Apprentissage Automatique, TP1

M1, année 2016-2017

Arbres de décision et langage R

1 Documentation et références

Librairie rpart : `https://cran.r-project.org/web/packages/rpart/rpart.pdf`
Refcard R : `https://cran.r-project.org/doc/contrib/Short-refcard.pdf`

2 Introduction

L'objectif de ce TP est de vous faire manipuler des arbres de décision. On utilisera pour cela la bibliothèque `rpart` de R. Celle-ci contient les fonctions permettant de construire et d'exploiter un arbre de décision ; l'algorithme utilisé ici est CART.

Créer d'abord un répertoire dédié au travail sur R. Puis créer à l'intérieur un nouveau répertoire consacré au TP1. Allez dans ce répertoire et lancez R. Vous aurez la possibilité d'enregistrer votre travail en cours en quittant R. Il est fortement conseillé de le faire !

3 La construction d'un arbre de décision

3.1 Chargement de la bibliothèque rpart

Pour pouvoir construire des arbres de décision, nous allons utiliser la bibliothèque `rpart` de l'environnement R. Il faut tout d'abord la rendre accessible. Pour cela, on tape la commande suivante :

```
library(rpart)
```

3.2 Chargement du jeu de données

Nous allons tout d'abord découvrir cette bibliothèque sur l'exemple du jeu de tennis. Commençons par charger le jeu de données à partir d'un fichier, vous pouvez soit enregistrer le fichier distant sur votre machine ou directement le charger via le réseau via la fonction :

```
tennis <- read.table("tennis_data.txt")
```

La structure de donnée R ici stockée est un `data.frame`. Analysez les données de ce `data.frame` dans R.

Question 1. Description des données : quelles sont les attributs ? Quel est l'attribut cible (classe) ? Pour quel but allons-nous construire un arbre de décision ?

3.3 Construction d'un arbre de décision

Les commandes suivantes permettent de construire l'arbre de décision. Tout d'abord, on doit spécifier quelques paramètres qui précisent comment l'arbre de décision doit être construit. On tapera la commande suivante :

```
ad.tennis.cnt <- rpart.control (minsplit = 1)
```

La variable `ad.tennis.cnt` stocke les paramètres de l'algorithme. `minsplit = 1` signifie que le nombre minimal d'exemples nécessaires à la création d'un nœud est 1. La valeur par défaut est 20. Comme le jeu de données contient moins de 20 exemples, utiliser la valeur par défaut ne produirait pas d'arbre du tout, juste une racine !
Remarque : le nom utilisé pour cette variable, `ad.tennis.cnt` suit la convention R : il indique qu'il s'agît d'un arbre de décision (préfixe `ad`), pour le jeu de tennis (tennis !) et qu'il s'agît des paramètres de contrôle (`cnt`) ; des points (.) séparent ces différentes informations. Tout cela est complétement libre ; on aurait pu l'appeler `toto`, R ne nous l'aurait pas interdit. Vous prendrez donc l'habitude de nommer les variables en suivant ce principe.

On va maintenant construire l'arbre de décision en indiquant :
— l'attribut qui doit être prédit (la classe) ;
— les attributs qui doivent être utilisés pour effectuer cette prédiction ;
— le jeu d'exemples avec lequel on construit l'arbre ;
— le nom de la variable qui contient les paramètres.

Question 2. Complétez la commande suivante pour créer l'arbre de décision :

```
ad.tennis <- rpart (Classe~Attribut1 + ... + AttributN, Données, control = paramètres)
```

Remarque : la notation (formule R) `Classe~Attribut1 + ... + AttributN` est courante en R, lorsqu'il s'agît de construire un modèle de prédiction (classification supervisée ou régression). Elle signifie : prédire l'attribut `Classe` en fonction d'autres attributs.

3.4 Génération de l'arbre de décision

3.4.1 Représentation textuelle

Visualisez l'arbre produit dans la variable `ad.tennis` et prenez le temps de comprendre ce qui est affiché.

3.4.2 Représentation graphique

On peut également obtenir une représentation graphique grâce à la commande `plot`. Il est possible d'ajouter du texte avec la fonction `text` :

```
plot (ad.tennis)
text (ad.tennis)
```
Voyez si ce que vous voyez graphiquement est compatible avec ce que vous aviez compris dans la représentation textuelle. L'aspect graphique de l'arbre peut être paramétré. Essayez (interprétez chaque commande et toutes les valeurs affichées) :

- `plot (ad.tennis, uniform=T)`
- `text (ad.tennis, use.n=T, all=T)`
- `plot (ad.tennis, branch=0)`
- `plot (ad.tennis, branch=.7)`
- `text (ad.tennis, use.n=T)`
- `plot (ad.tennis, branch=.4, uniform=T, compress=T)`
- `text (ad.tennis, all=T,use.n=T)`
- `plot (ad.tennis, branch=.2, uniform=T, compress=T, margin=.1)`
- `text (ad.tennis, all=T, use.n=T, fancy=T)`

3.5 La prédiction de la classe d'une donnée par un arbre de décision

La fonction `predict()` utilise un arbre de décision pour prédire la classe de nouvelles données. Elle prend en paramètres l'arbre et un `data.frame` qui contient les données dont il faut prédire la classe. Pour prédire la classe des données du jeu d'exemples (avec lesquels on a construit l'arbre de décision), on tapera la commande :
`predict(ad.tennis, tennis)`

Question 3. Expliquez la sortie de cette commande.

Nous allons utiliser l'arbre pour donner une prédiction dans les situations suivantes :

	Ciel	**Température**	**Humidité**	**Vent**
1	Ensoleillé	30	90	Faible
2	Couvert	25	70	Fort
3	Pluie	15	86	Fort

Pour cela, utilisez les deux méthodes suivantes pour charger les données :
Question 4.1 Mettre ces données dans un fichier dont le format est similaire à `tennis_data.txt` puis chargez le.
Question 4.2 Créer directement un `data.frame` (commande `data.frame(Vecteur1, ..., VecteurN)`)
Question 4.3 Pour chaque ligne, prédire si le match aura lieu. Est-ce conforme au modèle construit ?

4 Voitures : un exemple un peu plus réaliste

On va utiliser ici le jeu de données `car.test.frame`. Affichez son contenu (il fait partie du package `rpart`). Vous obtiendrez plus de renseignements sur ce jeu de données avec la commande d'aide `?car.test.frame`.

Question 5. Construisez un arbre de décision à partir de ce jeu de données pour prédire la variable `Type` à partir des autres variables. Vous utiliserez les mêmes paramètres que pour le tennis (`minsplit=1`). Que pensez-vous de l'arbre obtenu ? Comment pourrait-on l'améliorer ?

4.1 Validation croisée, élagage et estimation des erreurs apparente et réelle

4.1.1 Élagage

Il est possible d'élaguer automatiquement un arbre de décision avec la fonction `prune()` (qui veut dire élaguer, en anglais, ça veut dire prune aussi d'ailleurs). Essayez par exemple, sur l'arbre des voitures `prune(arbre, 0.02)`. La valeur passée en paramètre (0.02) s'appelle le « complexity parameter » (cf. documentation).
Question 6. Essayez plusieurs valeurs de ce paramètre et observez son effet sur l'arbre élagué.

4.1.2 Détermination de l'abre optimal

La question se pose maintenant de déterminer quel est l'arbre optimal (celui qui risque le moins de faire des erreurs de prédiction). Par défaut, `rpart()` effectue un élagage de l'arbre et une validation croisée à 10 plis sur chaque arbre élagué. Les mesures effectuées au long de cette procédure sont stockées dans une table dénommée la `cptable`.
Question 7. Affichez cette table pour l'arbre de décision sur les voitures grâce à la syntaxe :

```
ad.car$cptable
```

Il est possible que vous n'obteniez pas les mêmes résultats entre vous car le choix des exemples dans la validation croisée est aléatoire. `rel error` mesure l'erreur apparente (erreur d'entraînement). `xerror` mesure le taux d'erreur dans la validation croisée à 10 plis que l'on considère comme un estimateur correct de l'erreur réelle. `xstd` est l'écart-type de l'erreur de validation croisée. L'arbre qui nous intéresse est celui qui minimise `xerror` + `xstd` (l'erreur moyenne estimée + 1 écart-type). Si plusieurs arbres minimisent cette valeur, on prend toujours le plus petit (à performances équivalentes, on choisit le plus petit modèle).

4.1.3 Les autres arbres

Vous pouvez obtenir tous les autres arbres aussi (les 7 autres, ici). Pour cela, il faut indiquer le `cp` dans la fonction `prune()`. Il faut bien comprendre comment se lit cette table et la relation entre valeur de `cp` et l'arbre élagué correspondant. Notez bien que la borne inférieure de l'intervalle est exclue.
Question 8. À quel arbre correspond alors la valeur de `cp` = 0,2 ?

4.1.4 Détermination de la probabilité d'erreur de prédiction d'un arbre de décision

Question 9.. Combien d'exemples sont mal classés à la racine (aidez-vous de la sortie standard d'un arbre) ?

Pour visualiser facilement quel est l'arbre optimal, vous allez tracer deux courbes :
Question 10. Avec les fonctions `plot()` puis `lines()`, tracez d'abord la courbe de `rel error` en fonction du nombre de nœuds (nombre de nœuds = `nsplit` + 1).
Astuce : pour récupérer rapidement les valeurs à partir du `cptable`, R propose une syntaxe très simple, inspirez-vous de ces exemples. *Récupérer toutes les valeurs de la colonne d'indice 2 :* `tableau[,2]`, *toutes les valeurs de la ligne 5 :* `tableau[5,]`, *la valeur de la colonne d'indice 2*

et de la ligne 5 : `tableau[5,2]`.
Question 11. Faites de même (sur le même graphique) avec la courbe du `xerror` toujours en fonction du nombre de noeuds.

La technique d'analyse (brutale) de ce type de graphique est celle dite du « coude » ; si à un endroit du graphique, la ou les courbe(s) fléchissent telle(s) un coude d'un bras, on peut estimer que la valeur optimale se trouve à cet endroit.

Question 12. En utilisant cette technique (efficace !), déterminez la valeur de `cp` optimale. Créer l'arbre correspondant dans une nouvelle variable grâce à la fonction `prune` vue précédemment :
`ad.car.optimal <- prune (ad.car, cp=...)`

4.1.5 Autres prédictions

On peut aussi essayer de prédire un autre attribut qualitatif que le `Type`. On peut ainsi tenter de prédire `Country` ou encore `Reliability`.
Remarque : l'attribut `Reliability` est un attribut qualitatif mais sa valeur dans le tableau est un nombre. Dans ce cas, `rpart` interprète que c'est un attribut quantitatif. Aussi, pour répondre correctement à la question, il faut d'abord transformer l'attribut `Reliability` en un attribut qualitatif, ayant donc des valeurs non numériques. Par exemple, vous pourrez remplacer les valeurs numériques 1, 2, 3, 4 et 5 par les valeurs « un », « deux », « trois », « quatre » et « cinq ».
Astuce : remplacer chaque occurrence des 5 valeurs possibles se fait par exemple via la syntaxe R suivante :
`tableau$NomColonne[table$NomColonne==Valeur]<-NouvelleValeur`

Pour être plus rigoureux, on pourrait même copier la colonne pour ne pas altérer les données originales :
`tableau$NouvelleColonne <- tableau$Colonne`

Question 13. Construisez des arbres de décision pour ces deux attributs comme on l'a fait précédemment pour `Type`. Lequel de ces trois attributs est le mieux prédit ?

Repartons de la prédiction du `Type`. Parmi les autres attributs, on veut déterminer celui qui est le moins important pour prédire ce type. Pour cela, on va construire des arbres de décision utilisant tous les attributs sauf un. L'attribut manquant qui sera associé au meilleur arbre de décision est l'attribut le moins important.

Créez maintenant un script R (ouvrir un nouveau terminal et créer un simple fichier texte avec pour extension .R) que vous sauvegarderez dans le répertoire de travail courant.

Question 14. Déterminez quel attribut est le moins important et celui qui est le plus important en affichant les meilleurs arbres pour chaque fonction de `Type` possible (enlever un des attributs à chaque fois dans la formule R donnée en premier paramètre de `rpart`). En sortie, le script affichera le score `xerror` + `xstd` pour chaque attribut ignoré.
Rappel : le meilleur arbre est celui qui minimise `xerror` + `xstd`.

Indications : utilisez une boucle sur les colonnes du tableau (fonction `colnames(tableau)` en prenant bien garde à exclure `Type`). La fonction `which()` permet d'effectuer un filtre sur des données. La syntaxe `tableau[which(tableau)==Valeur]` ne sélectionne que les données testées positivement. La fonction `as.formula()` va aussi vous être utile pour créer dynamiquement la formule R.