INTRODUCTION AU MACHINE LEARNING

ESGI – 2024/2025

Duchelle KEPSU – duchellekepsu@yahoo.com

école supérieure de génie informatique

Séance 2: Les modèles d’apprentissage supervisés et mise en application sur des jeux de données

Rappel: Qu'est ce qu'un modèle d'apprentissage supervisé?

- **L'apprentissage supervisé** est une approche de Machine Learning dans laquelle un modèle est entrainé à partir d'un ensemble de données étiquetées (chaque observation inclut des caractéristiques d'entrées et une sortie cible connue)

- Les modèles supervisés se divisent généralement en **deux** grandes catégories:
  - Les modèles de régression
  - Les modèles de classification

- Dans la suite de ce cours, nous allons étudier ces deux catégories de modèles supervisés, les différents algorithmes que constitue chacun d'eux, leur fonctionnement et comment les implémenter avec Python

- Travail Pratique sur les modèles d'apprentissage supervisés

Partie I: LES MODELES DE REGRESSION

I.1- Définition:

Un **modèle de régression** est un modèle de Machine Learning qui prédit une **valeur continue** à partir des données d'entrées.

Voici une liste non – exhaustive des exemples de modèles de régression:

- La régression Linéaire: utilisée pour établir une relation linéaire entre les variables d'entrées et la sortie
  Ex: Prédire le prix d'une maison

- La régression Polynomiale: Qui est une extension non-linéaire de la régression linéaire, et est utile lorsque la relation entre les variables est non – linéaire

- Régression Ridge et Lasso: Qui sont des variantes de la régression linéaire qui intègre des termes de régularisation pour réduire le surajustement

  NB: le surajustement (ou **Overfitting**) est un problème courant en Machine Learning un modèle apprend trop bien les détails et les variations du jeu de données d'entrainement au point où il perd sa capacité à généraliser sur de nouvelles données

- Régression des forêts aléatoires ou **Random Forest:** Modèle basé sur des arbres de décisions

Partie I: LES MODELES DE REGRESSION

➢ Dans ce cours, nous n'allons pas étudier tous ces types de modèles. Mais nous allons nous focaliser sur:

La régression linéaire **Le Random Forest**

I.2- Etude de la régression Linéaire:

❖ Principe fondamental:

L'objectif de la régression linéaire est de trouver une relation linéaire (une droite ou un hyperplan) entre les variables d'entrées X et la sortie Y. L'équation de base est:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon$$

- Y : Variable cible (valeur que l'on veut prédire).
- $X_1, X_2, \ldots, X_p$: Variables explicatives ou indépendantes.
- β_0 : Interception (valeur de Y lorsque toutes les entrées sont nulles).
- $\beta_1, \beta_2, \ldots, \beta_p$: Coefficients des variables $X_1, X_2, \ldots, X_p$, qui mesurent l'impact de chaque variable sur Y.
- ϵ : Terme d'erreur ou résidu (différence entre les valeurs réelles et prédites).

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Principe fondamental:

La régression linéaire est le modèle le plus simple pour modéliser et prédire des valeurs continues en fonction d'autres variables.
Il existe deux types de régression linéaire:

- **La régression linéaire simple:** Pensez-y comme une ligne droite qui relie les points dans un nuage de données captant ainsi la tendance générale des observations.

  Lorsqu'on prédit une variable cible Y, à partir d'une variable de prédiction ou d'entrée X, la régression linéaire simple **consiste à trouver les coefficients β_0 et β_1 dans l'équation de la droite qui va être plus proche de tous les points du nuage de point:**

$$Y = \beta_0 + \beta_1 X_1$$

- **La régression linéaire multiple:** Lorsqu'on prédit une variable cible Y à partir d'au moins 2 variables de prédiction, on parle de régression linéaire multiple. Pour deux variables de prédiction, le modèle correspond à l'équation d'un plan défini par:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Processus d'apprentissage de la régression Linéaire:

1 – Estimation des coefficients β

- L'objectif est de minimiser l'erreur € = J(β) (qui est une fonction dépendant des coefficients β) entre les valeurs prédites (Ŷ) et les valeurs réelles (Y)

$$J(\beta) = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 X_{i1} + \cdots + \beta_p X_{ip}))^2 \quad \Longleftrightarrow \quad J(\beta) = ||Y - X\beta||^2$$

L'objectif est de trouver le vecteur β qui minimise J(β)

La bonne nouvelle c'est qu'avec la programmation informatique, cette étape est gérée par des fonctions qui font le travail. Nous allons les voir dans la partie implémentation avec python

2- Prédiction

- Une fois les coefficients β on peut prédire Y pour de nouvelles valeurs de X en utilisant:

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Implémentation en python:

Prenons un exemple d'implémentation avec sickit-learn pour prédire les prix de maison (dataset fictif)

- Importation des librairies:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Pour la régression linéaire, nous avons besoin de deux modules clés de la bibliothèque sickit-learn et de deux fonctions clés:

- Dans le module **sklearn.linear_model**: la fonction **LinearRegression** est utilisée pour réaliser le processus d'apprentissage de la régression linéaire. C'est cette fonction qui calcule directement les coefficients β des variables de prédiction

- Dans le module **sklearn.model_selection**: la fonction **train_test_split** est utilisée pour diviser le jeu de donnée en jeu d'entrainement et jeu de test

NB: Numpy et Pandas sont d'autres bibliothèques pour la gestion des données le traitement et les calculs numériques

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Implémentation en python:

- Collecte des données:

Nous avons pour cet exemple crée des données fictives:

```
# Exemple de données
data = pd.DataFrame({
    'Surface': [50, 60, 70, 80, 90],
    'Chambres': [1, 2, 2, 3, 3],
    'Prix': [150000, 180000, 200000, 240000, 275000]
})
```

En vue réelle nous avons ceci:

Surface	Chambres	Prix
50	1	150000
60	2	180000
70	2	200000
80	3	240000
90	3	275000

- Séparation des caractéristiques X (Surface et Chambre) et la cible à prédire Y (Prix)

```
# Séparation des caractéristiques (X) et de la cible (Y)
X = data[['Surface', 'Chambres']]
y = data['Prix']
```

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Implémentation en python:

- Division en données d'entrainement et de test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

On note la présence de paramètres dans notre fonction:

- **test_size = 0.2**: qui donné la taille de nos données de test: 20% de l'ensemble de nos données et donc notre modèle s'entraine sur 80% de l'ensemble de données

- **Random_state = 42**: 42 est un chiffre utilisé de manière conventionnelle pour garantir qu'à chaque exécution de notre code, la division des données soit toujours la même

- Création du modèle linéaire:

```
# Modèle de régression linéaire
model = LinearRegression()
model.fit(X_train, y_train)
```

  - On instancie un objet de la classe LinearRegression et on l'affecte à la variable « model »

  - Ce modèle peut donc être utilisé sur nos données en appliquant la méthode **.fit()** et en mettant en paramètres de cette fonction les variables de prédiction d'entrainement **X_train** et les données cibles d'entrainement **y_train**

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Implémentation en python:

- On fait la prédiction sur les données de test :

```
# Prédictions
y_pred = model.predict(X_test)
```

  - ➢ En utilisant la fonction **.predict()** sur l'objet « model » instancié et en prenant en paramètre les variables de prédiction test

  - ➢ Les prédictions sont stockées dans la variable **y_pred**

- Affichage des résultats:

```
results = pd.DataFrame({
    "Surface_m2": X_test["Surface_m2"],
    "Chambres": X_test["Chambres"],
    "Prix Réel": y_test,
    "Prix Prédit": y_pred
})
print(results)
```

Partie I: LES MODELES DE REGRESSION

I.2- Etude de la régression Linéaire:

❖ Implémentation en python:

- Affichage des résultats:

Surface (m^2)	Chambres	Prix Réel (€)	Prix Prédit (€)
70	2	200000	204000
50	1	150000	158000

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Principe fondamental:

- Un **arbre de décision** est un modèle de prédiction qui divise l'espace des données en fonction des critères de décision de sorte à minimiser l'erreur dans les prédictions de la cible. Il crée des branches en fonction des caractéristiques des données et fournit une valeur cible (par exemple le prix d'une maison)

- Au lieu d'utiliser un seul arbre de décision, pour prédire une valeur, le modèle **Random Forest** utilise un ensemble d'arbres de décision d'où le terme **forêt**

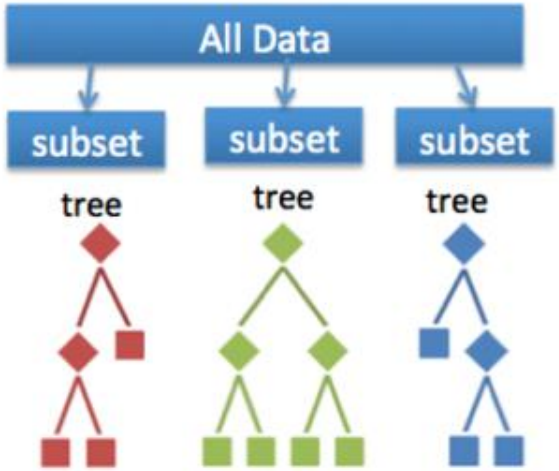
- Chaque arbre de la forêt est construit avec un sous – ensemble aléatoire des données d'entrainement et une sélection aléatoire des caractéristiques (variables de prédiction). Cela permet d'augmenter la diversité des arbres et d'éviter les pbs d'overfitting

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Principe fondamental:

- Une représentation graphique d'une forêt aléatoire



❖ Processus d'apprentissage de la forêt aléatoire de régression (Random Forest):

1- Echantillonnage Bootstrap:

- A partir du jeu d'entrainement, on extrait plusieurs échantillons de manière aléatoire avec remplacement pour créer les sous-ensembles sur lesquels chaque arbre sera formé

2- Sélection des caractéristiques à chaque nœud: Une fois l'échantillon Bootstrap prêt, la construction de l'arbre de décision se poursuit avec la sélection des features

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Processus d'apprentissage de la forêt aléatoire de régression (Random Forest):

2- Sélection des caractéristiques à chaque nœud:

- A chaque nœud de l'arbre, une sous-partie aléatoire des caractéristiques est sélectionnée. Le modèle évalue chaque caractéristique sélectionnée pour trouver la meilleure coupure possible qui minimise l'erreur de prédiction à ce nœud.

- Le processus de division est répété pour chaque nouveau nœud et de nouvelles caractéristiques sont sélectionné de manière aléatoire

- La régression continue jusqu'à ce qu'un **critère d'arrêt** soit atteint

- Critère d'arrêt de l'arbre: Si aucune division n'apporte de division significative de l'erreur de prédiction, le processus s'arrête

- Et ces étapes se reproduise pour la création de chaque arbre

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Processus d'apprentissage de la forêt aléatoire de régression (Random Forest):

3- Agrégation des arbres:

- Après la construction de tous les arbres, la forêt aléatoire fait des prédictions pour de nouvelles données en agrégeant les prédictions de chaque arbre: la prédiction finale est la moyenne des prédictions de tous les arbres

❖ Implémentation en python:

Reprenons notre exemple précédent sur la prédiction du prix d'une maison. Et cette fois, appliquons la méthode des forêts aléatoire de régression:

- Importation des librairies:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Implémentation en python:

- Importation des librairies:

  Pour les forêts aléatoires de régression, nous avons besoin également de deux modules de la bibliothèques sickit-learn et de deux fonctions:

  - Dans le module **sklearn.ensemble:** La fonction **RandomForestRegressor** est utilisée pour réaliser le processus d'apprentissage de la forêt aléatoire de régression. Elle prédit les valeurs continues en combinant les prédictions de multiples arbres de décisions individuels.

  - Dans le module **sklearn.model_selection**: la fonction **train_test_split** joue le même rôle que dans le cadre de la régression linéaire; celui de la division des données en jeu d'entrainement et de test

- Collecte des données:

Nous allons utiliser les mêmes données fictives qui ont été utilisé dans l'exemple précédent

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Implémentation en python:

- Collecte des données:

```
# Exemple de données
data = pd.DataFrame({
    'Surface': [50, 60, 70, 80, 90],
    'Chambres': [1, 2, 2, 3, 3],
    'Prix': [150000, 180000, 200000, 240000, 275000]
})
```

- Séparation des caractéristiques X (Surface et Chambre) et la cible à prédire Y (Prix)

```
# Séparation des caractéristiques (X) et de la cible (Y)
X = data[['Surface', 'Chambres']]
y = data['Prix']
```

- Division en données d'entrainement et de test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Implémentation en python:

- Création du modèle:

```
# Initialisation du modèle
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Entraînement du modèle
rf_model.fit(X_train, y_train)
```

➢ On instancie un objet de la classe **RandomForestRegressor** et on l'affecte à la variable « rf_model »:
On note la présence de deux paramètres dans notre fonction:

- n_estimators: qui représente le nombre d'arbres de décisions dans la forêt. La valeur par défaut dans la bibliothèque sickit-learn est 100.

- Random_state: vu précédemment

➢ Ce modèle peut donc être utilisé sur nos données en appliquant la méthode **.fit()** et en mettant en paramètres de cette fonction les variables de prédiction d'entrainement **X_train** et les données cibles d'entrainement **y_train**

Partie I: LES MODELES DE REGRESSION

I.3- Etude du Random Forest ou Régression des forêts aléatoires :

❖ Implémentation en python:

- On fait la prédiction sur les données de test :

```
# Prédictions sur l'ensemble de test
y_pred = rf_model.predict(X_test)
```

- Affichage des résultats:

```
results = pd.DataFrame({
    "Surface_m2": X_test["Surface_m2"],
    "Chambres": X_test["Chambres"],
    "Prix Réel": y_test,
    "Prix Prédit": y_pred
})
print(results)
```

Surface_m2	Chambres	Prix Réel (€)	Prix Prédit (€)
70	2	200000	205000

Partie II: LES MODELES DE CLASSIFICATION

II.1- Définition:

Un **modèle de classification** est un modèle de Machine Learning qui prédit une **catégorie** ou une **étiquette discrète** à partir des données d'entrées.

Voici une liste non – exhaustive des exemples de modèles de classification:

- La régression logistique: qui est un modèle probabiliste utilisé pour les classifications binaires (par exemple: vérifier si un mail est un spam ou non)
- Les arbres de décision: modèle basé sur des règles pour catégoriser les données
- Forêts aléatoires (Random Forest): modèle d'ensemble d'arbre de décision
- K-Nearest Neighbors (KNN): modèle qui classifie une donnée en fonction de la majorité des étiquettes de ses voisins les plus proches
- Gradient Boosting Machine (GBM): Des modèles comme Xgboost, AdaBoost qui sont des modèles basés sur les arbres de décision mais plus robustes

Partie II: LES MODELES DE CLASSIFICATION

➢ Dans ce cours, nous n'allons pas étudier tous ces types de modèles. Mais nous allons nous focaliser sur:

La régression logistique

Les arbres de décisions

II.1- Etude de la régression Logistique:

❖ Principe fondamental:

Le modèle de régression logistique cherche à établir une relation entre les variables d'entrée (features) et la probabilité d'appartenir à une certaine classe (par exemple, « 1 » ou « 0 »). Il repose sur une fonction sigmoïde (ou fonction logistique) pour transformer une combinaison linéaire des variables d'entrée en une probabilité comprise entre 0 et 1.

La fonction sigmoïde est définie par: $$\text{sigmoïde}(z) = \frac{1}{1+e^{-z}}$$

Où z est la combinaison linéaire des variables d'entrées, soit : $$z = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \cdots + \beta_n \cdot x_n$$

- Ici, β_0 est l'ordonnée à l'origine (intercept), et $\beta_1, \beta_2, \ldots, \beta_n$ sont les **coefficients** associés aux variables d'entrée $x_1, x_2, \ldots, x_n$.

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Principe fondamental:

- La fonction sigmoïde transforme cette somme linéaire pour produire une probabilité p (comprise entre 0 et 1) que l'observation appartienne à une certaine classe

- Prédiction de la classe: la probabilité p de la classe, correspondant à la valeur calculée par la fonction sigmoïde suit cette logique:

  - Si $p \geq 0.5$ le modèle prédit la classe « 1 » (positif)

  - Si $p < 0.5$ le modèle prédit la classe « 0 » (négatif)

La bonne nouvelle c'est qu'avec la programmation informatique, cette étape est gérée par des fonctions qui font le travail. Nous allons les voir dans la partie implémentation avec python

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

L'exemple que nous allons prendre ici est celui de la prédiction des espèces de fleurs de la base de données Iris: Cette base de données contient des données sur 3 espèces de fleurs: Setosa, Versicolor, et Virginica.

Les caractéristiques ou variables explicatives permettant de déterminer à quelle espèce appartient chaque fleur et la variable cible (species = espèce) sont les suivantes:

- **sepal length (cm)** : Longueur du sépale en centimètres.
- **sepal width (cm)** : Largeur du sépale en centimètres.
- **petal length (cm)** : Longueur du pétale en centimètres.
- **petal width (cm)** : Largeur du pétale en centimètres.
- **species** : Label représentant l'espèce de la fleur (0 : *Setosa*, 1 : *Versicolor*, 2 : *Virginica*).

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

Voici à quoi ressemble la base de données Iris:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0

NB: Dans notre exercice, nous allons simplifier le problème à deux classes, nous allons uniquement prédire les classes **Setosa (0)** ou **non-Setosa (1)** (appartenance à la classe Virginica ou Versicolor)

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

- Importation des librairies:

```python
# 1. Importation des bibliothèques nécessaires
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Pour la régression logistique, nous avons besoin également de deux modules de la bibliothèques sickit-learn et de deux fonctions:

- Dans le module **sklearn.linear_model:** La fonction **LogisticRegression** est utilisée pour réaliser le processus d'apprentissage de la régression linéaire. Elle la probabilité d'une espèce à appartenir soit à la classe Setosa soit à la classe non-Setosa

- Dans le module **sklearn.model_selection**: la fonction **train_test_split** joue le même rôle, celui de la division des données en jeu d'entrainement et de test

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

- Collecte des données: Le dataset Iris est présent dans la bibliothèque sickit-learn. Nous allons l'importer directement:

```
# 2. Chargement du dataset Iris
iris = load_iris()
X = iris.data  # Caractéristiques (lor
y = iris.target  # Classes des fleurs
```

➢ Les caractéristiques ou variables explicatives (longueur et largeur des sépales, longueur et largeur des pétales) sont stockées dans la variable **x**

➢ La variable cible ou variable à prédire (espèce des fleurs) est stockée dans la variable **y**

- Modification de la variable cible: Ici on va faire une petite modification: en paramétrant les espèces de fleurs avec 0 = Setosa et 1= Non-Setosa

```
# Pour simplifier, on ne
y = (y != 0).astype(int)
```

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

- Division en données d'entrainement et de test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Création du modèle:

```
# 4. Création et entraînement du modèle de régression logistique
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
```

➢ On instancie un objet de la classe **LogisticRegression** et on l'affecte à la variable « log_reg »

➢ Ce modèle peut donc être utilisé sur nos données en appliquant la méthode **.fit()** et en mettant en paramètres de cette fonction les variables de prédiction d'entrainement **X_train** et les données cibles d'entrainement **y_train**

- On fait la prédiction sur les données de test :

```
# 5. Prédiction sur les données de test
y_pred = log_reg.predict(X_test)
```

Partie II: LES MODELES DE CLASSIFICATION

II.1- Etude de la régression Logistique:

❖ Implémentation en python:

- Affichage des probabilités prédites:

```
# 7. Affichage des probabilités prédites pour les données test
y_prob = log_reg.predict_proba(X_test)  # Probabilités pour chaque classe
print("\nProbabilités prédites (premières 5 observations):\n", y_prob[:5])
```

```
Probabilités prédites (premières 5 observations):
 [[0.982 0.018]
  [0.014 0.986]
  [0.003 0.997]
  [0.924 0.076]
  [0.002 0.998]]
```

NB: La 1ère probabilité correspond à la probabilité que l'observation appartienne à la classe « **positive** » (dans notre cas c'est la probabilité que la fleur soit d'espèce Setosa). Et la somme des deux probabilités donne 1

II.2- Etude des arbres de décision:

❖ Principe fondamental:

Le modèle **d'arbre de décision** divise les données en sous ensembles homogènes en fonction d'une règle basée sur les caractéristiques.

Précédemment, dans le cadre des modèles de régression, nous avons étudié le Random Forêt: qui est un ensemble d'arbres de décision. La différence est que dans ce cas, le modèle construit un seul arbre qui vérifie des critères sur les caractéristiques pour faire des prédictions

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Principe fondamental:

- Méthodes pour choisir les divisions:

1- On considère toutes les caractéristiques disponibles

2- On teste les divisions possibles:

- ➢ Si une caractéristique est continue (par exemple, une valeur numérique comme la taille), plusieurs seuils sont testés (par exemple, « taille > 5 »)
- ➢ Si une caractéristique est catégorielle (par exemple, couleurs rouge vert, bleu), on examine chaque catégorie ou combinaison

3- On évalue la qualité de chaque division: On calcule une mesure de « gain ». Les métriques suivantes nous permettent d ’évaluer le gain:

- ➢ L’indice de Gini: qui mesure la probabilité qu’un élément soit mal classé, l’impureté d’un ensemble. Il est calculé de la manière suivante:

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Principe fondamental:

- Méthode pour choisir les divisions:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

où p_i est la proportion des observations appartenant à la classe i.

✓ L'objectif est de choisir la division, caractéristique et seuil (pour les variables continues) qui **minimise l'indice de Gini**

➢ L'entropie et le gain d'information: L'entropie mesure l'incertitude d'un ensemble. Elle se calcule de la manière suivante:

$$Entropie = -\sum_{i=1}^{C} p_i \cdot \log_2(p_i)$$

où p_i est la proportion des observations dans la classe i.

Le **gain d'information** est défini comme la réduction d'entropie après une division

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Principe fondamental:

$$Gain\ d'information = Entropie_{parent} - \sum \left(\frac{|Ensemble\ fils|}{|Ensemble\ parent|} \cdot Entropie_{fils} \right)$$

✓ L'objectif est de choisir la division qui **maximise le gain d'information**

Les divisions sont effectuées de manières récursives, jusqu'à ce que la construction de l'arbre s'arrête.

4- Conditions d'arrêt de la construction de l'arbre:

La construction de l'arbre s'arrête lorsque une des condition suivante est atteinte:

1. Tous les exemples dans un sous-ensemble appartiennent à la même classe.
2. Il n'y a plus de caractéristiques pour diviser.
3. Une profondeur maximale ou un nombre minimal d'exemples par feuille est atteint (pour éviter le surapprentissage).

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Implémentation en python:

Nous allons reprendre notre exemple précédent, celui de la prédiction de l'espèce des fleurs de la base de données Iris. Ici, nous prédirons chaque espèce de fleur: **Setosa**, **Virginica** et **Versicolor**

1- Importation des librairies:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

- Dans le module **sklearn.tree:** La fonction **DecisionTreesClassifier** est utilisée pour réaliser le processus d'apprentissage de l'arbre de décision. Elle prédit les classes auxquelles appartiennent les ensembles de données en les divisant à chaque nœud en fonction des caractéristiques.

- Dans le module **sklearn.model_selection**: la fonction **train_test_split** joue le même rôle que dans le cadre de la régression linéaire; celui de la division des données en jeu d'entrainement et de test

2- Collecte des données:

Le dataset Iris est présent dans la bibliothèque sickit-learn. Nous allons l'importer directement:

```
# Charger le dataset Iris
iris = load_iris()
X = iris.data
y = iris.target
```

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Implémentation en python:

- Division en données d'entrainement et de test:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Création du modèle:

```
tree_clf = DecisionTreeClassifier(
    criterion='gini', max_depth=3, random_state=42)
tree_clf.fit(X_train, y_train)
```

➢ On instancie un objet de la classe **DecisionTreeClassifier** et on l'affecte à la variable « tree_clf ». On note la présence de paramètres dans notre fonction:

- criterion = « gini »: défini la métrique utilisée pour mesurer la qualité des divisions. Dans notre cas, nous avons utilisé l'indice de gini

- max_depth = 3: qui défini la limite de la profondeur de l'arbre

➢ Ce modèle peut donc être utilisé sur nos données en appliquant la méthode **.fit()** et en mettant en paramètres de cette fonction les variables de prédiction d'entrainement **X_train** et les données cibles d'entrainement **y_train**

Partie II: LES MODELES DE CLASSIFICATION

II.2- Etude des arbres de décision:

❖ Implémentation en python:

- On fait la prédiction sur les données de test :

```
y_pred = tree_clf.predict(X_test)
```

Pour chaque exemple de fleur X_test, y_pred prédit l'espèce à laquelle appartient la fleur.

PARTIE III:TP Apprentissage supervisé, les modèles de régression et de classification