

TP N°4**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

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

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

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

```
1 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 7^{ème} 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 répliquable pour conserver les mêmes résultats
 - Le package `caret` utilise la méthode `createDataPartition()` pour partitionner les données en deux ensembles: d'apprentissage et de test. On passe 3 paramètres: le paramètre "y" prend la valeur de la variable de partitionnement des données, dans notre cas, ce la variable **V7**, d'où on passe `car_df$V7`.
 - Le paramètre "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 et 30% pour les tests (2/3 et 1/3).
 - Le paramètre "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 construction), 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:

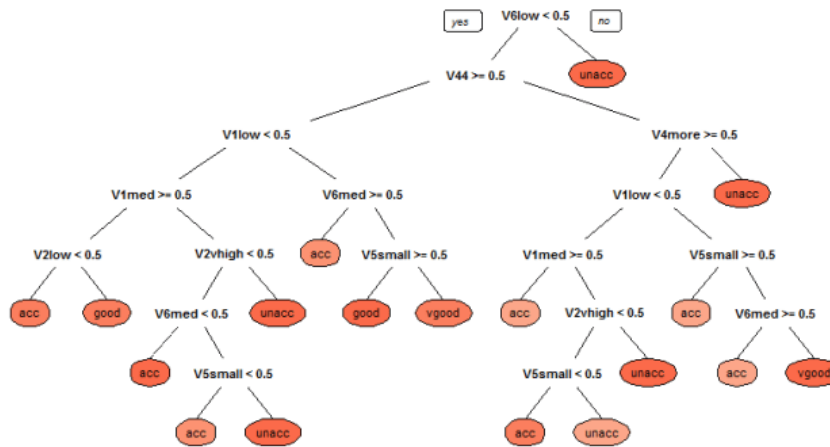
<i>cp</i>	<i>Accuracy</i>	<i>Kappa</i>
<i>0.01123596</i>	<i>0.8600447</i>	<i>0.6992474</i>
<i>0.01404494</i>	<i>0.8487633</i>	<i>0.6710345</i>
<i>0.01896067</i>	<i>0.8309266</i>	<i>0.6307181</i>
<i>0.01966292</i>	<i>0.8295492</i>	<i>0.6284956</i>
<i>0.02247191</i>	<i>0.8130381</i>	<i>0.5930024</i>
<i>0.02387640</i>	<i>0.8116674</i>	<i>0.5904830</i>
<i>0.05337079</i>	<i>0.7772599</i>	<i>0.5472383</i>
<i>0.06179775</i>	<i>0.7745300</i>	<i>0.5470675</i>
<i>0.07584270</i>	<i>0.7467212</i>	<i>0.3945498</i>
<i>0.08426966</i>	<i>0.7202717</i>	<i>0.1922830</i>

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 décision

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



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

1- Test d'une 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(dtrees_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(dtrees_fit, newdata = testing)
2 > confusionMatrix(test_pred, testing$V7) #check accuracy
3 Confusion Matrix and Statistics
4
5 Reference
6 Prediction acc good unacc vgood
7 acc 102 19 36 3
8 good 6 4 0 3
9 unacc 5 0 318 0
10 vgood 11 1 0 8
11
12 Overall Statistics
```

13

14 Accuracy : 0.8372

15 95% CI : (0.8025, 0.868)

16 No Information Rate : 0.686

17 P-Value [Acc > NIR] : 3.262e-15

18

19 Kappa : 0.6703

20 McNemar's Test P-Value : NA

21

22 Statistics by Class:

23

24 Class: acc Class: good Class: unacc Class: vgood

25 Sensitivity 0.8226 0.166667 0.8983 0.57143

26 Specificity 0.8520 0.981707 0.9691 0.97610

27 Pos Pred Value 0.6375 0.307692 0.9845 0.40000

28 Neg Pred Value 0.9382 0.960239 0.8135 0.98790

29 Prevalence 0.2403 0.046512 0.6860 0.02713

30 Detection Rate 0.1977 0.007752 0.6163 0.01550

31 Detection Prevalence 0.3101 0.025194 0.6260 0.03876

32 Balanced Accuracy 0.8373 0.574187 0.9337 0.77376