Projet Intelligence Artificielle – Printemps 2020

Task 1 :

Il s’agissait de créer un arbre de décision basé sur l’algorithme ID3 donné dans les solutions des exercices. Il fallait tout d’abord convertir le fichier CSV en tableau de données avec notre fonction csv\_reader dans task\_1/csv\_reader.py. Les entrées du tableau sont des string, car cela permettait un affichage plus simple par la suite. Puis nous avons créer l’arbre en s’aidant des données d’entrainement.

Pour afficher les statistiques, nous avons créé une classe Statistiques avec 3 fonctions, prenant chacune l’arbre dont on veut afficher les statistiques en paramètre.

compteur\_de\_feuille incrémente un compteur à chaque fois qu’elle arrive à un nœud terminal.

maximum\_longueur\_branche ajoute à une liste la taille de la branche terminée par le nœud terminal qu’elle rencontre. On se retrouve donc avec une liste contenant toutes les longueurs de chaque branche de l’arbre et on en sort le maximum.

nombre\_enfants ajoute à chaque nouvel enfant qui n’est pas terminal le nombre d’enfants de ce dernier à une liste. On fait la somme de la liste à la fin pour connaître le nombre d’enfants total. Nous nous sommes basés sur l’observation qu’un enfant n’avait qu’un seul parent.

Task 2 :

Pour pouvoir prédire la précision de prédictions correctes, nous avons utilisé les données de test, et comparions le résultat target prédit par l’algorithme id3 (1 ou 0) par rapport à la « vraie » valeur présente dans les données. Nous obtenions 56.25%

Task 3 :

Nous avions 4 sous-tâches dans la tâche 3 :

1. générer les règles
2. générer les faits
3. trouver la règle justifiant un fait
4. afficher la règle justifiant un fait

Pour ce faire, nous avons créer 3 fonctions implémentant chaque sous-tâche respectivement.

[3.1] Nous nous sommes inspirés de la manière dont ID3 parcourait un arbre pour l’afficher et avons fait la même chose pour en créer les règles et avoir pour chaque nœud terminal, la règle correspondante.

L’utilisation du dictionnaire permet de ne garder que la dernière valeur d’un attribut sur le parcours de l’arbre. Ainsi, si les faits d’une branche sont séparés n fois par le même attribut, seul la dernière combinaison attribut/valeur est prise en compte pour la création de la règle. Ce qui facilitait les deux autres sous-tâches.

[3.2] Globalement, la même chose que pour la tâche 1, dans la lecture du fichier csv, on transforme chaque ligne en dictionnaire attribut → valeur.

[3.3] Une fois qu’on a déduit toutes les règles de l’arbre, on parcourt la liste de règle et pour chaque règle, on parcourt le dictionnaire d’attribut→valeur et si une combinaison n’est pas satisfaite dans le fait, on abandonne la règle. Si après avoir parcouru tout le dictionnaire, la règle est valide, alors c’est la règle qui justifie le fait.

[3.4] On trouve la règle, on l’affiche et on affiche le fait.

Nous avons utilisé la classe RegleSansVariables uniquement pour afficher une règle.

Task 4 :

Pour la tâche 4, nous devions être en mesure de proposé une sorte de diagnostic pour un patient malade. Pour cela nous avons raisonné de la manière suivante :

Pour qu’un patient soit sain, il y a une règle qui affirme pourquoi il est sain, nous nommerons ce genre de règle les « bonnes » règles.. Si un patient est malade, il suit alors une règle affirmant qu’il est malade, nous nommerons ce genre de règle les « mauvaises » règles. Pour pouvoir le soigner, on doit changer un de ses attributs pour qu’il suive une bonne règle.

Pour ce faire :

* On trie les règles et on ne garde que les bonnes
* On parcourt les règles triées et pour chaque règle on regarde quels attributs ne sont pas satisfaits chez le patient.
  + Si un des attributs pas satisfait est le sexe ou l’âge, on abandonne cette règle, car ce ne sont pas des attributs que nous pouvons changer chez le patient pour que celui-ci respecte la règle.
  + Pour chaque règle, on compte le nombre d’attributs non satisfaits par la règle.
* On ne garde que la règle avec le minimum d’attributs non satisfaits. On garde aussi en mémoire le nombre d’attributs non satisfaits. Ce nombre nous sera utile pour la suite.

C’est le but de la fonction trouve\_meilleure\_regle du fichier abduction.py

Ensuite, une fois qu’on a la meilleure règle et le nombre d’attributs non satisfaits de cette règle, on commence notre diagnostic :

* Si un patient n’a aucun attribut non satisfait, alors il suit la règle et n’est par conséquent pas malade.
* Si un patient a + de 2 attributs non satisfaits, alors on part du principe qu’on ne peut pas le soigner.
* Autrement : On soigne le patient

Pour soigner le patient, on change juste la valeur des attributs non satisfaits de manière à ce qu’ils satisfassent la règle.

Une fois que nous avons tout ça, nous pouvons itérer sur tous les patients et faire le raisonnement pour chaque patient. La fonction nb\_patients\_sauvables nous retournera le nombre total de patients, le nombre de patients que nous ne pourrons sauver, le nombre de patients que nous pouvons sauver en changeant 1 ou 2 attributs et le nombre de patients qui ne sont pas malades.

Task 5 :

Nous devions modifier l’algorithme ID3 pour qu’il puisse traiter des données continues, c’est-à-dire créer un arbre avec des nœuds binomiaux : à gauche les données plus petites qu’une certaine valeur de séparation, à droite les valeurs plus grandes ou égales à cette valeur.

Tout d’abord il fallait séléctionner la combinaison « attribut = valeur » qui réduisait au maximum l’entropie, et cette valeur devenait la valeur de séparation (fonction construit\_arbre\_recur dans task\_5/id3\_continuous).

Puis il fallait séparer les données en 2 nouveaux set de données qui correspondent à chaque nouveau nœud (fonction partitionne dans task\_5/id3\_continuous). Il fallait aussi ajuster le tableau attributs en fonction des valeurs qui seraient conservées dans le nouveau nœud. Puis enfin nous créons les 2 enfants, et finalement nous créons l’entité NœudDeDecision\_continuous.

Dans NœudDeDecision\_continuous, il fallait modifier l’affichage : dans classifie, il fallait séparer les données dans l’enfant de gauche ou de droite, puis modifier l’affichage (< et >=).

Une fois cette arbre avancé fini, nous pouvions calculer la précision des prédictions. Nous obtenons 51.249999999. (Légèrement en dessous de la précision trouvée au task 2, nous nous attendions à trouver une prédiction plus élevée).