

# Oraux blancs MPI

## type CCINP

Florian Bourse

### Exercice A.

Voici un ensemble de données  $E$  :

V	W	X	Y
F	F	F	F
F	V	F	V
V	F	F	V
V	V	F	F
V	V	V	F

On souhaite construire un arbre de décision pour classifier la variable  $Y$ , contenant le moins de nœuds possible. *On rappelle qu'un arbre de décision est un arbre binaire dont les nœuds internes sont étiquetés par les attributs et les feuilles par  $\{V, F\}$ . Les fils gauches correspondent à une réponse  $F$  et les fils droits à une réponse  $V$ .*

1. Rappeler le principe de l'apprentissage supervisé.
2. L'entropie d'un ensemble  $S$  d'exemples est définie par :

$$H(S) = -\frac{n_+}{n} \log_2\left(\frac{n_+}{n}\right) - \frac{n_-}{n} \log_2\left(\frac{n_-}{n}\right)$$

où  $n_+$ ,  $n_-$  et  $n$  désignent respectivement le nombre d'éléments de  $S$  dont l'étiquette est  $+$ , le nombre d'éléments de  $S$  dont l'étiquette est  $-$  et enfin le nombre total d'éléments de  $S$ . Dans le cas où  $k = 0$ , on prend la convention  $k \log_2(k) = 0$ .

Par exemple, l'entropie de l'ensemble de toutes les données  $E$  ci-dessus est  $H(E) = 0,970\,950$  (valeur approchée).

Étant donné un attribut  $A$ , on définit le gain de  $A$  par rapport à  $S$  par

$$IG(S, A) = H(S) - \frac{n_{A=V}}{n} H(S_{A=V}) - \frac{n_{A=F}}{n} H(S_{A=F})$$

où  $S_{A=V}$  désigne le sous ensemble des éléments de  $S$  dont l'attribut  $A$  est  $V$  et  $n_{A=V}$  désigne son cardinal, de même pour  $F$  et  $n$  désigne toujours le cardinal de  $S$ . Par exemple, le gain d'information de l'attribut  $X$  par rapport à  $E$  est  $IG(E, X) = 0,170\,950$  (valeur approchée).

On donne aussi la valeur particulière  $-\frac{2}{3} \log_2\left(\frac{2}{3}\right) - \frac{1}{3} \log_2\left(\frac{1}{3}\right) = 0,918\,295$ .

Calculer les gains d'information  $IG(E, V)$  et  $IG(E, W)$ .

Quel attribut serait sélectionné en premier par l'algorithme ID3 ?

3. Donner l'arbre de décision construit par ID3 en entier, sans élagage.
4. Une idée pour élaguer l'arbre est de commencer à la racine, et d'enlever les tests pour lesquels le gain d'information est plus petit qu'un certain seuil  $\varepsilon$ . On parle d'élagage de haut en bas. Quel est l'arbre de décision renvoyé pour  $\varepsilon = 0,0001$  ? Tracer la matrice de confusion de cet arbre sur les données d'entraînement.

5. Une autre possibilité est de commencer aux feuilles, et d'élaguer les sous-arbres dont le gain d'information d'un test est plus petit qu'un certain seuil  $\varepsilon$ . Avec cette méthode, aucun ancêtre d'enfants avec un haut gain d'information ne sera élagué. On parle d'élagage de bas en haut. Quel est l'arbre de décision renvoyé pour  $\varepsilon = 0,0001$ ? Tracer la matrice de confusion de cet arbre sur les données d'entraînement.
6. Comparer les taux d'erreurs et la complexité des deux approches et discuter dans quels cas l'élagage de bas en haut serait préférable à l'élagage de haut en bas et vice versa.
7. Quelle est la profondeur de l'arbre renvoyé par ID3 avec élagage de bas en haut? Peut-on trouver un arbre de profondeur plus petite qui classe aussi parfaitement  $Y$  sur le jeu de données d'entraînement? Quelles conclusions peut-on en tirer sur les performances de l'algorithme ID3?

## Exercice B. L'exercice suivant est à traiter dans le langage OCaml

Cet exercice cherche à déterminer la plus grande séquence consécutive d'entiers présente parmi les éléments d'une liste.

*cf annexe pour un rappel sur le module `Hashtbl`*

On définit de plus dans le code donné la fonction `get` qui agit comme `Hashtbl.find` mais renvoie 0 à la place de lever une exception.

On admettra dans ce sujet que la complexité des opérations sur les tables de hachage est en  $O(1)$  en espérance.

1. Écrire une fonction `mem : 'a -> 'a list -> bool` permettant de tester l'appartenance d'un élément à une liste.
2. Donner la complexité de la fonction `has_duplicate` proposée dans le code donné.
3. On souhaite améliorer la complexité de cette opération en utilisant des tables de hachage.  
Écrire une fonction `occurences : 'a list -> ('a, int) Hashtbl.t` qui prend en entrée une liste  $\ell$  et renvoie une table de hachage contenant pour chaque élément  $k$  présent dans la liste  $\ell$  l'association  $k \mapsto v$ , où  $v$  est le nombre d'occurrences de  $k$  dans  $\ell$ .
4. En déduire une nouvelle fonction `has_duplicate` de complexité  $O(|\ell|)$ .
5. Proposer sans l'implémenter une méthode pour déterminer la plus grande séquence consécutive d'entiers présente parmi les éléments d'une liste triée et évaluer sa complexité.
6. Si la liste n'est pas triée, on propose l'algorithme suivant :  
On commence par créer la table des occurrences, ce qui permet de tester en  $O(1)$  si un élément est présent ou non dans la liste.  
On construit ensuite une table de hachage contenant pour chaque entier  $x$  de la liste la longueur de la plus grande séquence consécutive commençant à  $x$ .  
Il ne reste qu'à chercher la plus grande valeur de cette table.  
Implémenter l'algorithme proposé et discuter de sa complexité.

## A Aide à la programmation

On rappelle les syntaxes des fonctions du module `Hashtbl` pour manipuler des tables de hachages :

- `Hashtbl.create n` : créé une nouvelle table de hachage de taille  $n$  (la taille est modifiée dynamiquement donc la valeur de  $n$  est peu importante) ;
- `Hashtbl.add ht k v` : ajoute une association  $k \mapsto v$  dans la table `ht` ;
- `Hashtbl.mem ht k` : renvoie **true** si il existe une association pour la clé  $k$  dans la table `ht`, et **false** sinon ;
- `Hashtbl.find ht k` : renvoie  $x$  tel que l'association  $k \mapsto v$  existe dans la table `ht` ou lève l'exception `Not_found` ;
- `Hashtbl.find_opt ht k` : renvoie `Some x` si une association  $k \mapsto v$  existe dans la table `ht` ou `None` sinon.