Algorithme ID3, Arbre de Décision

Exercie 1

On souhaite construire un arbre de décision permettant de prédire si un étudiant révise ou non, à partir des attributs suivants :

— **Météo** : Soleil, Pluie, Nuageux

— **Humeur** : Bonne, Mauvaise

— **Temps libre** : Oui, Non

Voici la base de données :

#	Météo	Humeur	Temps libre	Révision
1	Soleil	Bonne	Oui	Oui
2	Pluie	Mauvaise	Non	Non
3	Soleil	Bonne	Non	Oui
4	Nuageux	Bonne	Oui	Oui
5	Soleil	Mauvaise	Oui	Oui
6	Pluie	Bonne	Non	Non
7	Nuageux	Mauvaise	Oui	Non
8	Soleil	Bonne	Oui	Oui

Travail à faire :

1. Calculer l'entropie de la variable cible `Révision`.
2. Calculer le gain d'information de chaque attribut.
3. Choisir l'attribut optimal pour la racine de l'arbre.
4. Construire l'arbre de décision en suivant l'algorithme ID3.
5. Représenter l'arbre final.

Exercie 2

Cet exercice vous guide dans la création progressive d'un arbre de décision en Python, en implémentant une à une les fonctions de la classe `ID3Classifier`.

Q1. Initialisation de la classe. Créez la classe `ID3Classifier` avec la méthode `__init__` qui initialise deux attributs :

- `self.tree_` (pour l'arbre final)
- `self.splits_` (liste vide pour enregistrer les divisions)

Q2. Entropie. Implémentez la méthode `_entropy(self, y)` qui reçoit une série ou une liste de classes, et retourne son entropie (utilisez `Counter`, `numpy` et `log2`).

Q3. Gain d'information. Implémentez la méthode `_info_gain(self, df, attr, target)` qui :

- Calcule l'entropie globale de la cible.
- Calcule l'entropie conditionnelle pondérée après division par `attr`.
- Retourne la différence entre les deux (le gain).

Q4. Construction récursive de l'arbre. Implémentez la méthode `_build_tree(self, df, features, target)` :

- Cas 1 : si toutes les cibles sont identiques, retournez la classe.
- Cas 2 : si plus d'attributs disponibles, choisissez celui avec le plus grand gain.
- Ajoutez un enregistrement dans `self.splits_` avec :
  - l'attribut sélectionné
  - l'entropie globale
  - le gain
  - la taille du nœud
- Construisez récursivement les sous-arbres.

Q5. Apprentissage. Implémentez la méthode `fit(self, df, features, target)` qui appelle `_build_tree` et stocke l'arbre final dans `self.tree_`.

Q6. Affichage en console. Implémentez la méthode `print_tree(self, tree=None, indent="")` pour afficher l'arbre sous forme textuelle, avec indentation croissante.

Q7. Visualisation graphique. Implémentez la méthode `visualize(self, filename="id3_tree")` qui :

- Utilise le module `graphviz` pour créer un graphe.
- Affiche et sauvegarde le graphe généré.

Q8. Analyse des divisions. Implémentez la méthode `get_splits_dataframe(self)` qui retourne un `DataFrame pandas` contenant toutes les divisions faites (chaque split avec les infos stockées dans `self.splits_`).

Q9. Test complet. Créez un script principal `if __name__ == "__main__"` qui :

a) charge un fichier CSV (`data_revision.csv`)
b) apprend l'arbre avec `fit`
c) affiche l'arbre avec `print_tree` et `visualize`
d) affiche les splits avec `get_splits_dataframe`

Proposition

Testez chaque méthode isolément dans un notebook ou un script avant de les combiner. Cela facilitera le débogage et la compréhension du fonctionnement de chaque composant.