Topics in Data Science

Lecture 12

Department of Mathematics and Statistics

Decision Trees

Decision Trees are useful supervised Machine learning algorithms that have the ability to perform both regression and classification tasks.

- It is characterized by nodes and branches, where the tests on each attribute are represented at the nodes, the outcome of this procedure is represented at the branches and the class labels are represented at the leaf nodes.
- It uses a tree-like model based on various decisions that are used to compute their probable outcomes.
- These types of tree-based algorithms are one of the most widely used algorithms due to the fact that these algorithms are easy to interpret and use.
- The predictive models developed by this algorithm are found to have good stability and a descent accuracy due to which they are very popular.

Decision Trees

- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training observations in the region to which it belongs.
- In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the class proportions among the training observations that fall into that region.
- A decision tree reaches its decision by performing a sequence of tests.

Decision Trees

- Tree that is consistent with the examples and is as small
- With some simple heuristics, we can find a good approximate solution: a small (but not smallest) consistent tree.
- The DECISION-TREE-LEARNING algorithm adopts a greedy divide-and-conquer strategy
- Always test the most important attribute first.
- By "most important attribute" we mean the one that makes the most difference to the classification of an example. $\rightarrow$ how ?

Decision Trees

A decision tree is a graphical representation of possible solutions to a decision based on certain conditions. It is called a decision tree because it starts with a single variable, which then branches off into a number of solutions, just like a tree. A decision tree has three main components

- **Root Node:** The top most node is called Root Node. This is the node that begins the splitting process by finding the variable that best splits the target variable.
- **Decision/Internal Node :** The nodes in which predictors (independent variables) are tested and each branch represents an outcome of the test.
- **Node Purity:** Decision nodes are typically impure, or a mixture of both classes of the target variable. Pure nodes are those that have one class – hence the term pure. They either have green or red dots only in the image.
- **Leaf/Terminal Node:** It holds a class label (category) - Yes or No (Final Classification Outcome).

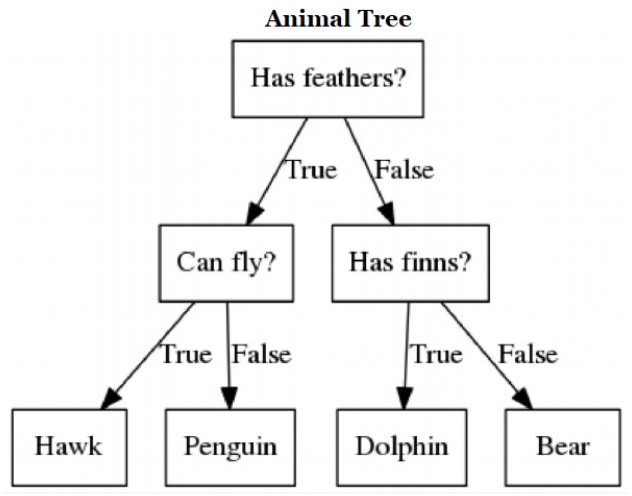
Splitting and Pruning

How to determine the root node? To identify the root node, we would evaluate the impact of all the variables we have on the target variable to identify the variable that divides the classes into the most homogeneous groups.

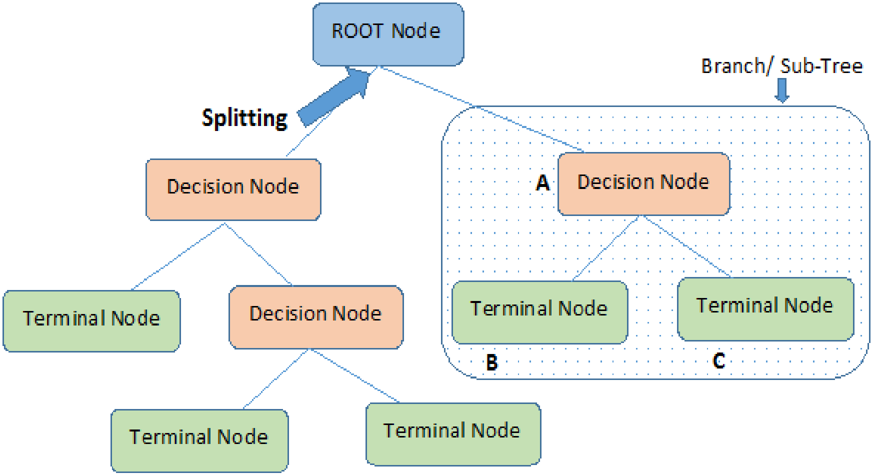
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

Decision Tree-Example

Animals: {Hawk, Penguin, Dolphin, Bear}



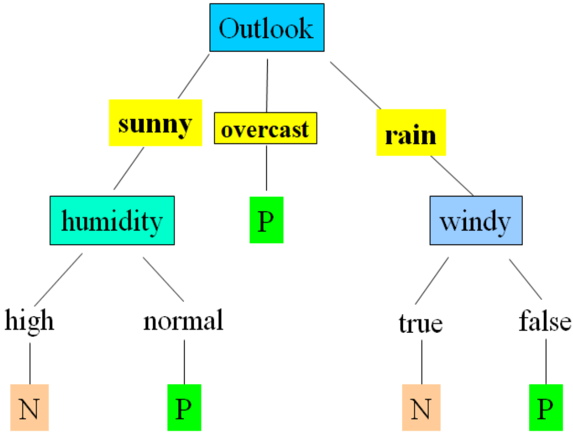
Decision Tree



Decision Tree-Example

Whether or not an individual will play golf depends on various factors such as weather outlook, temperature, humidity, and windy condition.

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N



Note: Overcast is pure subset as there are 4 yes and 0 no

Advantages and Disadvantages of Decision Tree

- **Advantages:**
  - Decision tree is easy to interpret.
  - Decision Tree works even if there is nonlinear relationships between variables. It does not require the linearity assumption.
  - Decision Tree is not sensitive to outliers.
- **Disadvantages**
  - The decision tree model generally overfits. This means that it does not perform well on the validation sample. This problem gets solved by setting constraints on model parameters and pruning
  - It assumes all independent variables interact each other, It is generally not the case every time.

Constructing Decision Trees

How do we choose the attributes and the order in which they appear in a DT?

- Recursive partitioning of the original data table
- Heuristic - each generated partition has to be less random (entropy reduction) than previously generated partitions

How it works?

Since we plan to assign an observation in a given region to the most common class of training observations in that region, the classification error rate is simply the fraction of training observations in that region that do not belong to the most common class:

$$E = 1 - max_k(\hat{p}_{mk})$$

where, $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class. However, it turns out that classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

- Gini index
- Entropy

Gini Index and Entropy

Gini index and entropy both are broadly similar and seek to determine which variable would split the data to make the underlying child nodes more homogeneous or pure. Both are used in different Decision Tree algorithms.

- **Entropy:** In the context of Decision Trees, entropy is a measure of disorder or impurity in a node. Thus, a node with more variable composition, would be considered to have higher Entropy than a node which has only one outcome.If all the instances belong to one class then entropy is zero. The maximum level of entropy or disorder is given by 1 and minimum entropy is given by a value 0 and is defined by

  $$E = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}).$$

  where K represents the no. of classes in the target variable and $\hat{p}_{mk}$ represents the proportion of training observations in the region m that are from class k.

  The Entropy has a maximum impurity of 1 and maximum purity is 0.

Gini Index and Entropy

- **Gini Index:** The other way of splitting a decision tree is via the Gini Index. The Entropy and Information Gain method focuses on purity and impurity in a node. The Gini Index or Impurity measures the probability for a random instance being misclassified when chosen randomly. The lower the Gini Index, the better the lower the likelihood of misclassification. The formula for Gini index is

$$Gini = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

  where K represents the no. of classes in the target variable and $\hat{p}_{mk}$ represents the proportion of training observations in the region m that are from class k.

The measure of the degree of probability of a particular variable being wrongly classified when it is randomly chosen is called the Gini index or Gini impurity.
The Gini index has a maximum impurity is 0.5 and maximum purity is 0.The feature with the least Gini index is selected.

Deviance

- For classification trees, the deviance is given by the summary() function and can be calculated via

$$-2\sum_{m}\sum_{k} n_{mk} log(\hat{p}_{mk})$$

  where n_{mk} is the number of observations in the mth terminal node that belongs to class k.
- A small deviance indicates a tree that provides a good fit to the (training) data
- The residual mean deviance reported is simply the deviance divided by $n - |T_0|$.

Information Gain

- We want to determine which attribute in a given set of training feature vectors is most useful for discriminating between the classes to be learned.
- Information gain tells us how important a given attribute of the feature vectors is.
- We will use it to decide the ordering of attributes in the nodes of a decision tree

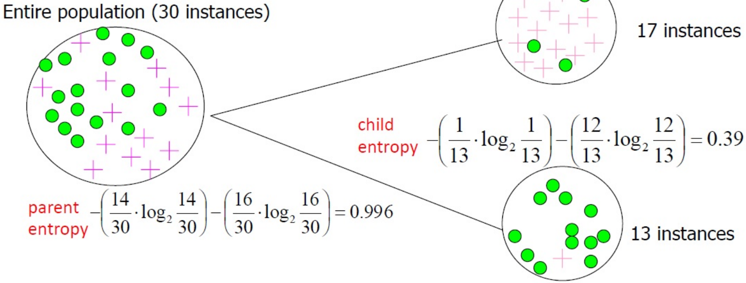
Information Gain is the expected reduction in entropy of target variable Y for data sample S, due to sorting

Information Gain

Calculating Information Gain

Information Gain = entropy(parent) – [average entropy(children)]

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$



child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$

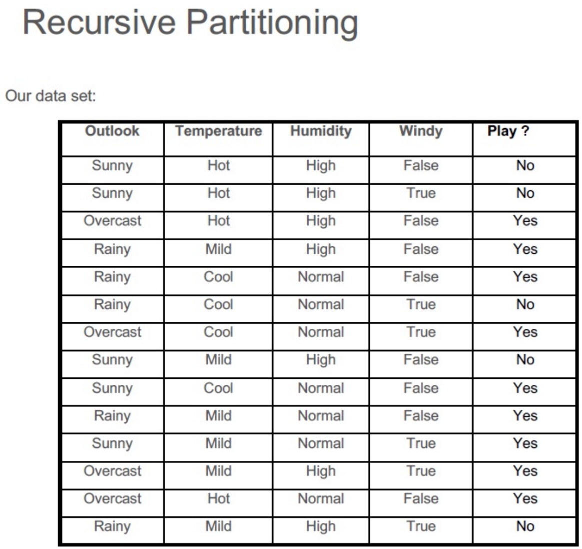
parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

Based on slide by Pedro Domingos

Partitioning the Data Set



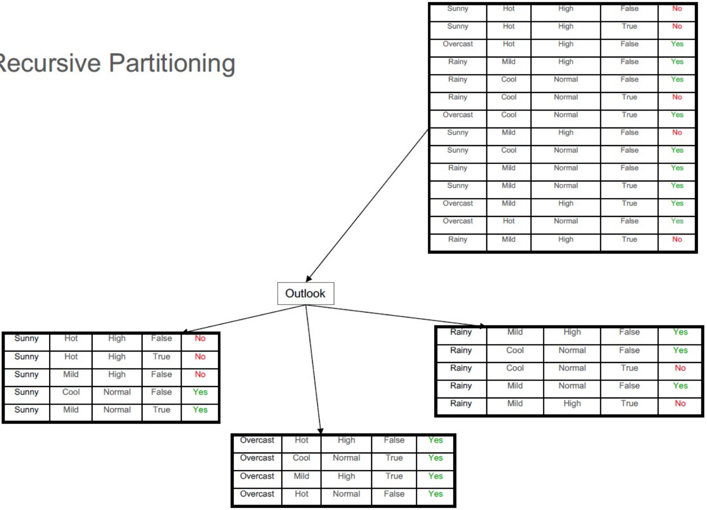
Recursive Partitioning

Our data set:

Outlook	Temperature	Humidity	Windy	Play ?
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Partitioning the Data Set

Recursive Partitioning



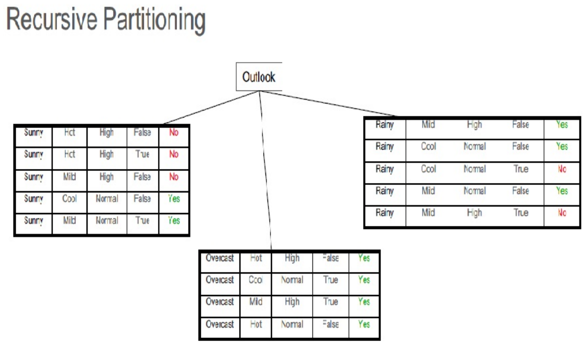
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Partitioning the Data Set

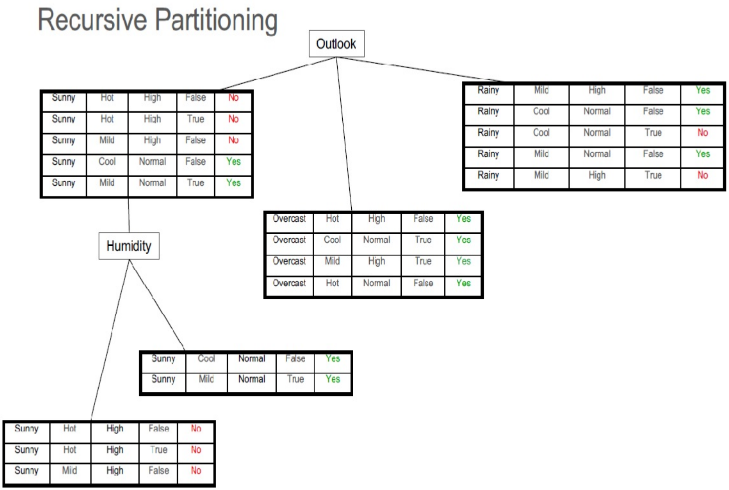


Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Partitioning the Data Set


Recursive Partitioning

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

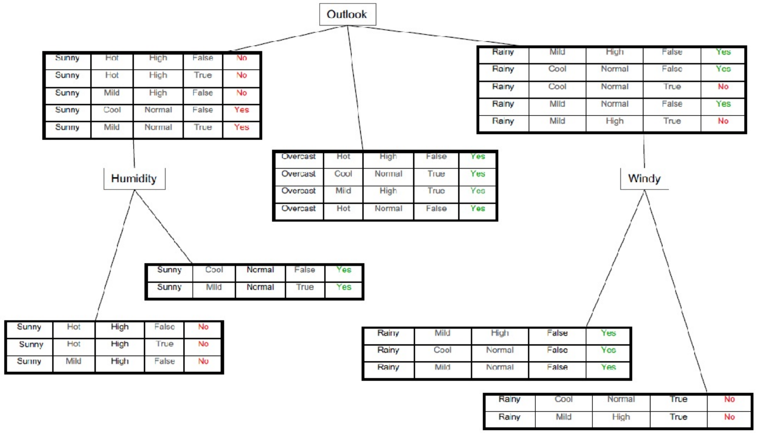
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No

Partitioning the Data Set

Recursive Partitioning



Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No

Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes

Rainy	Cool	Normal	True	No
Rainy	Mild	High	True	No

Calculating the Entropy

$$\begin{aligned}
Entropy(PlayGolf) &= 0.940 \\
Entropy(PlayGolf, Outlook) &= 0.693 \\
Entropy(PlayGolf, Temperature) &= 0.911 \\
Entropy(PlauGolf, Humidity) &= 0.788 \\
Entropy(PlayGolf, Windy) &= 0.892
\end{aligned}$$

The next step is to calculate the information gain for each of the attributes. The information gain is calculated from the split using each of the attributes. Then the attribute with the largest information gain is used for the split

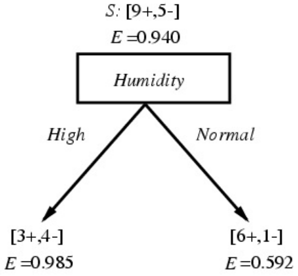
$$\begin{aligned}
Gain(PlayGolf, Outlook) &= Entropy(PlayGolf) - Entropy(PlayGolf, Out \\
Gain(PlayGolf, Outlook) &= 0.94 - 0.693 = 0.247 \\
Gain(PlayGolf, Temperature) &= 0.94 - 0.911 = 0.029 \\
Gain(PlayGolf, Humidity) &= 0.94 - 0.788 = 0.152 \\
Gain(PlayGolf, Windy) &= 0.94 - 0.892 = 0.048
\end{aligned}$$

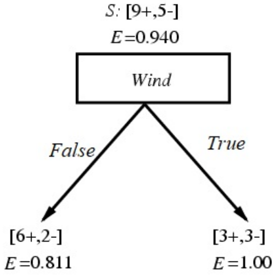
Calculating the Entropy using R

```
> library(DescTools)
> PlayGolf=c(9,5)
> Entropy(PlayGolf)
[1] 0.940286
```

Selecting the Next Attribute

Consider the golf data:



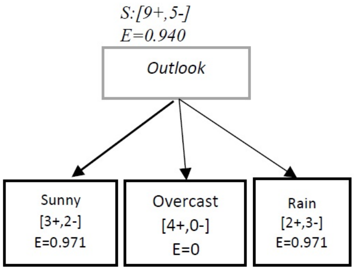


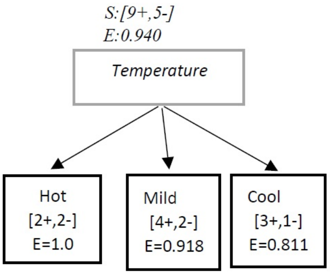
Note that :

$$\begin{aligned} Gain(S, Humidity) &= 0.940 - 7/14 \times 0.985 - 7/14 \times 0.592 \\ &= 0.151 \\ Gain(S, Windy) &= 0.940 - 8/14 \times 0.811 - 6/14 \times 1.00 \\ &= 0.048 \end{aligned}$$

Selecting the Next Attribute

Consider the golf data:





Note that :

$$\begin{aligned} Gain(S, Outlook) &= 0.940 - 5/14 \times 0.971 - 4/14 \times 0 - 5/14 \times 0.971 \\ &= 0.246 \\ Gain(S, Temp) &= 0.940 - 4/14 \times 1.0 - 6/14 \times 0.918 - 4/14 \times 0.811 \\ &= 0.029 \end{aligned}$$

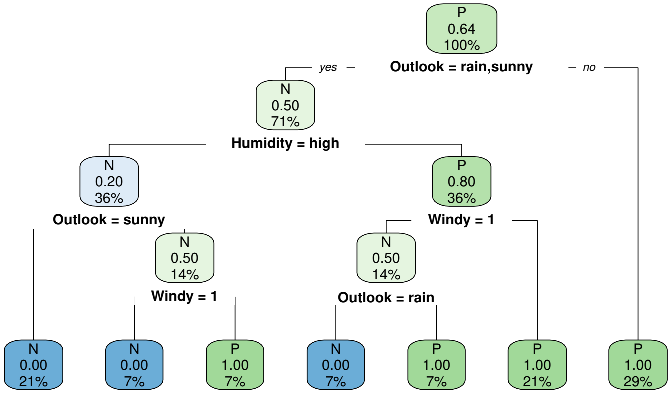
Decision Tree

Consider problems where the inputs have discrete values and the output has exactly two possible values; this is Boolean classification, where each example input will be classified as true (a positive example) or false (a negative example). A decision tree reaches its decision by performing a sequence of tests.

```
library(rpart)
library(rpart.plot)
data=read.csv("C:\\Users\\aryalg\\golf.csv")
attach(data)
options(repr.plot.width=6,repr.plot.height=5)
tree1=rpart(Class~., data=data, method = "class",
            minsplit = 2, minbucket = 1)
rpart.plot(tree1, space=1, split.cex=1, nn.border.col=0)
```

Tree Diagram-Example

Consider the data related to the playgolf data



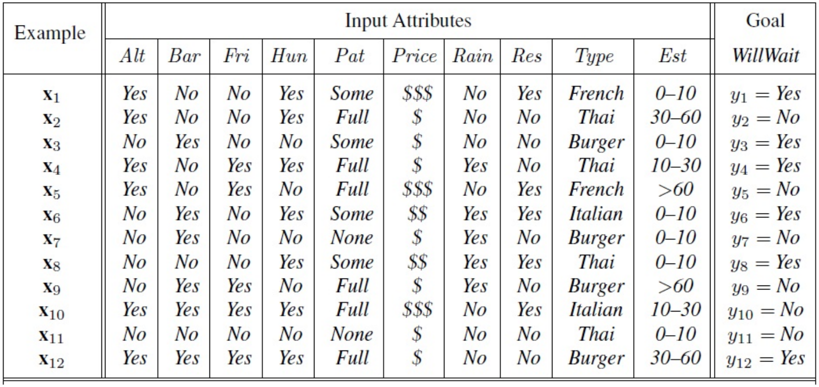
Decision Tree-Example

We will build a decision tree to decide whether to wait for a table at a restaurant. The aim here is to learn a definition for the goal predicate WillWait (T or F). First we list the attributes that we will consider as part of the input:

- Alternate: whether there is a suitable alternative restaurant nearby.
- Bar: whether the restaurant has a comfortable bar area to wait in.
- Fri/Sat: true on Fridays and Saturdays.
- Hungry: whether we are hungry.
- Patrons: how many people are in the restaurant (values are None, Some, and Full).
- Price: the restaurant's price range ($, $$, $$$).
- Raining: whether it is raining outside.
- Reservation: whether we made a reservation.
- Type: the kind of restaurant (French, Italian, Thai, or burger).
- WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, or >60)

Decision Tree-Example contd.

The task at hand is to develop an application that advises whether we should wait for a table at a restaurant or not.



Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Decision Tree-Example-Contd.

We can view this problem as playing a 20 questions game, where every question we ask:

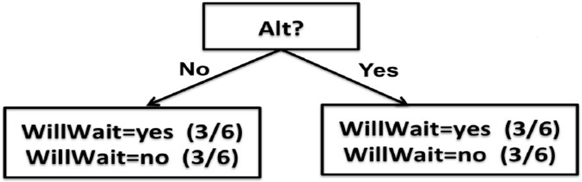
- should help us in narrowing down the value of the target WillWait
- depends on the previous questions that we may have already asked

Here, features represent questions and the answer is the specific feature value for a data instance. What happens when we ask a question?

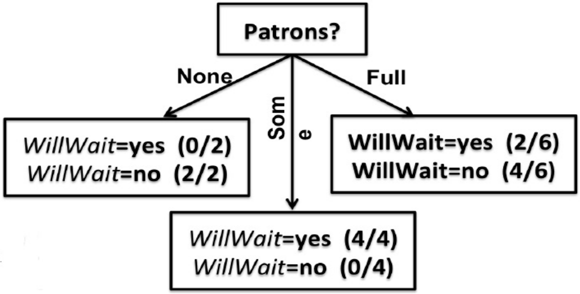
- Depending upon the number of possible answers the data is split into multiple subsets
- If a subset has the same value for the target concept(e.g. WillWait) then we have our answer
- If the subset has different values for the target concept then we need to ask more questions

What happens when we ask different questions?

Whether an alternate restaurant is available nearby or not



How many patrons in the restaurants?



Poor attribute

Note that Type: the kind of restaurant (French, Italian, Thai, or burger) is a poor attribute. It has no role in making a decision as each of which has the same number of positive and negative values.

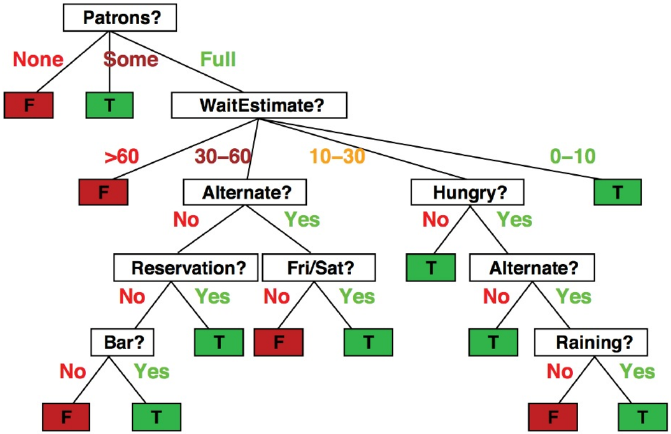
Type	Response(Willwait)
French	1 Yes, 1 No
Italian	1 Yes, 1 No
Thai	2 Yes, 2 No
Burger	2 Yes, 2 No

Decision Tree-Example Contd.

Observation

- In the case of asking the question whether an alternate restaurant is available nearby or not, we can see that the question is not informative in regards to answering our question because no matter what the answer is we have a fifty-fifty chance of being correct. This is as good as deciding randomly (flipping a fair coin).
- If we ask the question about the number of patrons in the restaurant then for two out of the three possible answers we can predict the target concept with 100% confidence.
  If we keep asking questions till we reach nodes in which we can make the predictions with acceptable chances of error or run out of questions we will get a decision tree built using the training data.

Decision Tree- Example, contd..



How does a decision tree works?

- A decision tree maps the input to the final decision value by performing a sequence of tests on the different feature values.
- For a given data instance, each internal node of the tree tests a single feature value to select one of its child nodes.
- This process continues till we reach a leaf node which assigns the final decision value to the instance.

It should be noted here that by changing the order in which the features are tested we can end up with an entirely different tree

Decision Tree Algorithm

```
Data: Training data with $m$ features and $n$ instances
Result: Induced Decision Tree
Begin with an empty tree;
while stopping criterion is not satisfied do
    ■ select best feature;
    ■ split the training data on the best feature;
    ■ repeat last two steps for the child nodes resulting from the split;
end
```

Decision Tree- Example (R code)

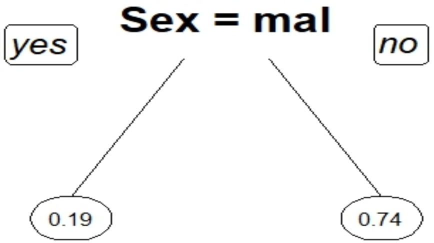
```
> data=read.csv("C:\\Users\\aryalg\\waitdata.csv")
> data
> attach(data)
> library(rpart)
> library(rpart.plot)
> tree_model <- rpart(WillWait ~ Pat + Est + Hun+Fri + Bar
 + Price + Rain + Res + Type, data = data, method = "class",
 control = rpart.control(minsplit = 1, maxdepth = 10))

# Plot the decision tree
> rpart.plot(tree_model, type = 4, extra=101)
```

Decision Tree-Example

Consider the Titanic dataset with various variables

```
library(rpart)
library(rpart.plot)
data=read.csv("C:\\Users\\aryalg\\train.csv")
attach(data)
options(rper.plot.width=1, repr.plot.height=5)
gender_tree=rpart(Survived~Sex, data=data)
prp(gender_tree, space=2, split.cex=2, nn.border=1)
```

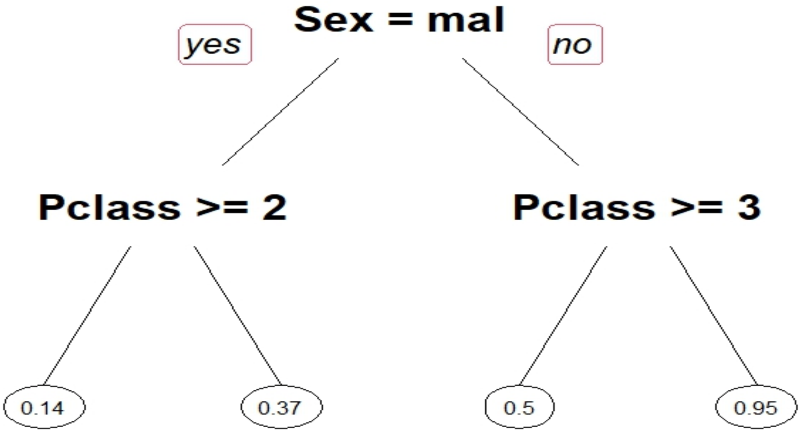
Decision tree: predicting survival based on gender



Using P-class

```
> library(rpart)
> library(rpart.plot)
> data=read.csv("C:\\Users\\aryalg\\train.csv")
> attach(data)
> class_tree=rpart(Survived~Sex+Pclass, data=data)
> prp(class_tree, space=2, split.cex=2, nn.border=2)
```

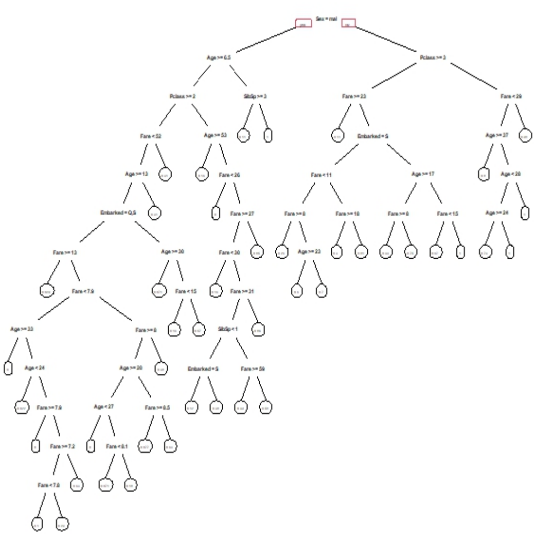
Decision tree: predicting survival based on passenger class



Complex Tree

```
> library(rpart)
> library(rpart.plot)
> data=read.csv("C:\\Users\\aryalg\\train.csv")
> attach(data)
> complex_tree=rpart(Survived~Sex+Pclass+
              Age+SibSp+Fare+Embarked,
              cp=0.001, data=data) #cp is complexity parameter
> prp(complex_tree, space=2, split.cex=2, nn.border=2)
```

Predicting survival based on passenger class and more



Pruning a decision tree

Pruning is a technique used in decision tree algorithms, to simplify the model by reducing its size and complexity. The main goals of pruning are to enhance the model's generalization ability, prevent overfitting, and improve interpretability. Pruning removes parts of the tree that provide little to no additional power in predicting target variables, ultimately resulting in a more robust and manageable model.
Decision trees can grow very deep, leading to complex models that perfectly fit the training data but perform poorly on new, unseen data due to overfitting. Pruning helps to reduce this complexity by removing sections of the tree that provide little power in predicting target variables.

Prune the tree using R

```
> library(rpart)
> library(rpart.plot)
> library(caret)
> data(iris)
> trainindex <- createDataPartition(iris$Species, p = 0.8, list = FALSE)
> traindata <- iris[trainindex, ]
> testdata <- iris[-trainindex, ]
> model <- rpart(Species ~ .,
                  data = traindata,
                  method = "class",
                  control = rpart.control(minsplit = 10, cp = 0.01))

> rpart.plot(model, box.palette = "auto", nn = TRUE)

> predictions <- predict(model, testdata, type = "class")
> confusionMatrix(predictions, testdata$Species)
> printcp(model)
> optimalcp <-model$cptable[which.min(model$cptable[,"xerror"]), "CP"]
> optimalcp
> prunedtree <- prune(model, cp = optimalcp)
> rpart.plot(prunedtree)
```

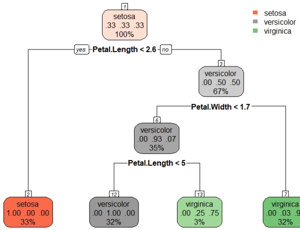


Figure: Before pruning

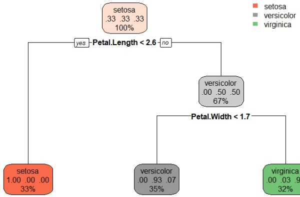


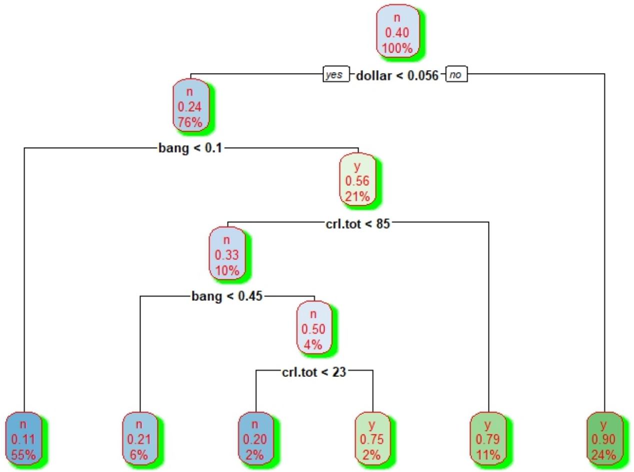
Figure: After pruning

Decision Tree-Example

The data consist of 4601 email items, of which 1813 items were identified as spam and is available in DAAG package

```
> library(DAAG)
> library(party)
> library(rpart)
> library(rpart.plot)
> library(mlbench)
> library(caret)
> library(pROC)
> library(tree)
> data("spam7")
> mydata=spam7
> ind <- sample(2, nrow(mydata), replace = T, prob = c(0.5, 0.5)
> ind
> train <- mydata[ind == 1,]
> test <- mydata[ind == 2,]
> tree <- rpart(yesno ~., data = train)
> rpart.plot(tree, shadow.col="green", col="red")
```

Decision tree

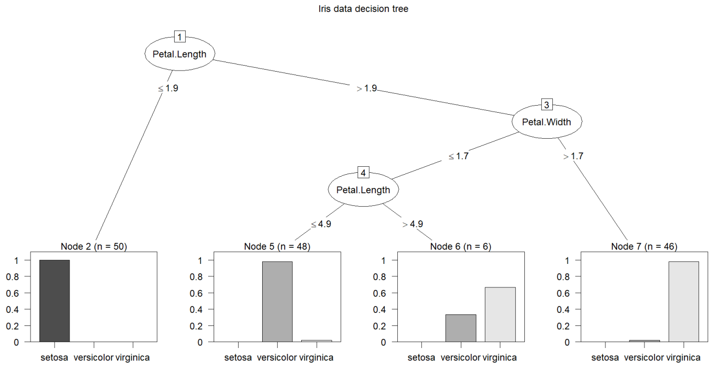


C5.0 Algorithm of Decision Tree

- The ID3 algorithm is a decision tree algorithm that constructs a tree by iteratively selecting the attribute that provides the highest information gain to split the data, aiming for homogeneous subsets.
- The C5 algorithm, created by J. Ross Quinlan, is a development of the ID3 decision tree method. By recursively dividing the data according to information gain it constructs decision trees.
- The C5 technique guarantees that the decision tree is pruned efficiently, avoiding overfitting and enhancing the tree's capacity for generalization by adhering to these stages.
- A method called **winnowing** is used to find and eliminate noisy or unnecessary features that might make a decision tree perform worse. This entails assessing each attribute's information gain and eliminating those that make little contributions to the overall entropy reduction.

C5.0 using R

```
> library(C50)
> data(iris)
> irTree <- C5.0(iris[,-5], iris[,5])
> summary(irTree)
> plot(irTree, main = "Iris data decision tree")
```



Iris data decision tree
1
Petal.Length
≤ 1.9
> 1.9
3
Petal.Width
≤ 1.7
> 1.7
4
Petal.Length
≤ 4.9
> 4.9
Node 2 (n = 50)
Node 5 (n = 48)
Node 6 (n = 6)
Node 7 (n = 46)
1
0.8
0.6
0.4
0.2
0
setosa
versicolor
virginica