TP3 : arbre de décision

Nous allons manipuler dans ce TP des arbres de décision avec la bibliothèque rpart de l'environnement R.

```
>library(rpart)
```

Exercice 1 : Construire un arbre de décision

Pour se familiariser avec les fonctions de construction d'arbre de décisions, nous allons manipuler un petit jeu de données composé de 14 observations. Il s'agit de savoir si on joue ou pas au tennis en fonction des conditions météorologique. Charger le jeu de données "tennis.csv" dans un data frame *data.tennis* et analysez le.

La fonction *rpart*() permet de construire l'arbre de décision. Nous devons lui préciser :

- la classe cible à prédire (la classe : Jouer).
- les attributs qui doivent être utilisés pour effectuer cette prédiction (Ciel + Temperature + Humidite + Vent).
- la data frame contenant les données d'apprentissage *data.tennis*.

```
> tree.tennis <- rpart (Jouer ~ Ciel + Temperature
+ Humidite + Vent, data.tennis)
```

- Afficher le modèle *tree.tennis* obtenu. Que constatez vous ?
- Consultez l'aide sur la fonction *rpart*() puis sur *rpart.control*().
- Quel est le rôle du paramètre *minsplit* et sa valeur par défaut ? Pouvez-vous maintenant justifier le résultat obtenu.
- Modifiez la valeur par défaut du paramètre *minsplit* en le mettant à 1 puis reexécutez la fonction *rpart*().

```
> tennis.cnt <- rpart.control (minsplit = 1)
> tree.tennis <- rpart (Jouer ~ Ciel + Temperature
+ Humidite + Vent, tennis, control = tennis.cnt)
> tree.tennis
```

Analyser et commentez le résultat obtenu. Vous pouvez visualiser l'arbre avec les commandes suivantes :

```
> plot (tree.tennis)
> text (tree.tennis)
```

Interprétez les résultats obtenus par les commandes suivantes :

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

Donner une prédiction des situations suivantes avec *predict* :

Ciel	Temperature	Humidite	Vent
Ensoleille	30	90	Faible
Couvert	25	70	Fort
Pluie	15	86	Fort

Évaluer les performances de votre modèle (taux d'erreur) sur l'ensemble d'apprentissage en affichant la matrice de confusion.

```
> predict(tree.tennis, data.tennis)
> table(predict(tree.tennis,data.tennis,"class"),
data.tennis$Jouer)
```

Mettez le paramètre $minsplit = 5$ puis recalculez le nouveau taux d'erreur de l'arbre obtenu sur l'ensemble d'apprentissage.

Exercice 2 : Élagage

La librairie *rpart* contient un jeu de données "car.test.frame". Pour le charger :

```
> data(car.test.frame)
> data.cars <- car.test.frame
```

— Affichez les premières lignes du ce data frame *data.cars*.
— Donnez le nombre d'observations/exemples de *data.cars*.
— Donnez le type de chaque attributs.
— Vous remarquez que les données ont des noms. Affichez les caractéristiques de la Nissan Maxima V6.
— On va essayer de prédire le type de voiture en fonction des autres attributs. Quel est le nombre de classes (les valeurs de l'attribut cible Type) ?
— Construisez le vecteur des effectifs par type de voiture.
— Faites un graphique.

```
> plot(table(data.cars$Type))
> barplot(table(data.cars$Type))
```

— Étudiez les dépendances 2 à 2 entre attributs en visualisant le jeu de données. Vous serez attentif aux affichages en fonction des types des attributs. Par exemple :

```
> plot(data.cars$Weight, data.cars$Disp)
```

— Construisez un arbre de décision *tree.cars* à partir de ce jeu de données pour prédire l'attribut "Type" à partir des autres attributs. Vous utiliserez $minsplit = 1$.
— Que pensez-vous de l'arbre obtenu (taille, lisibilité,...) ?
— L'arbre produit peut être élagué à l'aide de la fonction *prune*(). Que représente le paramètre *cp* dans la fonction *prune*()

```
> tree.cars.pruned <- prune(ad.car, cp= 0.1)
> tree.cars.pruned
```

— Augmentez progressivement la valeur de ce paramètre pour visualiser les différents élagages de l'arbre *tree.cars* jusqu'à obtenir un élagage qui correspond à un seul noeud avec la classe majoritaire.
— La fonction *rpart*() construit un arbre avec une erreur d'apprentissage petite mais calcule également les arbres élagués, leur erreur sur l'ensemble d'apprentissage et une estimation de l'erreur réelle. Pour

estimer cette erreur réelle, *rpart*() effectue une validation croisée sur chaque arbre élagué. Les mesures effectuées au long de cette procédure sont stockées dans la table *cptable*. La commande suivante affiche cette table :

```
> tree.cars$cptable
```

— La table précédente contient les résultats de différents élagages de l'arbre initial. Combien d'arbres élagués ont été considérés ?
— Où se trouve l'arbre le plus élagué ? Quel est l'intervalle de son *cp* ?
— Il est possible d'obtenir tous les arbres en jouant sur le paramètre *cp* de la fonction *purge*(). Visualisez les arbres obtenus pour ces différentes valeurs de *cp*.
— Dans la *cptable*, les valeurs d'erreur sont normalisées pour en simplifier l'interprétation. Cette normalisation consiste à faire en sorte que l'erreur d'apprentissage de l'arbre complètement élagué (réduit à une racine feuille) soit égale à 1. Pour cela, les valeurs des colonnes *rel error*, *xerror* et *xstd* ont été divisé par 45/60.
— Calculez l'erreur d'apprentissage (non normalisée) de chacun des arbres construits par *rpart*.

```
> err.app <- tree.cars$cptable[,3] * (45/60)
> err.app
> err.eststd <- tree.cars$cptable[,4] * (45/60)+
  tree.cars$cptable[,5]
> err.eststd
```

— Quel arbre minimise l'erreur réelle estimée ? En déduire la valeur optimale de *cp*
— Générer l'arbre optimal *tree.cars.opt* en élaguant l'arbre *tree.cars* avec la valeur optimale de *cp*.
— Pour visualiser les erreurs, vous pouvez tracer l'évolution des erreurs apparente et réelle en fonction de la taille de l'arbre (nombre de noeuds = $nsplit + 1$) avec les commandes *lines*() et *points*().
— On veut déterminer l'attribut le moins important pour prédire le type. Pour cela, on va construire des arbres de décision utilisant tous les attributs sauf 1. L'attribut manquant qui sera associé au meilleur arbre de décision est l'attribut le moins important.