Rapport du Projet d'Algorithmique 1. Exploration et Interrogation d'un Arbre de Décision

Stéphane Badi Budu

7 Mai 2024

1 Introduction

Dans ce projet, nous avons dû implémenter un code qui nous permettait de créer, explorer et interroger un arbre de décision. Par la suite, nous avons dû afficher cet arbre de deux façons :

1. D'une façon classique grâce à la fonction "Display".
2. Comme un arbre booléen qui ressemble à une longue expression logique permettant de déterminer si un champignon est comestible ou non.

Enfin, nous avons décidé de réaliser la fonction bonus nommée "to_python", qui crée un script explicite correspondant à l'arbre de décision créé et permet de déterminer si un champignon est comestible ou non.

Dans ce rapport, nous allons vous expliquer comment nous nous sommes pris pour réaliser ce projet.

2 Chargement des données

Pour le chargement des données, nous utilisons la fonction "load_dataset" qui récupère tous les éléments du fichier mushrooms.csv dans une matrice. Par la suite, nous créons les champignons de type Mushroom en lisant toutes les listes de la matrice sauf la première, qui contient tous les attributs des champignons. Elle va nous être utile pour ajouter les différentes caractéristiques des champignons via la méthode "add_attribute".

3 Création de l'arbre de décision

3.1 Calcul de l'entropie

Pour créer l'arbre de décision, nous avons besoin de calculer l'entropie non seulement de l'ensemble des champignons, mais aussi des champignons en fonction de la valeur de l'attribut étudié. Pour ce faire, nous avons implémenté une fonction "entropy" qui prend en paramètre une liste de champignons et qui retourne l'entropie de cette liste si la proportion de champignons comestibles dans cette liste est différente de 0 et 1. Sinon, elle retourne 0.

Si ce n'est pas le cas, on retourne 0 et on ne calcule pas l'entropie.

En ce qui concerne la formule de l'entropie, je l'ai simplifiée grâce à la règle de simplification des logarithmes :

$$\begin{aligned} \text{Entropie} &= p_Y \times \log_2\left(\frac{1-P_Y}{p_Y}\right) - (1-p_Y) \times \log_2(1-p_Y) \\ &= p_Y(\log_2(1-P_Y) - \log_2(p_Y)) - \log_2(1-p_Y) \\ &= p_Y \times \log_2(1-p_Y) - p_Y \times \log_2(p_Y) - \log_2(1-p_Y) \\ &= \log_2(1-p_Y) \times (p_Y - 1) - p_Y \times \log_2(p_Y) \end{aligned} \tag{1}$$

4 Calcul du gain d'information et de l'attribut à choisir

Pour calculer le gain d'information, nous transmettons à notre fonction "get_information_gain" l'attribut à étudier et la liste des champignons. En fonction des valeurs de l'attribut, on effectue un produit de l'entropie de la liste de champignons ayant la valeur à traiter avec la proportion de champignons comestibles qu'il y a dans cette liste par rapport à l'ensemble des champignons reçus. Pour gérer la création de listes ne contenant les champignons qui ont la valeur "v", nous avons créé une fonction "Ca_v" qui retourne un dictionnaire ayant comme clés les valeurs possibles de l'attribut étudié et comme valeurs les listes de champignons ayant cette valeur. La somme de ces produits effectués pour chaque valeur de l'attribut sera soustraite à l'entropie de la liste de champignons initiale, ce qui nous donne le gain d'information de l'attribut. Pour trouver le meilleur attribut à choisir, nous calculons le gain d'information de chaque attribut et nous choisissons celui qui a le gain d'information le plus élevé.

5 Les classes Node et Edge

Dans la classe Node, nous stockons l'attribut choisi pour diviser la liste de champignons, les valeurs possibles de cet attribut, un booléen "is_leaf" qui spécifie si le nœud est une feuille ou non, et une liste des arêtes liées au nœud en question dans une liste d'éléments de type Edge. Dans la classe Edge, nous stockons la valeur de l'attribut nommé "label", le nœud parent de type "Node" qui est celui qui divise au mieux la liste des champignons, et le nœud enfant qui est celui qui est lié à l'arête.

6 Création de l'arbre

Pour créer des arbres, nous avons dû implémenter une fonction récursive "build_decision_tree" qui prend en paramètre une liste de champignons et retourne un arbre de décision de type Node. Pour commencer, nous trouvons l'attribut qui divise au mieux la liste de champignons en faisant appel à la fonction "get_best_attribute". Ensuite, nous créons un nœud de type "Node" avec cet attribut et nous ajoutons les arêtes correspondant à chaque valeur possible de l'attribut. Pour chaque arête, si l'entropie de la liste d'éléments avec la valeur traitée est nulle, nous créons une feuille avec la valeur de l'attribut et nous spécifions si le champignon est comestible ou non. Sinon, nous appelons récursivement la fonction "build_decision_tree" avec la liste de champignons ayant la valeur traitée et nous ajoutons le nœud retourné comme enfant de l'arête.

7 Affichage de l'arbre

7.1 Affichage classique

Pour afficher l'arbre de manière classique, nous avons implémenté une fonction "display" qui prend en paramètre un nœud de type "Node" qui est la racine de l'arbre. Nous avons également implémenté une fonction imbriquée "display_r" qui prend en paramètre un nœud et un entier "tab" qui est utilisé pour afficher les tabulations. Pour chaque arête du nœud, nous affichons la valeur de l'attribut de l'arête et nous appelons récursivement la fonction "display_r" avec le nœud enfant et "tab + 1". Et lorsque nous tombons sur une feuille, nous affichons la valeur de l'attribut et si le champignon est comestible ou non avec une tabulation en plus.

7.2 Affichage booléen

Pour afficher l'arbre de manière booléenne, nous avons implémenté une classe BooleanTree qui stocke le nœud racine de l'arbre courant, une instance tab qui est utilisée pour afficher les tabulations, une liste des arêtes menant à un champignon comestible, l'arête courante et une instance qui compte le nombre de d'arêtes traités. La fonction "boolean_tree" prend en paramètre un nœud de type "Node" qui est la racine de l'arbre et une chaine de caractère qui aura toute la séquence logique de l'arbre. La fonction fait appel à une fonction imbriquée "boolean_tree_r" qui prend une instance de type "BooleanTree" et une chaine de caractère vide par défaut. Pour chaque feuille de l'arbre, nous avons un OR qui le suit grâce à la fonction "print_leaf" qui prend en paramètre une instance de type "BooleanTree" et une chaine de caractère vide par défaut. Pour chaque arête de l'arbre, nous avons un AND qui le suit puis une grande parenthèses des différents cas où le champignon est comestible grâce à la fonction "print_not_leaf" qui a les mêmes paramètres que "print_leaf".

8 Fonction bonus "to_python"

Pour la fonction bonus "to_python", nous avons créé une classe PythonScript qui stocke le nœud racine de l'arbre courant, une instance tab qui est utilisée pour afficher les tabulations, une liste des arêtes menant à un champignon comestible, l'arête courante et un booléen qui spécifie si un if a été mis dans le script. La fonction "to_python" prend en paramètre un nœud de type "Node" qui est la racine de l'arbre et une chaîne de caractères qui est le nom du fichier où sera écrit le script python. Dans la fonction, nous créons une chaîne de caractères qui contiendra le script python et nous faisons appel à une fonction imbriquée "to_python_r" qui prend une instance de type "PythonScript" et une chaîne de caractères vide par défaut. Pour les nœuds qui ne sont pas des feuilles, nous faisons un print de la valeur de l'arête courante grâce à la fonction "print_if_not_leaves" et nous appelons récursivement la fonction "to_python_r" avec le nœud enfant. Pendant ce temps, tous les nœuds menant à une feuille de

champignon comestible sont stockés dans la liste de l'instance de type "PythonScript". À la fin de la boucle qui itère sur les arêtes conservées, nous faisons un print de la condition qui permet de déterminer, par rapport à l'attribut traité, si le champignon est comestible ou non suivant les valeurs stockées dans la liste de PythonScript.

9 Tests

9.1 Tests sur la partie 1

Pour les tests de la partie 1, nous avons décidé de tester les fonctions d'entropie et de gain d'information. Ce sont des fonctions cruciales car si elles sont fausses ou retournent les mauvaises valeurs, notre arbre de décision sera faux. Pour ce faire, nous testons l'entropie avec une liste contenant que des champignons comestibles, une liste contenant que des champignons non comestibles pour voir si la fonction retourne bien 0 dans les deux cas. Ensuite, nous testons l'entropie avec une liste de champignons reçue via "load_dataset" pour voir si la valeur qu'elle est censée retourner est bien celle dans le test. Pour la fonction de gain d'information et la fonction de meilleur attribut, nous avons testé avec une liste de champignons reçue via "load_dataset" pour voir si la fonction retourne bien l'attribut 'odor' qui est celui qui divise le mieux la liste de champignons. Ensuite, nous modifions la liste et nous conservons que ceux qui ont l'attribut 'odor' égal à 'None' pour voir si la fonction retourne bien l'attribut 'spore-print-color' qui divise le mieux cette liste. Dans les deux cas, nous vérifions si le gain d'information est bien celui attendu.

9.2 Tests sur la partie 2

Pour les tests de la partie 2, nous avons décidé de tester la fonction "is_edible", le bonus "to_python" et la fonction "display". Pour tester la fonction "is_edible", nous avons créé un champignon avec une odeur 'almond' censé être comestible et un champignon inodore de couleur verte censé être non comestible. Pour tester la fonction "to_python", nous utilisons l'arbre de décision avec la liste de tous les champignons et nous regardons si le script retourne bien la bonne réponse en fonction des champignons qu'on lui transmet. Pour tester la fonction "display", nous réutilisons l'arbre de décision que nous parcourons en profondeur. On compare dans l'ordre la comestibilité des champignons et la disposition des arêtes dans l'arbre.

9.3 Tests sur l'arbre booléen

Pour tester l'arbre booléen, nous générons dans le premier test 3 arbres différents et nous comptons le nombre de parenthèses ouvrantes et fermantes pour voir si le nombre est bien égal. Dans le deuxième test, nous utilisons un arbre que nous parcourons en profondeur et nous vérifions si la séquence logique est bien celle attendue.

10 Conclusion

Pour conclure, nous avons réussi à implémenter un arbre de décision qui permet de déterminer si un champignon est comestible ou non. Nous avons également réussi à afficher cet arbre de deux façons différentes et à créer un script python qui permet de déterminer si un champignon est comestible ou non. Par la suite nous avons fait des tests pour vérifier si notre implémentation répond bien à ce que nos attentes. Le projet était amusant, intéressant et nous a permis de mettre en pratique les notions vues telles que les arbres, la dérécursification, le parcours d'arbres et la récursivité.