INFO-F103 – Algorithmique I
Projet 2 - Exploration et Interrogation d'un Arbre de Décision

Youcef Bouharaoua
youcef.bouharaoua@ulb.be

Année académique 2023-2024

Introduction

Un arbre de décision est une structure de données qui représente une série de décisions séquentielles, utilisée fréquemment pour la classification et la prédiction. Chaque nœud de l'arbre pose une question sur une caractéristique des données, et chaque branche représente une réponse possible, conduisant à un nouveau nœud ou à une conclusion finale. Cette méthode est particulièrement efficace pour traiter des décisions complexes en les décomposant en une série de choix plus simples, ce qui rend l'arbre de décision central dans notre projet.

Dans le cadre de ce projet, vous allez plonger dans le monde fascinant des structures de données et de l'apprentissage automatique. La première partie vous initiera au chargement et à l'analyse de données réelles issues d'un fichier CSV, concernant des caractéristiques variées de champignons. Vous apprendrez à distinguer les champignons comestibles des non comestibles en appliquant des concepts fondamentaux tels que l'entropie et le gain d'information pour construire un arbre de décision. Cette structure vous permettra de classer efficacement les données en fonction de critères précis, démontrant ainsi la puissance des arbres de décision dans la résolution de problèmes de classification.

La seconde partie du projet mettra l'accent sur l'exploration de cet arbre de décision. Vous développerez ensuite une requête spécifique pour interroger l'arbre afin de déterminer la comestibilité d'un champignon basée sur ses attributs.

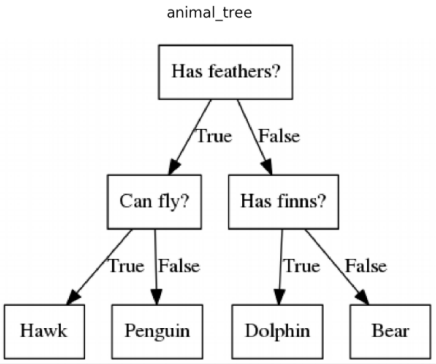


FIGURE 1 – Exemple d'un arbre de décision

Énoncé

Partie 1 – Chargement des données et construction de l'arbre

Dans cette première partie du projet, nous allons apprendre à charger des données à partir d'un fichier CSV et à utiliser ces données pour construire un arbre de décision. Cela implique de comprendre la structure de nos données et d'appliquer des concepts tels que l'entropie et le gain d'information.

Chargement des données

Le nom CSV veut dire *Comma Separated Values*, ou encore *valeurs séparées par des virgules* en français. Ce format de fichiers est donc très simple : chaque ligne représente une donnée composée d'attributs, tous séparés par une virgule. Habituellement, la première ligne du fichier contient le nom des colonnes, ou de manière équivalente, le nom des attributs.

Le fichier CSV que nous utilisons contient des informations sur différents champignons. Chaque ligne représente un champignon et inclut des caractéristiques comme sa forme, sa surface, sa couleur, etc.
La première colonne indique si le champignon est comestible ou non.

Voici en exemple les premières lignes du fichier `mushrooms.csv` qui vous est fourni [1] :

```
edible,cap-shape,cap-surface,cap-color,bruises,...
No,Convex,Smooth,Brown,Yes,...
Yes,Convex,Smooth,Yellow,Yes,...
Yes,Bell,Smooth,White,Yes,...
```

Étapes de chargement

Pour charger les données, nous utiliserons le module `csv` de Python. Écrivez la fonction `load_dataset(path: str) -> list[Mushroom]` qui lira le fichier et séparera les labels (*edible* ou non) des autres caractéristiques, et qui renvoie une liste d'objets `Mushroom`. Pour réaliser cette tâche, il vous faudra donc créer une classe, dont voici un squelette :

```
class Mushroom:
    def __init__(self, edible: bool):
        ...
    # fonction permettant de determiner
    # si un champignon est comestible
    def is_edible(self) -> bool:
        ...
    def add_attribute(self, name: str, value: str) -> None:
        ...
    def get_attribute(self, name: str) -> str:
        ...
```

Il est à noter que les méthodes spécifiées ci-dessus doivent impérativement faire partie de votre classe mais qu'il est encouragé de rajouter d'autres si vous en sentez le besoin. Aussi, les signatures des méthodes et des classes ne doivent pas être modifiées.

1. Ce jeu de donnée a été créé par la *National Audubon Society* et a été donnée par Jeff Schilmmer au répertoire des jeux de données de l'Université d'Irvine. Il est accessible par son DOI : 10.24432/C5959T.

Construction de l'arbre

Après avoir chargé les données, nous allons commencer la construction de l'arbre de décision. Afin de déterminer la manière de diviser les données à chaque nœud, nous utiliserons deux notions fondamentales en théorie de l'information : l'*entropie* et le *gain d'information* (également appelé *information mutuelle*).

Dans le cadre de ce projet, notre arbre de décision est un arbre dans lequel :

— chaque nœud correspond à un attribut des données ;

— les arêtes sortant d'un nœud correspondent à toutes les valeurs possibles pour cet attribut ;

— les feuilles correspondent à une *étiquette* (ici comestible vs non-comestible) ;

— chaque attribut ne peut apparaître qu'au plus une fois sur chaque branche.

Sur base des données chargées, il est possible de créer un très grand nombre d'arbres de décisions qui encoderont les mêmes informations. Cependant, ces arbres sont, en majorité, particulièrement lourds dans leur encodage et requièrent un assez grand nombre de sommets.
Prenons par exemple les données (tout à fait artificielles) représentées dans la table suivante, où A1, A2 et A3 représentent des attributs quelconques :

A1	A2	A3	Étiquette
1	1	1	Oui
1	2	1	Oui
1	3	1	Oui
2	1	1	Non
2	2	1	Non
2	3	1	Non

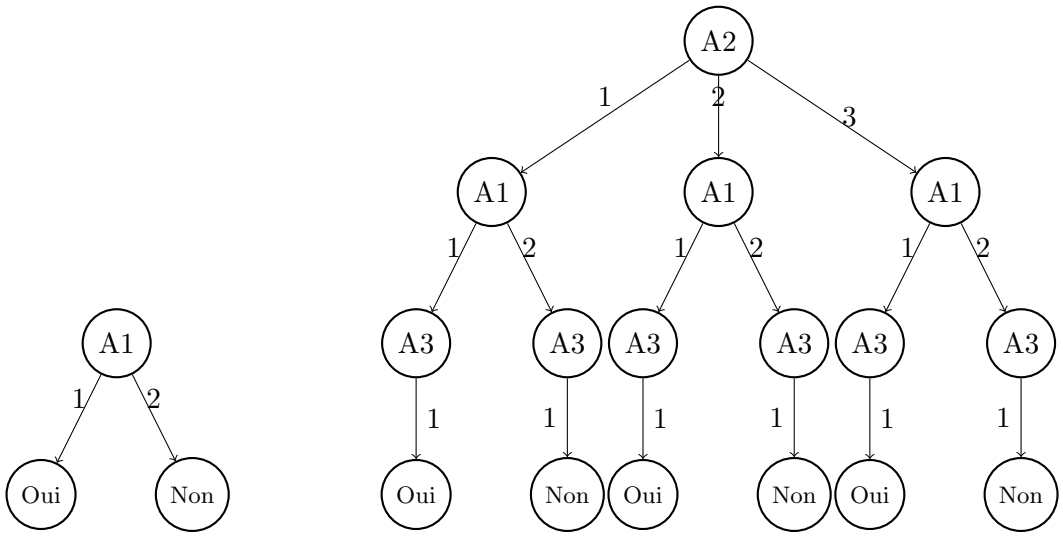
TABLE 1 – Données artificielles.

Nous pouvons y observer que seul l'attribut A1 est important puisque l'étiquette finale vaut *Oui* si l'attribut A1 vaut 1 et l'étiquette vaut *Non* si ce-même attribut vaut 2. Il est toutefois possible de construire un arbre alternatif utilisant tous les attributs, même si ces derniers ne servent à rien. La différence entre le meilleur cas et le pire cas est illustrée dans la figure 2.

L'objectif ici va être de construire l'arbre depuis sa racine jusqu'aux feuilles de manière à ce qu'à chaque étape l'attribut qui *discrimine* le plus les champignons restants soit choisi. En procédant de la sorte, nous devrons obtenir un arbre minimisant la longueur des chemins entre la racine et ses feuilles.

Pour la construction d'un tel arbre, vous aurez besoin d'implémenter les classes `Node` et `Edge` qui représentent respectivement un nœud et une arrête de l'arbre. Voici leurs signatures :

```
class Node:
    def __init__(self, criterion: str, is_leaf: bool=False):
        ...
        self.edges_ = [] # liste des arcs du noeud
    def is_leaf(self) -> bool:
        ...
    def add_edge(self, label: str, child: 'Node') -> None:
        ...
class Edge:
    def __init__(self, parent: Node, child: Node, label: str):
        self.parent_ = parent
        self.child_ = child
        self.label_ = label
```



(a) Meilleur arbre possible. (b) Pire arbre possible.

FIGURE 2 – Exemples extrêmes d'arbres de décision sur base des données dans le tableau 1.

Tout comme pour la classe `Mushroom`, vous êtes encouragés à rajouter d'autres méthodes si nécessaires.

Pour construire un tel arbre, il est également nécessaire de connaître les concepts suivants et de les incorporer dans votre solution :

1. **Introduction à l'entropie.** L'entropie est une mesure de l'*impureté* dans un ensemble de données. Dans le cadre de notre arbre de décision, elle nous aide à évaluer combien un ensemble de caractéristiques est mélangé en ce qui concerne les étiquettes (comestible ou non). L'entropie est maximale lorsque les instances sont réparties également entre les classes et minimale (zéro) lorsque toutes les instances appartiennent à la même classe.
2. **Calcul de l'entropie.**
   Pour un ensemble $\mathcal{C}$ de champignons, l'entropie $H(\mathcal{C})$ est calculée comme suit :
   $$H(\mathcal{C}) = \begin{cases} 0 & \text{si } p_Y = 0 \text{ ou } p_Y = 1 \\ p_Y \log_2 \dfrac{1-p_Y}{p_Y} - \log_2(1-p_Y) & \text{sinon.} \end{cases}$$
   où p_Y est la proportion des champignons comestibles.
3. **Utilisation de l'entropie pour la création de l'arbre.** À chaque nœud de notre arbre, nous chercherons à choisir l'attribut qui, une fois utilisé pour diviser les données, résulte en la plus grande réduction de l'entropie, ce qu'on appelle le gain d'information. Cela signifie que nous voulons que les sous-ensembles résultants soient les plus *purs* possibles, avec une entropie aussi basse que possible, relativement à l'entropie de l'ensemble initial.
   Nous calculons le gain d'information pour chaque attribut comme la différence entre l'entropie avant la division et la somme pondérée des entropies des sous-ensembles après la division. Plus précisément, si $\mathcal{C}_{A=v}$ désigne le sous-ensemble de $\mathcal{C}$ contenant uniquement les champignons dont l'attribut A vaut v, et si $p_{A=v}$ désigne la proportion des champignons de $\mathcal{C}$ qui sont dans $\mathcal{C}_{A=v}$ (i.e. $|\mathcal{C}_{A=v}| = p_{A=v}\,|\mathcal{C}|$), alors le gain d'information $I(\mathcal{C}|A)$ est défini par :
   $$I(\mathcal{C}|A) = H(\mathcal{C}) - \sum_v p_{A=v} \cdot H(\mathcal{C}_{A=v}).$$

L'attribut avec le gain d'information le plus élevé est choisi pour diviser les données à ce nœud.

Voici un *pseudo-code* récapitulatif de la création de l'arbre de décision :

1. Choisir l'attribut A qui sépare au mieux les champignons.
2. Créer le nœud associé r.
3. Pour chaque valeur v possible de cet attribut A :
   — Construire le sous-arbre $T_{A=v}$ sur base des champignons ayant la valeur v pour leur attribut A.
   — Ajouter la racine de $T_{A=v}$ aux enfants de r.

Pour résumer, on vous demande d'écrire une fonction `build_decision_tree` dont voici la signature : `build_decision_tree(mushrooms: list[Mushroom]) -> Node`. Cette fonction doit renvoyer la racine de l'arbre construit.

Tests

Dans cette partie du projet, deux catégories de tests sont nécessaires pour évaluer correctement la fonctionnalité et l'efficacité de votre implémentation.

Tests de chargement des données. Trois tests automatiques vous seront fournis pour vérifier que votre fonction `load_dataset` charge correctement les données du fichier CSV. Ces tests vérifieront si les données sont correctement extraites, si les attributs sont bien attribués aux instances de la classe `Mushroom`, et si les labels sont correctement définis. Échouer à ces tests entraînera une note de 0 sur 10 pour cette partie du chargement des données.

Tests de construction de l'arbre. Des tests spécifiques seront également fournis pour évaluer la qualité et l'optimalité de l'arbre de décision que vous construisez avec la fonction `build_decision_tree`. Ces tests examineront si l'arbre construit minimise la profondeur nécessaire pour arriver à une décision, et s'il utilise l'entropie et le gain d'information de manière efficace pour choisir les attributs de division.

Création de tests personnalisés. Au-delà des tests fournis, vous devez créer et fournir au moins deux tests personnalisés que vous discuterez dans votre rapport de projet. Ces tests devraient être conçus pour couvrir des scénarios que les tests automatiques pourraient ne pas vérifier, tels que des configurations spécifiques de données qui pourraient présenter des défis particuliers pour votre arbre de décision.

Chaque test personnalisé devra être accompagné d'une explication dans votre rapport, détaillant pourquoi vous avez choisi de tester ce scénario particulier, comment vous avez structuré le test, et ce que les résultats du test indiquent sur la robustesse et la performance de votre implémentation.

Partie 2 – Exploration

Dans cette partie du projet, vous apprendrez à explorer un arbre de décision et à effectuer des requêtes pour obtenir des informations spécifiques à partir des données. Cette exploration est fondamentale pour comprendre comment l'arbre prend des décisions et comment on peut interroger l'arbre pour extraire des connaissances.

Exploration de l'arbre

Nous vous demandons d'implémenter une fonction `display` qui parcourt l'arbre, et l'affiche sur le terminal. Nous vous laissons la liberté quant à la manière la plus judicieuse d'afficher un tel arbre et nous vous demandons de justifier vos choix dans le rapport.

Implémentation d'une requête

Après avoir compris comment parcourir l'arbre, l'étape suivante consiste à l'utiliser pour répondre à une requête spécifique. Dans le cadre de ce projet, nous vous demandons de mettre en œuvre une requête spécifique à l'aide de l'arbre de décision que vous avez construit. Nous vous demandons donc de trouver si un certain champignon avec des caractéristiques données est comestible ou non. Voici la signature de la fonction que nous vous demandons :
`is_edible(root: Node, mushroom: Mushroom) -> bool`

Transformation de l'arbre de décision en règles booléennes

Nous vous demandons de convertir l'arbre de décision que vous avez développé en un ensemble de règles booléennes. Cette transformation doit capturer toute la logique de décision de l'arbre de manière concise et claire.

Votre arbre de décision actuel utilise des attributs tels que l'odeur (*odor*), la couleur de la sporée (*spore print color*), l'habitat, etc. pour classifier la comestibilité des champignons. À partir de cet arbre, vous devrez créer une expression booléenne équivalente qui résume les chemins de décision.

Un exemple vous est donné ci-dessous.

```
odor = Almond
    Yes
odor = Pungent
    No
odor = Anise
    Yes
odor = None
    spore-print-color = Brown
        Yes
    spore-print-color = Black
        Yes
    spore-print-color = White
        habitat = Leaves
            cap-color = Cinnamon
                Yes
            cap-color = White
                No
            cap-color = Brown
                Yes
            cap-color = Yellow
                No
        habitat = Waste
            Yes
        habitat = Woods
            gill-size = Narrow
                No
            gill-size = Broad
                Yes
        habitat = Grasses
            Yes
        habitat = Paths
            Yes
    spore-print-color = Chocolate
        Yes
    spore-print-color = Green
        No
    spore-print-color = Orange
        Yes
    spore-print-color = Yellow
        Yes
    spore-print-color = Buff
        Yes
odor = Creosote
    No
odor = Foul
    No
odor = Fishy
    No
odor = Spicy
    No
odor = Musty
    No
```

Que nous pouvons réécrire comme :

```
((odor = Almond) OR (odor = Anise) OR (odor = None AND (
    (spore-print-color = Brown) OR (s-p-c = Black) OR
    (s-p-c = Chocolate) OR (s-p-c = Orange) OR
    (s-p-c = Yellow) OR (s-p-c = Buff) OR
    (s-p-c = White AND (
        (habitat = Waste) OR (habitat = Grasses) OR
        (habitat = Paths) OR
        (habitat = Woods AND (gill-size = Broad)) OR
        (habitat = Leaves AND (
            (cap-color = Cinnamon) OR (cap-color = Brown))
    )))
)))
```

Tests

Dans cette partie du projet, les tests seront essentiels pour s'assurer que les fonctionnalités implémentées fonctionnent comme prévu. Deux aspects principaux doivent être testés : l'implémentation de la requête et la conversion de l'arbre de décision en règles booléennes.

Tests de l'implémentation de la requête Trois tests seront fournis pour évaluer l'exactitude de votre fonction `is_edible`. Ces tests vérifieront si votre fonction peut correctement déterminer la comestibilité des champignons basée sur leurs attributs spécifiques, en utilisant l'arbre de décision que vous avez construit. Ces tests sont conçus pour couvrir différents cas de base et complexes pour assurer que votre implémentation répond correctement dans diverses situations. Il est crucial que votre code passe ces tests, car une défaillance pourrait signifier une note de 0 sur 10 pour cette section.

Tests de conversion en règles booléennes Vous devez également fournir au moins deux tests pour vérifier que les règles booléennes que vous générez à partir de l'arbre de décision reflètent fidèlement la logique de décision de l'arbre. Ces tests devraient vérifier que les règles peuvent être utilisées pour prédire correctement la comestibilité des champignons, tout comme l'arbre de décision le ferait.

Création de tests personnalisés En plus des tests fournis, vous êtes encouragé à développer et inclure dans votre rapport au moins deux tests personnalisés. Ces tests doivent être conçus pour évaluer des aspects de votre implémentation qui pourraient ne pas être couverts par les tests fournis. Dans le rapport, expliquez pourquoi ces tests sont pertinents, comment ils ont été structurés, et ce que leurs résultats suggèrent concernant la validité et l'efficacité de votre arbre de décision et des règles booléennes dérivées.

Bonus [2 points]

Nous proposons d'implémenter une fonction `to_python(dt: Node, path: str) -> None` qui prendra comme paramètre la racine de votre arbre de décision. Cette fonction sera chargée de générer un script Python qui reproduit la logique de l'arbre de racine `dt` en utilisant une série de structures conditionnelles if/else. Le script résultant sera sauvegardé dans un fichier Python spécifié par le chemin donné en paramètre. Cela pourrait être une manière amusante et instructive de visualiser le fonctionnement de l'arbre de décision dans un format de code exécutable.

Remise

Votre mission est d'appliquer les méthodes et fonctions nécessaires mentionnées précédemment et d'expliquer dans votre rapport les raisons de vos choix lorsque cela est requis. Vous devrez déposer votre projet sur l'Université Virtuelle.

Vous devez soumettre un fichier *.zip* qui contiendra un fichier *.py*, nommé *project.py*, incluant l'ensemble de votre code, un fichier *tests.py* regroupant les tests fournis ainsi que ceux que vous avez rédigés, et un fichier *.pdf*, nommé *rapport.pdf*, contenant toutes vos analyses.

Nous tenons à vous rappeler que la réalisation personnelle de vos projets est essentielle à votre apprentissage et à votre réussite universitaire. Si nous avons des raisons de croire que votre projet n'a pas été réalisé par vous-même, nous vous convoquerons pour une défense orale. Nous sommes conscients que certaines ressources peuvent être tentantes (e.g., AI), mais nous insistons sur le fait qu'une utilisation inappropriée de ces ressources peut nuire à votre compréhension de la matière et à votre avenir professionnel. Nous sommes là pour vous aider et vous guider, alors n'hésitez pas à nous contacter si vous avez des questions ou des préoccupations.

Le projet est à remettre pour le 07 mai 2024 à 23h59 sur l'UV. Tout manquement aux consignes ou retard sera sanctionné directement d'un 0/10.

Evaluation

Seront évaluées la qualité de votre code, la mise en pratique de la matière vue en cours, les optimisations mises en place et la qualité de vos tests. Veillez à ce que votre code soit commenté de manière concise pour mettre en valeur ses fonctionnalités et les optimisations appliquées. Dans le cas où l'exécution de votre code en ligne de commande produit une erreur, une note nulle sera reportée pour la partie exécution, veillez donc à bien tester votre code.