Rapport de projet

PROJ631 : Projet Algorithmique

Arbres de décision : de Id3 à C4.5

Lien GitHub : https://github.com/SimonGuilbert/DecisionTree

Enseignants

Galichet Sylvie

Huget Marc-Philippe

Mauffret Etienne

Vernier Flavien

Simon Guilbert

Table des matières

Introduction

Dans l'introduction de ce rapport, je vais expliquer certaines choses sur les arbres de décision. Pour cela, je vais m'appuyer sur les fichiers golf.csv et non pas soybean.csv car ces derniers ne contiennent que des termes qu'il est difficile de comprendre et d'expliquer si l'on n'est pas expert en botanique ou en biologie végétale.

A quoi sert un arbre de décision ?

Un arbre de décision est utilisé dans le cas où nous avons deux fichiers de données distincts et semblables à ceux-ci :

Outlook	Temperature	Humidity	Windy	Les conditions climatiques permettent-elles de jouer au golf ?
sunny	hot	high	false	**Non**
sunny	hot	high	true	**Oui**
overcast	hot	high	false	**Oui**
rain	mild	high	false	**Oui**
rain	cool	normal	false	**Oui**
rain	cool	normal	true	**Non**
overcast	cool	normal	true	**Oui**
sunny	mild	high	false	**Non**

Figure 1 - Fichier 1 : Ensemble d'apprentissage

Outlook	Temperature	Humidity	Windy	Les conditions climatiques permettent-elles de jouer au golf ?
sunny	cool	normal	false	**?**
rain	mild	normal	false	**?**
sunny	mild	normal	true	**?**
overcast	mild	high	true	**?**

Figure 2 - Fichier 2 : Ensemble de test/prédiction

Le premier fichier de données détaille les conditions météorologiques pour un jour donné à travers 4 *attributs* : **Outlook**, **Temperature**, **Humidity** et **Windy**. Chacun de ces *attributs* peut prendre différentes valeurs, par exemple sunny, rain ou overcast pour l'attribut **Outlook** ou encore true ou false pour l'attribut **Windy**. Chaque ligne du fichier de données correspond à un jour différent et ces lignes seront appelées dans la suite de ce rapport des objets ou bien des exemples. La cinquième colonne fonctionne de la même manière que les 4 premières, c'est-à-dire qu'on observe pour un jour donné si des gens se sont aventurés sur des terrains de golf ou pas et on note Oui ou Non. Ce premier fichier de données est appelé l'ensemble d'**apprentissage**.

Dans le 2ème fichier de données, comme pour le 1er, on observe et on attribue une valeur pour chacun des 4 *attributs* pour un jour donné. Mais cette fois-ci, nous ne connaissons la valeur de la cinquième colonne. En effet, ce que l'on veut maintenant c'est justement prédire si, en se basant sur les 4 éléments météorologiques, les conditions sont bonnes pour jouer au golf aujourd'hui ou pas. Cette information pourrait être utile pour estimer le nombre de clients potentiels par exemple, ou encore pour avoir une idée du soin nécessaire à la tonte de la pelouse ce matin. Ce deuxième fichier de données est appelé l'ensemble de **prédiction**.
C'est comme si on s'entraînait au lancer de javelot. La toute première fois où on lance un javelot à plusieurs dizaines de mètres, on n'a aucune idée précise de la distance nous séparant du point d'impact du javelot avec le sol. Mais au bout d'un certain nombre de lancers, on pourra plus facilement **prédire** cette distance grâce à l'**apprentissage** lors des précédents lancers.

Pour en revenir aux fichiers de données, la cinquième et dernière colonne fournit en fait l'information sur ce que l'on appelle une *classe*. Les valeurs "Oui" et "Non" visibles dans la cinquième colonne du 1er fichier ne s'appellent pas des *valeurs d'attributs* mais plutôt des *classes*. C'est ici que l'arbre de décision intervient : son rôle sera de déterminer la meilleure *classe possible* pour chaque objet de l'ensemble de prédiction en se servant des valeurs des *attributs*.

Fonctionnement d'un arbre de décision

Un arbre de décision est en fait une arborescence dont les nœuds sont des *attributs,* les arcs sont les *valeurs* possibles des attributs et les feuilles sont les *classes*. Voici à quoi il peut ressembler dans notre exemple :

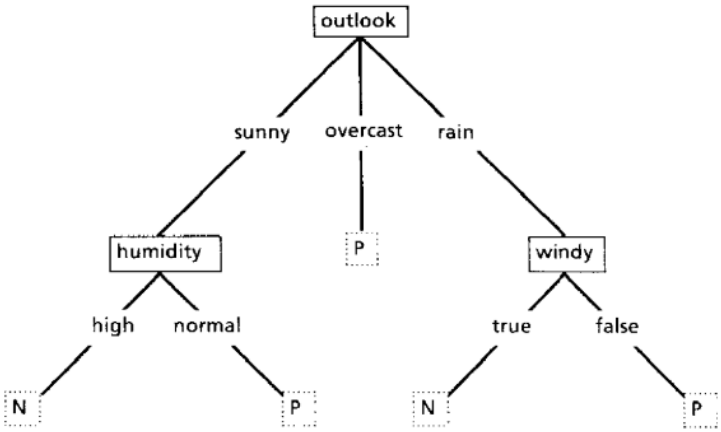

Figure 3 - Première représentation de l'arbre de décision

Cet arbre a été créé en se basant sur l'ensemble d'apprentissage (l'ensemble où on connaît la *classe* de chaque objet). Une fois créé, il est très simple de déterminer la classe la plus adaptée pour un objet de l'ensemble de prédiction. Par exemple, si on considère l'objet suivant

Outlook	Temperature	Humidity	Windy	Les conditions climatiques permettent-elles de jouer au golf ?
sunny	cool	normal	false	**?**

Figure 4 - un objet de l'ensemble de prédiction dont on ne connaît pas la classe

Il suffit de suivre le chemin de l'arbre de la Figure 3 en se guidant avec les valeurs des attributs de notre objet :

- On part du nœud racine **Outlook**
- Quelle est la valeur de l'attribut **Outlook** pour notre objet ?
- Sunny → on prend le chemin de gauche
- On arrive au nœud **Humidity**
- Quelle est la valeur de l'attribut **Humidity** pour notre objet ?
- normal → on prend le chemin de droite
- On arrive à la feuille "P" (qui correspond à la classe "Oui")
- C'est terminé, la classe retenue est "Oui"

Outlook	Temperature	Humidity	Windy	Les conditions climatiques permettent-elles de jouer au golf ?
sunny	cool	normal	false	**Oui**

Figure 5 - un objet de l'ensemble de prédiction avec sa classe prédite

L'algorithme à créer pour simuler ce que l'on vient de faire pour déterminer la classe est simple, rapide et efficace. L'algorithme le plus compliqué à faire est celui qui crée l'arbre de décision à partir de l'ensemble d'apprentissage. L'une des principales difficultés est de déterminer l'ordre des nœuds. Pourquoi le nœud racine choisi correspond à l'attribut **Outlook** et pas à **Temperature** par exemple ? Tout simplement parce que sinon l'arbre aurait ressemblé à ceci :

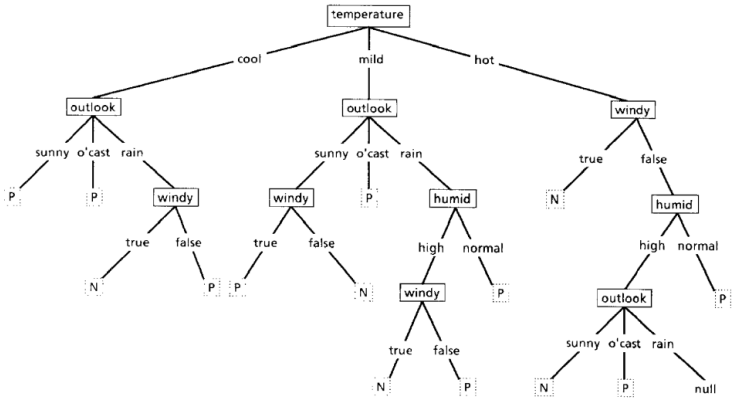

Figure 6 - une deuxième représentation de l'arbre de décision

L'arbre visible sur la Figure 6 fonctionne aussi bien que celui de la Figure 3 pour traiter les données de l'ensemble d'apprentissage mais il a plusieurs inconvénients :

- Il est plus complexe et donc compliqué à comprendre
- Il a de fortes chances de ne pas être aussi performant que l'arbre de la Figure 3 pour classer les données ne provenant pas de l'ensemble d'apprentissage. Typiquement, les données de l'ensemble de prédiction seront moins bien classées[1]

L'une des solutions qui existe pour créer l'un des meilleurs arbres de décision possibles est d'utiliser l'algorithme Id3.

L'algorithme Id3

Création du nœud racine

La première étape de l'algorithme correspond à créer le nœud racine de l'arbre. Il faut :

- Lire le fichier .csv
- Insérer les données dans un ArrayList<ArrayList<String>> appelé donnees
- A partir des données, créer :
    - Arbre.listeClasse (liste des classes)
    - Arbre.dicAttributs (dictionnaire ayant pour clé un attribut et pour valeur une liste des valeurs possibles pour cet attribut)
- Calculer le meilleur gain qui déterminera la nature du nœud

Voici le schéma de cette étape en prenant pour exemple les données du fichier golf.csv :

[1] D'après J.R. Quinlan dans son livre "Induction of Desicion Trees" page 87, 2ème paragraphe

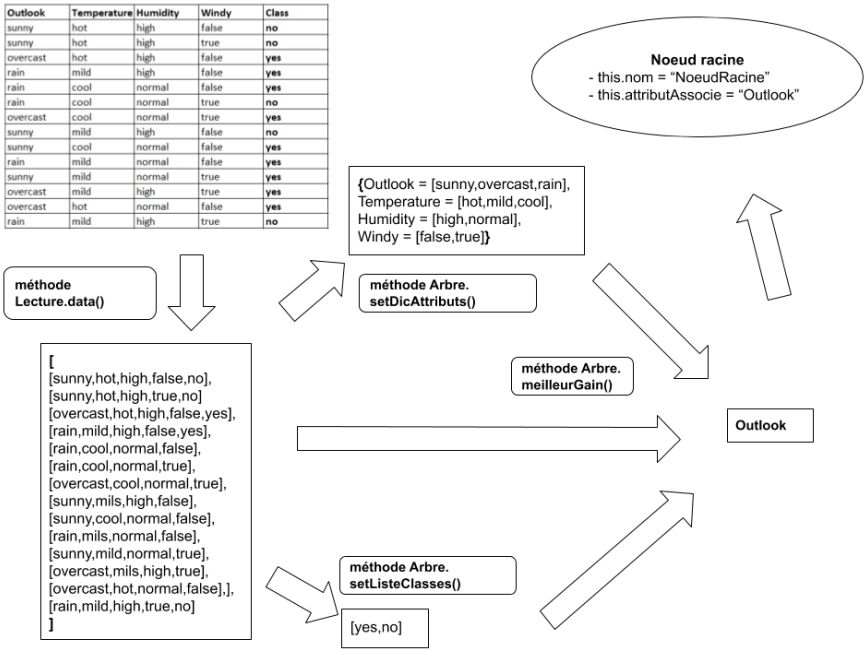

Figure 7 - Processus de création du nœud racine

Le calcul du meilleur gain se fait grâce aux fonctions I et E trouvées à la page 89 de l'article de référence du projet :

$$I(p, n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

et

$$E(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(p_i, n_i)$$

Ici, l'exemple ne fonctionne que pour 2 classes nommées p et n et il faut évidemment l'adapter au nombre de classes étudiées.

Construction du reste de l'arbre

Une fois le nœud racine créé, les nœuds fils sont ajoutés par la fonction récursive Arbre.setFils(). Avant chaque nouvelle création d'un nœud, l'algorithme divise l'ensemble des données (stocké dans Arbre.donnees) en plusieurs sous-ensembles grâce à la méthode Arbre.sousEnsemble(). Ce schéma illustre son fonctionnement :

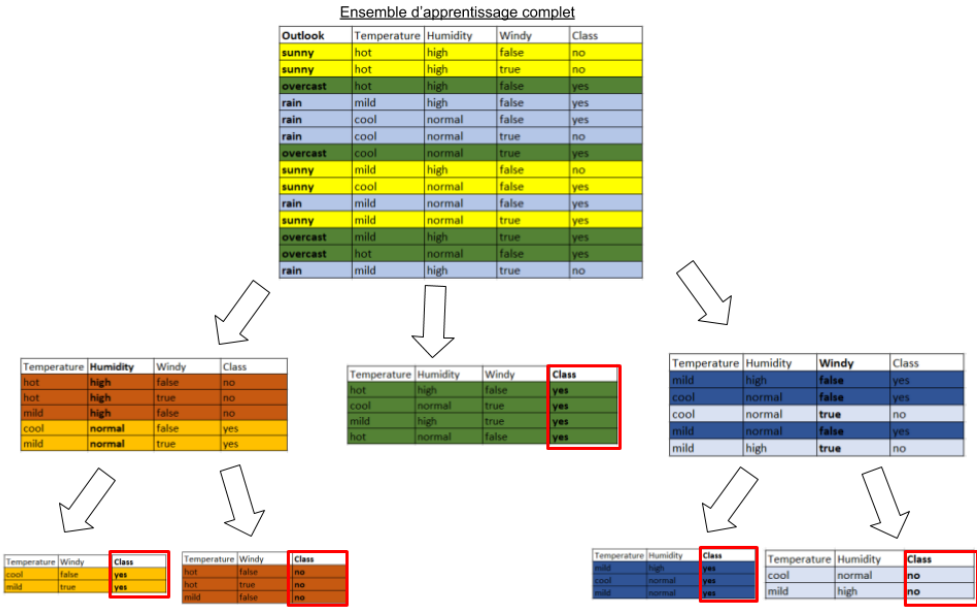

Figure 8 : Fonctionnement de la méthode sousEnsemble()

Lorsque tous les exemples d'un sous-ensemble sont de la même classe (la méthode Arbre.classesIdentiques() permet de le savoir), la récursivité s'arrête (dans la méthode setFils()) et le prochain nœud sera une feuille dont attributAssocie portera le nom de cette unique classe. C'est pourquoi aucun nouveau sous-ensemble n'est créé sur la Figure 8 lorsque c'est toujours la même classe (encadré en rouge).

Une amélioration : l'algorithme C4.5

L'utilisateur a le choix entre deux algorithmes.

Figure 9 - Choix de l'arbre

L'algorithme C4.5 se base beaucoup sur l'algorithme Id3, et on peut noter principalement 4 avantages :

- Adaptation de la fonction du gain d'information
- Gestion des attributs à valeurs continues
- Post-étalage de l'arbre
- Gestion des attributs à valeurs manquantes

Deux de ces fonctionnalités ont été traitées :

- Adaptation de la fonction du gain d'information
- Gestion des attributs à valeurs manquantes

Adaptation de la fonction du gain d'information

Ce premier ajout apporte peu de changement par rapport à l'ancienne méthode du gain d'information mais il permet d'améliorer un peu le calcul, notamment en évitant aux attributs possédant beaucoup de valeurs de biaiser le résultat[2]. L'amélioration consiste à diviser le gain d'information par une fonction appelée IV mais seulement dans le cas où la valeur du gain est supérieure à la moyenne des gains de tous les attributs (la méthode Arbre.moyenne calcule cette moyenne).

$$IV(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p+n} \log_2 \frac{p_i + n_i}{p+n}$$

Là encore, il faut adapter la formule au nombre de classes que l'on étudie.

[2] D'après l'article de référence de JR Quinlan, page 101

Gestion des valeurs manquantes

Il existe quatre manières de traiter les valeurs manquantes, et ce choix est donné à l'utilisateur dans le cas où au moins une valeur d'un des fichiers de données (soybean-app ou soybean-pred) est égale à un point d'interrogation.

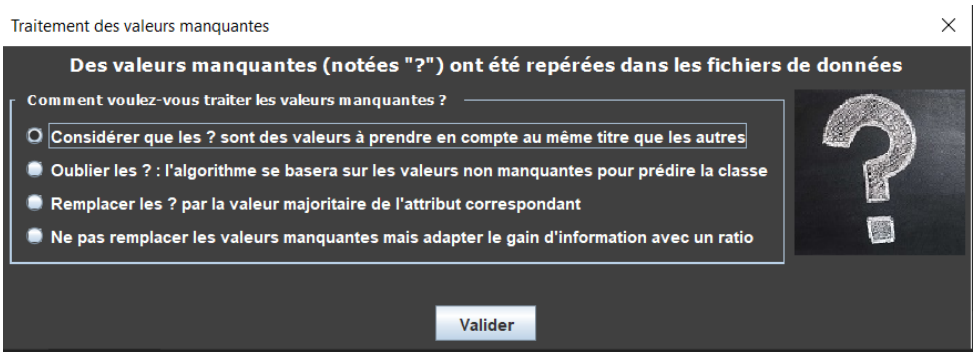

Figure 10 - Choix du traitement des valeurs manquantes

La première option permet de faire en sorte que les ? soient considérés comme une vraie valeur qui a la même importance que n'importe quelle autre valeur. Par exemple, si au moins un des objets comporte une valeur manquante pour l'attribut Outlook, une partie du dicAttributs associé à un arbre à son initialisation aura la forme :

{..., Outlook=[sunny,overcast,rain,**?**], …}

Cela signifie qu'au moins un noeud de l'arbre aura pour attributAssocie un point d'interrogation.
La deuxième option permet tout simplement de faire comme si les points d'interrogation n'existaient pas. Même s'il y a des valeurs manquantes pour l'attribut Outlook, le dicAttributs sera de la forme :

{..., Outlook=[sunny,overcast,rain], …}

Et comme la création des noeuds via la méthode Arbre.setFils() se base essentiellement sur ce dictionnaire, les données ayant des valeurs manquantes sont perdues et ne serviront pas à créer de nœud fils. Pour combler ce manque d'information au moment du parcours de l'arbre pour prédire la classe, un fils aléatoire est choisi parmi ceux du nœud posant problème pour que le parcours de l'arbre puisse se poursuivre. Cela expliquera en partie les différences dans les matrices de confusion même si on lance plusieurs fois le même programme avec les mêmes paramètres.

La troisième possibilité permet de remplacer les valeurs manquantes par la valeur de l'attribut associé majoritaire recherché dans Arbre.donnees. Ce remplacement est fait par la méthode Arbre.changeAttributMajoritaire().

Enfin, le dernier choix permet de prendre en compte les valeurs manquantes (mais sans les remplacer) pour le calcul du gain d'information en faisant intervenir un « ratio ».[3]

En se basant sur les pourcentages de la Figure 12 de quelques tests réalisés, voici les résultats obtenus (le pourcentage est donné à titre indicatif : ce n'est pas toujours le même) :

[3] Méthode expliquée à la page 98 de l'article de référence

Arbre	Traitement valeurs manquantes	% Apprentissage	% Prédiction
Id3	Choix 1	99.78%	90.78%
Id3	Choix 2	85.05%	71.92%
Id3	Choix 3	99.78%	79.82%
Id3	Choix 4	84,83%	73.24%
C4.5	Choix 1	99.78%	93.85%
C4.5	Choix 2	83.95%	75.43%
C4.5	Choix 3	99.78%	81.57%
C4.5	Choix 4	85.27%	77.19%

Figure 11 - Résultats observés

Le pourcentage montre la proportion d'objets dont la classe a été correctement prédite. Le choix 2 ne semble pas être une bonne idée au vu des résultats, ce qui est logique puisque on perd un certain nombre d'informations. Le choix 1 obtient les meilleurs résultats, mais là encore je ne sais pas s'il est convenable de considérer des valeurs manquantes comme de "vraies" valeurs. La forte ressemblance entre les résultats obtenus avec les choix 2 et 4 viennent du fait que dans ce fichier de données en particulier, les valeurs inconnues sont très souvent regroupées sur les mêmes lignes, ce qui réduit l'impact du ratio. Enfin la dernière chose que l'on peut observer est que l'algorithme C4.5 semble fournir de meilleurs résultats que l'algorithme Id3 dans tous les cas.

Matrices de confusion

Une fois l'arbre de décision parcouru et la classe prédite (méthode getPredClasse()), le résultat sous forme de deux listes (une pour les vraies classes et une une autre pour la prédiction) sert de constructeur à un objet Matrice. L'appel de la méthode Matrice.getMatrice() associée à la méthode Fenetre.creationJLabels() de la classe Fenetre fournit ce résultat :

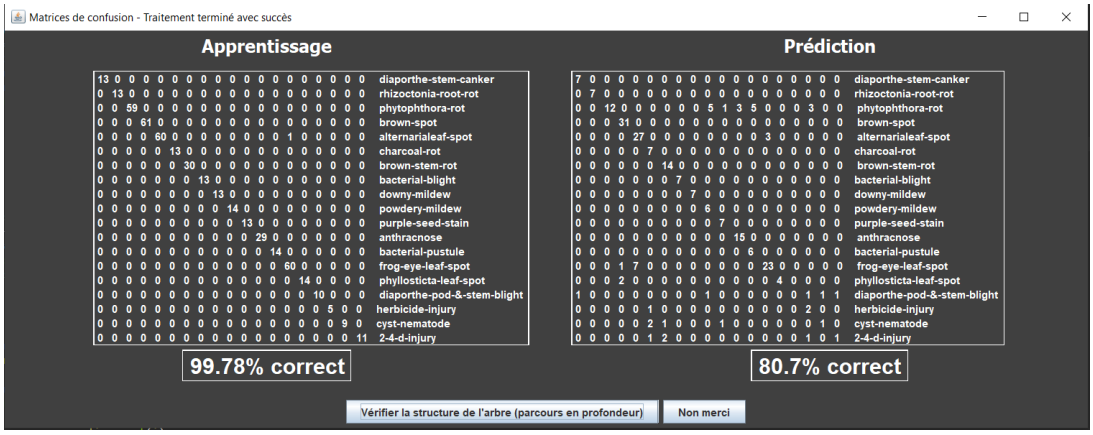

Figure 12 - Exemple pour un arbre C4.5 et remplacement des valeurs manquantes par l'attribut majoritaire

Vérification de la structure de l'arbre

Sur la fenêtre d'affichage des matrices de confusion, un bouton permet d'afficher la structure de l'arbre d'apprentissage dans la console via un parcours en profondeur. Il y figure :

- L'attribut du noeud père
- La valeur d'attribut qui permet de se diriger vers un noeud fils
- L'attribut du noeud fils
- Dans le cas où un noeud fils est une feuille, c'est donc une classe et le mot CLASSE est affiché en majuscules

Figure 13 : Arbre Id3 choix 3 valeurs manquantes

Figure 14 : Arbre C4.5 choix 3 valeurs manquantes

On peut observer que les algorithmes Id3 et C4.5 construisent des arbres différents. Ici, le nœud racine n'est par exemple pas le même.

Conclusion – Ce qu'il reste à faire

Pour conclure, le programme fonctionne correctement et parvient à afficher les matrices de confusion pour l'ensemble d'apprentissage et l'ensemble de prédiction. Les résultats observés dans le tableau à la Figure 11 semblent cohérents. Enfin, du point de vue personnel, ce projet m'a beaucoup aidé à apprendre à utiliser le langage Java, qui me posait de nombreuses difficultés au début.

Pour ce qu'il reste à faire, j'aurais voulu trouver un autre moyen de remplacer les valeurs manquantes. L'article de référence fourni avec ce projet parle d'une méthode consistant à remplacer les valeurs manquantes à l'aide d'un arbre de décision. Cette idée m'a parue intéressante, d'autant plus que la proportion de bons remplacements de valeur est assez élevée (de 78% à 81% d'après l'article). J'ai commencé à réfléchir à cette solution, mais rien n'a abouti. De plus, il reste deux fonctionnalités de l'algorithme C4.5 que je n'ai pas du tout eu le temps de traiter :

- Gestion des attributs à valeurs continues
- Post-étalage de l'arbre

Enfin, il y a un problème d'alignement que je n'ai pas réussi à corriger dans l'affichage des matrices de confusion. Il faudrait peut-être utiliser un autre Layout que BorderLayout qui est simple à utiliser mais les possibilités sont limitées.