y))
```

1. LogisticRegression
2. KFold
3. DecisionTreeClassifier

# Gini Vs Entropy 1

**Le critère d'impureté (impurity criterion) par défaut de l'algorithme d'arbre décisionnel de Scikit-learn est l'impureté de Gini**. Cependant, l'entropie a également été implémentée et vous pouvez choisir celui que vous souhaitez utiliser lorsque vous créez l'objet DecisionTreeClassifier.
Si vous allez dans la documentation, vous pouvez voir que l'un des paramètres est le critère.

Pour construire un arbre de décision qui utilise l'entropie, nous devons définir le paramètre de critère sur l'entropie. Ci-dessous le code permettant de construire un arbre de décision qui utilise l'entropie au lieu de l'impureté de Gini :
dt = DecisionTreeClassifer(criterion='entropy')

# Gini Vs Entropy 2

Nous pouvons maintenant comparer un arbre décisionnel utilisant le gini avec un arbre décisionnel utilisant l'entropie. Nous créons d'abord une division k-fold car lorsque nous comparons deux modèles, nous voulons qu'ils utilisent les mêmes divisions train/test pour être équitables. Ensuite, nous effectuons une validation croisée k-fold avec chacun des deux modèles possibles. Nous calculons l'accuracy, la précision et le rappel pour chacune des deux options.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score, precision_score,
recall_score
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
```

# Gini Vs Entropy 3

```
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
kf = KFold(n_splits=5, shuffle=True)
for criterion in ['gini', 'entropy']:
    print("Decision Tree - ".format(criterion))
    accuracy = []
    precision = []
    recall = []
    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        dt = DecisionTreeClassifier(criterion=criterion)
        dt.fit(X_train, y_train)
        y_pred = dt.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_pred))
        precision.append(precision_score(y_test, y_pred))
```

# Gini Vs Entropy 4

```
print("accuracy:", np.mean(accuracy))
print("precision:", np.mean(precision))
print("recall:", np.mean(recall))
```

**Attention !**

Nous constatons une très faible différence entre les performances de Gini et celles e l'Entropie. C'est normal puisqu'il ne s'agit pas vraiment de fonctions très différentes. Il est rare de trouver un ensemble de données où le choix ferait une différence.

# Exercice d'application 13

## Gini Vs Entropy

Lesquelles des propositions suivantes crée un arbre de décision avec Gini comme critère d'impureté ?

1. dt = DecisionClassifier()
2. dt = DecisionTreeClassifier(impurity_criterion ='gini')
3. dt = DecisionTreeClassifier(impurity ='gini')
4. dt = DecisionTreeClassifier(criterion ='gini')

# Graphique d'un arbre de décision 1

Si vous souhaitez **créer une image png** de votre graphique, comme celles présentées dans ce module, vous pouvez utiliser **la fonction export_graphviz()** de Scikit-learn.
Tout d'abord, importons la :

```
from sklearn.tree import export_graphviz
dot_file = export_graphviz(dt, feature_names=feature_names)
```

# Graphique d'un arbre de décision 2

Ensuite, nous utilisons la fonction export_graphviz(). Ici, **dt** est un **objet Arbre de Décision et feature_names est une liste de noms de variables**. Les objets graphiques sont stockés dans des fichiers .dot qui peuvent être utilisés par le programme GraphViz. Notre objectif est d'enregistrer un fichier image png. Nous pourrons convertir le fichier .dot en un fichier png, donc nous enregistrons d'abord le fichier .dot dans une variable, donc nous enregistrons le fichier .dot créé par la fonction export_graphviz afin de pouvoir le convertir en png.

Nous pouvons ensuite utiliser le module graphviz pour le convertir au format d'image png.

```
import graphviz
graph = graphviz.Source(dot_file)
```

# Graphique d'un arbre de décision 3

Finalement, nous pouvons utiliser la méthode render pour créer le fichier. Nous lui indiquons le nom et le format du fichier. Par défaut, il créera des fichiers supplémentaires qui ne nous intéressent pas, nous ajoutons donc le nettoyage pour lui dire de s'en débarrasser :

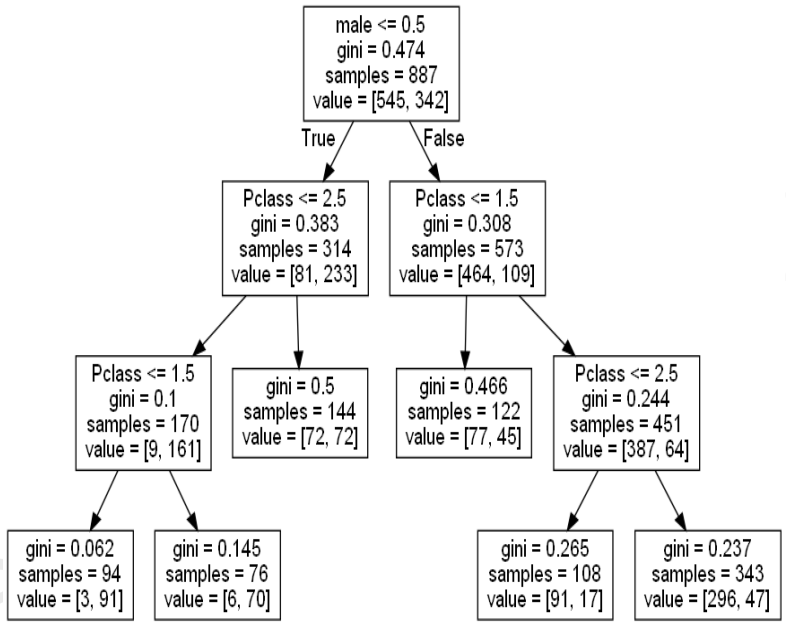
```
graph.render(filename='tree', format='png', cleanup=True)
```

Vous devriez maintenant avoir un fichier appelé tree.png sur votre ordinateur. Vous trouverez dans la prochaine slide le programme pour construire l'arbre pour l'ensemble de données Titanic avec seulement les variables Sex et Pclass.

# Graphique d'un arbre de décision 4

```
from sklearn.tree import export_graphviz
import graphviz
from IPython.display import Image
feature_names = ['Pclass', 'male']
X = df[feature_names].values
y = df['Survived'].values
dt = DecisionTreeClassifier()
dt.fit(X, y)
dot_file = export_graphviz(dt, feature_names=feature_names)
graph = graphviz.Source(dot_file)
graph.render(filename='tree', format='png', cleanup=True)
```

# Graphique d'un arbre de décision 5




**Attention !**

Si vous avez l'intention de l'exécuter sur votre ordinateur, assurez-vous d'installer graphviz en premier. Vous pouvez le faire avec "pip install graphviz".

# Exercice d'application 14

## Interprétation

Réarrangez les lignes de code suivantes pour construire un arbre de décision et sauvegarder l'image dans un fichier png.

1. graph = graphviz.Source(dot_file)
2. dt = DecisionTreeClassifier()
3. dot_file = export_grapviz(dt, feature_names=['PClass', 'Age', 'Fare']
4. graph.render(filename='tree', format='png', cleanup=True)
5. dt.fit(X,y)

# Machine Learning (ML)

## Chap 4 : Decision Trees

## Partie II

David TCHOUTA

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

April 11, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Surapprentissage
3. Élagage
4. Pros and Cons
5. Quiz

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue le problème de surapprentissage (overfitting) très fréquents dans les arbres de décision. Nous verrons également comment les solutionner.

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

- de comprendre les problèmes de surapprentissage (overfitting)
- de les solutionner

# Surapprentissage dans les arbres de décision 1

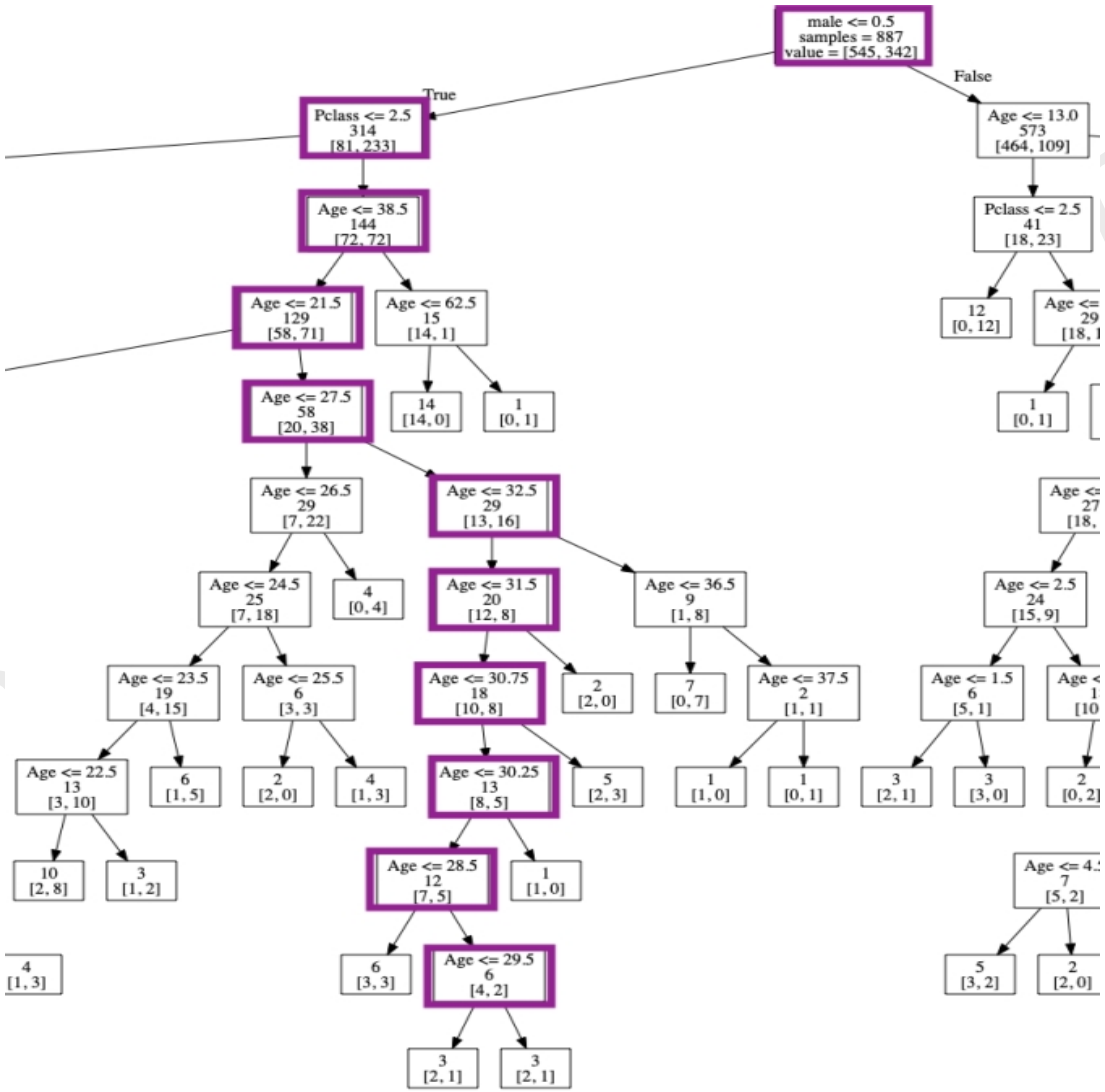
Rappelons que l'on parle d'ajustement excessif (surapprentissage) lorsque nous réussissons à construire un modèle pour l'ensemble d'apprentissage, mais qu'il n'est pas performant sur l'ensemble de test. **Les arbres de décision sont incroyablement enclins au surajustement(surapprentissage)**. Puisqu'ils peuvent continuer à avoir des nœuds supplémentaires dans l'arbre qui se divisent sur les variables, le modèle peut vraiment creuser profondément dans les spécificités de l'ensemble d'apprentissage. Selon les données, il peut en résulter un modèle qui ne capture pas la vraie nature des données et ne se généralise pas.

# Surapprentissage dans les arbres de décision 2

Peut-être que nous n'avons qu'un seul point de données qui va vers un nœud terminal. Il n'est peut-être pas utile d'avoir cette division supplémentaire.

Examinons le diagramme d'un arbre de décision pour l'ensemble de données Titanic. Il s'agit de l'arbre résultant de la construction d'un arbre de décision avec scikit-learn sur l'ensemble des données. Nous n'examinons qu'une partie de l'arbre de décision, car il est très volumineux. Nous avons mis en évidence un chemin particulier qui nous intéresse.

# Surapprentissage dans les arbres de décision 3



# Surapprentissage dans les arbres de décision 4

Si vous suivez le chemin en surbrillance, vous verrez que nous avons splités sur le sexe, la PClass, puis sur l'âge, 9 fois de suite avec différents seuils. **Le résultat est un graphique qui est très pointilleux sur l'âge**. Une passagère de la PClass 3 âgée de 31 ans va vers un nœud terminal différent de celui d'une passagère similaire âgée de 30.5, 30 ou 29 ans. Le modèle prédit qu'une passagère de 35 ans survit, que celle de 32 ans ne survit pas, que celle de 31 ans survit et que celle de 30 ans ne survit pas. **Ce modèle est probablement trop fin et donne trop de pouvoir aux points de données uniques de notre ensemble de données**. Vous pouvez voir que les nœuds terminaux ont tous peu de points de données et souvent un seul.

# Surapprentissage dans les arbres de décision 5

**Attention !**

Si vous laissez un arbre de décision continuer à se construire, il risque de créer un arbre trop ajusté (overfit) et qui ne capture pas l'essence des données.

# Exercice d'application 1

## Surappprentissage

En utilisant l'arbre de décision de la page 6, quelle est la prédiction pour les passagers suivants ? Chaque passager est une femme de la PClass 3 d'un âge différent. Inscrivez 0 pour " n'a pas survécu " et 1 pour " a survécu ".

1. 24 : .....
2. 25 : .....
3. 27 : .....

# Élagage (Pruning) 1

Afin de **résoudre ces problèmes de surapprentissage**, nous procédons à ce que l'on appelle **l'élagage de l'arbre (Pruning)**. Cela signifie que **nous réduisons la taille de l'arbre dans le but de réduire l'overfitting**.

Il existe deux types d'élagage : le **pré-élagage** et le **post-élagage**. Dans le cas du **pré-élagage**, nous avons des **règles pour savoir quand arrêter la construction de l'arbre**, de sorte que **nous arrêtons la construction avant que l'arbre ne devienne trop grand**. Dans le cas du **post-élagage**, **nous construisons l'arbre entier, puis nous l'examinons et décidons quels nœuds supprimer pour réduire la taille de l'arbre**.

# Élagage (Pruning) 2

**Attention !**

Le terme "élagage" vient de l'agriculture. Les agriculteurs coupent les branches des arbres et nous faisons de même avec notre arbre décisionnel.

# Exercice d'application 2

## Élagage

Dans lequel des cas suivants, nous construisons l'arbre de décision dans son intégralité, puis nous le réduisons ?

1. surapprentissage
2. post-élagage
3. pré-élagage

# pré-élagage 1

Nous allons nous concentrer sur les techniques de pré-élagage car elles sont plus faciles à mettre en œuvre. Nous disposons de plusieurs options pour limiter la croissance de l'arbre. Voici quelques techniques de pré-élagage couramment utilisées :

- **Profondeur maximale (Max depth)** : Ne faites pousser l'arbre que jusqu'à une certaine profondeur, ou hauteur de l'arbre. Si la profondeur maximale est de 4, il y aura au maximum 4 scissions pour chaque point de données.
- **Taille des nœuds terminaux (Leaf size)** : Ne pas diviser un nœud si le nombre d'échantillons à ce nœud est inférieur à un seuil.
- **Nombre de nœuds terminaux (Number of leaf nodes)** : Limite le nombre total de nœuds terminaux autorisés dans l'arbre.

# pré-élagage 2

**L'élagage est un équilibre**. Par exemple, si vous définissez une **profondeur maximale trop faible, vous n'aurez pas d'arbre et vous n'aurez pas de pouvoir prédictif**. C'est ce qu'on appelle le **sous-ajustement (underfitting)**. De même, si la taille des nœuds terminaux est trop importante ou si le nombre de nœuds terminaux est trop faible, vous aurez un modèle sous-ajusté (underfitting).

**Attention !**

Il n'y a pas de science exacte pour déterminer quelle méthode de pré-élagage donnera les meilleurs résultats. En pratique, nous essayons plusieurs valeurs différentes pour chaque paramètre et nous effectuons une validation croisée pour comparer leurs performances.

# Exercice d'application 3

## Pré-élagage

Lesquels des éléments suivants sont des techniques de pré-élagage ? (Choisir toutes les bonnes réponses)

1. Utiliser seulement la moitié des points de données
2. Limiter la depth (height) de l'arbre
3. Limiter le nombre de fois qu'on peut splitter une variable donnée
4. Limiter le nombre de nœuds terminaux de l'arbre
5. Limiter le splitting lorsqu'un nœud à un nombre minimal de points.

# Paramètres de pré-élagage 1

Scikit-learn a mis en œuvre un grand nombre de techniques de pré-élagage. Nous nous intéresserons plus particulièrement à **trois des paramètres : max_depth, min_samples_leaf et max_leaf_nodes**. Consultez la documentation sur les arbres décisionnels pour obtenir des explications complètes sur ces trois paramètres.

**Technique de pré-élagage 1** : limiter la profondeur (depth)

Nous utilisons le paramètre **max_depth** pour **limiter le nombre d'étapes** que l'arbre peut avoir entre le nœud racine et les nœuds terminaux.

# Paramètres de pré-élagage 2

**Technique de pré-élagage 2** : éviter les nœuds terminaux avec peu de points de données

Nous utilisons le paramètre min_samples_leaf pour indiquer au modèle d'arrêter la construction de l'arbre prématurément si le nombre de points de données dans un nœud est inférieur à un seuil.

**Technique de pré-élagage 3** : Limitation du nombre de nœuds terminaux

Nous utilisons max_leaf_nodes pour fixer une limite au nombre de nœuds terminaux dans l'arbre.

# Paramètres de pré-élagage 3

Voici le code pour créer un arbre de décision avec les propriétés suivantes :
- profondeur maximale de 3
- nombre minimum d'échantillons par nœud de 2
- nombre maximal de nœuds terminaux de 10

dt = DecisionTreeClassifier(max_depth=3, min_samples_leaf=2, max_leaf_nodes=10)

# Paramètres de pré-élagage 4

Vous pouvez maintenant entraîner le modèle et le tester comme nous l'avons fait précédemment.

Vous pouvez utiliser autant ou aussi peu de paramètres que vous le souhaitez.

**Attention !**

Pour déterminer les meilleures valeurs pour les paramètres de pré-élagage, nous utiliserons la validation croisée pour comparer plusieurs options potentielles.

# Exercice d'application 4

## Paramètres de pré-élagage

Écrivez le code pour créer un arbre de décision avec : une profondeur maximale de 3
nombre minimum d'échantillons par nœud de 2
nombre maximum de nœuds terminaux de 10

dt = DecisionTreeClassifier( ..... = 3, ..... = 2, ..... = 10)

# La fonction GridSearchCV() 1

Nous ne serons pas en mesure d'intuitionner les meilleures valeurs pour les paramètres de pré-élagage. Afin de décider lesquels utiliser, nous utilisons la **validation croisée et comparons les métriques**. Nous pourrions passer en boucle nos différentes options comme nous l'avons fait dans la leçon sur les arbres de décision dans Scikit-learn, mais Scikit-learn a intégré **une classe de recherche de grille qui le fera pour nous**.

Cette **classe** s'appelle **GridSearchCV**. Commençons par l'importer :

```
from sklearn.model_selection import GridSearchCV
```

# La fonction GridSearchCV() 2

**GridSearchCV** possède **quatre paramètres** que nous allons utiliser :

1. Le modèle (model) (dans ce cas un DecisionTreeClassifier)
2. La grille de paramètres (param grid) : un dictionnaire des noms des paramètres et de toutes les valeurs possibles
3. La métrique à utiliser (par défaut, l'accuracy)
4. Combien de folds pour la validation croisée k-fold ?

# La fonction GridSearchCV() 3

Créons la variable **param grid**. Nous allons donner une **liste de toutes les valeurs possibles** pour chaque paramètre que nous voulons essayer :

```
param_grid =
    'max_depth': [5, 15, 25],
    'min_samples_leaf': [1, 3],
    'max_leaf_nodes': [10, 20, 35, 50]
```

# La fonction GridSearchCV() 4

Maintenant, nous créons l'objet de **recherche de grille (grid search)**. Nous utiliserons la **grille de paramètres** ci-dessus, définirons **la métrique de notation au score F1**, et ferons **une validation croisée 5 fois**.

dt = DecisionTreeClassifier() gs = GridSearchCV(dt, param_grid, scoring='f1', cv=5)

Nous pouvons maintenant adapter l'objet de recherche de la grille. Cela peut prendre un peu de temps car toutes les combinaisons possibles des paramètres sont essayées.
gs.fit(X, y)

# La fonction GridSearchCV() 5

Comme nous avons 3 valeurs possibles pour max_depth, 2 pour min_samples_leaf et 4 pour max_leaf_nodes, nous avons 3 * 2 * 4 = 24 combinaisons différentes à essayer :

max_depth: 5, min_samples_leaf: 1, max_leaf_nodes: 10
max_depth: 15, min_samples_leaf: 1, max_leaf_nodes: 10
max_depth: 25, min_samples_leaf: 1, max_leaf_nodes: 10
max_depth: 5, min_samples_leaf: 3, max_leaf_nodes: 10

# La fonction GridSearchCV() 6

Nous utilisons l'attribut **best_params_** pour voir quel modèle est meilleur :

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
df = pd.read_csv('https://sololearn.com/uploads/files/titanic.csv')
df['male'] = df['Sex'] == 'male'
X = df[['Pclass', 'male', 'Age', 'Siblings/Spouses',
'Parents/Children', 'Fare']].values
y = df['Survived'].values
```

# La fonction GridSearchCV() 7

```
param_grid =
    'max_depth': [5, 15, 25],
    'min_samples_leaf': [1, 3],
    'max_leaf_nodes': [10, 20, 35, 50]
dt = DecisionTreeClassifier()
gs = GridSearchCV(dt, param_grid, scoring='f1', cv=5)
gs.fit(X, y)
print("best params:", gs.best_params_)
```

# La fonction GridSearchCV() 8

Ainsi, nous voyons que le meilleur modèle a une maximum depth of 15, maximum number of leaf nodes de 35 and minimum samples per leaf de 1.

L'attribut best_score_ nous indique le score du modèle gagnant : print("best score:", gs.best_score_ $\Longrightarrow$ 0.7753

**Attention !**

Il y a souvent quelques modèles qui ont des performances très similaires. Si vous l'exécutez plusieurs fois, vous pouvez obtenir des résultats légèrement différents en fonction du caractère aléatoire de la répartition des points entre les folds. En général, si nous avons plusieurs modèles avec des performances comparables, nous choisissons le modèle le plus simple.

# Exercice d'application 5

## La fonction GridSearchCV()

Soit le code suivant pour exécuter une recherche par grille :

```
param_grid =
'max_depth': [5, 15, 25, 35],
'max_leaf_nodes': [10, 20, 35, 50, 75]
dt = DecisionTreeClassifier()
gs = GridSearchCV(dt, param_grid, scoring='f1', cv=5)
gs.fit(X, y)
```

Combien de modèles différents sont comparés ?

.....

# Quantité de calculs 1

Lorsque nous parlons de la quantité de calcul requise pour un algorithme d'apprentissage automatique, nous la séparons en deux questions : la quantité de calcul requise pour construire le modèle et la quantité de calcul requise pour prédire.

La construction d'un arbre de décision est très coûteuse en termes de calcul. Cela est dû au fait qu'à chaque nœud, nous essayons chaque variable et chaque seuil comme une division possible. Nous devons calculer à chaque fois le gain d'information de chacune de ces divisions possibles. Cette opération est très coûteuse en termes de calcul.

# Quantité de calculs 2

En revanche, la prédiction à l'aide d'un arbre de décision est très peu coûteuse sur le plan informatique. Il suffit de poser une série de questions oui/non sur le point de données pour arriver à la prédiction.

**Attention !**

En général, nous nous soucions beaucoup plus du temps de calcul pour la prédiction que pour l'apprentissage. Les prédictions doivent souvent être effectuées en temps réel pendant que l'utilisateur attend un résultat.

# Exercice d'application 6

## Quantité de calculs

Les arbres de décision sont rapides à apprendre et lents à prédire.

1. Vrai
2. Faux

# Performance 1

Les arbres de décision peuvent donner de bons résultats en fonction des données, mais comme nous l'avons vu, ils ont tendance à s'adapter de manière excessive (overfitting). Étant donné qu'un nœud terminal ne peut avoir qu'un seul point de données qui y atterrit, il donne trop de pouvoir aux points de données individuels.

Pour remédier aux problèmes de surajustement (overfitting), les arbres de décision nécessitent généralement un certain réglage pour obtenir le meilleur modèle possible. Les techniques d'élagage sont utilisées pour limiter la taille de l'arbre et contribuent à atténuer l'overfitting.

# Performance 2

**Attention !**

Les arbres décisionnels nécessitent souvent du travail pour obtenir des performances comparables à celles d'autres modèles sans réglage.

# Exercice d'application 7

## Performance

Les arbres de décision sont sujets à l'overfitting :

1. Vrai
2. Faux

# Interprétation 1

La principale raison pour laquelle les gens aiment choisir les arbres de décision est qu'ils sont facilement interprétables. Selon la raison pour laquelle vous construisez un modèle, vous pouvez avoir besoin de donner une raison pour laquelle vous avez fait une certaine prédiction. Une personne non technique peut interpréter un arbre décisionnel, il est donc facile de donner une explication à une prédiction.

# Interprétation 2

Ceci est particulièrement vrai dans les situations juridiques. Imaginons que vous soyez une banque et que vous disposiez d'un modèle permettant de prédire si une personne doit se voir accorder un prêt ou non. Il est important de pouvoir expliquer pourquoi le modèle a pris cette décision, sinon vous pourriez cacher des pratiques discriminatoires dans le modèle.

**Attention !**

L'interprétabilité est le plus grand avantage des arbres décisionnels. La question de savoir si cela est important pour votre problème dépend de la situation.

# Exercice d'application 8

## Interprétation

Dans lequel des cas suivants un arbre de décision est-il le plus susceptible d'être le modèle approprié ?

1. Nous avons besoin que le modèle soit construit rapidement.
2. Nous ne voulons pas ajuster le modèle.
3. Nous devons expliquer les prédictions aux clients.

# Quiz 1

## 1) Prédiction

Dans notre jeu de données d'entraînement, parmi les passagers féminins de la classe 1, 10 ont survécu et 5 n'ont pas survécu. Il n'y a pas de passagères dans la classe 2. Parmi les passagères de la classe 3, 5 ont survécu et 7 n'ont pas survécu.

Le modèle prédit qu'une passagère de la classe 3 a survécu :

1. Vrai
2. Faux

# Quiz 2

## 2) Impureté et Gain d'information

Nous construisons un arbre de décision pour l'ensemble de données Titanic. Quelle est l'impureté de Gini des deux côtés de la répartition suivante :

Côté gauche : 10 points de données : 0 a survécu, 10 n'a pas survécu
Côté droit : 10 points de données : 5 ont survécu, 5 n'ont pas survécu

1. Gini impureté côté gauche 0.5 / Gini impureté côté droit 0.5
2. Gini impureté côté gauche 0.0 / Gini impureté côté droit 0.5
3. Gini impurity côté gauche 1.0 / Gini impurity côté droit 0.0

# Quiz 3

## 3) Split

Nous construisons un arbre de décision pour l'ensemble de données du Titanic. Nous avons un ensemble de 20 points de données où 6 ont survécu et 14 n'ont pas survécu (que nous écrirons [6, 14]).
Voici les trois divisions que nous envisageons :

Division A : [3, 2] / [3, 12]
Division B : [6, 0] / [0, 14]
Division C : [10, 0] / [6, 4]
Quel fractionnement présente le gain d'information le plus élevé ?

1. A
2. B
3. C

# Quiz 4

## 4) Élagage

Laquelle des affirmations est vraie :

1. Élagage est une manière de solutionner l'overfitting
2. Un arbre de décision élagué aura plus de nœuds qu'un arbre de décision non élagué
3. Plus d'élagage implique de meilleures performances
4. Limiter le nombre de nœuds terminaux d'un arbre de décision est un exemple de pré-élagage

# Quiz 5

## 5) Tuning Parameters

Lequel des éléments suivants crée un modèle d'arbre de décision avec une profondeur maximale de 3 et au maximum 20 nœuds terminaux ?

1. DecisionTreeClassifier(maximum_depth=3, maximum_leaf_nodes=20)
2. DecisionTreeClassifier(max_depth=3, max_leaf_nodes=20)
3. DecisionTreeClassifier(max_depth:3, max_leaf_nodes:20)
4. DecisionTreeClassifier()

# Quiz 6

## 6) GridSearchCV()

Soit le code suivant pour exécuter une recherche par grille :

```
param_grid =
'max_depth': [5, 15, 25, 35],
'max_leaf_nodes': [10, 20, 35]
dt = DecisionTreeClassifier()
gs = GridSearchCV(dt, param_grid, scoring='f1', cv=5)
gs.fit(X, y)
```

Combien de modèles différents sont comparés ?

.....

# Machine Learning (ML)

## Chap 5 : Random Forests

David TCHOUTA

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

April 12, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Introduction
3. Random Forests in Sklearn
4. Tuning
5. Pros and Cons
6. Quiz

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue le modèle et l'algorithme des Random Forests.

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

- de comprendre l'algorithme des random forests
- de réaliser le tuning des paramètres

# Améliorer les arbres de décision 1

Comme nous l'avons appris dans le chapitre précédent, le **principal inconvénient des arbres de décision est qu'ils ont tendance à se surajuster (overfitting)**. Nous avons vu que nous pouvions **améliorer leurs performances avec l'élagage**, mais dans ce chapitre, nous allons voir une façon d'utiliser les arbres de décision pour construire un meilleur modèle.

Les **arbres décisionnels** sont **très sensibles aux idiosyncrasies aléatoires** de l'ensemble de données d'entraînement. On dit que **les arbres de décision ont une variance élevée** car si vous changez aléatoirement l'ensemble de données d'apprentissage, vous pouvez vous retrouver avec un arbre très différent.

# Améliorer les arbres de décision 2

**L'un des avantages des arbres de décision** par rapport à un **modèle comme la régression logistique** est **qu'ils ne font aucune hypothèse sur la structure des données**. Dans **la régression logistique**, **nous supposons** que nous pouvons **tracer une frontière** pour diviser les données. Parfois, nos données ne sont tout simplement pas structurées de cette façon. Un arbre de décision a le potentiel d'atteindre l'essence des données, quelle que soit leur structure.

# Améliorer les arbres de décision 3

Dans ce module, nous allons nous familiariser avec les **forêts aléatoires (random forests)**, qui, comme vous pouvez le deviner d'après leur nom, **sont des modèles construits avec plusieurs arbres**. **L'objectif des forêts aléatoires est de profiter des avantages des arbres de décision tout en atténuant les problèmes de variance**.

**Attention !**

**Une forêt aléatoire** est **un exemple d'ensemble** car elle utilise plusieurs modèles d'apprentissage automatique pour créer un modèle unique.

# Exercice d'application 1

## Introduction

Sélectionnez toutes les affirmations qui sont vraies :

1. Une forêt aléatoire est construit avec plusieurs arbres
2. Une forêt aléatoire calcule une frontière qui sépare les données
3. Une forêt aléatoire est moins sujette à l'ajustement excessif (overfitting) qu'un arbre de décision.

# Bootstrapping 1

Un **échantillon bootstrap** est un échantillon aléatoire de points de données dans lequel nous sélectionnons au hasard, avec remplacement, des points de données de notre ensemble de données original pour créer un ensemble de données de la même taille. **La sélection aléatoire avec remplacement signifie que nous pouvons choisir le même point de données plusieurs fois**. Cela signifie que dans un échantillon bootstrap (bootstrapped), certains points de données de l'ensemble de données original apparaîtront plusieurs fois et d'autres pas du tout.

# Bootstrapping 2

Par exemple, si nous avons quatre points de données A, B, C, D, il pourrait s'agir de 3 rééchantillons :

A, A, B, C
B, B, B, D
A, A, C, C

Nous préférerions pouvoir obtenir plus d'échantillons de données de la population, mais comme nous ne disposons que de notre ensemble d'entraînement, nous l'utilisons pour générer des ensembles de données supplémentaires.

**Attention !**

Nous utilisons le bootstrapping pour imiter la création d'échantillons multiples.

# Exercice d'application 2

## Bootstrapping

Supposons que nous avons cet échantillon de 5 points de données : A, B, C, D, E.

Lesquels des éléments suivants sont des échantillons bootstrapped valides ?

1. A, A, A
2. A, A, A, A, A
3. A, B, C, D, E
4. A, A, B, B

# Bootstrap Aggregation (Bagging) 1

**L'agrégation bootstrap (ou Bagging)** est une technique permettant de réduire la variance d'un modèle individuel en créant un ensemble à partir de plusieurs modèles construits sur des échantillons bootstrap.

Pour mettre en sac (to bag) les arbres de décision, nous créons plusieurs (supposons 10) rééchantillons bootstrapped de notre ensemble de données d'apprentissage. Ainsi, si notre ensemble d'apprentissage compte 100 points de données, chacun des rééchantillons comprendra 100 points de données choisis au hasard dans notre ensemble d'apprentissage. Rappelons que nous effectuons une sélection aléatoire avec remplacement, ce qui signifie que certains points de données apparaîtront plusieurs fois et d'autres pas du tout.

# Bootstrap Aggregation (Bagging) 2

Nous créons un arbre de décision avec chacun de ces 10 rééchantillons.

Pour faire une prédiction, nous faisons une prédiction avec chacun des 10 arbres de décision, puis chaque arbre de décision reçoit un vote. **La prédiction qui obtient le plus de votes est la prédiction finale.**

# Bootstrap Aggregation (Bagging) 3

Lorsque nous bootstrapons l'ensemble de formation, nous essayons d'éliminer la variance de l'arbre de décision. La moyenne de plusieurs arbres ayant des ensembles d'apprentissage différents créera un modèle qui s'approchera plus précisément de l'essence des données.

**Attention !**

La mise en sac des arbres décisionnels est un moyen de réduire la variance du modèle.

# Exercice d'application 3

## Bootstrap Aggregation (Bagging)

Supposons que nous disposions de 100 points de données et que nous utilisions 10 arbres de décision en sac. Le premier point de données serait utilisé pour construire combien de ces 10 arbres ?

1. Exactement 10
2. Nous ne le savons pas car c'est aléatoire
3. Moins de 10
4. Plus de 10

# Decorrélation des arbres 1

Avec les arbres de décision en sac (bagged), les arbres peuvent encore être trop similaires pour avoir bien créé le modèle idéal. Ils sont construits sur des rééchantillons différents, mais ils ont tous accès aux mêmes variables. Nous ajouterons certaines restrictions au modèle lors de la construction de chaque arbre de décision afin que les arbres présentent davantage de variation. Nous appelons cela la décorrélation des arbres (decorrelating the trees).

# Decorrélation des arbres 2

Si vous vous souvenez, lors de **la construction d'un arbre de décision**, **à chaque nœud, nous comparons tous les seuils de division pour chaque variable afin de trouver la meilleure variable et le meilleur seuil de division**. Dans **un arbre de décision pour une forêt aléatoire**, **à chaque nœud, nous sélectionnons aléatoirement un sous-ensemble de variables à considérer**. Ainsi, à chaque étape, nous choisissons une bonne, mais pas la meilleure, variable à diviser. Il est important de noter que la sélection aléatoire des variables se fait à chaque nœud. Ainsi, au premier nœud, nous pouvons considérer les variables Sex et Fare, puis au deuxième nœud, les variables Fare et Age.

# Decorrélation des arbres 3

Le choix standard pour le nombre de variables à considérer à chaque fractionnement est la racine carrée du nombre de variables. Ainsi, si nous avons 9 variables, nous en considérerons 3 à chaque nœud (choisies au hasard).

**Si nous mettons en sac ces arbres de décision, nous obtenons une forêt aléatoire**.

**Attention !**

Chaque arbre de décision au sein d'une forêt aléatoire est probablement pire qu'un arbre de décision standard. Mais lorsque nous en faisons la moyenne, nous obtenons un modèle très solide.

# Exercice d'application 4

## Decorrélation des arbres

Laquelle des propositions suivantes est correcte ?

1. Nous considérons généralement une moitié aléatoire des variables
2. Nous ajoutons la sélection aléatoire de variables afin de décorréler les arbres
3. Dans un arbre de décision d'une forêt aléatoire, nous prenons une sélection aléatoire des variables à utiliser avant de construire l'arbre

# Ensemble de données sur le Cancer du Sein 1

Nous utiliserons l'**ensemble de données sur le cancer du sein** dans cette section. Rappelez-vous que cet ensemble de données **contient des mesures de différents attributs d'une tumeur dans le tissu mammaire et une étiquette/label indiquant si la tumeur est cancéreuse ou non**.

Ci-dessous le code pour extraire les données de sklearn.

# Ensemble de données sur le Cancer du Sein 2

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
cancer_data = load_breast_cancer()
df = pd.DataFrame(cancer_data['data'],
columns=cancer_data['feature_names'])
df['target'] = cancer_data['target']
X = df[cancer_data.feature_names].values
y = df['target'].values
print('data dimensions', X.shape)
```

L'ensemble de données comprend **569 points de données** et **30 variables**.

# Exercice d'application 5

## Données sur le Cancer du Sein

Réorganisez ces lignes de code pour créer la matrice de variables X de l'ensemble de données sur le cancer du sein.

1. from sklearn.datasets import load_breast_cancer
2. X = df[d.feature_names].values
3. df = pd.DataFrame(d['data'], columns = d['feature_names'])
4. d = load_breast_cancer()

# Random Forest dans Sklearn 1

La syntaxe pour construire et utiliser un modèle Random Forest est la même que pour la régression logistique et les arbres de décision. Les créateurs de scikit-learn ont intentionnellement fait en sorte qu'il soit facile de passer d'un modèle à l'autre et de comparer les différents modèles.

Voici l'instruction d'importation du modèle Random Forest pour la classification :

```
from sklearn.ensemble import RandomForestClassifier
```

# Random Forest dans Sklearn 2

Nous allons d'abord diviser l'ensemble de données en un ensemble d'entraînement et un ensemble de test :

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=3)
```

Nous avons ajouté le paramètre d'état aléatoire ici afin qu'il effectue la même répartition à chaque fois que nous exécutons le code. Sans l'état aléatoire, nous pourrions nous attendre à des points de données différents dans les ensembles d'apprentissage et de test à chaque fois que nous exécutons le code, ce qui peut rendre le test du code plus difficile.

# Random Forest dans Sklearn 3

Ensuite, nous créons l'objet RandomForestClassifier et utilisons la méthode fit pour construire le modèle sur l'ensemble d'entraînement.

rf = RandomForestClassifier() rf.fit(X_train, y_train)

Nous pouvons maintenant utiliser le modèle pour faire une prédiction. Par exemple, prenons la première ligne de l'ensemble de test et voyons quelle est la prédiction. **Rappelez-vous que la méthode predict prend un tableau de points, donc même si nous n'avons qu'un seul point, nous devons le mettre dans une liste.**

# Random Forest dans Sklearn 4

```
import pandas as pd from sklearn.datasets import
load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
cancer_data = load_breast_cancer()
df = pd.DataFrame(cancer_data['data'],
columns=cancer_data['feature_names'])
df['target'] = cancer_data['target']
X = df[cancer_data.feature_names].values
y = df['target'].values
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=3)
```

# Random Forest dans Sklearn 5

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
first_row = X_test[0]
print("prediction:", rf.predict([first_row])) $\Longrightarrow$ [1]
print("true value:", y_test[0]) $\Longrightarrow$ 1

**Ces résultats signifient que le modèle a prédit que la tumeur était cancéreuse, ce qui était correct**.
Nous pouvons utiliser la méthode des scores pour calculer la précision sur l'ensemble du jeu de tests.

print("random forest accuracy:", rf.score(X_test, y_test))

La précision est donc de 96.5%. Nous pouvons voir comment cela se compare au modèle de l'arbre de décision.

# Random Forest dans Sklearn 6

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
print("decision tree accuracy:", dt.score(X_test, y_test)) $\Longrightarrow$
0.9020

Ainsi, l'accuracy du modèle d'arbre de décision est de 90.2%, bien pire que celle de la forêt aléatoire.

**Attention !**

Notez à quel point le code de scikit-learn est similaire à celui de la régression logistique et des arbres décisionnels. Cela rend très facile l'essai et la comparaison de différents modèles.

# Exercice d'application 6

## Random Forest dans Sklearn

Dans le programme ci-dessous, le modèle peut être une instance de laquelle des classes suivantes ? (sélectionnez toutes les bonnes réponses)

model.predict([[1, 2]])

1. KFold
2. RandomForestClassifier
3. DecisionTreeClassifier
4. LogisticRegression

# Paramètres des forêts aléatoires 1

Lorsque vous consultez la documentation de scikit-learn pour le **RandomForestClassifier**, vous verrez un certain nombre de paramètres que vous pouvez contrôler. Nous allons en examiner quelques-uns, mais vous pouvez les explorer tous en détail dans la documentation.

Puisqu'une forêt aléatoire est composée d'arbres de décision, nous avons tous les mêmes paramètres de réglage (**tuning parameters**) pour le pré-élagage que pour les arbres de décision : max_depth, min_samples_leaf, et max_leaf_nodes. Avec les forêts aléatoires, il n'est généralement pas important de régler ces paramètres car l'ajustement excessif (overfitting) n'est généralement pas un problème.

# Paramètres des forêts aléatoires 2

Nous allons voir deux nouveaux paramètres de réglage : **n_estimators (le nombre d'arbres)** et **max_features (le nombre de variables à considérer à chaque division)**.

La valeur par défaut pour **max_features est la racine carrée de p, où p est le nombre de variables (ou prédicteurs)**. La valeur par défaut est généralement un bon choix pour le maximun de variables et nous n'aurons généralement pas besoin de la modifier, mais vous pouvez la définir à un nombre fixe comme suit :

rf = RandomForestClassifier(max_features=5)

# Paramètres des forêts aléatoires 3

**Le nombre d'estimateurs (nombre d'arbres de décision) par défaut est de 10**. Cela fonctionne souvent bien mais peut dans certains cas être trop faible. Vous pouvez le fixer à un autre nombre comme suit. Nous verrons dans les parties suivantes comment choisir la meilleure valeur.

rf = RandomForestClassifier(n_estimators=15)

**Attention !**

L'un des grands avantages des forêts aléatoires est qu'elles nécessitent très peu de réglages. Les valeurs par défaut fonctionnent bien sur la plupart des ensembles de données.

# Exercice d'application 7

## Paramètres des forêts aléatoires

Lequel des éléments suivants compléterait le code permettant de créer une forêt aléatoire avec 17 arbres.

rf = RandomForestClassifier( ..... )

1. n_estimators=17
2. max_estimators=17
3. max_features=17

# La fonction GridSearchCV() 1

Si vous vous souvenez du module Decision Tree, scikit-learn a intégré une classe Grid Search pour nous aider à trouver le choix optimal des paramètres.

Utilisons Grid Search pour comparer les performances d'une forêt aléatoire avec différents nombres d'arbres.

Rappelez-vous que nous devons **définir la grille des paramètres (parameter grid)** que nous voulons **faire varier et donner une liste des valeurs à essayer**.

param_grid = 'n_estimators': [10, 25, 50, 75, 100],

# La fonction GridSearchCV() 2

Nous pouvons maintenant créer un **Random Forest Classifier** et un **Grid Search**. Rappelons que le Grid Search effectuera une validation croisée k-fold pour nous. Nous définissons cv=5 pour une validation croisée 5-fold.

rf = RandomForestClassifier()
gs = GridSearchCV(rf, param_grid, cv=5)

Maintenant, nous utilisons la **méthode fit()** pour exécuter la recherche sur la grille. Les meilleurs paramètres seront stockés dans l'attribut **best_params_**.

gs.fit(X, y)
print("best params:", gs.best_params_) $\Longrightarrow$ 50

# La fonction GridSearchCV() 3

Ce sont les paramètres qui donnent la plus grande accuracy car c'est la métrique par défaut. Notez que vous pouvez obtenir des résultats légèrement différents à chaque fois que vous exécutez cette méthode, car la division aléatoire des 5 folds peut affecter le meilleur score d'accuracy.

**L'accuracy nous convient dans ce cas, car les classes de l'ensemble de données sur le cancer du sein sont raisonnablement équilibrées. Si les classes sont déséquilibrées, nous voudrions utiliser une métrique alternative, comme le score f1.** Nous pouvons changer la métrique en notant le paramètre "f1" comme ci-dessous. Pour éviter d'afficher un meilleur paramètre différent à chaque fois, on peut définir le random_state dans le classifier.

# La fonction GridSearchCV() 4

param_grid = 'n_estimators': [10, 20, 50, 75, 100],
rf = RandomForestClassifier(random_state=3)
gs = GridSearchCV(rf, param_grid, scoring='f1', cv=5)
gs.fit(X, y)
print("best params:", gs.best_params_) $\Longrightarrow$ 20

**Attention !**

Vous pouvez ajouter des paramètres supplémentaires, par exemple max_features, et des valeurs de paramètres au dictionnaire param_grid pour comparer davantage d'arbres de décision.

# Exercice d'application 8

## La fonction GridSearchCV()

Avec la grille de paramètres suivante, combien de modèles Random Forest différents la recherche par grille va-t-elle construire ?

param_grid = { 'max_features': [None, 10, 20], 'n_estimators': [10, 100], }

1. 4
2. 5
3. 6
4. 2
5. 3

# Graphique du Coude 1

Avec un paramètre tel que le nombre d'arbres dans une forêt aléatoire, l'augmentation du nombre d'arbres n'affectera jamais les performances. L'augmentation du nombre d'arbres augmentera les performances jusqu'à un point où elles se stabiliseront. Cependant, plus il y a d'arbres, plus l'algorithme est complexe. Un algorithme plus complexe est plus gourmand en ressources. En général, il vaut la peine d'ajouter de la complexité au modèle si cela améliore les performances, mais nous ne voulons pas ajouter inutilement de la complexité.
Nous pouvons utiliser ce que l'on appelle **un Elbow Graph (graphique du coude)** pour trouver le point idéal. **Le graphique coudé est un modèle qui optimise les performances sans ajouter de complexité inutile**.

# Graphique du Coude 2

Pour trouver la valeur optimale, faisons une recherche par grille en essayant toutes les valeurs de 1 à 100 pour n_estimators.

```
n_estimators = list(range(1, 101))
param_grid = { 'n_estimators': n_estimators, }
rf = RandomForestClassifier()
gs = GridSearchCV(rf, param_grid, cv=5)
gs.fit(X, y)
```

# Graphique du Coude 3

Au lieu de ne regarder que les meilleurs paramètres comme nous l'avons fait auparavant, nous allons utiliser l'ensemble des résultats de la grille de recherche. Les valeurs sont situées dans l'attribut **cv_results_**. Il s'agit d'un **dictionnaire** contenant de nombreuses données, mais nous n'aurons besoin que d'une seule des clés : **mean_test_score**. Extrayons ces valeurs et stockons-les dans une variable :

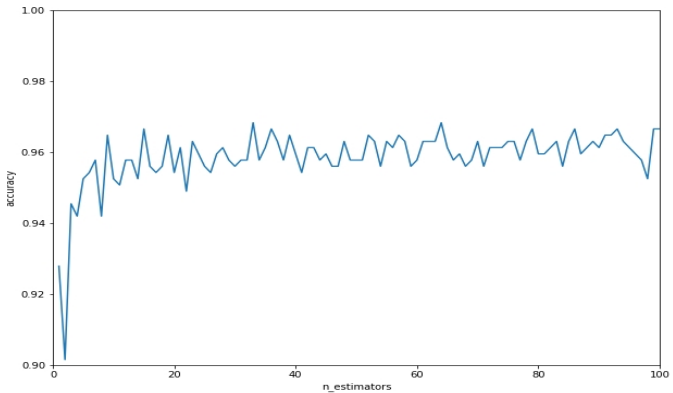
scores = gs.cv_results_['mean_test_score'] $\Longrightarrow$ [0.91564148, 0.90685413, ...]

# Graphique du Coude 4

Utilisons maintenant matplotlib pour représenter graphiquement les résultats.

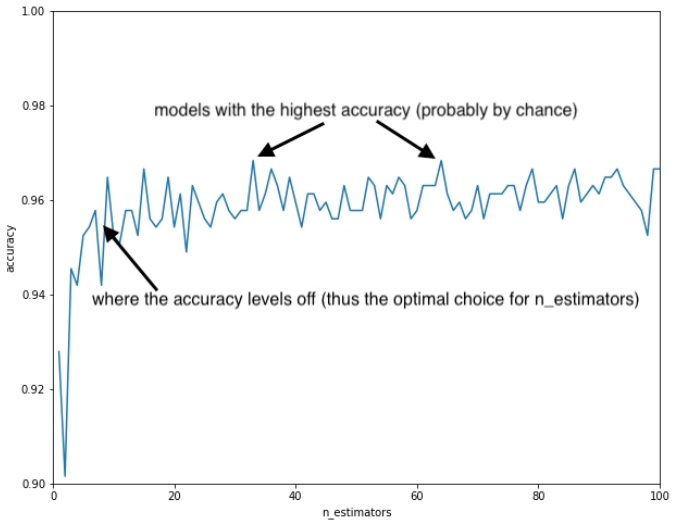
```
import matplotlib.pyplot as plt
scores = gs.cv_results_['mean_test_score']
plt.plot(n_estimators, scores)
plt.xlabel("n_estimators")
plt.ylabel("accuracy")
plt.xlim(0, 100)
plt.ylim(0.9, 1)
plt.show()
```

# Graphique du Coude 5




Si nous examinons ce graphique, nous constatons qu'autour de 10 arbres, le graphique se stabilise. Le meilleur modèle est apparu à n_estimators=33 et n_estimators=64, mais étant donné la volatilité du modèle, cela est probablement dû au hasard. Nous devrions choisir environ 10 comme nombre d'estimateurs, parce que nous voulons le nombre minimum d'estimateurs qui donnent encore une performance maximale.

# Graphique du Coude 6




Nous pouvons maintenant construire notre modèle de forêt aléatoire avec le nombre optimal d'arbres.

rf = RandomForestClassifier(n_estimators=10)
rf.fit(X, y)

# Graphique du Coude 7

**Attention !**

Vous verrez les graphiques en forme de coude apparaître dans de nombreuses situations différentes, lorsque nous ajoutons de la complexité à un modèle et que nous voulons déterminer la quantité minimale de complexité qui permettra d'obtenir des performances optimales.

# Exercice d'application 9

## Graphique du Coude

Si, en observant le graphique du coude, vous constatez que la précision d'un modèle de forêt aléatoire est de 70% avec 5 arbres, de 80% avec 10 arbres, de 90% avec 20 arbres et de 90% avec 30 arbres, combien d'arbres devez-vous utiliser ?

1. 30
2. 5
3. 20
4. 10

# L'importance des variables endogènes 1

Il y a 30 variables endogènes dans l'ensemble de données sur le cancer. Chaque prédicteur contribue-t-il de manière égale à la construction d'un modèle ? Sinon, quel sous-ensemble de prédicteurs doit-on utiliser ? C'est une question **de sélection de prédicteurs**.

# L'importance des variables endogènes 2

Les forêts aléatoires fournissent une méthode simple pour la sélection des prédicteurs : **la diminution moyenne de l'impureté**. Rappelons qu'une forêt aléatoire se compose de nombreux arbres de décision et que, pour chaque arbre, le nœud est choisi pour diviser l'ensemble de données en fonction de la diminution maximale de l'impureté, généralement **l'impureté de Gini ou l'entropie** dans la classification.
Ainsi, pour un arbre, on peut calculer combien d'impureté chaque prédicteur diminue dans un arbre. Puis, pour une forêt, on peut faire la moyenne de la diminution de l'impureté de chaque prédicteur. Considérons cette mesure comme une métrique de l'importance de chaque prédicteur, n**ous pouvons alors classer et sélectionner les prédicteurs en fonction de leur importance**.

# L'importance des variables endogènes 3

Scikit-learn fournit une variable **feature_importances_** avec le modèle, qui indique l'importance relative de chaque prédicteur. Les scores sont réduits de manière à ce que la somme de tous les scores soit égale à 1.
Trouvons les importances des prédicteurs dans une forêt aléatoire avec n_estimateur = 10 en utilisant l'ensemble de données d'entraînement, et affichons-les dans l'ordre décroissant :

```
rf = RandomForestClassifier(n_estimators=10, random_state=3)
rf.fit(X_train, y_train)
ft_imp = pd.Series(rf.feature_importances_,
index=cancer_data.feature_names).sort_values(ascending=False)
print(ft_imp.head(10))
```

# L'importance des variables endogènes 4

A partir des résultats, nous pouvons voir que parmi toutes les caractéristiques, le **worst radius est le plus important (0.31)**, suivi par les **mean concave points** et les **worst concave points**.

**Attention !**

Dans la régression, nous calculons **l'importance des variables** en utilisant plutôt la variance.

# Exercice d'application 10

## L'importance des variables endogènes

Pour calculer l'importance des variables dans les problèmes de classification, nous pouvons utiliser : (Sélectionnez toutes les réponses correctes)

1. L'impureté de Gini
2. La variance
3. L'entropie

# Nouveau modèle sur les variables sélectionnées 1

Pourquoi devrions-nous **effectuer une sélection de variables** ? Les principales raisons sont les suivantes : **elle nous permet d'entraîner un modèle plus rapidement** ; **elle réduit la complexité d'un modèle**, ce qui le rend plus facile à interpréter. Et si **le bon sous-ensemble est choisi, il peut améliorer l'accuracy d'un modèle**. Le choix du bon sous-ensemble dépend souvent de la connaissance du domaine, d'un peu d'art et d'un peu de chance.

Dans notre jeu de données, nous avons remarqué que les variables avec le "worst" semblent avoir une plus grande importance. Par conséquent, nous allons construire un nouveau modèle avec les variables sélectionnées et voir si cela améliore l'accuracy. Rappelez-vous le modèle de la dernière section.

# Nouveau modèle sur les variables sélectionnées 2

rf = RandomForestClassifier(n_estimators=10, random_state=3)
rf.fit(X_train, y_train)
print(rf.score(X_test, y_test)) $\Longrightarrow$ 0.9650

Nous trouvons d'abord les variables dont le nom contient le mot "worst" :
worst_cols = [col for col in df.columns if 'worst' in col]
print(worst_cols)

# Nouveau modèle sur les variables sélectionnées 3

Il y a dix variables de ce type. Nous créons maintenant un autre dataframe avec les variables sélectionnées, suivi d'un split des données entre le test et l'entraînement avec le même état aléatoire

```
X_worst = df[worst_cols]
X_train, X_test, y_train, y_test = train_test_split(X_worst, y,
random_state=3)
```

Entraînons le modèle et calculons l'accuracy :

```
rf.fit(X_train, y_train)
print(rf.score(X_test, y_test)) ⟹ 0.9720
```

# Nouveau modèle sur les variables sélectionnées 4

Nous sommes en mesure d'améliorer l'accuracy en utilisant un sous-ensemble de variables, un tiers du total des variables pour être exact. Cela s'explique par le fait que nous avons supprimé une partie du bruit et des variables fortement corrélées, ce qui a permis d'améliorer l'accuracy. L'avantage de construire un meilleur modèle en utilisant moins de variables sera plus prononcé lorsque la taille de l'échantillon est importante.

**Attention !**

Il n'existe pas de meilleure méthode de sélection des variables, du moins pas de manière universelle. Nous devons plutôt découvrir ce qui fonctionne le mieux pour un problème spécifique et tirer parti de l'expertise du domaine pour construire un bon modèle.
Scikit-learn offre un moyen facile de découvrir l'importance des variables.

# Exercice d'application 11

## Nouveau modèle sur les variables sélectionnées

Un classificateur construit avec toutes les variables est toujours plus performant que celui qui utilise un sous-ensemble des variables.

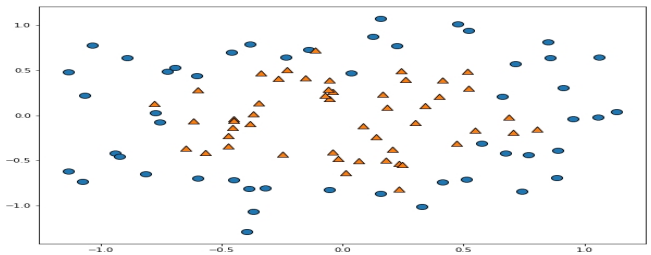
1. Faux
2. Vrai

# Performance 1

Le plus grand **avantage des forêts aléatoires** est probablement **qu'elles sont généralement performantes sans aucun réglage**. **Ils sont également assez performants sur presque tous les ensembles de données**.

Un modèle linéaire, par exemple, ne peut pas être performant sur un ensemble de données qui ne peut pas être divisé par une ligne. Il n'est pas possible de diviser l'ensemble de données ci-dessous avec une ligne sans manipuler les variables. Cependant, une forêt aléatoire sera très performante sur cet ensemble de données.

# Performance 2



Nous pouvons le constater en examinant le code permettant de générer le faux ensemble de données ci-dessus et en comparant un modèle de régression logistique avec un modèle Random Forest. La fonction **make_circles() crée un ensemble de données de classification avec des cercles concentriques**. Nous utilisons la validation croisée kfold pour comparer les scores d'accuracy et nous constatons que le modèle de régression logistique est moins performant que l'estimation aléatoire, mais que le modèle Random Forest est assez performant.

# Performance 3

```
from sklearn.datasets import make_circles
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import numpy as np
X, y = make_circles(noise=0.2, factor=0.5, random_state=3)
kf = KFold(n_splits=5, shuffle=True, random_state=3)
lr_scores = [ ]
rf_scores = [ ]
```

# Performance 4

```
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    lr = LogisticRegression(solver='lbfgs')
    lr.fit(X_train, y_train)
    lr_scores.append(lr.score(X_test, y_test))
    rf = RandomForestClassifier(n_estimators=100)
    rf.fit(X_train, y_train)
    rf_scores.append(rf.score(X_test, y_test))
print("LR accuracy:", np.mean(lr_scores)) ⟹ 0.36
print("RF accuracy:", np.mean(rf_scores)) ⟹ 0.82
```

# Performance 5

**Attention !**

Lorsque l'on cherche à obtenir un point de référence pour un nouveau problème de classification, il est courant de commencer par construire un modèle de régression logistique et un modèle Random Forest, car ces deux modèles ont le potentiel d'être performants sans aucun réglage. Vous obtiendrez ainsi des valeurs pour vos paramètres à essayer de battre. Souvent, il est presque impossible de faire mieux que ces valeurs de référence trouvés.

# Exercice d'application 12



## Performance

Les forêts aléatoires nécessitent beaucoup de réglages pour obtenir des performances décentes :

1. Vrai
2. Faux

# Interprétation 1

Les **forêts aléatoires**, bien qu'elles soient composées d'arbres décisionnels, **ne sont pas faciles à interpréter**. Une forêt aléatoire comporte plusieurs arbres de décision, dont chacun n'est pas un très bon modèle, mais qui, lorsqu'on en fait la moyenne, créent un excellent modèle. Les forêts aléatoires ne sont donc pas un bon choix lorsqu'on recherche la facilité d'interprétation.

**Attention !**

Dans la plupart des cas, l'interprétation n'est pas importante.

# Exercice d'application 13



## Interprétation

Les forêts aléatoires, puisqu'elles sont constituées d'arbres de décision, sont faciles à interpréter :

1. Faux
2. Vrai

# Quantité de calculs 1

Les forêts aléatoires peuvent être un peu lentes à construire, surtout si la forêt aléatoire contient beaucoup d'arbres. La construction d'une forêt aléatoire implique la construction de 10 à 100 (généralement) arbres de décision. Chacun des arbres de décision est plus rapide à construire qu'un arbre de décision standard, car nous ne comparons pas chaque variable à chaque fractionnement, mais étant donné la quantité d'arbres de décision, la construction est souvent lente.

# Quantité de calculs 2

De même, la prédiction avec une forêt aléatoire sera plus lente qu'avec un arbre de décision, car nous devons effectuer une prédiction avec chacun des 10-100 arbres de décision afin d'obtenir notre prédiction finale.

**Attention !**

Les forêts aléatoires ne sont pas le modèle le plus rapide, mais ce n'est généralement pas un problème puisque la puissance de calcul des ordinateurs est considérable.

# Exercice d'application 14



## Interprétation

Les forêts aléatoires sont gourmandes en ressources informatiques car elles sont composées de nombreux modèles.:

1. Faux
2. Vrai

# Quiz 1

## 1) Random Forests 1

Une forêt aléatoire est une collection d'arbres de décision. En quoi ces arbres de décision sont-ils différents d'un arbre de décision standard construit sur l'ensemble d'apprentissage ?

1. ils sont construits chacun sur un échantillon boostrapped
2. ils sont construits chacun sur un petit jeu de données
3. A chaque split dans l'arbre de décision, un sous-ensemble aléatoire des variables est évalué
4. Chaque arbre de décision a seulement un sous-ensemble des variables

# Quiz 2

## 2) Random Forests 2

Quel paramètre utilise-t-on pour contrôler le nombre d'arbres dans un RandomForestClassifier ?

1. n_trees
2. num_estimators
3. num_trees
4. n_estimators

# Quiz 3

## 3) Random Forests 3

Ajouter plus d'arbres à une forêt aléatoire peut rendre le modèle moins performant :

1. Faux
2. Vrai

# Quiz 4

## 4) Random Forests 4

Quel est le plus grand avantage des forêts aléatoires ?

1. Ils ont généralement de bonnes performances sans aucun réglage
2. Ils sont rapides à entraîner
3. Ils sont faciles à interpréter

# Quiz 5

## 5) Random Forests 5

Complétez le programme pour construire un modèle de forêt aléatoire avec 100 arbres, le fitter à l'ensemble d'apprentissage (xtrain, ytrain) et trouver le score d'accuracy sur l'ensemble de test (xtest, ytest).

rf = RandomForestClassifier(n_estimators= ..... )
rf. ..... (xtrain, ytrain)
print(rf. ..... (xtest, ytest))

# Quiz 6

## 6) Random Forests 6

Complétez le programme pour utiliser Grid Search afin de trouver les meilleurs paramètres pour un modèle Random Forest. Considérez [10, 50, 100] pour le nombre d'arbres et [3, 5] pour le max features.

```
param_grid = {      ..... : [10, 50, 100],      ..... : [3, 5], }
rf = RandomForestClassifier()
gs = GridSearchCV(rf, ..... , cv=5)
gs.fit(X, y)
print("best params:", gs.best_params_)
```

# Machine Learning (ML)

## Chap 6 : Neural Networks (les réseaux de neurones)

## Partie I

David TCHOUTA

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

April 13, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Introduction
3. Multi-Layer Perceptron
4. Training
5. Sklearn

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue les réseaux de neurones (Multi-Layer Perceptron, entraînement, etc.) à la fois sur le plan théorique et sur le plan pratique avec Scikit-learn.

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

- de créer et de manipuler les données artificielles
- de comprendre comment fonctionne les réseaux de neurones
- de manipuler un perceptron muti-couche (MLP)
- d'entraîner avec les réseaux de neurones

# Cas d'utilisation des réseaux neuronaux 1

Les **réseaux neuronaux (neural networks)** sont des modèles d'apprentissage automatique incroyablement populaires et puissants. Ils sont souvent **performants** dans les cas où nous avons beaucoup de prédicteurs, car **ils effectuent automatiquement l'ingénierie des variables exogènes** sans nécessiter de connaissances du domaine pour les restructurer.

# Cas d'utilisation des réseaux neuronaux 2

Dans ce module, nous allons **utiliser des données d'image**. Comme **chaque pixel de l'image est un prédicteur**, nous pouvons avoir un **ensemble de variables dépendantes très large**. Les réseaux neuronaux sont tous couramment utilisés dans les **données textuelles**, car celles-ci possèdent également un **grand ensemble de variables**. La **reconnaissance vocale** est un autre exemple où les réseaux neuronaux sont meilleurs.

**Attention !**

Les réseaux neuronaux fonctionnent souvent bien sans qu'il soit nécessaire d'utiliser la connaissance du domaine pour effectuer une quelconque ingénierie des variables dépendantes.

# Exercice d'application 1

## Cas d'utilisation des réseaux neuronaux

Les réseaux neuronaux fonctionnent souvent bien dans les domaines comportant de grands ensembles de variables non structurés :

1. Faux
2. Vrai

# Réseaux neuronaux biologiques

Un terme plus précis pour les réseaux neuronaux est celui de **réseaux neuronaux artificiels (Artificial Neural Networks or ANN)**. Ils ont été inspirés par le fonctionnement des réseaux neuronaux biologiques dans le cerveau humain.

**Le réseau neuronal d'un cerveau** est composé d'environ **86 milliards de neurones**. Les neurones sont connectés par ce qu'on appelle des **synapses**. Il y a environ 100 trillions de synapses dans le cerveau humain. Les neurones s'envoient des signaux par l'intermédiaire des synapses.

# Exercice d'application 2

## Réseaux neuronaux biologiques

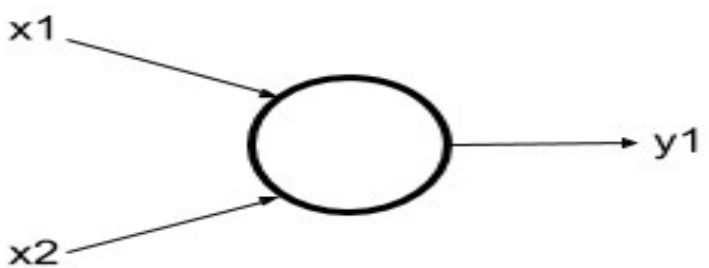
Un réseau neuronal est composé de neurones qui sont connectés par quoi ?

1. les signaux
2. les synapses
3. les terminaisons nerveuses

# Définition 1

Un **neurone artificiel (souvent appelé nœud) est modelé sur un neurone biologique**. C'est un objet simple qui peut recevoir des entrées (inputs), effectuer des calculs avec ces entrées et produire une sortie (outputs).

Nous représentons visuellement les neurones comme suit. x1 et x2 sont les inputs. À l'intérieur du neurone, un calcul est effectué sur la base de x1 et x2 pour produire l'output y1.



# Définition 2

Les neurones peuvent recevoir un nombre quelconque d'inputs et produire un nombre quelconque de outputs.

**Attention !**

Chaque neurone n'est capable que d'un petit calcul, mais lorsqu'ils travaillent ensemble, ils deviennent capables de résoudre des problèmes importants et compliqués.

# Exercice d'application 3

## Définition

Un neurone dans un réseau neuronal artificiel est également appelé un :

.....

# Calculs effectués par des neurones 1

À l'intérieur du neurone, pour effectuer le calcul qui produira l'output, nous mettons d'abord les inputs dans l'équation suivante (comme dans la régression logistique) :

$w_1 x_1 + w_2 x_2 + b$

Rappelons que x1 et x2 sont les inputs. Dans la **régression logistique, les valeurs w1, w2 et b sont appelées coefficients**. Dans les **réseaux neuronaux**, nous appelons **w1 et w2 les poids, et b le biais**.

# Calculs effectués par des neurones 2

Nous introduisons cette valeur dans ce que l'on appelle une **fonction d'activation**. Le résultat de l'équation ci-dessus peut être un nombre réel quelconque. **La fonction d'activation le condense dans une plage fixe (souvent entre 0 et 1)**.

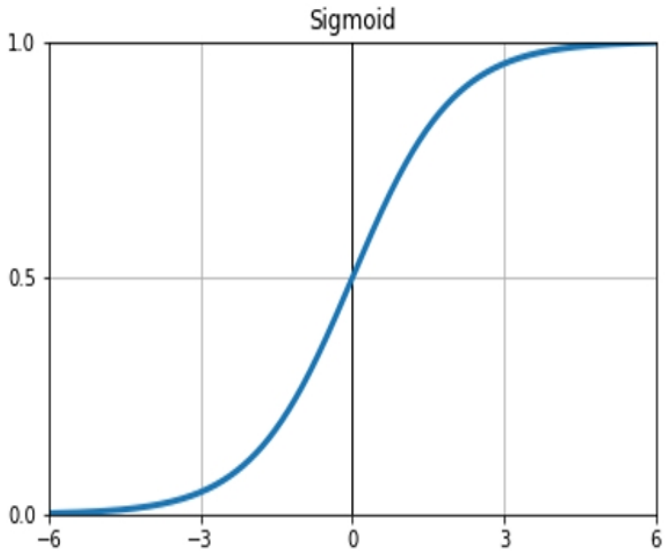
**Une fonction d'activation** couramment utilisée est **la fonction sigmoïde**, la même que celle utilisée dans la régression logistique. Rappelons que cette fonction produit une valeur comprise entre 0 et 1.

# Calculs effectués par des neurones 3

Elle est définie comme suit :

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

La fonction sigmoïde a la forme suivante :



# Calculs effectués par des neurones 4

Pour obtenir la sortie à partir des entrées, nous effectuons le calcul suivant. Les poids, w1 et w2, et le biais, b, contrôlent ce que fait le neurone. Nous appelons ces valeurs (w1, w2, b) les paramètres. La fonction f est la fonction d'activation (dans ce cas, la fonction sigmoïde). La valeur y est la sortie du neurone :

$$y = f(w_1 x_1 + w_2 x_2 + b) = \frac{1}{1+e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

**Attention !**

Cette fonction peut être généralisée pour avoir un nombre quelconque d'entrées (xi) et donc le nombre correspondant de poids (wi).

# Exercice d'application 4

## Calculs effectués par des neurones

D'après l'équation suivante, que représente "f" ?

1. la fonction d'activation
2. l'output
3. les poids
4. le biais

# Autres fonctions d'activation

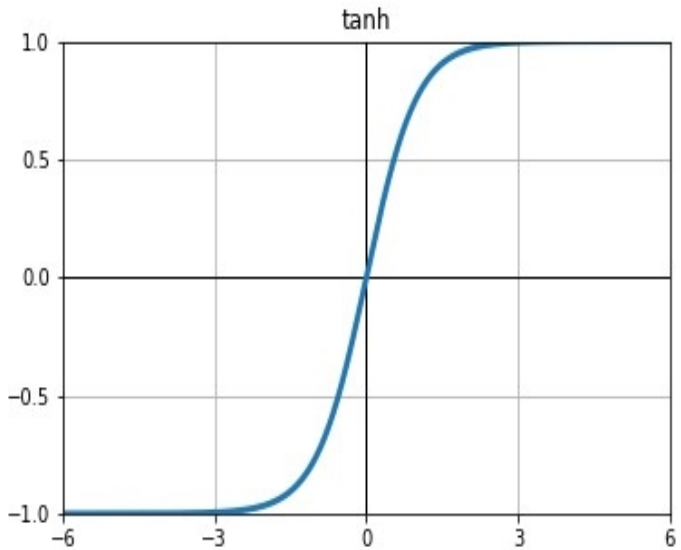
Il existe trois fonctions d'activation couramment utilisées : **sigmoïde (de la partie précédente), tanh et ReLU**.

Tanh a une forme similaire à celle de la sigmoïde, mais va de -1 à 1 au lieu de 0 à 1. **Tanh est la fonction tan hyperbolique** et est définie de la manière suivante :

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

# Graphique de la fonction tanh

Ci-dessous le graphique de la fonction tanh :



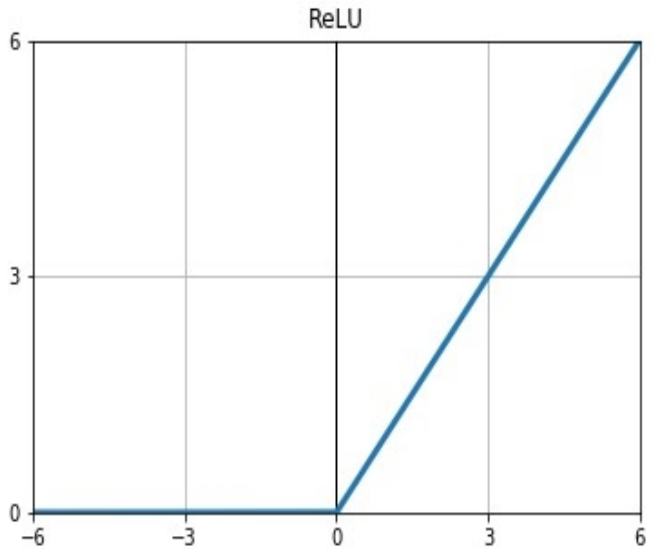
# La fonction ReLU

**ReLU est l'abréviation de Rectified Linear Unit (unité linéaire rectifiée)**.
C'est **la fonction d'identité pour les nombres positifs et elle renvoie les nombres négatifs à 0**.

Ci-dessous l'équation et le graphique.

$$ReLU = \begin{cases} 0 & \text{si x} <= 0 \\ x & \text{si x} > 0 \end{cases}$$

# Graphique de la fonction ReLU




**Attention !**

N'importe laquelle de ces fonctions d'activation fera l'affaire. Pour choisir laquelle utiliser dépendra des spécificités des données. En pratique, nous déterminons laquelle utiliser en comparant les performances de différents réseaux neuronaux.

# Exercice d'application 5

## Autres fonctions d'activation

Lesquelles des fonctions suivantes sont des fonctions d'activation couramment utilisées ? Sélectionnez toutes les bonnes réponses

1. ReLU
2. Sigmoid
3. Tanh
4. Log
5. Sine

# Exemple 1

Supposons que nous ayons un neurone qui prend 2 inputs et produit 1 output et dont la fonction d'activation est la sigmoïde. Les paramètres sont les suivants :

Pondérations (w1, w2) = [0, 1]
Biais (b) = 2

# Exemple 2

Si nous donnons au neurone un input (1, 2), nous obtenons le calcul suivant :

$$\begin{aligned} w_1 x_1 + w_2 x_2 + b &= 0 * 1 + 1 * 2 + 2 \\ &= 0 + 2 + 2 \\ &= 4 \\ y &= f(w_1 x_1 + w_2 x_2 + b) \\ y &= \frac{1}{1 + e^{-4}} \\ &= 0.982013 \end{aligned}$$

**Le neurone renvoie un output de 0.9820**

# Exemple 3

Alternativement, si nous donnons au neurone l'input (2, -2), nous obtenons le calcul suivant :

$$\begin{aligned} w_1x_1 + w_2x_2 + b &= 0 * 2 + 1 * -2 + 2 \\ &= 0 - 2 + 2 \\ &= 0 \\ y &= f(w_1x_1 + w_2x_2 + b) \\ y &= \frac{1}{1 + e^0} \\ &= 0.5 \end{aligned}$$

**Le neurone renvoie un output de 0.5**

# Exemple 4

**Attention !**

Un neurone seul n'a pas beaucoup de pouvoir, mais lorsque nous construisons un réseau de neurones, nous pouvons voir à quel point ils sont puissants tous ensemble.

# Exercice d'application 6

## Exemple

Nous avons un neurone avec les poids et le biais suivants :

Pondérations (w1, w2) = [0, 1]
Biais (b) = 2

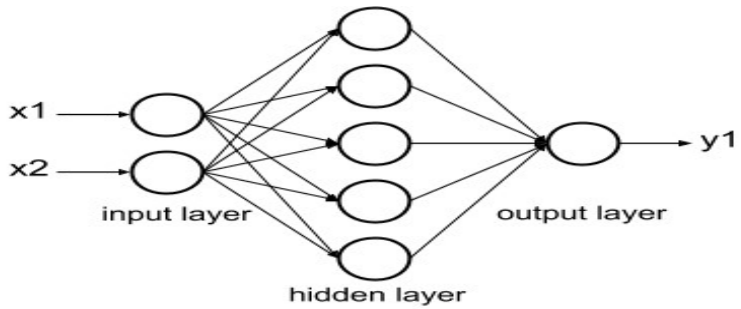
Si f est la fonction d'activation et y est la sortie, laquelle des équations suivantes est la bonne pour le point (2, 1).

1. f(2)
2. f(3)
3. f(5)
4. f(1)

# Perceptron multicouche (Multi-Layer Perceptron) 1

Pour **créer un réseau neuronal**, nous **combinons des neurones** entre eux de sorte **que les sorties de certains neurones soient les entrées d'autres neurones**. Nous travaillerons avec **des réseaux neuronaux à action directe**, ce qui signifie que les neurones n'envoient des **signaux que dans une seule direction**. En particulier, nous travaillerons avec ce que l'on appelle un **perceptron multicouche (MLP pour Multi-Layer Perceptron)**. Ce réseau neuronal comporte plusieurs couches, représentées ci-dessous.

# Perceptron multicouche (Multi-Layer Perceptron) 2




**Un perceptron multicouche** comporte **toujours une couche d'entrée**, avec un neurone (ou nœud) pour chaque entrée. Dans le réseau neuronal ci-dessus, il y a deux entrées et donc deux nœuds d'entrée. Il aura **une couche de sortie**, avec un nœud pour chaque sortie. Dans le réseau ci-dessus, il y a un nœud de sortie pour une seule valeur de sortie. Il peut avoir **un nombre quelconque de couches cachées et chaque couche cachée peut avoir un nombre quelconque de nœuds**. Ci-dessus, **il y a une couche cachée avec 5 noeuds**.

# Perceptron multicouche (Multi-Layer Perceptron) 3

Les nœuds de la couche d'entrée prennent une seule valeur d'entrée et la transmettent. **Les nœuds des couches cachées et de la couche de sortie peuvent prendre plusieurs entrées, mais ils produisent toujours une seule sortie**. Parfois, les nœuds doivent transmettre leur sortie à plusieurs nœuds. Dans l'exemple ci-dessus, **les nœuds de la couche d'entrée transmettent leur sortie à chacun des cinq nœuds de la couche cachée**.

**Attention !**

Un perceptron monocouche est un réseau neuronal sans couche cachée. Ces réseaux sont rarement utilisés. La plupart des réseaux neuronaux sont des perceptrons multicouches, généralement avec une ou deux couches cachées.

# Exercice d'application 7

## Perceptron multicouche

Les données peuvent circuler de la couche cachée à la couche d'entrée dans un perceptron multicouche.

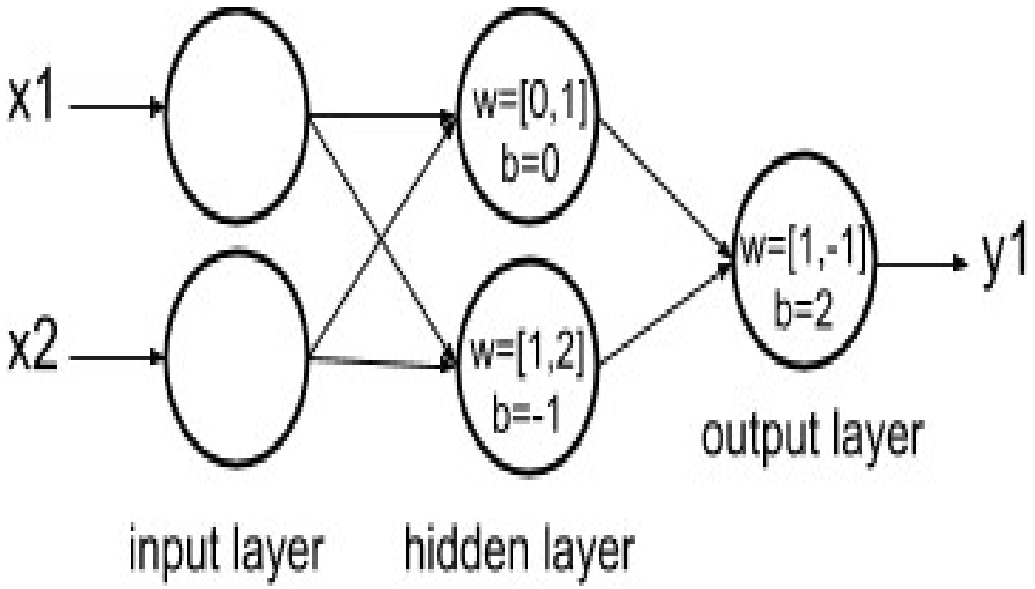
1. Faux
2. Vrai

# Exemple de Perceptron multicouche 1

Voyons plus en détail comment cela fonctionne à l'aide d'un exemple. Un réseau neuronal qui résout un problème réel sera trop volumineux pour être interprété. Nous allons donc examiner un exemple simple.

Nous avons **un réseau neuronal avec deux entrées, une seule couche cachée avec deux nœuds et une sortie**. Les poids et le biais sont indiqués dans les nœuds ci-dessous. Tous les nœuds utilisent la **fonction d'activation sigmoïde**.

# Exemple de Perceptron multicouche 2



# Exemple de Perceptron multicouche 3

Voyons quelle sortie nous obtenons pour l'entrée (3,2)
Ci-dessous la sortie pour le premier nœud de la couche cachée :

$$\begin{aligned} h_1 &= f(0 * x_1 + 1 * x_2 + 0) \\ &= f(0 * 3 + 1 * 2 + 0) \\ &= f(2) \\ &= \frac{1}{1 + e^{-2}} \\ &= 0.88079 \end{aligned}$$

# Exemple de Perceptron multicouche 4

Ci-dessous la sortie pour le deuxième nœud de la couche cachée :

$$\begin{aligned} h_2 &= f(1 * x_1 + 2 * x_2 - 1) \\ &= f(1 * 3 + 2 * 2 - 1) \\ &= f(6) \\ &= \frac{1}{1 + e^{-6}} \\ &= 0.99752 \end{aligned}$$

# Exemple de Perceptron multicouche 5

Ci-dessous la sortie du nœud de la couche de sortie. Notez que ce nœud prend les sorties de la couche cachée comme entrée :

$$\begin{aligned} y_1 &= f(1 * h_1 - 1 * h_2 + 2) \\ &= f(1 * 0.88079 - 1 * 0.99752 + 2) \\ &= f(1.88327) \\ &= \frac{1}{1 + e^{-1.88327}} \\ &= 0.86798 \end{aligned}$$

**Par conséquent, pour l'entrée (3, 2), la sortie du réseau neuronal est de 0.86798**

# Exemple de Perceptron multicouche 6

**Attention !**

Pour modifier les performances du réseau neuronal, nous pouvons modifier les valeurs de pondération et de biais.

# Exercice d'application 8

## Exemple de Perceptron multicouche

Quelles sont les entrées de la couche de sortie ?

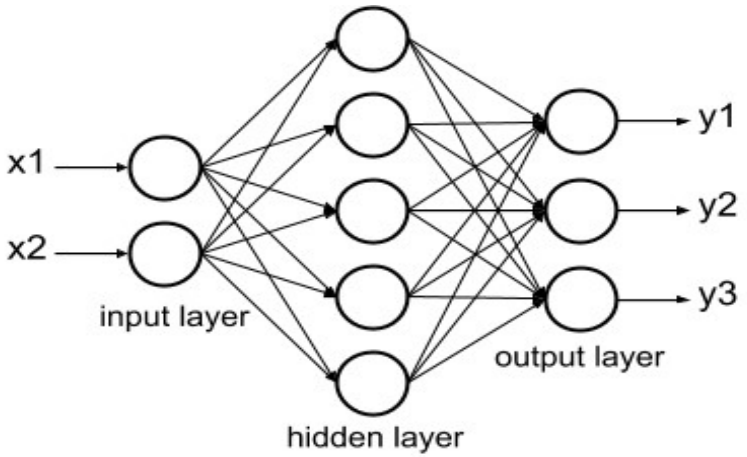
1. Les entrées de la couche cachée
2. Les sorties de la couche cachée
3. Entrées du réseau neuronal

# Plus de 2 variables cibles 1

**L'avantage d'un classificateur MLP** est qu'il s'étend facilement aux problèmes qui ont **plus de deux valeurs cibles**. Dans les modules précédents, nous nous sommes occupés de prédire 0 ou 1 (vrai ou faux, survivant ou non, cancéreux ou non, ). Dans certains cas, nous devrons choisir parmi **3 sorties possibles ou plus. Un réseau neuronal fait cela naturellement. Il suffit d'ajouter des nœuds supplémentaires à la couche de sortie**.

Par exemple, si nous essayons de prédire si une image est un oiseau, un chat ou un chien, nous aurons trois nœuds de sortie. Le premier (y1) mesure si l'image est un oiseau, le deuxième (y2) mesure si l'image est un chat, et le troisième (y3) mesure si l'image est un chien. **Le modèle choisit la sortie ayant la valeur la plus élevée**.

# Plus de 2 variables cibles 2




Par exemple, pour une entrée d'image donnée, les sorties du réseau neuronal y1=0,3, y2=0,2 et y3=0,5, le modèle détermine alors que l'image contient un chien (y3).

**Attention !**

Nous pouvons utiliser n'importe quel classificateur pour un problème multi-classes, mais les réseaux neuronaux se généralisent naturellement.

# Exercice d'application 9

## Plus de 2 variables cibles

Un réseau neuronal peut avoir un nombre quelconque de sorties :

1. Faux
2. Vrai
3. ça depend

# La fonction Loss 1

Pour **entraîner** un réseau neuronal, il faut **définir une fonction de perte**. **Il s'agit d'une mesure de la distance qui sépare notre réseau neuronal de la perfection**. Lorsque nous entraînons le réseau neuronal, **nous optimisons une fonction de perte**.

Nous utiliserons **l'entropie croisée (cross entropy)** comme **fonction de perte**. Cette fonction est identique à la vraisemblance utilisée dans la régression logistique, mais elle porte un nom différent dans ce contexte. Nous calculons l'entropie croisée de la manière suivante :

$$\text{cross entropy} = \begin{cases} p & \text{si y} = 1 \\ 1 - p & \text{si y} = 0 \end{cases}$$

# La fonction Loss 2

Nous multiplions entre elles les valeurs d'entropie croisée pour tous les points de données.

Suposons que nous avons deux modèles à comparer sur un petit ensemble de données avec 4 points de données. Ci-dessous un tableau des valeurs réelles, des probabilités prédites pour le modèle 1 et des probabilités prédites pour le modèle 2 :

| **Variable cible** | **Prédiction Model 2** | **Prédiction Model 2** |
|---|---|---|
| 1 | 0.6 | 0.5 |
| 1 | 0.8 | 0.9 |
| 0 | 0.3 | 0.1 |
| 0 | 0.4 | 0.5 |

# La fonction Loss 3

L'entropie croisée du modèle 1 est la suivante :
$0.6 * 0.8 * (1 - 0.3) * (1 - 0.4) = 0.2016$

L'entropie croisée du modèle 2 est la suivante :
$0.5 * 0.9 * (1 - 0.1) * (1 - 0.5) = \mathbf{0.2025}$

L'entropie croisée sera d'autant plus élevée que le modèle est bon. Ainsi, **puisque le modèle 2 a une entropie croisée plus élevée que le modèle 1, il est le meilleur modèle**.

**Attention !**

Tout comme nous l'avons fait avec la fonction de vraisemblance dans la régression logistique, nous utilisons la fonction de perte pour trouver le meilleur modèle possible.

# Exercice d'application 10

## La fonction Loss

Sur la base des valeurs cibles et prédictives suivantes, quelle serait l'équation correcte pour calculer l'entropie croisée (en conservant les valeurs dans le même ordre) ?

Valeurs cibles : 0, 0, 1, 1

Prédictions : 0.6, 0.3, 0.6, 0.8

1. 0.0 * 0.9 * 1.0 * 1.4
2. 0.4 * 0.7 * 0.6 * 0.8
3. 0.0 * 1.0 * 0.9 * 1.4

# Backpropagation (rétropropagation) 1

Un réseau neuronal a beaucoup de paramètres que nous pouvons contrôler. Il y a plusieurs coefficients pour chaque nœud et il peut y avoir beaucoup de nœuds ! Le processus de mise à jour de ces valeurs pour converger vers le meilleur modèle possible est assez compliqué. **Le réseau neuronal travaille à rebours à partir du nœud de sortie, en actualisant de manière itérative les coefficients des nœuds**. Ce processus de **retour en arrière dans le réseau neuronal est appelé rétropropagation ou backprop**.

# Backpropagation 2

Nous n'allons pas entrer dans tous les détails ici car il implique le calcul de dérivées partielles, mais **l'idée est que nous initialisons toutes les valeurs des coefficients et que nous les modifions de manière itérative afin de constater à chaque itération une amélioration de la fonction de perte**. Finalement, nous **ne pouvons plus améliorer la fonction de perte** et nous avons alors trouvé **notre modèle optimal**.

**Attention !**

Avant de créer un réseau neuronal, nous fixons le nombre de nœuds et le nombre de couches. Ensuite, nous utilisons le backprop pour mettre à jour de manière itérative toutes les valeurs des coefficients jusqu'à ce que nous convergions vers un réseau neuronal optimal.

# Exercice d'application 11

## Backpropagation

La rétropropagation (Backpropagation) est :

1. Le processus de mise à jour des coefficients dans un réseau neuronal
2. Le processus de détermination de la fonction d'activation dans un réseau neuronal
3. processus d'initialisation des coefficients dans un réseau neuronal

# Création d'un ensemble de données artificielles 1

Parfois, **afin de tester des modèles**, il est utile de **créer un ensemble de données artificielles**. Nous pouvons créer un ensemble de données de la taille et de la complexité nécessaires. Ainsi, nous pouvons **créer un ensemble de données plus facile à utiliser qu'un ensemble de données réelles**. Cela peut nous **aider à comprendre le fonctionnement des modèles** avant de les appliquer à des données désordonnées du monde réel.

# Création d'un ensemble de données artificielles 2

Nous allons utiliser **la fonction make_classification** de scikit-learn. Elle **génère une matrice de variables X et un tableau cible y**. Nous lui donnerons les paramètres suivants :

- n_samples: nombre de points de données
- n_features: nombre de variables
- n_informative: nombre de variables informatives
- n_redundant: nombre de variables redondantes
- random_state: état aléatoire pour garantir le même résultat à chaque fois

Vous pouvez consulter la documentation complète pour voir d'autres paramètres que vous pouvez modifier pour changer le résultat.

# Création d'un ensemble de données artificielles 3

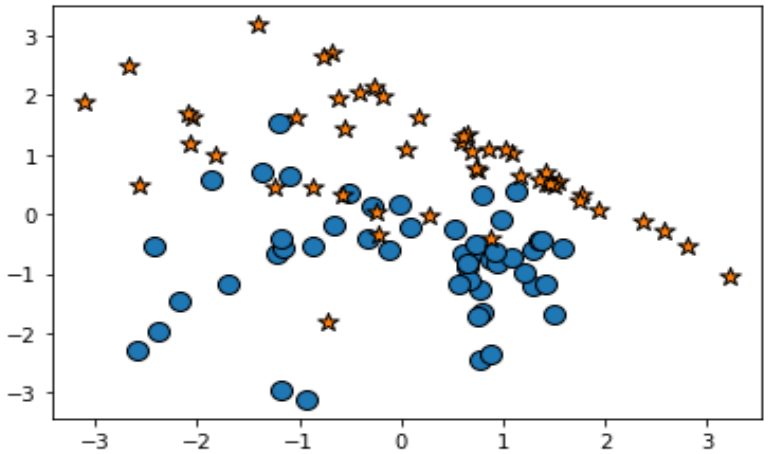
Ci-dessous le code pour générer un jeu de données :

```
from sklearn.datasets import make_classification
X, y = make_classification(n_features=2, n_redundant=0,
n_informative=2, random_state=5)
print(X)
print(y)
```

Ci-dessous le code pour construire le graphe des données afin que nous puissions les observer :

```
from matplotlib import pyplot as plt
plt.scatter(X[y==0][:, 0], X[y==0][:, 1], s=100, edgecolors='k')
plt.scatter(X[y==1][:, 0], X[y==1][:, 1], s=100, edgecolors='k',
marker='*')
plt.show()
```

# Création d'un ensemble de données artificielles 4




**Attention !**

Scikit-learn dispose de quelques fonctions autres que make_classification pour créer des ensembles de données de classification avec différentes propriétés. Regardez make_circles et make_moons si vous voulez jouer avec des ensembles de données plus artificiels.

# Exercice d'application 12

## Création d'un ensemble de données artificielles

Lesquelles des formes suivantes sont les formes correctes pour X et y respectivement après avoir exécuté le code ci-dessous :

X, y = make_classification(n_features=3, n_redundant=1, n_informative=2)

1. (100, 3), (100, 1)
2. (100, 2), (100,)
3. (100, 3), (100,)

# MLPClassifier 1

Scikit-learn possède **une classe MLPClassifier**, qui est un perceptron multicouche pour la classification. La classe peut être importée de scikit-learn, un objet MLPClassifier peut être créé et la méthode fit peut être utilisée pour l'entraînement :

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_features=2, n_redundant=0,
n_informative=2, random_state=5)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=5)
mlp = MLPClassifier()
mlp.fit(X_train, y_train)
```

# MLPClassifier 2

Message d'avertissement :

**ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.**
**% self.max_iter, ConvergenceWarning)**

Vous remarquerez que nous obtenons un ConvergenceWarning. Cela signifie que le réseau neuronal a besoin de plus d'itérations pour converger vers les coefficients optimaux. Le nombre d'itérations par défaut est de 200. Augmentons cette valeur à 1000.

mlp = MLPClassifier(max_iter=1000)

# MLPClassifier 3

Lorsque nous exécutons ce code, le réseau neuronal converge. Nous pouvons maintenant utiliser la méthode des scores pour calculer l'accuracy sur l'ensemble de test :

```
mlp = MLPClassifier(max_iter=1000)
mlp.fit(X_train, y_train)
print("accuracy:", mlp.score(X_test, y_test))
```

**Attention !**

Les réseaux neuronaux sont incroyablement compliqués, mais scikit-learn les rend très faciles à utiliser.

# Exercice d'application 13

## MLPClassifier

Que faire si vous obtenez un avertissement de convergence lors de la formation d'un MLPClassifier ?

1. Augmenter la valeur de max_iter
2. Diminuer la valeur de max_iter

# Paramètres du MLPClassifier 1

Il existe **quelques paramètres** que vous pouvez **être amené à modifier dans le MLPClassifier**.

Vous pouvez configurer **le nombre de couches cachées** et le **nombre de nœuds dans chaque couche**. Le MLPClassifier par défaut aura une seule couche cachée de 100 noeuds. Cela fonctionne souvent très bien, mais nous pouvons expérimenter avec différentes valeurs. Ceci créera un MLPCLassifier avec deux couches cachées, une de 100 nœuds et une de 50 nœuds.

mlp = MLPClassifier(max_iter=1000, hidden_layer_sizes=(100, 50))

# Paramètres du MLPClassifier 2

Nous avons vu **max_iter** dans la partie précédente. Il s'agit du **nombre d'itérations**. **En général, plus vous avez de données, moins vous avez besoin d'itérations pour converger**. Si la valeur est trop grande, l'exécution du code prendra trop de temps. Si la valeur est trop faible, le réseau neuronal ne convergera pas vers la solution optimale.
Il arrive également que l'on doive modifier **alpha**, qui est **la taille du pas**. Il s'agit de **la mesure dans laquelle le réseau neuronal modifie les coefficients à chaque itération**. Si la valeur est trop faible, vous risquez de ne jamais converger vers la solution optimale. Si la valeur est trop grande, vous risquez de manquer la solution optimale. Initialement, vous pouvez laisser cette valeur par défaut. **La valeur par défaut de alpha est de 0,0001**. Notez que **la diminution de alpha nécessite souvent une augmentation de max_iter**.

# Paramètres du MLPClassifier 3

Parfois, vous voudrez **changer le solveur**. Il s'agit de **l'algorithme utilisé pour trouver la solution optimale**. Tous les solveurs fonctionnent, mais il se peut que vous trouviez pour votre ensemble de données qu'un solveur différent trouve la solution optimale plus rapidement. Les **options pour le solveur sont 'lbfgs', 'sgd' et 'adam'**.

Exécutez le code ci-dessous et essayez de modifier les paramètres du MLPClassifier. Le code utilise un random_state pour s'assurer que chaque fois que vous exécutez le code avec les mêmes paramètres, vous obtiendrez le même résultat.

# Paramètres du MLPClassifier 4

```
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
X, y = make_classification(n_features=2, n_redundant=0,
n_informative=2, random_state=5)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=5)
mlp = MLPClassifier(max_iter=1000, hidden_layer_sizes=(100,
50), alpha=0.0001, solver='adam', random_state=3)
mlp.fit(X_train, y_train)
print("accuracy:", mlp.score(X_test, y_test))
```

**Attention !**

Si vous consultez la documentation, vous trouverez plusieurs autres paramètres que vous pouvez modifier dans les réseaux de neurones.

# Exercice d'application 14

## Paramètres du MLPClassifier

Réorganisez le code pour créer un réseau neuronal avec deux couches cachées, la première de taille 10 et la seconde de taille 5.

1. (hidden_layer_sizes = (10,5))
2. from sklearn.neural_network
3. mlp = MLPClassifier
4. import MPLClassifier

# Machine Learning (ML)

## Chap 6 : Neural Networks (les réseaux de neurones)
## Partie II

David TCHOUTA

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

April 13, 2022

*"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison*

# Table de matières

1. Objectifs
2. Prediction
3. MLP Weights
4. Pros/Cons
5. Quiz

# Objectifs du chapitre

Dans ce chapitre, nous allons passer en revue les prédictions et les couches cachées dans un MLP.

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

- de travailler avec l'Open ML
- de capable de réaliser une prédiction avec un MLP
- de calculer les coefficients des couches cachées et des couches de sortie
- de comprendre et d'afficher les couches cachées

# L'ensemble de données MNIST 1

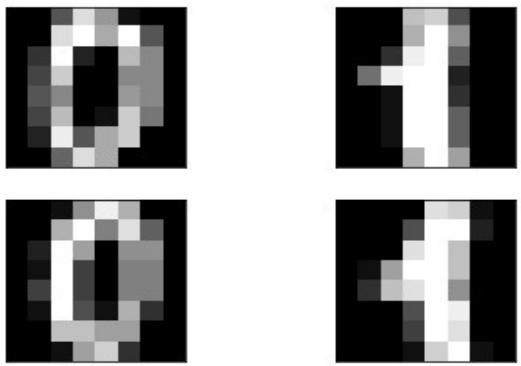
Dans cette section, nous allons travailler avec un nouveau jeu de données, **la base de données MNIST** de chiffres manuscrits. NIST est le National Institute of Standards and Technology et le M signifie Modified.

Il s'agit **d'une base de données d'images de chiffres manuscrits**. Nous allons construire un classifier pour **déterminer quel chiffre se trouve dans l'image.**

Nous commencerons par la version de la base de données MNIST qui est intégrée à Scikit-learn. **Les images ont seulement 8 par 8 pixels, donc elles sont floues**.

# L'ensemble de données MNIST 2

Voici quelques exemples d'images :



Dans scikit-learn, nous pouvons charger le jeu de données en utilisant **la fonction load_digits**. Pour simplifier le problème, nous ne travaillerons initialement qu'avec deux chiffres (0 et 1), nous utilisons donc le paramètre **n_class pour limiter le nombre de valeurs cibles à 2**.

# L'ensemble de données MNIST 3

```
from sklearn.datasets import load_digits
X, y = load_digits(n_class=2, return_X_y=True)
```

Nous pouvons voir les dimensions de X et y et ce à quoi ressemblent les valeurs de la manière suivante :

```
print(X.shape, y.shape)
print(X[0])
print(y[0])
```

# L'ensemble de données MNIST 4

(360, 64) (360,)

[ 0. 0. 5. 13. 9. 1. 0. 0. 0. 0. 13. 15. 10. 15. 5. 0. 0. 3. 15. 2. 0. 11. 8. 0. 0. 4. 12. 0. 0. 8. 8. 0. 0. 5. 8. 0. 0. 9. 8. 0. 0. 4. 11. 0. 1. 12. 7. 0. 0. 2. 14. 5. 10. 12. 0. 0. 0. 0. 6. 13. 10. 0. 0. 0.]

0

Nous voyons que nous avons **300 points de données et que chaque point de données a 64 prédicteurs**. Nous avons 64 prédicteurs parce que l'image est de 8 x 8 pixels et que nous avons 1 prédicteur par pixel. **La valeur est sur une échelle de gris où 0 est noir et 16 est blanc**.

# L'ensemble de données MNIST 5

Pour obtenir une vue plus intuitive du point de données, modifions le tableau pour qu'il soit 8x8 :

print(X[0].reshape(8, 8))

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

# L'ensemble de données MNIST 6

Nous pouvons voir qu'il s'agit d'un 0, mais nous verrons dans la partie suivante que nous pouvons dessiner l'image plus clairement.

**Attention !**

Il existe différentes versions de ce jeu de données avec plus de pixels et avec des couleurs (pas en niveaux de gris). Nous allons voir que même avec ces images simplifiées, nous pouvons construire un bon classificateur.

# Exercice d'application 15

## L'ensemble de données MNIST

Que signifie chaque prédicteur de l'ensemble de données MNIST ?

1. Le degré d'obscurité d'un pixel
2. Le nombre de pixels dans l'image
3. Le nombre que l'image représente

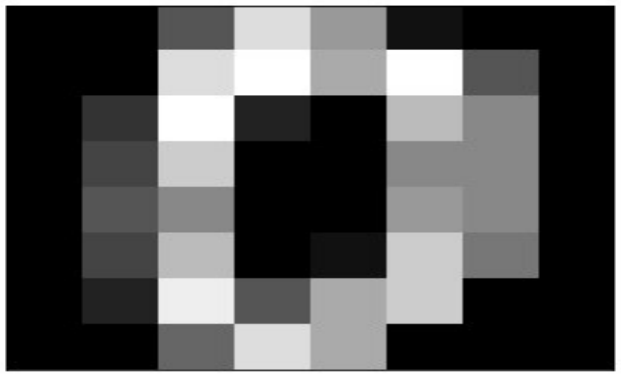
# Afficher l'image des chiffres manuscrits 1

Vous pouvez construire un modèle sans jamais regarder une représentation visuelle des images, mais il peut parfois être utile de dessiner l'image.

Nous utilisons la **fonction matplotlib matshow** pour **dessiner l'image**. Le **paramètre cmap est utilisé pour indiquer que l'image doit être en niveaux de gris plutôt que colorée**.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
X, y = load_digits(n_class=2, return_X_y=True)
plt.matshow(X[0].reshape(8, 8), cmap=plt.cm.gray)
plt.xticks(()) ⟹ remove x tick marks
plt.yticks(()) ⟹ remove y tick marks
plt.show()
```

# Afficher l'image des chiffres manuscrits 2



**Attention !**

Vous pouvez voir qu'avec seulement 64 pixels, l'image est très pixellisée. Même avec ces images floues, nous pouvons construire un excellent modèle.

# Exercice d'application 16

## Afficher l'image des chiffres manuscrits

Quelle fonction de matplotlib utilise-t-on pour dessiner une image à partir de valeurs de pixels ?

1. matshow
2. plot
3. scatter

# MLP sur MNIST Dataset 1

Utilisons maintenant le MLPClassifier pour construire un modèle pour l'ensemble de données MNIST.

Nous allons faire une séparation train/test et entraîner un MLPClassifier sur l'ensemble d'entraînement.

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=5) mlp = MLPClassifier() mlp.fit(X_train, y_train)

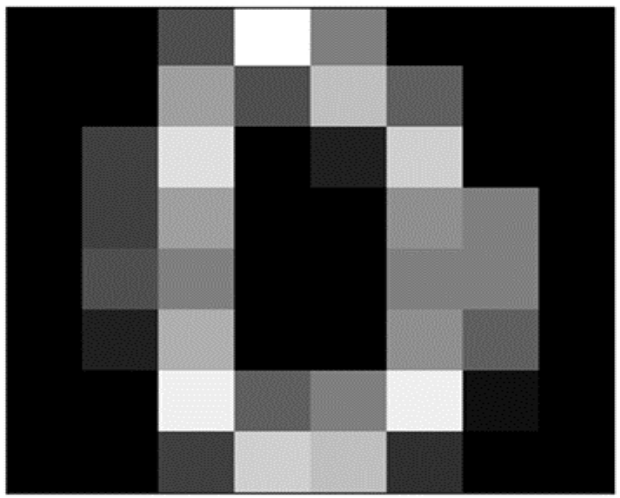
Nous ne recevons pas d'avertissement, donc le nombre d'itérations par défaut est adéquat dans ce cas.

# MLP sur MNIST Dataset 2

Voyons comment le modèle prédit le premier point de données de l'ensemble de test. Nous utilisons matplotlib pour dessiner les images et ensuite montrer la prédiction du modèle :

```
x = X_test[0]
plt.matshow(x.reshape(8, 8), cmap=plt.cm.gray)
plt.xticks(())
plt.yticks(())
plt.show()
print(mlp.predict([x])) ⟹ 0
```

# MLP sur MNIST Dataset 3



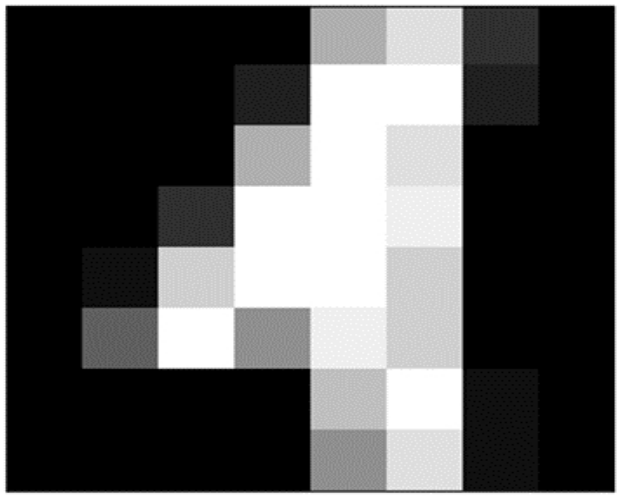
Nous pouvons voir que c'est un 0 et que notre modèle prédit correctement 0.

# MLP sur MNIST Dataset 4

De même, examinons le deuxième point de données :

```
x = X_test[1]
plt.matshow(x.reshape(8, 8), cmap=plt.cm.gray)
plt.xticks(())
plt.yticks(())
plt.show()
print(mlp.predict([x])) ⟹ 1
```

# MLP sur MNIST Dataset 5



Il s'agit clairement d'un 1 et notre modèle obtient à nouveau une prédiction correcte.

# MLP sur MNIST Dataset 6

Nous pouvons également voir l'accuracy du modèle sur l'ensemble du jeu de test :
print(mlp.score(X_test, y_test)) $\Longrightarrow$ 1

Nous pouvons également voir la précision du modèle sur l'ensemble du jeu de test. Ainsi, notre modèle obtient une précision de 100%.

**Attention !**

0 et 1 sont deux des chiffres les plus faciles à distinguer, mais nous verrons que le modèle peut également être performant pour distinguer des exemples plus difficiles.

# Exercice d'application 17

## MLP sur MNIST Dataset

Réorganisez le code pour charger les deux premiers chiffres, faire un split de données en test et en entraînement, construire un MLPClassifier et ensuite calculer le score d'accuracy.

1. = train_test_split(X, y, random_state=5)
2. print(mlp.score(X_test, y_test))
3. mlp = MLPClassifier().fit(X_train, y_train)
4. X_train, X_test, y_train, y_test
5. X, y = load_digits(n_class=2, return_X_y = True)

# Classer les 10 chiffres 1

Comme les réseaux neuronaux se généralisent facilement pour traiter des sorties multiples, nous pouvons simplement utiliser le même code pour construire un classificateur permettant de distinguer les dix chiffres.

# Classer les 10 chiffres 2

Cette fois, lorsque nous chargeons les chiffres, nous ne limitons pas le nombre de classes :

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
X, y = load_digits(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=2)
mlp = MLPClassifier(random_state=2)
mlp.fit(X_train, y_train)
print(mlp.score(X_test, y_test)) ⟹ 0.96
```

# Classer les 10 chiffres 3

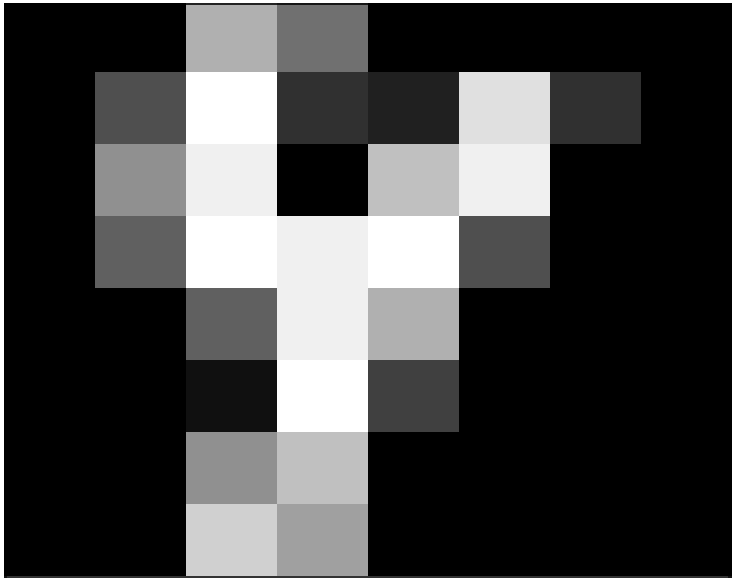
Nous avons donc obtenu 96% de points de données corrects dans l'ensemble de test. Regardons ceux qui sont incorrects. Nous utilisons un masque (sous-ensemble de données respactant une condition) numpy pour extraire uniquement les points de données qui sont incorrects. Récupérons les valeurs x, la vraie valeur y ainsi que la valeur prédite.

```
y_pred = mlp.predict(X_test)
incorrect = X_test[y_pred != y_test]
incorrect_true = y_test[y_pred != y_test]
incorrect_pred = y_pred[y_pred != y_test]
```

# Classer les 10 chiffres 4

Regardons la première image que nous avons mal interprétée et ce que nous avions prévu :

```
j = 0
plt.matshow(incorrect[j].reshape(8, 8), cmap=plt.cm.gray)
plt.xticks(())
plt.yticks(())
plt.show()
```



# Classer les 10 chiffres 4

print("true value:", incorrect_true[j]) $\Longrightarrow$ 4
print("predicted value:", incorrect_pred[j]) $\Longrightarrow$ 9

Vous pouvez voir en regardant l'image qu'un humain pourrait aussi être confus. Il n'est pas évident de savoir si c'est un 4 ou un 9.

**Attention !**

Vous pouvez modifier le code pour voir tous les points de données que le modèle a prédits de manière incorrecte.

# Exercice d'application 18

## Classer les 10 chiffres

Laquelle des propositions suivantes charge les 10 chiffres ?
(Sélectionnez toutes les bonnes réponses)

1. X, y = load_digits(n_class=5, return_X_y = True)
2. X, y = load_digits(n_class=10, return_X_y = True)
3. 1X, y = load_digits(n_class=2, return_X_y = True)
4. X, y = load_digits(return_X_y = True)

# L'Open ML 1

Pour cette section, nous allons utiliser une version plus granulaire de l'ensemble de données MNIST. Au lieu d'utiliser la version de scikit-learn qui comporte des images de 64 pixels, nous utiliserons une version d'Open ML qui comporte 784 pixels (28 x 28).

Open ML dispose d'une base de données de grands ensembles de données qui peuvent être utilisés pour une variété de problèmes d'apprentissage automatique. Scikit-learn dispose d'une **fonction fetch_openml pour télécharger directement des ensembles de données** depuis la base de données Open ML.

# L'Open ML 2

Utilisez le code suivant pour obtenir le jeu de données :

```
from sklearn.datasets import fetch_openml
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

Nous pouvons examiner brièvement la forme des tableaux, la plage des valeurs des prédicteurs et les premières valeurs du tableau cible (variable endogène) pour mieux comprendre l'ensemble de données.

```
print(X.shape, y.shape)
print(np.min(X), np.max(X))
print(y[0:5])
```

# L'Open ML 3

(70000, 784) (70000,)
0.0 255.0
['5' '0' '4' '1' '9']

Nous pouvons voir que nous avons 70 000 points de données avec 784 variables dépendantes. Les valeurs des prédicteurs vont de 0 à 255 (que nous interprétons sur une échelle de gris, 0 étant blanc et 255 étant noir). Les valeurs cibles sont les chiffres 0 à 9. Notez que les valeurs cibles sont stockées sous forme de chaînes de caractères et non d'entiers.

Pour notre exemple, nous n'utiliserons que les chiffres 0-3, nous pouvons donc utiliser le code de la slide suivante pour segmenter cette partie de l'ensemble de données.

# L'Open ML 4

X5 = X[y <= '3']
y5 = y[y <= '3']

Nous allons modifier certains des paramètres par défaut du MLPClassifier pour construire le modèle. **Comme notre objectif est de visualiser les poids de la couche cachée, nous n'utiliserons que 6 nœuds dans la couche cachée afin de pouvoir tous les examiner. Nous utiliserons 'sgd' (stochastic gradient descent) comme solveur, ce qui nous oblige à diminuer alpha (le taux d'apprentissage)**.

# L'Open ML 5

```
mlp=MLPClassifier(
      hidden_layer_sizes=(6,),
      max_iter=200, alpha=1e-4,
      solver='sgd', random_state=2)
mlp.fit(X5, y5)
```

Si nous exécutons ce code, nous verrons qu'il converge.

# Exercice d'application 19

## L'Open ML

Quelles sont les différences entre la version Open ML du jeu de données MNIST que nous utilisons et la version Sklearn ? (Plusieurs réponses possibles)

1. L'Open ML a plus de variables cibles
2. L'Open ML a plus de points de données
3. L'Open ML a plus de pixels
4. L'Open ML a plus de couleur

# Coefficients de la classe MLPClassifier 1

La classe MLPClassifier stocke les **coefficients** dans **l'attribut coefs_**. Voyons à quoi cela ressemble :

print(mlp.coefs_)

[array([[-0.01115555, -0.08262704, 0.00865575,
-0.01127276, -0.01387922,
-0.0295712 ],
...

# Coefficients de la classe MLPClassifier 2

Nous constatons qu'il s'agit d'une liste avec deux éléments :

print(len(mlp.coefs_)) $\Longrightarrow$ 2

Les deux éléments de la liste correspondent aux deux couches : **la couche cachée et la couche de sortie**. Nous avons un tableau de coefficients pour chacune de ces couches. Regardons la forme des coefficients pour la couche cachée :

print(mlp.coefs_[0].shape) $\Longrightarrow$ **(784, 6)**

# Coefficients de la classe MLPClassifier 3

Nous voyons que nous avons un tableau bidimensionnel de taille 784 x 6. **Il y a 6 nœuds et 784 valeurs d'entrée** alimentant chaque nœud, et nous avons un poids pour chacune de ces connexions.

**Attention !**

Afin d'interpréter les valeurs, nous devrons utiliser une représentation visuelle.

# Exercice d'application 20

## Coefficients de la classe MLPClassifier

D'après la sortie de ce code, laquelle des affirmations suivantes est correcte concernant notre ensemble de données et notre réseau neuronal ?

print(len(mlp.coefs_)) $\Longrightarrow$ 2
print(mlp.coefs_[0].shape) $\Longrightarrow$ (10, 50)
print(mlp.coefs_[1].shape) $\Longrightarrow$ (50, 4)

1. 50 entrées / 10 nœuds dans la couche cachée / 4 sorties
2. 4 entrées / 50 noeuds dans la couche cachée / 10 sorties
3. 10 entrées / 50 noeuds dans la couche cachée / 4 sorties

# Représentation graphique de la couche cachée 1

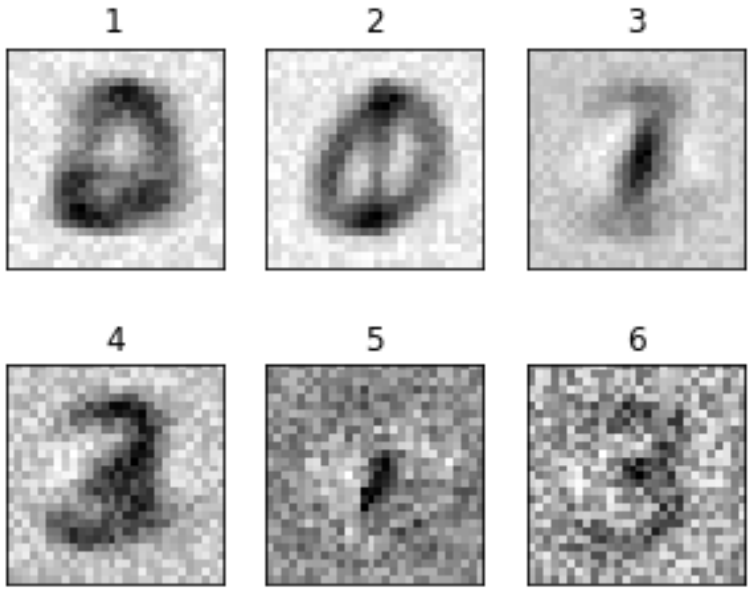
Pour mieux comprendre ce que fait le réseau neuronal, nous pouvons visualiser les poids de la couche cachée pour avoir une idée de ce que fait chaque nœud.

Nous allons à nouveau utiliser la **fonction matshow de matplotlib pour visualiser les images**. Dans matplotlib, nous pouvons utiliser **la fonction subplots pour créer plusieurs graphiques dans un seul graphique**.

# Représentation graphique de la couche cachée 2

```
fig, axes = plt.subplots(2, 3, figsize=(5, 4))
for i, ax in enumerate(axes.ravel()):
    coef = mlp.coefs_[0][:, i]
    ax.matshow(coef.reshape(28, 28), cmap=plt.cm.gray)
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(i + 1)
plt.show()
```

# Représentation graphique de la couche cachée 3




Vous pouvez voir que **les nœuds 4 et 6 déterminent si le chiffre est un 3**. **Le nœud 1 détermine si le chiffre est un 0 ou un 2 puisque vous pouvez voir ces deux valeurs dans l'image**. Tous les nœuds cachés n'ont pas une utilité évidente.

# Représentation graphique de la couche cachée 4

**Attention !**

Si vous modifiez l'état aléatoire dans le MLPClassifier, vous obtiendrez probablement des résultats différents. Il existe de nombreux réseaux neuronaux équivalents et optimaux qui fonctionnent différemment.

# Exercice d'application 21

## Représentation graphique de la couche cachée

Complétez le code pour répresenter graphiquement le premier nœud caché :

coef = ..... .coefs_[0][:, 0]
plt. ..... (coef. ..... (28, 28), cmap=plt.cm.gray)

# Interprétation

Bien que l'on puisse visualiser les nœuds de la couche cachée pour comprendre à un haut niveau ce que fait le réseau neuronal, il est **impossible de répondre à la question "Pourquoi le point de données x a-t-il obtenu la prédiction y ?"** Comme **il y a tellement de nœuds**, chacun avec ses propres coefficients, **il n'est pas possible d'obtenir une explication simple de ce que fait le réseau neuronal**. Cela en fait un modèle difficile à interpréter et à utiliser dans certains cas d'utilisation commerciale.

**Attention !**

Les réseaux neuronaux ne sont pas une bonne option pour l'interprétation.

# Exercice d'application 22

## Interprétation

Les réseaux neuronaux sont faciles à interpréter :

1. Vrai
2. Faux

# Quantité de calcul

**L'apprentissage des réseaux neuronaux peut prendre un temps considérable**. Chaque nœud a ses propres coefficients et, pour s'entraîner, ils sont mis à jour de manière itérative, ce qui peut prendre beaucoup de temps. Cependant, comme ils sont parallélisables, il est possible d'utiliser la puissance de l'ordinateur pour accélérer leur apprentissage.

**Attention !**

Une fois construits, les réseaux neuronaux ne sont pas lents à faire des prédictions, mais ils ne sont pas aussi rapides que certains autres modèles.

# Exercice d'application 23

## Quantité de calcul

Les réseaux neuronaux prennent plus de temps pour prédire que pour apprendre :

1. Vrai
2. Faux

# Performance 1

**Le principal attrait des réseaux neuronaux est leur performance**. Pour de nombreux problèmes, leurs performances sont tout simplement imbattables par d'autres modèles. Ils peuvent nécessiter un certain réglage des paramètres pour trouver la performance optimale, mais ils ont l'avantage de nécessiter un minimum d'ingénierie des prédicteurs avant de construire le modèle.

Pour de nombreux problèmes plus simples, vous pouvez obtenir des performances équivalentes avec un modèle plus simple comme la régression logistique, mais avec **de grands ensembles de données non structurées, les réseaux neuronaux surpassent les autres modèles**.

# Performance 2

**Attention !**

Le principal avantage des réseaux neuronaux est leur capacité de performance.

# Exercice d'application 24

## Performance

Le principal avantage des réseaux neuronaux est leur performance :

1. Vrai
2. Faux

# Quiz 1

## 1) Quelles sont les affirmations correctes ?

1. La fonction d'activation renvoie toujours une valeur entre 0 et 1
2. Un neurone est configuré par son poids
3. Un neurone a exactement un input
4. Un neurone a exactement une sortie

# Quiz 2

## 2) Fonction d'activation

Nous avons un neurone avec les poids et le biais suivants :

Poids (w1, w2) = [1, 0]
Biais (b) = 0

Si f est la fonction d'activation et y la sortie, quelle est l'équation correcte pour le point (2, 1) ?

1. y = f(1)
2. y = f(0)
3. y = f(3)
4. y = f(2)

# Quiz 3

## 3) Couche cachée

Lesquels de ces nombres de couches cachées sont valables pour un perceptron multicouche ?

1. 2
2. 0
3. 3
4. 1

# Quiz 4

## 4) MLP

Si vous avez deux couches cachées dans un perceptron multicouche, quelles sont les entrées de la deuxième couche cachée ?

1. Les sorties de la première couche cachée
2. Les entrées du réseau neuronal
3. Les mêmes que les entrées de la première couche cachée.

# Quiz 5

## 5) MLPClassifier

Laquelle des propositions suivantes crée un MLP avec 2 couches cachées ?

1. MLPClassifier(hidden_layer_sizes=(100, 50))
2. MLPClassifier()
3. MLPClassifier(hidden_layers = 2)
4. MLPClassifier(hidden_layer_sizes=(10, 10))

# Quiz 6

## 6) chargement, entraîner, accuracy

Complétez le code pour charger l'ensemble de données des chiffres, faire une répartition train-test, construire un MLPClassifier sur l'ensemble d'entraînement et afficher le score d'accuracy sur l'ensemble de test.

```
X, y = ..... (return_X_y=True)
X_train, X_test, y_train, y_test = ..... (X, y)
mlp = MLPClassifier().fit(X_train, y_train)
print( ..... (X_test, y_test))
```

# Classer les 10 chiffres 1

## Attention !

# Exercice d'application 9

## Interprétation

Regardons l'enfant gauche du nœud ?

1. 144
2. 532
3. 170