Rapport arbre de décision

Introduction

Les arbres de décision sont des outils d'apprentissage automatique largement utilisés pour la classification et la prédiction dans divers domaines tels que la science des données, l'intelligence artificielle et la recherche opérationnelle. Ils sont particulièrement attrayants en raison de leur capacité à représenter des décisions complexes de manière claire et explicite, ce qui les rend facilement interprétables par les humains.

Dans le cadre du cours d'algorithmique nous avons été charger de construire un arbre de décision qui a comme but de déterminer si un champignon est comestible ou non. Dans ce rapport nous tenterons d'analyser la structure du code qui consiste à examiner la dépendance entre les fonctions, l'utilité des fonctions auxiliaires, nous choix algorithmique et pour finir nous expliquerons nos choix de tests et leur implémentation.

La structure du code

Tout d'abord, nous avons la classe **Mushroom** Cette classe représente un champignon. Elle possède un attribut **edible** qui indique si le champignon est comestible ou non, ainsi qu'un dictionnaire **attributes** pour stocker ses caractéristiques. Les méthodes **is_edible, add_attribute** et **get_attribute** permettent respectivement de vérifier si le champignon est comestible, d'ajouter une caractéristique et de récupérer la valeur d'une caractéristique.

La fonction **load_dataset** est essentielle pour charger les données à partir d'un fichier CSV et les préparer pour la construction de l'arbre de décision. Cette fonction créer une liste de d'objet champignon.

Ensuite nous avons la classe Node, cette classe représenter un nœud de l'arbre. Chaque nœud a un critère de division **criterion_** et un indicateur **leaf** indiquant s'il s'agit d'une feuille de l'arbre. La méthode **is_leaf** permet de vérifier si le nœud est une feuille, et la méthode **add_edge** permet d'ajouter une arête sortante vers un autre nœud.

La classe Edge représente une arête reliant deux nœuds de l'arbre de décision. Chaque arête a un nœud parent **parent_**, un nœud enfant **child_** et un libellé **label_** indiquant le critère de division associé à cette arête.

Pour l'arbre de décision, chaque fonction joue un rôle spécifique dans le processus global. Voici une explication de la dépendance entre ces fonctions et l'utilité de chacune.

Premièrement, nous avons la fonction Entropie qui calcule l'entropie a l'aide de cette formule donner dans l'énonce.

$$H(\mathcal{C}) = \begin{cases} 0 & \text{si } p_Y = 0 \text{ ou } p_Y = 1 \\ p_Y \log_2 \dfrac{1-p_Y}{p_Y} - \log_2(1-p_Y) & \text{sinon.} \end{cases}$$

Deuxièmement, nous implémentons **calculates_p_y(mushrooms)** qui calcule le P_y qui dépend du nombre de champignon comestible divisée par le nombre de champignon totale. Elle est utilisée comme entropie parente lors du calcul du gain d'information pour déterminer quel attribut diviser.

Troisièmement, la fonction **information_gain(parent_entropy, subsets)** le gain d'information. Elle compare l'entropie du parent (avant la partition) à l'entropie pondérée des sous-ensembles (après la partition).

Quatrièmement, **calculate_information_gain_for_attribute(mushrooms, attribute)** qui calcule le gain d'information pour un attribut spécifique en partitionnant les données selon cet attribut et en utilisant l'entropie donné par **calculates_p_y(mushrooms)** comme entropie parente.

Cinquièmement, nous avons **choose_best_attribute(mushrooms)** calcule le gain d'information de tous les attributs et renvoie le meilleur.

Enfin, nous avons **build_decision_tree(mushrooms: list[Mushroom])** Cette fonction utilise récursivement le gain d'information pour construire l'arbre de décision. Elle sélectionne d'abord le meilleur attribut pour diviser les données à chaque étape, puis répète le processus sur les sous-ensembles résultants jusqu'à ce que les feuilles de l'arbre soient des 'Yes' ou 'No.

L'emploi d'une fonction récursive dans **build_decision_tree** est justifié par la nature arborescente inhérente à la construction de l'arbre de décision. Dans cette structure, chaque nœud représente une décision basée sur un attribut et chaque branche correspond à une valeur de cet attribut. En adoptant une approche récursive, nous pouvons efficacement traverser cette structure arborescente, en examinant chaque niveau de l'arbre jusqu'à ce que nous atteignions les feuilles.

Cette même approche récursive a été utilisé pour la fonction **is_edible** , juste le cas de base est que lorsque on tombe sur une feuille le programme s'arrête et détermine si le champignon est comestible ou pas. De même , la fonction **display** utilise la récursivité pour explorer la structure de l'arbre de décision, en affichant chaque nœud et chaque arête de manière itérative et indentée, le cas de base ici est lorsque on se situe sur une feuille on print simplement 'Yes'' or No'. Cela permet d'obtenir une représentation visuelle claire et compréhensible de l'arbre de décision.

Test de vérification des types de données

Ce test vise à vérifier que les données chargées à partir du fichier CSV sont correctement typées, c'est-à-dire que tous les attributs sont des chaînes de caractères et que le label est un booléen. Cette vérification est importante car elle garantit que les données sont conformes aux attentes de l'algorithme d'arbre de décision qui pourrait s'appuyer sur ces types de données pour prendre des décisions.

Méthode

Nous itérons sur chaque instance de champignon chargée à partir du fichier CSV. Pour chaque champignon, nous vérifions que tous les attributs sont des chaînes de caractères en utilisant la fonction isinstance(value, str) pour chaque valeur attribut.

Nous vérifions également que le label est un booléen en utilisant la fonction isinstance(mushroom.label, bool). Si l'une de ces vérifications échoue pour au moins une instance de champignon, le test échoue, ce qui indique qu'il y a un problème avec le chargement des données en termes de types de données.

Test de validation des valeurs manquantes

Ce test vise à vérifier qu'aucune valeur d'attribut n'est manquante dans les données chargées à partir du fichier CSV. La présence de valeurs manquantes pourrait entraîner des erreurs lors de la construction de l'arbre de décision, il est donc important de s'assurer que toutes les valeurs sont présentes et complètes.

Méthode

Nous itérons sur chaque instance de champignon chargée à partir du fichier CSV.

Pour chaque champignon, nous vérifions s'il y a une valeur manquante dans ses attributs en utilisant la fonction any(value is None for value in mushroom.attributes.values()).

Si une valeur manquante est trouvée pour au moins une instance de champignon, le test échoue, ce qui indique qu'il y a des valeurs manquantes dans les données chargées.

Ces deux tests complémentaires fournissent une vérification supplémentaire de la qualité du chargement des données et aident à garantir l'intégrité des données utilisées pour construire l'arbre de décision. Ils permettent également de détecter rapidement les problèmes éventuels dans le processus de chargement des données, ce qui facilite le débogage et l'amélioration de l'implémentation

Test de la classe Node

Ce test évalue la classe **Node**, qui est responsable de la représentation des nœuds de l'arbre de décision. L'objectif de ce test est de vérifier le bon fonctionnement de la classe **Node** et de s'assurer que ses attributs et méthodes sont correctement implémentés car s'il y a des erreurs cela peut nuire à la construction de l’arbre.

Méthode

Dans un premier temps, le test crée deux instances de la classe **Node** : un nœud parent avec le critère "odor" et un nœud enfant avec le critère "Almond". Le nœud enfant est marqué comme une feuille (**is_leaf=True**), indiquant qu'il n'a pas d'enfants.

Ensuite, le test appelle la méthode **add_edge** sur le nœud parent pour ajouter une arête reliant le nœud parent au nœud enfant avec le libellé "Almond".

Au niveau des vérifications, nous avons 5 assert, le premier vérifie la méthode .**criterion**_ qui doit renvoyer ‘odor’. Puis nous testons la méthode **is_leaf** avec un assertFalse car ‘odor’ est un critère de division et non une feuille. Par la suite nous vérifions si un élément a bien été ajouter aux en enfants du node avec un assertequal qui vérifie le la longueur de la liste des edges est bien 1. Ensuite nous regardons si l’enfant ajouté un bien ‘Almond’ et pour finir nous testons si l’enfant est bien égal avec le child_node.

Test du calcul de gain d’information

Cette vérification a comme but tester si le calcul du gain d’information est effectué correctement car une erreur de calcul peut entrainer des failles lors de la construction de l’arbre de décision nous devons donc vérifier le bon fonctionnement du gain d’information.

Méthode

Nous faisons appel ici à la fonction **calculate_information_gain** qui calcul le gain, il prend en paramètre la liste de champignons chargée à partir du fichier csv et un attribut. Dans ce test nous avons décidé de faire 2 vérifications une avec ‘odor’ et une avec ‘spore-print-color' pour renforcer la crédibilité du test.

Cela renforce la fiabilité de l'algorithme de construction de l'arbre de décision, car le calcul précis du gain d'information est essentiel pour prendre les décisions appropriées lors de la sélection des attributs pour diviser l'ensemble de données. En garantissant que le calcul du gain d'information est correct, nous avons une assurance. Supplémentaire de la qualité de l'arbre de décision généré et de sa capacité à classifier correctement les champignons comme comestibles ou non comestibles.

Test d’un 2ieme attribut de division de l’arbre

Ce test cherche à vérifier un deuxième critère de division de l’arbre. Ce test vérifie si l'arbre de décision prend en compte efficacement les critères de division successifs pour classifier les données. Dans ce contexte, nous évaluons la capacité de l'arbre à sélectionner et à utiliser correctement un deuxième attribut de division après avoir évalué le premier attribut.

Méthode

Nous avons légèrement changé t**est_tree_main_attribute,** en effet nous avons supprimé tous les éléments dans liste nos et on a rajouté ‘None’ car ce n’est pas une caractéristique qui déterminé si le champignon est comestible ou pas.

Ensuite, nous avons simplement repris la structure du test précèdent en changeant la liste nos.

Test d’arbre de décision simple

Ce test vise à vérifier la fonction **decision_tree_to_boolean_expression** pour un arbre de décision simple. L'objectif est de s'assurer que la fonction génère correctement une expression booléenne à partir de l'arbre de décision donné.

Méthode

Dans ce test, nous construisons un arbre de décision simple avec un seul attribut de division.Nous utilisons ensuite la fonction **decision_tree_to_boolean_expression** pour générer une expression booléenne à partir de cet arbre. Enfin, nous comparons l'expression générée avec une expression attendue pour vérifier si elles correspondent.

Test de l'arbre de décision complexe

Ce test vise à vérifier la fonction **decision_tree_to_boolean_expression** pour un arbre de décision complexe qui contient plusieurs critères de division. L'objectif est de s'assurer que la fonction génère correctement une expression booléenne à partir de l'arbre de décision donné.

Méthode

Dans ce test, nous construisons un arbre de décision complexe avec plusieurs niveaux et différentes valeurs d'attributs. Nous utilisons ensuite la fonction **decision_tree_to_boolean_expression** pour générer une expression booléenne à partir de cet arbre. Enfin, nous comparons l'expression générée avec une expression attendue pour vérifier si elles correspondent.