Tutorial: Decision Trees (important)

Q1: We have been given the following medical dataset and wish to create a decision tree (using the ID3 algorithm). The dataset includes four attributes. Calculate which attribute needs to be selected for the first (root) node. Justify your response.

Training	fever	vomiting	diarrhea	shivering	Classification
d_1	no	no	no	no	healthy (H)
d_2	average	no	no	no	influenza (I)
d_3	high	no	no	yes	influenza (I)
d_4	high	yes	yes	no	salmonella poisoning (S)
d_5	average	no	yes	no	salmonella poisoning (S)
d_6	no	yes	yes	no	bowel inflammation (B)
d_7	average	yes	yes	no	bowel inflammation (B)

For the calculation of log2, remember, from maths, that: $\log_a b = \dfrac{\log_{10} b}{\log_{10} a}$

$\log_2$ approx.

log2(1/8) = -3.0p
log2(1/4) = -2.00
log2(1/3) = -1.58
log2(3/8) = -1.42
log2(3/7) = -1.22
log2(1/2) = -1.00
log2(4/7) = -0.81
log2(5/8) = -0.68
log2(2/3) = -0.58
log2(3/4) = -0.42
log2(7/8) = -0.19

Solution

The parent entropy (i.e. total) is calculated as:

$$Entropy(S) = -\frac{1}{7}\log_2\left(\frac{1}{7}\right) - \frac{2}{7}\log_2\left(\frac{2}{7}\right) - \frac{2}{7}\log_2\left(\frac{2}{7}\right) - \frac{2}{7}\log_2\left(\frac{2}{7}\right) = 1.948$$

We need to check the entropy for each attribute individually and then find which attribute maximizes the information gain.

Fever attribute:

Values	H	I	S	B	Entropy(S_i)
S_1 (no)	x			x	[1/2,0,0,1/2]
S_2(average)		x	x	x	[0,1/3,1/3,1/3]
S_3(high)		x	x		[0,1/2,1/2,0]

$$Entropy(S_1) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - 0 - 0 - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1.0$$

$$Entropy(S_2) = 0 - \frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 1.585$$

$$Entropy(S_3) = 0 - \frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) - 0 = 1.0$$

Therefore, the entropy for this attribute is:

$$Entropy(S \mid Fever) = \frac{2}{7}\cdot 1 + \frac{3}{7}\cdot 1.585 + \frac{2}{7}\cdot 1 = 1.2507$$

Vomiting attribute:

Values	H	I	S	B	Entropy(S_i)
S_1 (yes)			x	xx	[0,0,1/3, 2/3] = 0.918
S_2(no)	x	xx	x		[1/4, 2/4, 1/4, 0] = 1.5

Therefore, the entropy for this attribute is:

$$Entropy(S \mid Vomiting) = \frac{3}{7}\cdot 0.918 + \frac{4}{7}\cdot 1.5 = 1.2505$$

Diarrhea attribute:

Values	H	I	S	B	Entropy(S_i)
S_1 (yes)			xx	xx	[0,0,2/4, 2/4] = 1.0
S_2(no)	x	xx			[1/3, 2/3, 0, 0] = 0.918

Therefore, the entropy for this attribute is:

$$Entropy(S \mid Diarrhea) = \frac{4}{7}\cdot 1.0 + \frac{3}{7}\cdot 0.918 = 0.965$$

Shivering attribute:

Values	H	I	S	B	Entropy(S_i)
S_1 (yes)		x			[0,1,0,0] = 0
S_2(no)	x	x	xx	xx	[1/6, 1/6, 2/6, 2/6] = 1.918

Therefore, the entropy for this attribute is:

$$Entropy(S \mid Shivering) = \frac{1}{7}\cdot 0 + \frac{6}{7}\cdot 1.918 = 1.644$$

So, we choose the attribute that maximizes the overall information gain

- Fever: Gain(S) = Ent(S) − Ent(S|Fever) = 1.948 − 1.2507 = 0.6973
- Vomiting: Gain(S) = Ent(S) − Ent(S|Vomit) = 1.948 − 1.2505 = 0.6975
- Diarrhea: Gain(S) = Ent(S) − Ent(S|Diarrh) = 1.948 − 0.965 = 0.983
- Shivering: Gain(S) = Ent(S) − Ent(S|Shiver) = 1.948 − 1.644 = 0.304

Thus, "Diarrhea" is the most effective one that maximizes the information gain.

Q2: We would like to predict the gender of a person based on two binary attributes: leg-cover (pants or skirts) and beard (beard or bare-faced). We assume we have a dataset of 20000 individuals, 12000 of which are male and 8000 of which are female. 80% of the 12000 males are barefaced. Skirts are present on 50% of the females. All females are bare-faced and no male wears a skirt. Calculate which attribute needs to be selected for the first (root) node. Justify your response. For the calculation of $\log_2$, remember, from maths, that: $\log_a b = \frac{\log_{10} b}{\log_{10} a}$

Solution

We have 12000 males and 8000 females. Thus, the analogy for males is 12000/20000 = 3/5, while for females 2/5.

The parent entropy (i.e. total) is calculated as:

$$Entropy(S) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = 0.971$$

We need to check the entropy for each attribute individually and then find which attribute maximizes the information gain.

Bare-faced attribute:

Values	Male	Female	Entropy(S_i)
S_1 (yes)	9600	8000	[9600/17600, 8000/17600]
S_2(no)	2400	0	[2400/2400, 0] = [1, 0]

$$Entropy(S_1) = -\frac{96}{176}\log_2\left(\frac{96}{176}\right) - \frac{80}{176}\log_2\left(\frac{80}{176}\right) = 0.994$$

$$Entropy(S_2) = -\log_2(1) - 0 = 0$$

Therefore, the entropy for this attribute is:

$$Entropy(S \mid bare-faced) = \frac{17600}{20000}\cdot 0.994 + \frac{2400}{20000}\cdot 0 = 0.8747$$

Leg-cover attribute:

Values	Male	Female	Entropy(S_i)
S_1 (yes)	0	4000	[0, 1]
S_2(no)	12000	4000	[12000/16000, 4000/16000]

$$Entropy(S_1) = 0 - \log_2(1) = 0$$

$$Entropy(S_2) = -\frac{12}{16}\log_2\left(\frac{12}{16}\right) - \frac{4}{16}\log_2\left(\frac{4}{16}\right) = 0.8112$$

Therefore, the entropy for this attribute is:

$$Entropy(S \mid leg-cover) = \frac{4000}{20000}\cdot 0 + \frac{16000}{20000}\cdot 0.8112 = 0.649$$

So, we choose the attribute that maximizes the overall information gain

- Bare-faced: Gain(S) = Ent(S) − Ent(S|bare-faced) = 0.971 – 0.8747 = 0.0963
- Leg-cover: Gain(S) = Ent(S) − Ent(S|leg-cover) = 0.971-0.649 = 0.322

Thus, "Leg-cover" is the most effective one that maximizes the information gain.

Q3: We would like to indicate what factor(s) may affect sunburn. We have the following dataset of 8 samples, with attributes like Hair, Height, Weight and Lotion. The question is practically to create a decision tree, based on this dataset, and try to construct the tree structure. Do we need all these attributes?

Name	Hair	Height	Weight	Lotion	Result
Sarah	blonde	average	light	no	sunburned (positive)
Dana	blonde	tall	average	yes	none (negative)
Alex	brown	short	average	yes	none
Annie	blonde	short	average	no	sunburned
Emily	red	average	heavy	no	sunburned
Pete	brown	tall	heavy	no	none
John	brown	average	heavy	no	none
Katie	blonde	short	light	yes	none

Solution

We have 3+ (positives) and 5- (negatives) cases, from the result column.
The parent entropy (i.e. total) is calculated as:

$$Entropy(S) = -\frac{3}{8}\log_2\left(\frac{3}{8}\right) - \frac{5}{8}\log_2\left(\frac{5}{8}\right) = 0.9544$$

We need to check the entropy for each attribute individually and then find which attribute maximizes the information gain (in order to be used at root position).

Hair attribute:

Values	positive	negative	Entropy(S_i)
S_{Blonde}	2	2	[2/4, 2/4] = [1/2, 1/2]
S_{Brown}	0	3	[0, 3/3] = [0, 1]
S_{Red}	1	0	[1, 0]

$$Entropy(S_{Blonde}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

$$Entropy(S_{brown}) = 0 - \log_2(1) = 0$$

$$Entropy(S_{Red}) = -\log_2(1) - 0 = 0$$

Therefore, the entropy for this attribute is:

$$Entropy(S \mid hair) = \frac{4}{8}\cdot 1.0 + \frac{3}{8}\cdot 0 + \frac{1}{8}\cdot 0 = 0.5$$

The information gain for Hair is: Entropy(S) – Entropy(S | hair) = 0.9544-0.5=0.4544

Height attribute:

Values	positive	negative	Entropy(S_i)
$S_{average}$	2	1	[2/3, 1/3]
S_{tall}	0	2	[0, 2/2] = [0, 1]
S_{short}	1	2	[1/3, 2/3]

Entropy ($S_{Average}$) = 0.91829 check these results
Entropy (S_{Tall}) = 0
Entropy (S_{Short}) = 0.91829

Therefore, the entropy for this attribute is:
Entropy(S | Height) = (3/8)*0.91829 + (2/8)*0 + (3/8)*0.91829 = 0.6887
The information gain for Height is: Entropy(S) – Entropy(S | height) = 0.9544-0.6887=0.2657

Weight attribute:

Values	positive	negative	Entropy(S_i)
S_{light}	1	1	[1/2, 1/2]
$S_{average}$	1	2	[1/3, 2/3]
S_{heavy}	1	2	[1/3, 2/3]

Entropy (S_{light}) = 1 check these results
Entropy ($S_{average}$) = 0.91829
Entropy (S_{heavy}) = 0.91829

Therefore, the entropy for this attribute is:
Entropy(S | Weight) = (2/8)*1 + (3/8)*0.91829 + (3/8)*0.91829 = 0.9387
The information gain for Weight is: Entropy(S) – Entropy(S | weight) = 0.9544-0.9387=0.0157

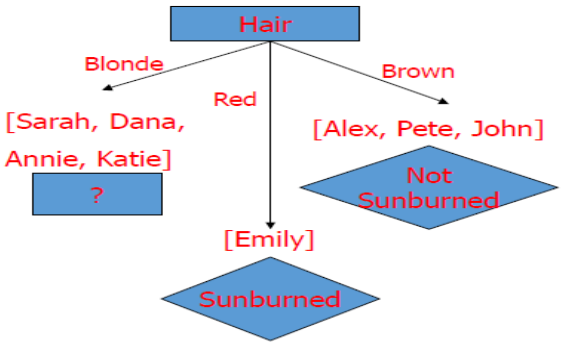
Lotion attribute:

Values	positive	negative	Entropy(S_i)
S_{yes}	0	3	[0, 1]
S_{no}	3	2	[3/5, 2/5]

Entropy (S_{yes}) = 0 check these results
Entropy (S_{no}) = 0.97095

Therefore, the entropy for this attribute is:
Entropy(S | Lotion) = (3/8)*0 + (5/8)*0.97095 = 0.6068
The information gain for Lotion is: Entropy(S) – Entropy(S | lotion) = 0.9544-0.6068=0.3475

Thus, based on the information gain for each attribute, Hair is the one chosen for the root node. But we need to proceed to the creation of the remaining decision tree.



As we can see, Hair seems to be a good choice, solved the tree regarding brown and red hairs, but not for blonde. So for this, we need another attribute to look for. We need to concentrate only for the "blonde" samples.

Name	Hair	Height	Weight	Lotion	Sunburned
Sarah	Blonde	Average	Light	No	Yes
Dana	Blonde	Tall	Average	Yes	No
Annie	Blonde	Short	Average	No	Yes
Katie	Blonde	Short	Light	Yes	No

Practically, we repeat the same procedure, but only for this limited number of samples.

We have 2+ (positives) and 2- (negatives) cases, from the result column.
The parent entropy (i.e. total) is calculated as:

$$Entropy(S) = -\frac{2}{4}\log_2\left(\frac{2}{4}\right) - \frac{2}{4}\log_2\left(\frac{2}{4}\right) = 1$$

We need to check the entropy for each attribute individually (except hair) and then find which attribute maximizes the information gain (in order to be used at that missing position).

Height attribute:

Values	positive	negative	Entropy(S_i)
$S_{average}$	1	0	[1, 0]
S_{tall}	0	1	[0, 1]
S_{short}	1	1	[1/2, 1/2]

Entropy ($S_{Average}$) = 0 check these results
Entropy (S_{Tall}) = 0
Entropy (S_{Short}) = 1

Therefore, the entropy for this attribute is:
Entropy(S | Height) = (1/4)*0 + (1/4)*0 + (2/4)*1 = 0.5
The information gain for Height is: Entropy(S) – Entropy(S | height) = 1.0-0.5=0.5

Weight attribute:

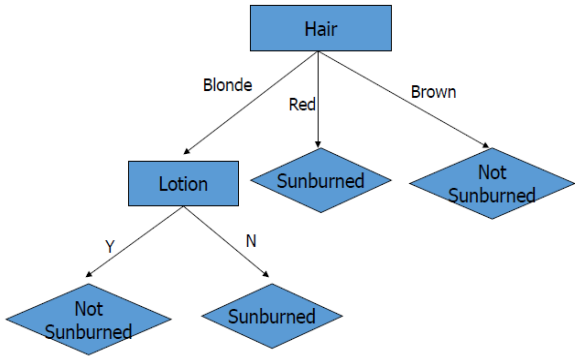
$S_{Average}$ = [1+, 1-] $E(S_{Average}) = 1$
S_{Light} = [1+, 1-] $E(S_{Light}) = 1$
Information Gain(S, Weight) = 1 – [(2/4)*1 + (2/4)*1] = 0

Lotion attribute:

S_{Yes} = [0+, 2-] $E(S_{Yes}) = 0$
S_{No} = [2+, 0-] $E(S_{No}) = 0$

Information Gain(S, Lotion) = 1 – [(2/4)*0 + (2/4)*0] = 1

So, Lotion is the chosen one.



R-implementations

First attempt on DT using Iris dataset

Decision Trees (DT) are popular supervised machine learning algorithms. You will often find the abbreviation CART when reading up on decision trees. CART stands for Classification and Regression Trees. This is one specific type of DT. An alternative scheme is C4.5/C5.0. CART uses the Gini index as

an attribute selection measure in order to build a decision tree, while C5.0 utilizes entropy. Both schemes have advantages and disadvantages. In this example we are going to create a Classification Tree. Meaning we are going to attempt to classify our data into one of the (three in this case) classes. In this first example we are going to be using the Iris data set native to R.

```
#start code##########################
iris
head(iris)
```

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> |
```

As you can see, our data has 5 variables – Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species. The first 4 variables refer to measurements of flower parts and the species identifies which species of iris this flower represents. What we are going to attempt to do here is develop a predictive model that will allow us to identify the species of iris based on measurements. The species we are trying to predict are setosa, virginica, and versicolor. These are our three classes we are trying to classify our data as. In order to build our decision tree, first we need to install the correct package.

```
library(rpart)
```

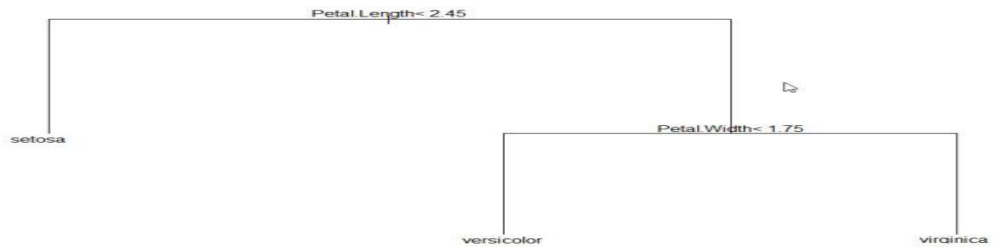
Next we are going to create our tree. Since we want to predict Species – that will be the first element in our fit equation. Pay attention to the "class" selection in the method.

```
fit <- rpart(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, method="class", iris)
```

Now, let's take a look at the tree.

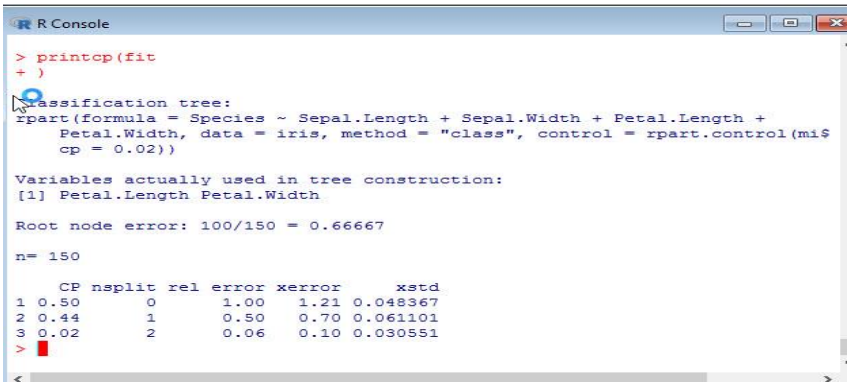
```
par(xpd = NA)   # otherwise on some devices the text is clipped
plot(fit)
text(fit, digits = 3)
print(fit, digits = 2)
```

To understand what the output say, according to our model, if the Pedal.Length is < 2.45 then the flower is classified as setosa. If not, it goes to the next split – Petal Width. If < 1.75 then versicolor, else virginica.



Now, we want to take a look at how good the model is.

printcp(fit)



Let's look down at the table on the bottom. The first row has a CP = 0.50. This means (approx) that the first split reduced the relative error by 0.5. You can see this in the relative error in the second row. Now the 2nd row CP = 0.44, so the second split improved the relative error in the third row to 0.06. If we wish to see another metric for the goodness of the model, look at the xerror (cross validation error) of the final row. 0.10 is a nice low number. Let's make now a prediction. Start by creating some test data.

testData <-data.frame (Sepal.Length = 1, Sepal.Width = 4, Petal.Length =1.2, Petal.Width=0.3)

Now let's predict

predict(fit, testData, type="class")

```
> predict(fit, testData, type="class")
     1
setosa
Levels: setosa versicolor virginica
>
```

As you can see, the model predicted setosa. If you look back at the tree, you will see why. Let's do one more prediction

```
newdata<-data.frame(Sepal.Length=c(3,8,7,5),
 Sepal.Width=c(2,3,2,6),
 Petal.Length=c(5.4,3.2,4.6,5.3),
 Petal.Width=c(4,3,6,1.3))
 predict (fit, newdata, type="class")
```

```
> newdata<-data.frame(Sepal.Length=c(3,8,7,5),
+   Sepal.Width=c(2,3,2,6),
+   Petal.Length=c(5.4,3.2,4.6,5.3),
+   Petal.Width=c(4,3,6,1.3))
>
> predict (fit, newdata, type="class")
          1           2           3           4
 virginica  virginica  virginica versicolor
Levels: setosa versicolor virginica
>
```

The model predicts 1, 2, 3 are virginica and 4 is versicolor.

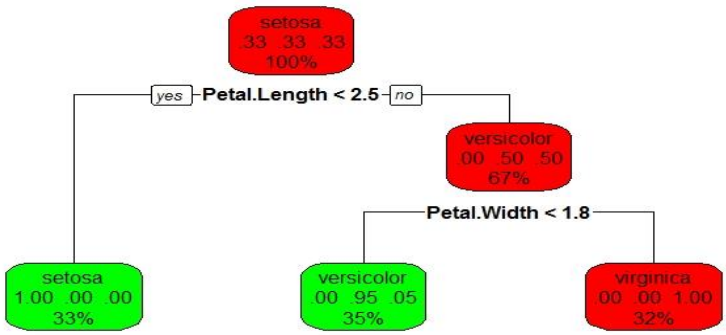
Second attempt on DT using Iris dataset (a bit more complicated)

Here we are going to create separate training/testing files.

```
## Decision Tree for Iris dataset
library(rpart)
library(rpart.plot)
library(caret)

str(iris)
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .

summary(iris)
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50

v <- iris$Species    # the 5th column – i.e. desired output
table(v)
## v
##     setosa versicolor  virginica
##         50         50         50

set.seed(522)
# runif function returns a uniform distribution which can be further conditionally split into 75-25 ratio
iris[, 'train'] <- ifelse(runif(nrow(iris)) < 0.75, 1, 0)
# specify separate training / testing sets
trainSet <- iris[iris$train == 1,]
testSet <- iris[iris$train == 0, ]

trainColNum <- grep('train', names(trainSet))

trainSet <- trainSet[, -trainColNum]
testSet <- testSet[, -trainColNum]

treeFit <- rpart(Species~.,data=trainSet,method = 'class')  # check the class method
print(treeFit)
```

```
## n= 111
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 111 74 setosa (0.33333333 0.33333333 0.33333333)
##   2) Petal.Length< 2.45 37  0 setosa (1.00000000 0.00000000 0.00000000) *
##   3) Petal.Length>=2.45 74 37 versicolor (0.00000000 0.50000000 0.50000000)
##     6) Petal.Width< 1.75 39  2 versicolor (0.00000000 0.94871795 0.05128205) *
##     7) Petal.Width>=1.75 35  0 virginica (0.00000000 0.00000000 1.00000000) *
```

```
rpart.plot(treeFit, box.col=c("red", "green"))
```




```
Prediction1 <- predict(treeFit,newdata=testSet[-5],type = 'class')
## Print the confusion matrix to check the accuracy and other statistics
confusionMatrix(Prediction1,testSet$Species)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction   setosa versicolor virginica
##   setosa         13          0         0
##   versicolor      0         12         3
##   virginica       0          1        10
##  Overall Statistics
##
##                Accuracy : 0.8974
##                  95% CI : (0.7578, 0.9713)
##     No Information Rate : 0.3333
##     P-Value [Acc > NIR] : 3.435e-13
##
##                   Kappa : 0.8462
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: setosa Class: versicolor Class: virginica
## Sensitivity                1.0000           0.9231          0.7692
## Specificity                1.0000           0.8846          0.9615
## Pos Pred Value             1.0000           0.8000          0.9091
## Neg Pred Value             1.0000           0.9583          0.8929
## Prevalence                 0.3333           0.3333          0.3333
```

```
## Detection Rate           0.3333    0.3077    0.2564
## Detection Prevalence     0.3333    0.3846    0.2821
## Balanced Accuracy        1.0000    0.9038    0.8654
```

We can also check details of our Tree, via the summary command

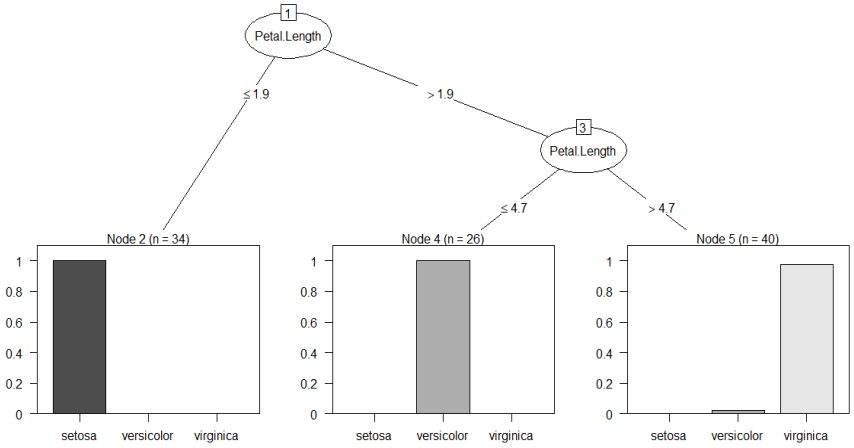
```
summary (treeFit)   # in case of large trees, the information included under summary could be huge

Call:
rpart(formula = Species ~ ., data = trainSet, method = "class")
  n= 111

       CP nsplit  rel error     xerror       xstd
1 0.500000      0 1.00000000 1.18918919 0.05770481
2 0.472973      1 0.50000000 0.78378378 0.07111459
3 0.010000      2 0.02702703 0.06756757 0.02952873

Variable importance
 Petal.Width Petal.Length Sepal.Length  Sepal.Width
          35           32           20           13

Node number 1: 111 observations,    complexity param=0.5
  predicted class=setosa      expected loss=0.6666667  P(node) =1
    class counts:    37    37    37
   probabilities: 0.333 0.333 0.333
  left son=2 (37 obs) right son=3 (74 obs)
  Primary splits:
      Petal.Length < 2.45 to the left,  improve=37.00000, (0 missing)
      Petal.Width  < 0.75 to the left,  improve=37.00000, (0 missing)
      Sepal.Length < 5.45 to the left,  improve=25.31579, (0 missing)
      Sepal.Width  < 3.15 to the right, improve=14.33589, (0 missing)
  Surrogate splits:
      Petal.Width  < 0.75 to the left,  agree=1.000, adj=1.000, (0 split)
      Sepal.Length < 5.45 to the left,  agree=0.919, adj=0.757, (0 split)
      Sepal.Width  < 3.25 to the right, agree=0.829, adj=0.486, (0 split)

Node number 2: 37 observations
  predicted class=setosa      expected loss=0  P(node) =0.3333333
    class counts:    37     0     0
   probabilities: 1.000 0.000 0.000

Node number 3: 74 observations,    complexity param=0.472973
  predicted class=versicolor  expected loss=0.5  P(node) =0.6666667
    class counts:     0    37    37
   probabilities: 0.000 0.500 0.500
  left son=6 (39 obs) right son=7 (35 obs)
  Primary splits:
      Petal.Width  < 1.75 to the left,  improve=33.205130, (0 missing)
      Petal.Length < 4.75 to the left,  improve=28.190480, (0 missing)
      Sepal.Length < 5.75 to the left,  improve= 7.966507, (0 missing)
      Sepal.Width  < 2.45 to the left,  improve= 3.865672, (0 missing)
  Surrogate splits:
      Petal.Length < 4.75 to the left,  agree=0.905, adj=0.800, (0 split)
      Sepal.Length < 6.15 to the left,  agree=0.716, adj=0.400, (0 split)
      Sepal.Width  < 2.95 to the left,  agree=0.649, adj=0.257, (0 split)
```

```
Node number 6: 39 observations
  predicted class=versicolor  expected loss=0.05128205  P(node) =0.3513514
    class counts:     0    37     2
   probabilities: 0.000 0.949 0.051

Node number 7: 35 observations
  predicted class=virginica   expected loss=0  P(node) =0.3153153
    class counts:     0     0    35
   probabilities: 0.000 0.000 1.000

printcp(treeFit)
Classification tree:
rpart(formula = Species ~ ., data = trainSet, method = "class")

Variables actually used in tree construction:
[1] Petal.Length Petal.Width

Root node error: 74/111 = 0.66667

n= 111

       CP nsplit rel error   xerror     xstd
1 0.50000      0  1.000000 1.189189 0.057705
2 0.47297      1  0.500000 0.783784 0.071115
3 0.01000      2  0.027027 0.067568 0.029529
```

Another on DT using Iris dataset (C5.0 algorithm this time)

```
library(caret)
library(C50)
library(printr)

str(iris)
summary(iris)
## Decision Tree for Iris dataset

##Split iris data into train and test
train.indices <- sample(1:nrow(iris), 100)
iris.train <- iris[train.indices, ]
iris.test <- iris[-train.indices, ]

###Train decision tree
model <- C5.0(Species ~., data=iris.train)
### Test
results <- predict(object=model, newdata=iris.test, type="class")
###Look at confusion matrix
table(results, iris.test$Species)
results      setosa versicolor virginica
  setosa         14          0         0
  versicolor      0         20         2
  virginica       0          2        12
```

```
plot(model)
```



Cars Evaluation Data Set Description

The Cars Evaluation data set consists of 7 attributes, 6 as feature attributes and 1 as the target attribute. All the attributes are categorical. We will try to build a classifier for predicting the Class attribute. The index of target attribute is 7th.

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

For implementing Decision Tree in R, we need to import "caret" package & "rpart.plot". Caret helps to perform various tasks for our machine learning work, while the rpart.plot package will help to get a visual plot of the decision tree.

```
library(caret)
library(rpart.plot)
```

Data Import

For importing the data and manipulating it, we are going to use data frames. First of all, we need to download the dataset. All the data values are separated by commas. After downloading the data file, you need to set your working directory via console else save the data file in your current working directory.

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

For importing data into an R data frame, we can use **read.csv()** method with parameters as a file name and whether our dataset consists of the 1st row with a header or not. If a header row exists then, the header should be set **TRUE** else header should set to **FALSE**. For checking the structure of data frame we can call the function **str()** over car_df:

```
str(car_df)
'data.frame': 1728 obs. of  7 variables:
$ V1: Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
$ V2: Factor w/ 4 levels "high","low","med",..: 4 4 4 4 4 4 4 4 4 4 ...
$ V3: Factor w/ 4 levels "2","3","4","5more": 1 1 1 1 1 1 1 1 1 1 ...
$ V4: Factor w/ 3 levels "2","4","more": 1 1 1 1 1 1 1 1 1 2 ...
$ V5: Factor w/ 3 levels "big","med","small": 3 3 3 2 2 2 1 1 1 3 ...
$ V6: Factor w/ 3 levels "high","low","med": 2 3 1 2 3 1 2 3 1 2 ...
$ V7: Factor w/ 4 levels "acc","good","unacc",..: 3 3 3 3 3 3 3 3 3 3 ...
```

The above output shows us that our dataset consists of 1728 observations each with 7 attributes. To check top 5-6 rows of the dataset, we can use head().

```
head(car_df)
    V1    V2 V3 V4    V5   V6    V7
1 vhigh vhigh  2  2 small  low unacc
2 vhigh vhigh  2  2 small  med unacc
3 vhigh vhigh  2  2 small high unacc
4 vhigh vhigh  2  2   med  low unacc
5 vhigh vhigh  2  2   med  med unacc
6 vhigh vhigh  2  2   med high unacc
```

All the features are categorical, so normalization of data is not needed.

Data Slicing

Data slicing is a step to split data into train and test set. Training data set can be used specifically for our model building. Test dataset should not be mixed up while building model.

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

The **set.seed()** method is used to make our work replicable. The caret package provides a function createDataPartition() for partitioning our data into train and test sets. We utilize 3 parameters. The "y" parameter takes the value of variable according to which data needs to be partitioned. In our case, target variable is at **V7**, so we are passing **car_df$V7**. The "p" parameter holds a decimal value in the range of 0-1. It's to show that percentage of the split. We are using p=0.7. It means that data split should be done in 70:30 ratio. The "list" parameter is for whether to return a list or matrix. We are passing FALSE for not returning a list. The createDataPartition() method is returning a matrix "intrain" with record's indices. By passing values of intrain, we are splitting training data and testing data. The line training <- car_df[intrain,] is for putting the data from data frame to training data. Remaining data is saved in the testing data frame, testing <- car_df[-intrain,].

For checking the dimensions of our training data frame and testing data frame, we can use these:

```
#check dimensions of train & test set
dim(training); dim(testing);
```

To check whether our data contains missing values or not, we can use **anyNA()** method. Here, NA means Not Available.

```
anyNA(car_df)
[1] FALSE
```

Since it's returning **FALSE**, it means we don't have any missing values.

```
summary(car_df)
V1 V2 V3 V4 V5 V6 V7
high :432 high :432 2 :432 2 :576 big :576 high:576 acc : 384
low :432 low :432 3 :432 4 :576 med :576 low :576 good : 69
med :432 med :432 4 :432 more:576 small:576 med :576 unacc:1210
vhigh:432 vhigh:432 5more:432
```

Training the Decision Tree classifier with criterion as information gain

Caret package provides **train()** method for training our data for various algorithms. We just need to pass different parameter values for different algorithms. Before **train()** method, we will first use **trainControl()** method. It controls the computational nuances of the **train()** method.

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

We are setting 3 parameters of **trainControl()** method. The "method" parameter holds the details about re-sampling method. We can set "method" with many values like "boot", "boot632", "cv", "repeatedcv", "LOOCV", "LGOCV" etc. For this exercise, we'll use repeatedcv i.e. repeated cross-validation. The "number" parameter holds the number of re-sampling iterations. The "repeats" parameter contains the complete sets of folds to compute for our repeated cross-validation. We are using setting number=10 and repeats=3. This **trainControl()** methods returns a list. We are going to pass this on our **train()** method.

Before training our Decision Tree classifier, **set.seed()**.

For training Decision Tree classifier, **train()** method should be passed with "method" parameter as "rpart". There is another package "rpart", it is specifically available for decision tree implementation. Caret links its train function with others to make our work simple. We are passing our target variable V7. The "V7~." denotes a formula for using all attributes in our classifier and V7 as the target variable. The "trControl" parameter should be passed with results from our **trianControl()** method. We can use different criterions while splitting our nodes of the tree. To select the specific strategy, we need to pass a parameter "parms" in our train() method. It should contain a list of parameters for our rpart method. For splitting criterions, we need to add a "split" parameter with values either "information" for information gain & "gini" for gini index. In the above snippet, we are using information gain as a criterion.

Trained Decision Tree classifier results

We can check the result of our **train()** method by a print dtree_fit variable. It is showing us the accuracy metrics for different values of cp. Here, cp is complexity parameter for our dtree.

```
dtree_fit
      CART
```

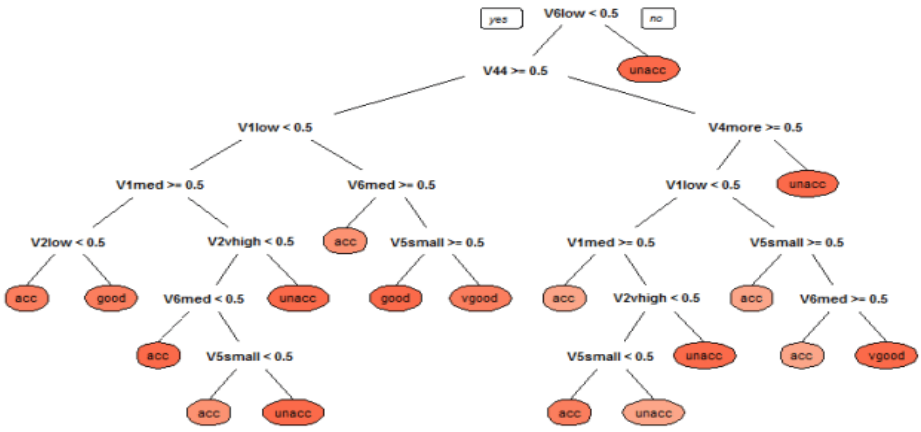
```
1212 samples
  6 predictor
  4 classes: 'acc', 'good', 'unacc', 'vgood'

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.
```

Plot Decision Tree

We can visualize our decision tree by using prp() method.

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

The decision tree visualization shown above indicates its structure. It shows the attribute's selection order for criterion as information gain.



Prediction

Now, our model is trained with cp = 0.01123596. We are ready to predict classes for our test set. We can use predict() method. Let's try to predict target variable for test set's 1st record.

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

For our 1st record of testing data classifier is predicting class variable as "unacc". Now, its time to predict target variable for the whole test set.

```
test_pred <- predict(dtree_fit, newdata = testing)
confusionMatrix(test_pred, testing$V7 )  #check accuracy  # why there is a mistake here?

 gg <- as.factor(testing$V7)
confusionMatrix(test_pred, gg)
```

```
Confusion Matrix and Statistics

          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
```

The above results show that the classifier with the criterion as information gain is giving 83.72% of accuracy for the test set. Try to program a decision tree classifier using splitting criterion as gini index. What is the outcome?