Dupuis Octave
Science des Données

Arbre de décision et forêt aléatoire

Problème 1 : Arbres de décision et étude de la validité

1) Le taux d'erreur est défini par $R(t) = 1 - \frac{N_{Y(t)}}{N(t)}$

TP1 Q1

Attribut	Regles	Taux d'erreur	Taux d'erreur total
Outlook	Sunny-no/ Overcast-yes /rainy-yes	R1=0.4 R2=0 R3=0.4	R(t)=1/14(5R1+4R1 +5R3)=0.286
Temperature	Hot-no/ mild-yes/ cool-yes	R1=0.5 R2=0.33 R3=0.25	R(t)=1/14(4R1+6R2 4R3)=0.357
Humidity	High-no/ normal-yes	R1=0.429 R2=0.143	R(t)=1/14(7R1+7R2) =0.786
Windy	False-yes/ true-yes	R1=0.25 R2=0.5	R(t)=1/14(8R1+6R2) =0.357

R(T) = Sigma((N(ti)/n)*R(ti)

2) On obtient le taux d'erreur le plus faible pour l'attribut Outlook. On le choisi pour la classification OneR. Les règles sont : sunny = no, overcast = yes, rainy = yes.

3)

```
> model<-OneR(data,verbose=TRUE)

    Attribute   Accuracy
1 * outlook     71.43%
1   humidity    71.43%
3   temperature 64.29%
3   windy       64.29%
---
Chosen attribute due to accuracy
and ties method (if applicable): '*'
```

```
Contingency table:
     outlook
play  overcast rainy sunny Sum
  no         0     2   * 3   5
  yes      * 4   * 3     2   9
  Sum        4     5     5  14
---
Maximum in each column: '*'
```

Detailed accuracy by class :

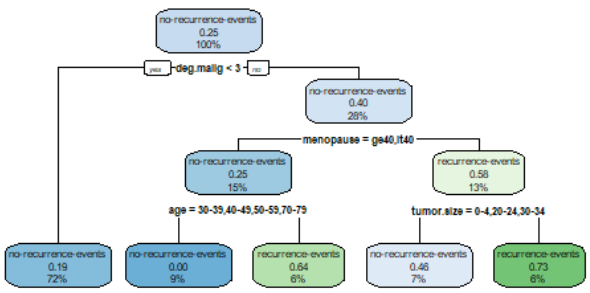
Overcast : 100 %

Rainy : $\frac{3}{5} = 60$ %

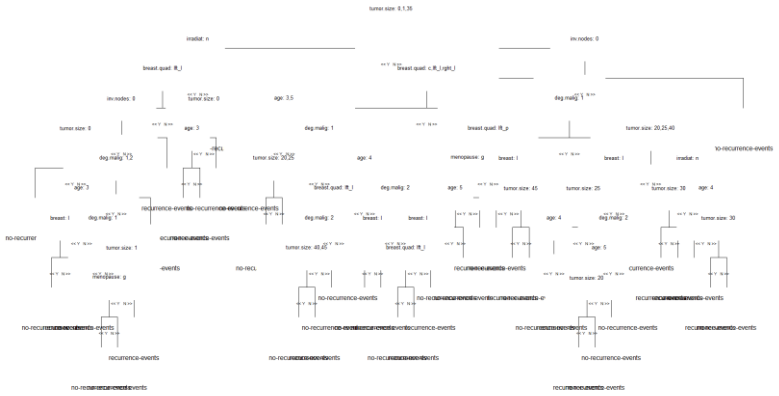
Sunny : $\frac{3}{5} = 60$ %

4) Après avoir utilisé la librairie rpart, nous avons essayé de construire un arbre avec la librairie randomForest. Pour créer un seul arbre à partir de cette librairie, nous avons créé une forêt aléatoire avec un seul arbre.

En utilisant la librairie rpart, on obtient l'arbre suivant :



Malheuresement, nous n'avons pas réussi à obtenir une représentation satisfaisante avec la librairie randomForest ; voici le meilleur résultat que nous avont obtenue :



Pour les erreurs de classification, on étudie les la spécificité et la sensibilité des deux models ; on obtient les résultats suivants :

	rpart	randomForest
Spécificité	0.33	0.421
Sensibilité	1	0.921

On observe que les résultats obtenus sont de même ordre de grandeur.

Problème 2 : Arbre de décision « From Scratch »

Nous avons commencé une ébauche de code sous python, nous vous le joignons en compte rendu.

Problème 3 : Challenge

1) On divise nos données en deux parties : une, *train*, qui servira à créer l'arbre et la seconde, *test*, qui servira à évaluer la pertinence de l'arbre de décision ainsi créé. Comme suggéré dans l'énoncé, on prend respectivement 70% et 30% des données. Afin d'avoir les mêmes données test et d'entrainement à chaque exécution du code, on prend soin d'imposer une graine de temps à l'aide de la commande « set.seed(1234) ».

2) La librairie « rpart » permet de créer des arbres décisionnels sous R à partir de données. Il existe deux paramètres qui permettent d'influencer la réalisation de ces arbres à savoir *minsplit* et minbucket.
Le paramètre *minbucket* correspond au nombre minimum d'observations dans un nœud terminal tandis que le paramètres *minsplit* spécifie le nombre minimum d'individus présents à l'étape d'un nœud pour envisager une coupure. Par défaut, si seulement l'un des deux est spécifié, la fonction « rpart » considère que *minsplit = 3 * minbucket*. On peut également régler la profondeur de l'arbre à l'aide de la commande *maxdepth*.
Il convient dès lors de jouer sur ces paramètres afin d'obtenir le meilleur modèle. On pourra juger de la performance de notre arbre de décision grâce à la courbe ROC et plus particulièrement à l'aide de l'aire sous cette courbe.

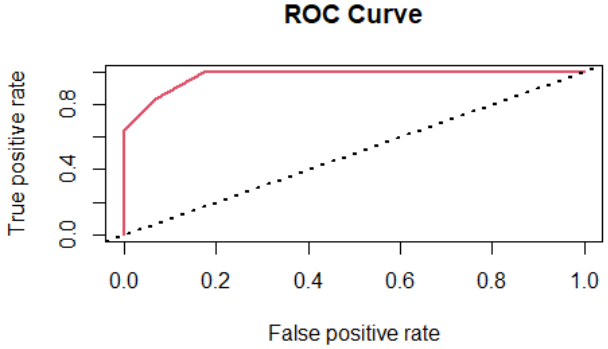
3) On va chercher à trouver les conditions optimales des trois paramètres précédemment cités (*minsplit, minbucket, maxdepth*) afin d'obtenir le meilleur score au sens de la valeur de l'aire sous la courbe ROC maximale (AUC).
Il convient de rechercher les valeurs de *minsplit* et *minbucket* optimale dans un premier temps. On montrera plus tard que dans ce cadre bien précis, le paramètre *maxdepth* n'est pas pertinent.

On cherche dès lors à maximiser la valeur de l'aire sous la courbe (AUC) à l'aide du code R fournit en annexe. On choisit une méthode « naïve » pour cela dans la mesure où on ne peut, à priori, pas prédire la zone où sera situé le meilleur résultat. Dès lors on regarde pour chaque paramètre le score obtenu pour chaque valeur prise entre 1 et 140. On rappelle que 140 constitue la taille de notre échantillon de données réservées pour l'apprentissage.

On trouve dès lors les valeurs suivantes : $\begin{cases} minsplit = 1 \\ minbucket = 11 \end{cases}$.

On obtient ainsi une valeur d'AUC = 0.9738598. Ce qui est plutôt un bon score car très proche de la valeur 1.

La courbe ROC ci-dessous témoigne de même d'une bonne prédiction :



Concernant le paramètre *maxdepth* : on se rend vite compte qu'il est un frein à un score correct lorsqu'il est inférieur à 5 et devient inutile une fois supérieur à ce nombre car l'arbre est de toute façon moins long.

On propose ci-dessous un récapitulatif des valeurs prises en fonction de la valeur de ce paramètre, pour les valeurs de *minsplit* et *minbucket* optimales.

Valeur de maxdepth :	**Valeur de l'aire sous la courbe (AUC)**
1	0.6501669
2	0.6501669
3	0.8904338
4	0.8904338
5	0.9738598
>6	0.9738598