UMB M'sila Master2 SIGL

Département de l'Informatique Module DMRI

TP N°2

Implémentation d'un arbre de décision avec R

Objectif du TP :

- Savoir comment installer les packages de la classification, y compris les arbres de décision.
- Savoir les différentes étapes de la classification par arbre de décision
- Savoir les différentes méthodes de R pour utiliser les arbres de décision

Etapes du TP

1- **Installation du package « caret »**

```
install.packages("caret")
```

Pour installer le package à partir d'un répertoire local, on écrit la commande suivante

```
install.packages("caret",destdir="repertoire de sauvegarde des packages")
```

2- **Ensemble de données (le fichier car.data)**

L'ensemble de données contient les informations sur les véhicules (six attributs), le 7ème est la classe.

1	buying	vhigh, high, med, low
2	maint	vhigh, high, med,low
3	doors	2, 3, 4, 5 , more
4	persons	2, 4, more
5	lug_boot	small, med, big.
6	safety	low, med, high

7	Car Evaluation – Target Variable	unacc, acc, good, vgood

3- Import des bibliothèques de l'arbre de décision

```
1 library(caret)  #Fonctions et methods des arbres de décision
2 library(rpart.plot) #Visualisation des arbres de décision
```

4- Import des données des véhicules (car.data)

```
1 data_url <- c("https://archive.ics.uci.edu/ml/machine-learning-databases/car/car.data")
2 download.file(url = data_url, destfile = "car.data")
3
4 car_df <- read.csv("car.data", sep = ',', header = FALSE)
```

Vérification de la structure du fichier: *str(car_df)*

Voir les 6 premières lignes (tuples) : *head(car_df)*

5- Répartition de l'ensemble de données

```
1 set.seed(3033)
2   intrain <- createDataPartition(y = car_df$V7, p= 0.7, list = FALSE)
3 training <- car_df[intrain,]
4 testing <- car_df[-intrain,]
```

- set.seed(3033) pour garder le travail réplicable pour conserver les mêmes résultats
- Le package caret utilise la méhode *createDataPartition()* pour partitionner les données en deux ensembles: d'apprentissage et de test. On passé 3 paramètres: le paramètre "y" prend la valeur de la variable de partionnement des données, dans notre cas, ce la variable**V7**, d'où on passé **car_df$V7.**
- Le parameter "p" prend une valeur décimale de 0 à 1. Il définit le pourcentage de répartition des données. On utilise p=0.7 qui correspond 70% des données pour l'apprentissage tet 30% pour les tests (2/3 et 1/3).
- Le parameter "list" pour retourner ou non l'ensemble *intrain.* On passe FALSE pour ne pas le faire
  Pour voir la dimension de chaque ensemble: dim(training); dim(testing);

6- Prétraitement et apprentissage

a- Prétraitement

Il faut s'assurer qu'il n'existe pas une valeur manquante dans les différents attributs de l'ensemble de données. Pour cela on vérifie si la valeur NA existe ou non :
anyNA(car_df)

b- Apprentissage en utilisant le gain en information (construction de l'arbre)

```
1 trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
2 set.seed(3333)
3 dtree_fit <- train(V7 ~., data = training, method = "rpart",
4                    parms = list(split = "information"),
5                    trControl=trctrl,
6                    tuneLength = 10)
```

- La méthode **trainControl()** est utilisée pour contrôler l'échantillonnage de données . Ici on utilise la validation croisée répétée "repeatedcv". Il existe d'autres methods comme "boot", "boot632", "cv", "LOOCV", "LGOCV" etc.
- Pour l'apprentissage de l'arbre de décision (sa conctruction), on utilise la méthode *train* qui utilise le package « rpart », on passe l'attribut V7 qui est la classe . La formule "V7~." Signifie la passation de tous les attributs avec l'attribut classe V7.
- Dans *params* on spécifie la méthode de sélection du meilleur attribut, "information" pour gain en formation gain & "gini" pour index de gini index.

Pour voir les résultats de l'arbre de décision, on peut afficher la variable dtree_fit

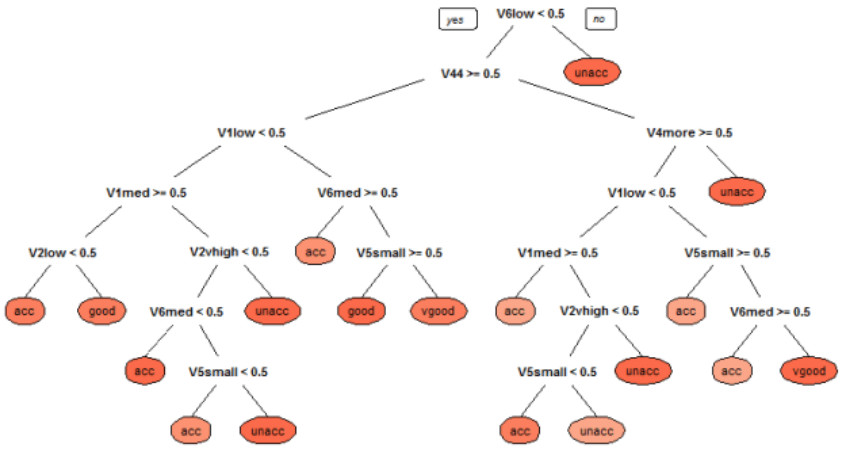
```
dtree_fit
CART

1212 samples
6 predictor
4 classes: 'acc', 'good', 'unacc', 'vgood'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 1091, 1090, 1091, 1092, 1091, 1091, ...
Resampling results across tuning parameters:

cp           Accuracy   Kappa
0.01123596  0.8600447  0.6992474
0.01404494  0.8487633  0.6710345
0.01896067  0.8309266  0.6307181
0.01966292  0.8295492  0.6284956
0.02247191  0.8130381  0.5930024
0.02387640  0.8116674  0.5904830
0.05337079  0.7772599  0.5472383
0.06179775  0.7745300  0.5470675
```

0.07584270 0.7467212 0.3945498
0.08426966 0.7202717 0.1922830

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01123596.

7- Affichage de l'arbre de decision

```
1 prp(dtree_fit$finalModel, box.palette = "Reds", tweak = 1.2)
```



8- Classification (test de l'arbre de décision)

1- Test d'un seule donnée de l'ensemble de test

```
1 > testing[1,]
2  V1 V2 V3 V4 V5 V6 V7
3 2 vhigh vhigh 2 2 small med unacc
4
5 > predict(dtree_fit, newdata = testing[1,])
6 [1] unacc
7 Levels: acc good unacc vgood
```

Pour le premier tuple de l'ensemble de test, à savoir testing[1,], l'arbre a donné la classe *unacc*.

2- Test des toutes les données de l'ensemble de test

```
1  > test_pred <- predict(dtree_fit, newdata = testing)
2  > confusionMatrix(test_pred, testing$V7 )  #check accuracy
3  Confusion Matrix and Statistics
4
5          Reference
```

```
Prediction acc good unacc vgood
     acc     102   19    36     3
     good      6   4     0      3
     unacc     5   0   318      0
     vgood    11   1     0      8

Overall Statistics

               Accuracy : 0.8372
                 95% CI : (0.8025, 0.868)
    No Information Rate : 0.686
    P-Value [Acc > NIR] : 3.262e-15

                  Kappa : 0.6703
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: acc   Class: good Class: unacc Class: vgood
Sensitivity              0.8226    0.166667      0.8983      0.57143
Specificity              0.8520    0.981707      0.9691      0.97610
Pos Pred Value           0.6375    0.307692      0.9845      0.40000
Neg Pred Value           0.9382    0.960239      0.8135      0.98790
Prevalence               0.2403    0.046512      0.6860      0.02713
Detection Rate           0.1977    0.007752      0.6163      0.01550
Detection Prevalence     0.3101    0.025194      0.6260      0.03876
Balanced Accuracy        0.8373    0.574187      0.9337      0.77376
```