

# MEME19903/MECG11103

## Predictive Modelling

### Topic 2b: Supervised Learning: “Tree” Models

Dr Liew How Hui

May 2022

# Outline

## 1 Methods of Classification

- Terminologies and Impurity Measures
- ID3 and Its Generalisations
- CART Decision Trees
- Other Decision Tree Algorithms

## 2 Results Interpretation

## 3 Models Comparison

## 4 Case Study

## 5 Lab Practice on Classification Method

# Methods of Classification

- Regression models from statistics
- kNN models
- Naive Bayes models
- Tree-based models

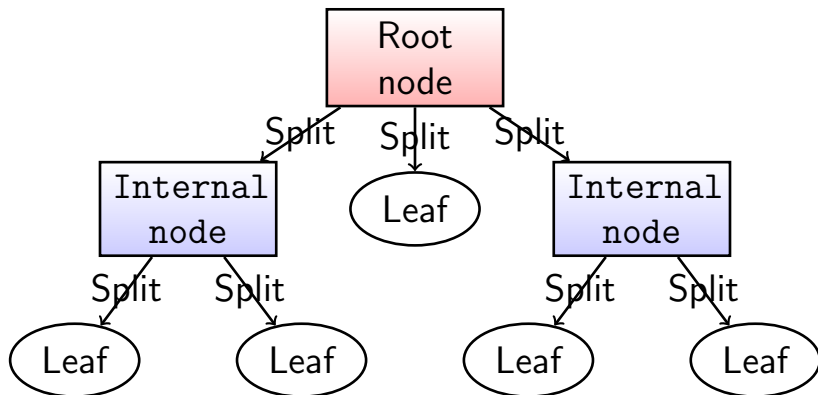
# Theory

A *decision tree* algorithm is a non-parametric method for classification (and regression). It can handle categorical features and extra the predicate logic from features. It is a “flowchart”-like structure in which each *internal node* (the **decision node**) represents a “test” on a “predictor”, each branch represents the outcome of the test, and each **leaf node** represents a class label (decision taken after computing all predictor).

The assumption of decision tree is “the data must be clean and logical”. Based on this assumption, the algorithm uses **information theory** to compress the data table into a “decision tree”.

# Theory / Terminologies (cont)

An illustrative example of a decision tree is given below.



# Theory / Terminologies (cont)

The goal in building a decision tree is to make sure it is “maximally compact” and “only has pure leaves”.

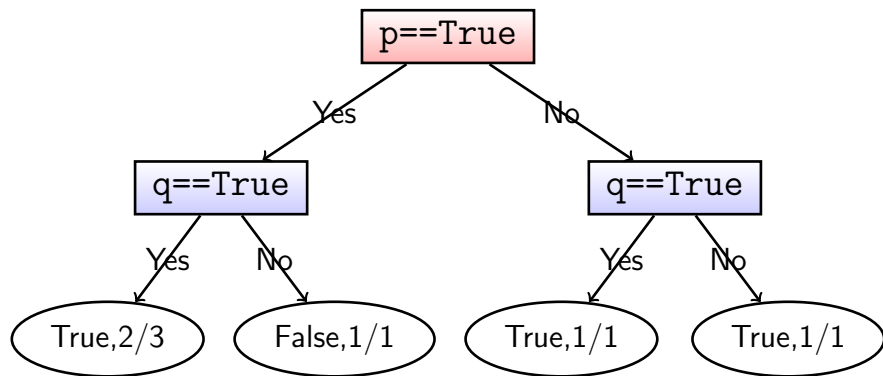
**Example 1:** Consider an Inconsistent Table:

$X_1$	$X_2$	$Y$
T	T	T
T	T	F
T	T	T
T	F	F
F	T	T
F	F	T

Construct a decision tree by encoding each entry as a leaf.

# Theory / Terminologies (cont)

## Example 1: (cont)



This tree is not compact. In general, a noncompact tree with  $n = 10000$  rows data will have 10000 leaves and the storage is the same as kNN.

# Theory / Terminologies (cont)

To build an “compact” decision tree representing the data, we need “smart splitting” of the data  $S$ ! The splitting is performed on the best predictor/attribute which gives close to ‘pure’ output for classification problems (things are more complex for regression problems).

The **splitting criteria** on the predictor depend on whether the predictor is categorical or numeric:

predictors	qualitative (categorical)	quantitative (numerical)
splits	disjoint subsets of the predictor categories	disjoint ranges of the predictors value



# Theory / Terminologies (cont)

For the 'categorical' predictors, there are two types of splitting criteria:

- 1 *Binary split*: Divides values into two subsets. E.g. Department = {A, A, C, A, B, B, C} can be divided into Department1 = {A, A, A} and Department2 = {B, B, C, C}.
- 2 *Multi-way split / L-way split*: Use as many partitions as distinct values. E.g. Department = {A, A, C, A, B, B, C} will be divided into DepartmentA = {A,A,A}, DepartmentB = {B,B}, DepartmentC = {C,C}.

# Theory / Terminologies (cont)

For the 'numeric' predictors, binary splitting is the only option to avoid the problem with multi-way split: For example,  $\text{Age} = \{25, 42, 33, 57, 43\}$  will be splitted into 5-way! This is **not** compact in most cases.

A binary splitting will give a split based on a certain cut-off. For example, the Age can be splitted in two as  $\text{Age} < 42.5 = \{25, 33, 42\}$  and  $\text{Age} > 42.5 = \{43, 57\}$ .

# Theory / Terminologies (cont)

With appropriate splittings, we may be able to get a compact decision tree and it has several nice advantages over the kNN (as a non-parametric model):

- 1 the compact decision tree is a compact representation of the training data and unlike the kNN, the training data does not need to be stored;
- 2 a compact and balanced decision tree is fast in deployment, as the inputs will traverse down the tree to a leaf leading to the predicted label;
- 3 a decision tree can be interpreted when it is drawn;
- 4 decision trees require no metric because the splits are based on feature thresholds and not distances.

# Theory / Terminologies (cont)

One way to choose the appropriate splitting is to use the “impurity” measures, which are from statistics and information theory ([Mac03], [CT06], [HHPDJ03]) to choose the “best” predictor

- Gini Impurity Index (default in ‘rpart’)
- Entropy and Information Gain
- Gain Ratio
- Deviance (used by ‘tree’)
- Variance Reduction (?)

The “impurity” measures generally measure the homogeneity of the target variable within the subsets.

Ref: [https://en.wikipedia.org/wiki/Decision\\_tree\\_learning](https://en.wikipedia.org/wiki/Decision_tree_learning)

# Gini Impurity (Index)

Gini impurity is used by the *CART* (*classification and regression tree*) algorithm for classification trees. It is defined as a measure of total variance across the  $K$  classes:

$$G(S) = \sum_{j=1}^K p_j(1 - p_j) = 1 - \sum_{j=1}^K p_j^2, \quad p_j = \frac{|S_j|}{|S|} \quad (1)$$

where  $p_j$  is the proportion of observations in  $S$  that belong to class  $j$  and  $S_j = \{(\mathbf{x}, y) \in S : y = j\} \subseteq S$  (all inputs with labels  $j$ ) such that  $S_i \cap S_k = \emptyset$  when  $i \neq k$  and  $S = S_1 \cup \dots \cup S_K$ .

## Gini Impurity (cont)

Gini impurity will be maximised when observations are heterogeneous, i.e. **impure**. For example, when all classes are **uniformly distributed**, i.e.

$p_1 = \dots = p_K = \frac{1}{K}$ , then

$$G(S) = 1 - \sum_{k=1}^K \frac{1}{K^2} = 1 - \frac{1}{K}.$$

Gini impurity will be minimised when observations are homogenous, i.e. **pure**. For example, when one class is having all the observations or is empty, then

$$G(S) = 1 - 1 = 0.$$

# Gini Impurity (cont)

The *Gini impurity of a splitting based on the predictor/attribute A* is defined as

$$G^T(\underbrace{\{S_1, \dots, S_L\}}_{S^A}) = \sum_{i=1}^L \frac{|S_i|}{|S|} G^T(S_i). \quad (2)$$

In the splitting of the data, we want a branch with more “pure” nodes, so the predictor with the **lowest** Gini impurity is selected as the splitting predictor and the tree algorithm is repeated for the subsequent nodes.

# Gini Impurity (cont)

**Example 1:** (Tennis / Golf Example from [Mit97, §3.4.2])

Day	Outlook	Temperature	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Compute the **Gini impurity** of each feature (excluding Day) with respect to the output “Play”.



# Gini Impurity (cont)

## Example 1: (cont)

Let  $S_{Weak}$  and  $S_{Strong}$  be the split of Wind is “Weak” and “Strong” respectively,  $S^{Wind} = \{S_{Weak}, S_{Strong}\}$ ,

$S_{Weak, Yes} = S_{Weak} \cap \{Play = Yes\}$ , ..., and

$S_{Strong, No} = S_{Strong} \cap \{Play = No\}$ , etc.

$$\mathbb{P}(S_{Weak}) = \frac{8}{14}; \quad \mathbb{P}(S_{Strong}) = \frac{6}{14}$$

$$\mathbb{P}(S_{Weak, Yes}) = \frac{6}{8}; \quad \mathbb{P}(S_{Weak, No}) = \frac{2}{8} \Rightarrow G(S_{Weak}) = 1 - \left(\frac{6}{8}\right)^2 - \left(\frac{2}{8}\right)^2 = 0.375$$

$$\mathbb{P}(S_{Strong, Yes}) = \frac{3}{6}; \quad \mathbb{P}(S_{Strong, No}) = \frac{3}{6} \Rightarrow G(S_{Strong}) = 1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2 = 0.5$$

$$G(S^{Wind}) = \frac{8}{14}(0.375) + \frac{6}{14}(0.5) = 0.4286$$

# Gini Impurity (cont)

## Example 1: (cont)

The calculations for the Gini impurity of Humidity:

- $\mathbb{P}(S_{High}) = \frac{7}{14}; \quad \mathbb{P}(S_{Normal}) = \frac{7}{14}$
- $\mathbb{P}(S_{High,Yes}) = \frac{3}{7}; \quad \mathbb{P}(S_{High,No}) = \frac{4}{7} \Rightarrow$   
 $G(S_{High}) = 1 - \left(\frac{3}{7}\right)^2 - \left(\frac{4}{7}\right)^2 = 0.4898$
- $\mathbb{P}(S_{Normal,Yes}) = \frac{6}{7}; \quad \mathbb{P}(S_{Normal,No}) = \frac{1}{7} \Rightarrow$   
 $G(S_{Normal}) = 1 - \left(\frac{6}{7}\right)^2 - \left(\frac{1}{7}\right)^2 = 0.2449$
- $G(S^{Humidity}) = \frac{7}{14}(0.4898) + \frac{7}{14}(0.2449) = 0.3674$

# Gini Impurity (cont)

## Example 1: (cont)

Try and see if you can find

$$G(S^{Temperature}) = 0.4405$$

$$G(S^{Outlook}) = 0.3429$$

# Gini Impurity (cont)

We can find Gini impurity index in both recursive partitioning tree algorithms.

```
library(tree)
tree(formula, data, weights, subset,
      na.action=na.pass, control=tree.control(nobs,...),
      method = "recursive.partition",
      split = c("deviance", "gini"),
      model = FALSE, x = FALSE, y = TRUE,
      wts = TRUE, ...)

library(rpart)
rpart(formula, data, weights, subset,
       na.action = na.rpart, method,
       model = FALSE, x = FALSE, y = TRUE, parms,
       control, cost, ...)
```

The parms's split can be 'gini' (default) or 'information'.

# Entropy and Information Gain

Information gain is used by the ID3, C4.5 and C5.0 tree-generation algorithms. Information gain is based on the concept of (Shannon) Information Entropy:

$$H(S) = - \sum_{i=1}^K p_i \log_2 p_i \quad (3)$$

where  $p_i$  is the proportion of observations in  $S$  that belong to class  $i \in \{1, \dots, K\}$  and  $S$  a collection with  $K$  number of classes.

Note that in physics, natural log is used (e.g. Boltzmann's entropy equation  $S = k \ln W$ ) instead of base 2 log.

# Entropy and Information Gain (cont)

The *entropy* over a splitting with attribute  $A$  with  $L$  classes is

$$H^T(\underbrace{\{S_1, \dots, S_L\}}_{S^A}) = \sum_{i \in \text{Values}(A)} \frac{|S_i|}{|S|} H(S_i). \quad (4)$$

The *information gain* measures how well *a given predictor separates the training data  $S$  according to their response variable classification*:

$$IG(S^A) = H(S) - H^T(S^A) \quad (5)$$

where  $S_i$  is the subset of  $S$  for which predictor  $A$  is belonging to class  $i$ .

Features that perfectly partition should give **maximal information gain** or **minimal entropy**.

# Entropy and Information Gain (cont)

**Example 2:** Consider the tennis dataset from **Example 1**. Compute the **information gain** of all features (note: Day is not a feature).  
The **overall entropy (of the target output) before splitting**:

$$H(S) = -\left(\frac{9}{14} \log_2 \frac{9}{14} + \frac{5}{14} \log_2 \frac{5}{14}\right) = \underline{0.9403}$$

# Entropy and Information Gain (cont)

## Example 2: (cont)

The entropy and information gain **after splitting on Wind** are

- $\mathbb{P}(S_{Weak,Yes}) = \frac{6}{8}; \quad \mathbb{P}(S_{Weak,No}) = \frac{2}{8} \Rightarrow$   
 $H(S_{Weak}) = -\left(\frac{6}{8} \log_2 \frac{6}{8} + \frac{2}{8} \log_2 \frac{2}{8}\right) = 0.8113$
- $\mathbb{P}(S_{Strong,Yes}) = \frac{3}{6}; \quad \mathbb{P}(S_{Strong,No}) = \frac{3}{6} \Rightarrow$   
 $H(S_{Strong}) = -\left(\frac{3}{6} \log_2 \frac{3}{6} + \frac{3}{6} \log_2 \frac{3}{6}\right) = 1$
- $H(S^{Wind}) = \frac{8}{14}H(S_{Weak}) + \frac{6}{14}H(S_{Strong}) = 0.8922$   
 $IG(S^{Wind}) = H(S) - H(S^{Wind}) = 0.9403 - 0.8922 = 0.0481$



# Entropy and Information Gain (cont)

## Example 2: (cont)

The information gain **after splitting on** Humidity:

- $\mathbb{P}(S_{High}, Yes) = \frac{3}{7}, \quad \mathbb{P}(S_{High}, No) = \frac{4}{7}$   
 $\Rightarrow H(S_{High}) = 0.9852$
- $\mathbb{P}(S_{Normal}, Yes) = \frac{6}{7}, \quad \mathbb{P}(S_{Normal}, No) = \frac{1}{7}$   
 $\Rightarrow H(S_{Normal}) = 0.5917$
- $H(S^{Humidity}) = \frac{7}{14}H(S_{High}) + \frac{7}{14}H(S_{Normal}) = 0.7885$   
 $IG(S^{Humidity}) = 0.9403 - 0.7885 \approx 0.1518$

# Entropy and Information Gain (cont)

## Example 2: (cont)

Try and see if you can find

$$H(S^{Temperature}) = 0.9111$$

$$IG(S^{Temperature}) = 0.0292$$

$$H(S^{Outlook}) = 0.6936$$

$$IG(S^{Outlook}) = 0.2467$$

# Entropy and Information Gain (cont)

A simple R implementation is listed below.

---

```
entropy = function(S) {  
  counts = table(S)  
  p = counts/sum(counts)  
  return(-sum(ifelse(p==0,0,p*log2(p))))  
}  
  
# A = split_attribute_name, R = target / response name  
info_gain = function(d.f,A,R="target") {  
  HS = entropy(d.f[R])  
  counts = table(unlist(d.f[A]))  
  p = counts/sum(counts)  
  HSi = unlist(lapply(split(d.f, d.f[A]), function(s) {  
    entropy(s[R]) })))  
  return(HS - sum(p*HSi))  
}  
  
d.f = read.csv('playtennis.csv')  
print(entropy(d.f$playtennis))  
print(info_gain(d.f, 'wind', 'playtennis'))
```

---

# Gain Ratio

There is a natural bias in information gain measure that **favours** predictors with large number of values. For example, use Day as a predictor to calculate the information gain of “tennis” data table.

- For  $i = 1, 2, 6, 8, 14$ ,  $\mathbb{P}(S_{D_i, \text{Yes}}) = 0$ ,  $\mathbb{P}(S_{D_i, \text{No}}) = 1$   
 $\Rightarrow H(S_{D_i}) = -(0 \times \log_2 0 + 1 \times \log_2 1) = 0$
- For  $i = 3, 4, 5, 7, 9, 10, 11, 12, 13$ ,  $\mathbb{P}(S_{D_i, \text{Yes}}) = 1$ ,  
 $\mathbb{P}(S_{D_i, \text{No}}) = 0$   
 $\Rightarrow H(S_{D_i}) = -(1 \times \log_2 1 + 0 \times \log_2 0) = 0$
- $IG(S^{\text{Day}}) = 0.9403 - \frac{1}{14} \times 0 - \dots - \frac{1}{14} \times 0 = 0.9403$ .

## Gain Ratio (cont)

If “Day” is an attribute, it will win out as the root but it is useless in prediction just like a person’s id number is useless in identifying the performance of a person.

Gain ratio is a modification of information gain that **reduces its bias** on highly branching predictors.

The *intrinsic information* of  $A$ , is the entropy of the “splitting factor” with  $L$  classes:

$$H(A) = - \sum_{i=1}^L \frac{|A_i|}{|A|} \log_2 \frac{|A_i|}{|A|}. \quad (6)$$

## Gain Ratio (cont)

The **gain ratio**,  $R(S^A)$  is defined as

$$R(S^A) = \frac{IG(S^A)}{H(A)} \quad (7)$$

where  $IG(S^A)$  is the information gain (5).

The predictor with the maximum gain ratio is selected as the splitting predictor and the algorithm is similar to the information gain.

# Gain Ratio (cont)

**Example 3:** Compute the **gain ratio** of the output “Play” against other features (excluding Day) for the data in **Example 1**.

**Solution:** The gain ratio of Wind:

- $H(S) = 0.9403$
- $IG(S^{Wind}) = 0.0481$
- $H(Wind) = -(\frac{8}{14} \log_2 \frac{8}{14} + \frac{6}{14} \log_2 \frac{6}{14}) = 0.9852$
- $R(S^{Wind}) = \frac{IG(S^{Wind})}{H(Wind)} = 0.0488$

# Gain Ratio (cont)

The calculation of gain ratio of Humidity:

- $IG(S^{Humidity}) = 0.9403 - \frac{7}{14} \times 0.9852 - \frac{7}{14} \times 0.5917 = 0.1519$
- $H(Humidity) = -(\frac{7}{14} \log_2 \frac{7}{14} + \frac{7}{14} \log_2 \frac{7}{14}) = 1$
- $R(S^{Humidity}) = 0.1519$

Try to use the steps above to check that

$$R(S^{Temperature}) = 0.0188$$

$$R(S^{Outlook}) = 0.1564.$$



# Gain Ratio (cont)

Apart from being used in C4.5, the gain ratio is still used in C5.0 but with adjustments (with penalty and boosting?).

```
library(C50)
C5.0(x, y, trials = 1, rules = FALSE, weights = NULL,
     control = C5.0Control(), costs = NULL, ...)

C5.0(formula, data, weights, subset, na.action = na.pass, ...)

C5.0Control(subset = TRUE, bands = 0, winnow = FALSE,
             noGlobalPruning = FALSE, CF = 0.25, minCases = 2,
             fuzzyThreshold = FALSE, sample = 0,
             seed = sample.int(4096, size = 1) - 1L,
             earlyStopping = TRUE, label = "outcome")
```

The `subset=TRUE` is for binary split and `subset=FALSE` is for multi-way split. The `minCases` is the minimum number of rows each leaf needs to have.

# Deviance

We can see CART as a way to allocate already  $n$  labeled individuals into arbitrary classes (in a classification context). Trees can be viewed as providing a probability model for individuals class membership. So, at each node  $i$ , we have a probability distribution  $p_{ik}$  over the classes. What is important here is that the leaves of the tree give us a random sample  $n_{ik}$  from a multinomial distribution specified by  $p_{ik}$ . We can thus define the **deviance** of a tree,  $D$ , as the sum over all leaves of

$$D_i = -2 \sum_k n_{ik} \log(p_{ik}),$$

following Venables and Ripley's notations (MASS, Springer 2002, 4th ed., p. 255 ff.).

## Deviance (cont)

Basically, the idea is to minimise, by pruning the tree,  $D + \alpha \#(T)$  where  $\#(T)$  is the number of nodes in the tree  $T$ . Here we recognise the cost-complexity trade-off. Here,  $D$  is equivalent to the concept of node impurity (i.e., the heterogeneity of the distribution at a given node) which are based on a measure of entropy or information gain, or the Gini index.

With a regression tree, the idea is quite similar, and we can conceptualise the deviance as sum of squares defined for individuals  $j$  by

$$D_i = \sum_j (y_j - \mu_i)^2,$$

summed over all leaves.

## Deviance (cont)

Here, the probability model that is considered within each leaf is a gaussian  $N(\mu_i, \sigma^2)$ .

Quoting Venables and Ripley (p. 256), “ $D$  is the usual scaled deviance for a gaussian GLM. However, the distribution at internal nodes of the tree is then a mixture of normal distributions, and so  $D_i$  is only appropriate at the leaves. The tree-construction process has to be seen as a hierarchical refinement of probability models, very similar to forward variable selection in regression.”

# Variance Reduction

According to Wikipedia, the variance reduction [BFSO84] is often employed in cases where the target variable is continuous (regression tree): The variance reduction of a node  $N$  is defined as the total reduction of the variance of the target variable  $x$  due to the split at this node:

$$I_V(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2} (x_i - x_j)^2 - \left( \frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2} (x_i - x_j)^2 + \frac{1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2} (x_i - x_j)^2 \right)$$

where  $S$ ,  $S_t$ , and  $S_f$  are the set of presplit sample indices, set of sample indices for which the split test is true, and set of sample indices for which the split test is false, respectively.

# Misclassification Error

The misclassification error is rarely used for tree construction but it is used in tree models comparison.

Misclassification errors can be computed as the total number of errors in all leaves or at each node.

The misclassification error handling function is available in `tree` and `rpart` libraries from

```
tree::misclass.tree(tree, detail = FALSE)

rpart::printcp(x, digits = getOption("digits") - 2)

rpart::summary(object, cp = 0, digits = getOption("digits"),
  file, ...)
```

# ID3 Algorithm and Its Extensions

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The training observations are sorted to the corresponding internal nodes. If the nodes only contain observations from a single class (pure), then the node will become a leaf node. Otherwise, it will be further expanded, by selecting the predictor with highest information gain relative to the new subsets of observations.

# ID3 Extensions (cont)

The ID3 algorithm is restricted in dealing with categorical data only. However, [Mit97, §3.7.2] mentioned it is possible to introduce a threshold to turn the numeric feature to a binary categorical data. [Mit97, §3.7.3] also pointed out that ID3 can be extended to use gain ratio. Note that decision trees are myopic in general and ID3-trees are prone to overfitting as the tree depth increases. Therefore, we need more sophisticated tree algorithms.



## ID3 Extensions (cont)

Since ID3 cannot handle numeric data, C4.5 is a successor to ID3 which removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. The C4.5 algorithm ([https://en.wikipedia.org/wiki/C4.5\\_algorithm](https://en.wikipedia.org/wiki/C4.5_algorithm)) constructs a decision tree based on the gain ratio. C4.5 is superseded in 1997 by a commercial system See5/C5.0 according to <https://rulequest.com/see5-comparison.html> and CART.

## ID3 Extensions (cont)

C5.0 is Quinlan's latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate. The C50 library implements C5.0 decision trees and rule-based models for pattern recognition extend the work of [Qui86]. The company <https://www.rulequest.com> has the pattern of C5.0 and provides Cubist (Rule- And Instance-Based Regression Modelling) and outlier tree (Explainable outlier detection through decision tree conditioning based on GritBot) for statistical inference with tree models.

# ID3 Extensions (cont)

**Example 4:** Use C5.0 to construct the ID3 decision tree for the tennis data.

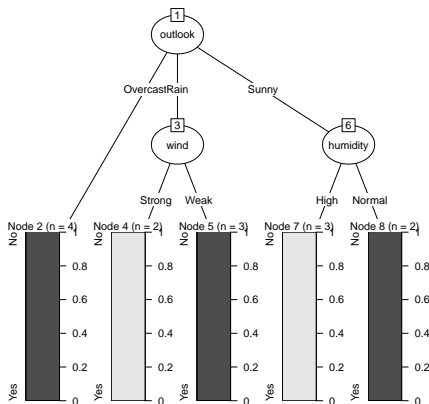
**Solution:**

```
tennis.data = read.csv("playtennis.csv", row.names=1)
tennis.data$outlook = factor(tennis.data$outlook)
tennis.data$temperature = factor(tennis.data$temperature)
tennis.data$humidity = factor(tennis.data$humidity)
tennis.data$wind = factor(tennis.data$wind)
tennis.data$playtennis = factor(tennis.data$playtennis)

# ID3 tree with C5.0
library(C50)
tree.model = C5.0(playtennis ~., data=tennis.data)
plot(tree.model)
# Textual tree representation may be easier to understand
print(summary(tree.model))
```

# ID3 Extensions (cont)

## Example 4: (cont)



## Summary of Model:

```
outlook = Overcast: Yes (4)
outlook = Rain:
...wind = Strong: No (2)
:   wind = Weak: Yes (3)
outlook = Sunny:
...humidity = High: No (3)
      humidity = Normal: Yes (2)
```

# ID3 Extensions (cont)

A comparison table of ID3, C4.5 to the CART is given below.

Aspect	Original ID3	C4.5	CART
Features	Discrete features only; cannot handle numeric features	continuous and discrete features	continuous and discrete features
missing values	Handles missing attributes by ignoring them in information gain computation		?
splitting criteria	<b>Multi-category</b> splitting which maximises information gain / min. entropy	maximises gain ratio	uses Gini impurity to perform <b>binary split</b>
shape	short and wide tree	short and wide tree	taller tree
pruning	no pruning, prone to overfitting	performs post-pruning (bottom-up pruning)	performs cost-complexity pruning

# CART Decision Tree

Classification And Regression Tree (CART) is a binary tree generation algorithm using Gini index as its splitting criteria which can handle both categorical and numeric predictors. CART handles missing values by surrogating tests to approximate outcomes.

A detail account of the algorithm is given in [BFSO84]. An introduction to recursive partitioning using the rpart routines is given by Atkinson and Therneau's rpart documentation.

# CART (cont)

For more general review (including bagging), refer to

- Moissen, G.G. (2008). Classification and Regression Trees. Ecological Informatics, pp. 582-588.
- Sutton, C.D. (2005). Classification and Regression Trees, Bagging, and Boosting, in Handbook of Statistics, Vol. 24, pp. 303-329, Elsevier.

In R, CART is supported by the `tree` and `rpart` libraries.

# CART (cont)

The `tree` library provides a limited functions of constructing, plotting and pruning trees. The `rpart`, which stands for “recursive partitioning” for classification, regression and survival trees, implements most of the functionality of [BFSO84]. The `rpart.plot` or `partykit::plot.party` can be used to draw nice diagrams for `rpart`.



# CART (cont)

CART is very similar to C4.5, but it differs in that it supports numerical target variables, i.e. it supports regression. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

The scikit-learn uses an optimised version of the CART algorithm (`sklearn.tree.DecisionTreeRegressor`); however, the implementation does not support categorical variables for now (if the input is categorical, it will convert to number?).

# CART (cont)

To build a regression tree,

- 1 use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
- 2 Beginning at the top of the tree, split it into 2 branches, creating a partition of 2 spaces and choose the split of the features that minimises the current SSE.
- 3 Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of  $\alpha$  that balances the depth of the tree and its goodness of fit to the training data.

# CART (cont)

**Example 5:** Construct a CART decision tree for the tennis data in **Example 1**.

**Solution:** We can either use tree library:

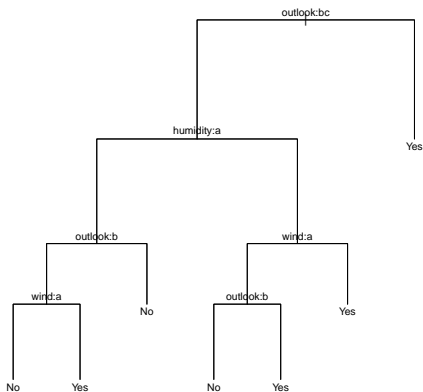
```
1 library(tree)
2 tennis.data = read.csv("playtennis.csv",row.names=1,
3   stringsAsFactor=T)
4 t.m = tree(playtennis~., data=tennis.data, control=
5   tree.control(nobs=nrow(tennis.data),minsize=1))
6 print(t.m)
7 plot(t.m); text(t.m, cex=0.8)
```

or the rpart library:

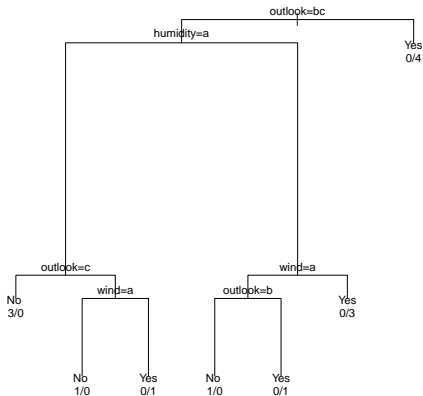
```
1 library(rpart)
2 tennis.data = read.csv("playtennis.csv",row.names=1)
3 #rpart can handle strings (no need to convert to factor)
4 dt = rpart(playtennis ~ ., data=tennis.data,
5   control=rpart.control(minsplit=2, minbucket=1, cp=0.001))
6 print(dt)
7 plot(dt, margin=0.05); text(dt, use.n=TRUE, cex=0.8)
```

# CART (cont)

## Example 5: (cont)



(a) tree



(b) rpart

# Other Decision Tree Algos

We have introduced the popular decision tree construction algorithms C5.0 and CART (Breiman et al., 1984) in the earlier slides, the less popular classification (and regression\*) tree algorithms are listed below:

- CTREE (Hothorn et al., 2006): <https://cran.r-project.org/web/packages/partykit/index.html>
- MOB (Model-Based Recursive Partitioning, Zeileis et al., 2008)\* <https://cran.r-project.org/web/packages/partykit/index.html>
- Random forest [Bre01]: <https://cran.r-project.org/web/packages/randomForest/index.html>
- Generalised Boosted Trees:  
<https://cran.r-project.org/web/packages/gbm/index.html>

# Other Decision Tree Algos (cont)

- BART (Chipman et al., 2010):

<https://cran.r-project.org/web/packages/BART/index.html>

- CHAID (Chi-squared Automatic Interaction Detector, an extension of THAID [MM72]) [Kas80]: Available in Stata scc and IBM SPSS
- GUIDE (Loh, 2002, 2009; Loh and Zheng, 2013; Loh et al., 2015):

<https://pages.stat.wisc.edu/~loh/guide.html>.

Other algos from the same author:

- ▶ CRUISE (Kim and Loh, 2001, 2003):  
<https://pages.stat.wisc.edu/~loh/cruise.html>
- ▶ QUEST (Loh and Shih, 1997):  
<https://github.com/hetianle/QuestDecisionTree>

# Other Decision Tree Algos (cont)

Single decision tree algorithms suffer from high variance, meaning if we split the training data into 2 parts at random, and fit a decision tree to both halves, the results that we get could be quite different!

In contrast, multi-decision-tree algorithms below can yield tree-based models which have lower variance:

- Boosting grows trees sequentially: each tree is grown using information from previously grown trees. Boosting does not involve bootstrap sampling; instead, each tree is fitted on a modified version of the original dataset.
- Bagging, or bootstrap aggregation, is a technique used to reduce the variance of your predictions by combining the result of multiple classifiers modeled on different sub-samples of the same dataset.

# Other Decision Tree Algos (cont)

- Random Forests provides an improvement over bagged trees by a small tweak that decorrelates the trees. As in bagging, you build a number of decision trees on bootstrapped training samples. But when building these decision trees, each time a split in a tree is considered, a random sample of  $m$  predictors is chosen as split candidates from the full set of  $p$  predictors. The split is allowed to use only one of those  $m$  predictors. This is the main difference between random forests and bagging; because as in bagging, the choice of predictor  $m = p$ .



# Outline

- 1 Methods of Classification
  - Terminologies and Impurity Measures
  - ID3 and Its Generalisations
  - CART Decision Trees
  - Other Decision Tree Algorithms
- 2 Results Interpretation
- 3 Models Comparison
- 4 Case Study
- 5 Lab Practice on Classification Method

# Results Interpretation

Tree-based algorithms produces results which can be interpreted as follows:

- The decision tree has a correspondence nested if statements.
- The decision trees from the recursive partition tree algorithms have the corresponding recursive partition diagram theoretically. However, it is only possible to plot the two-variable case practically.
- The most important variable is usually located at the root of the tree. Due to the sensitivity of trees to data, important variables are usually identified by a collection of trees.

## Case Study 2

To illustrate the relation of decision trees to nested if statements, we express the CART tree for the tennis data in **Example 5** as a nested if statements in Python.

```
def tree_model(outlook, humidity, wind):
    if outlook in ['Rain', 'Sunny']:
        if humidity == 'High':
            if outlook == 'Rain':
                if wind == "Strong": return "No"
                if wind == "Weak": return "Yes"
            if outlook == "Sunny": return "No"
        if humidity == "Normal":
            if wind == "Strong":
                if outlook == "Rain": return "No"
                if outlook == "Sunny": return "Yes"
            if wind == "Weak": return "Yes"
    if outlook == "Overcast": return "Yes"
```

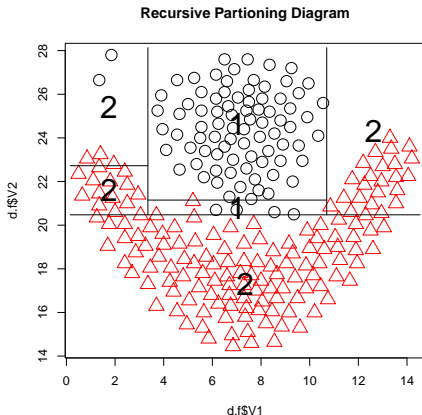
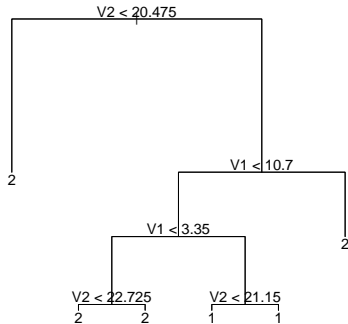
## Case Study 2 (cont)

Expressing the CART tree as a nested if statements in R.

```
tree_model = function(outlook, humidity, wind){  
  if (outlook %in% c('Rain', 'Sunny')) {  
    if (humidity == 'High') {  
      if (outlook == 'Rain') {  
        if (wind == "Strong") return("No")  
        if (wind == "Weak")  return("Yes")  
      }  
      if (outlook == "Sunny") return("No")  
    }  
    if (humidity == "Normal") {  
      if (wind == "Strong") {  
        if (outlook == "Rain")  return("No")  
        if (outlook == "Sunny") return("Yes")  
      }  
      if (wind == "Weak") return("Yes")  
    }  
  }  
  if (outlook == "Overcast") return("Yes")  
}
```

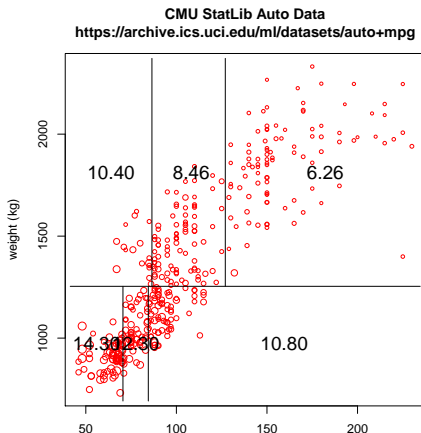
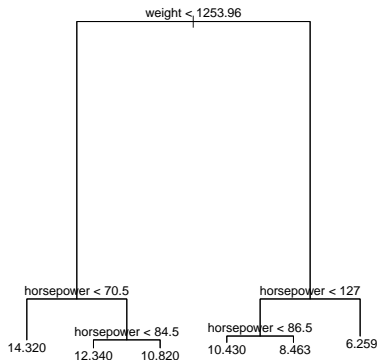
# Results Interpretation (cont)

To illustrate the interpretation of CART tree algorithm as a kind of data recursive partitioning algorithm, we study the CART tree of the 'flame' data.



# Results Interpretation (cont)

The CART tree of the UCI Auto MPG Data is illustrated on the left together with the corresponding recursive partitioning diagram on the right.



# Interpretation: Variable Importance

A variable may appear in the tree many times, either as a primary or a surrogate variable. An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable, plus goodness times adjusted agreement for all splits in which it was a surrogate.

Rpart's function below provides information about important variables for CART tree.

```
printcp(x, digits = getOption("digits") - 2)

summary(object, cp = 0, digits = getOption("digits"),
  file, ...)
```

# Results Interpretation (cont)

Despite the root of the tree would usually be regarded as the important variable that provides the 'most information' in constructing a tree, trees can be rather unstable!

A small change in the training data set may result in relatively large differences in the fitted trees! In this case, the root can be different and it is difficult to tell which variable is more important.

When a collection of trees are constructed, the important variables can be obtained from them.



# Variable Importance (cont)

For CART (Recursive Partitioning) tree: The reduction in the loss function (e.g. mean squared error) attributed to each variable at each split is tabulated and the sum is returned. Also, since there may be candidate variables that are important but are not used in a split, the top competing variables are also tabulated at each split.

Bagged Trees: The same methodology as a single tree is applied to all bootstrapped trees and the total importance is returned.

# Variable Importance (cont)

Random Forest: For each tree, the prediction accuracy on the out-of-bag portion of the data is recorded. Then the same is done after permuting each predictor variable. For regression, the MSE is computed on the out-of-bag data for each tree, and then the same computed after permuting a variable. The difference between the two accuracies/errors are then averaged over all trees, and normalised by the standard error. If the standard error is equal to 0 for a variable, the division is not done.

Boosted tree: It uses the same approach as a single tree, but sums the importances over each boosting iteration.

# Outline

- 1 Methods of Classification
  - Terminologies and Impurity Measures
  - ID3 and Its Generalisations
  - CART Decision Trees
  - Other Decision Tree Algorithms
- 2 Results Interpretation
- 3 Models Comparison
- 4 Case Study
- 5 Lab Practice on Classification Method

# Models Comparison

Compare to a complex decision tree model, a simpler decision tree model is preferred. In order to obtain a better tree model, we need to study the factors associated with tree generation algorithms.

Factors affecting tree models:

- tree pruning
- overfitting & bias
- algorithm complexity

# Models Comparison: Tree-Pruning

The purpose of tree pruning is to obtain a smaller tree. A smaller tree with fewer splits may lead to a lower variance, better interpretation maintaining misclassification errors, and slightly more “generalisation”.

There are two approaches in “pruning” a complex decision tree:

- **Pre-pruning:** This type of algorithms will stop growing the tree earlier, before it perfectly classifies the training set.
- **Post-pruning:** This type of algorithms will allow the tree to perfectly classify training set, and post prune the tree.

# Tree Pre-Pruning

Pre-pruning for C5.0 is given by C5.0Control mentioned in an earlier slide and for CART is listed below:

```
tree.control(nobs, mincut = 5, minsize = 10, mindev = 0.01)

rpart.control(minsplit = 20, minbucket = round(minsplit/3),
  cp = 0.01, maxcompete = 4, maxsurrogate = 5,
  usesurrogate = 2, xval = 10, surrogatestyle = 0,
  maxdepth = 30, ...)
```

where nobs is # observations in the training set;  
minsplit is the minimum # observations that must exist in a node in order for a split to be attempted;  
minsize is the smallest allowed node size; minbucket is the smallest allowed leaf (terminal node) size.

# Tree Post-Pruning

Post-pruning (not available for C5.0) is available for CART:

```
prune.tree(tree, k = NULL, best = NULL, newdata, nwts,  
  method = c("deviance", "misclass"), loss, eps = 1e-3)  
  
prune.misclass(tree, k = NULL, best = NULL, newdata,  
  nwts, loss, eps = 1e-3)  
  
prune.rpart(tree, cp, ...)
```

Note that `prune.rpart` has options similar to `rpart.control`.

# Models Comparison: Overfit & Bias

Single-decision-tree algorithms have the following problems:

- The generate tree is mostly **overfitting**, i.e. it is able to predict data similar to training data but performs badly on unseen data different from training data;
- bias: With different training data, the tree can give different results and it may be biased to one class than the other when the data is imbalanced.

To deal with these problems, random forests, Boosting trees, etc. are developed. Despite being better models, but the 'interpretability' will be lost.



# Models Comparison: Complexities

There are two kinds of complexities associated with the decision tree algorithms:

- The generated tree is **complex**, i.e. too many branches and leaves. If it can be pruned to a simpler tree with less branches and leaves while maintaining the missclassification errors, the **simpler pruned tree model** is better.
- The tree-generating algorithm may be complex.

# Complexity (cont)

## Algorithm Complexity

Let  $n$  be the number of samples and  $p$  be the number of features. According to <https://scikit-learn.org/stable/modules/tree.html>,

the **average** run time cost to construct a balanced binary tree is  $O(p \times n \log(n))$  and query time  $O(\log(n))$ .

Although the tree construction algorithm attempts to generate balanced trees, they will not always be balanced. So in real-world situation, the constructed tree may be imbalance and the query time may be slightly longer than  $O(\log(n))$ .

# Models Comparison (cont)

**Example 6:** (SRM-02-18, Q29) Determine which of the following considerations may make decision trees preferable to other statistical learning methods.

- i. Decision trees are easily interpretable.
- ii. Decision trees can be displayed graphically.
- iii. Decision trees are easier to explain than linear regression methods.

# Models Comparison (cont)

- (A) None
- (B) I and II only
- (C) I and III only
- (D) II and III only
- (E) The correct answer is not given by (A), (B), (C), or (D).

---

**Solution:** All three statements are true. See [JWHT13, Section 8.1]. The statement that trees are easier to explain than linear regression methods (as well as kNN, naive Bayes, etc.) may not be obvious. For those familiar with regression but just learning about trees, the reverse may be the case. However, for those not familiar with regression, relating the dependent variable to the independent variables, especially if the dependent variable has been transformed, can be difficult to explain.

**Answer:** (E)

# Outline

- 1 Methods of Classification
  - Terminologies and Impurity Measures
  - ID3 and Its Generalisations
  - CART Decision Trees
  - Other Decision Tree Algorithms
- 2 Results Interpretation
- 3 Models Comparison
- 4 Case Study
- 5 Lab Practice on Classification Method

# Case Study 1

## **Example 7:** (SRM-02-18, Q9)

A classification tree is being constructed to predict if an insurance policy will lapse. A random sample of 100 policies contains 30 that lapsed. Consider two splits:

**Split 1:** One node has 20 observations with 12 lapses and one node has 80 observations with 18 lapses.

**Split 2:** One node has 10 observations with 8 lapses and one node has 90 observations with 22 lapses.

# Case Study 1 (cont)

## **Example 7:** (cont)

The total Gini impurity after a split is the weighted average of the Gini impurity at each node, with the weights proportional to the number of observations in each node.

The total entropy after a split is the weighted average of the entropy at each node, with the weights proportional to the number of observations in each node.

# Case Study 1 (cont)

## Example 7: (cont)

Determine which of the following statements is/are true?

- I. Split 1 is preferred based on the total Gini impurity.
- II. Split 1 is preferred based on the total entropy.
- III. Split 1 is preferred based on having fewer classification errors.

- (A) I only
- (B) II only
- (C) III only
- (D) I, II, and III
- (E) The correct answer is not given by (A), (B), (C), or (D).



# Case Study 1 (cont)

## Solution:

$$\begin{aligned}\text{Gini}(\text{Split 1}) &= \frac{20}{100} \left[ 1 - \left( \frac{12}{20} \right)^2 - \left( \frac{8}{20} \right)^2 \right] + \frac{80}{100} \left[ 1 - \left( \frac{18}{80} \right)^2 - \left( \frac{62}{80} \right)^2 \right] \\ &= 0.2 \times 0.48 + 0.8 \times 0.34875 = 0.375\end{aligned}$$

$$\begin{aligned}\text{Gini}(\text{Split 2}) &= \frac{10}{100} \left[ 1 - \left( \frac{8}{10} \right)^2 - \left( \frac{2}{10} \right)^2 \right] + \frac{90}{100} \left[ 1 - \left( \frac{22}{90} \right)^2 - \left( \frac{68}{90} \right)^2 \right] \\ &= 0.1 \times 0.32 + 0.9 \times 0.3693827 = 0.3644\end{aligned}$$

⇒ **Lower Gini impurity** is better. ∴ Split 2 is preferred. I is wrong.

$$\begin{aligned}\text{H}(\text{Split 1}) &= -\frac{20}{100} \left[ \frac{12}{20} \log_2 \frac{12}{20} + \frac{8}{20} \log_2 \frac{8}{20} \right] - \frac{80}{100} \left[ \frac{18}{80} \log_2 \frac{18}{80} + \frac{62}{80} \log_2 \frac{62}{80} \right] \\ &= -0.2 \times (-0.9710) - 0.8 \times (-0.7692) = 0.8096\end{aligned}$$

$$\begin{aligned}\text{H}(\text{Split 2}) &= -\frac{10}{100} \left[ \frac{8}{10} \log_2 \frac{8}{10} + \frac{2}{10} \log_2 \frac{2}{10} \right] - \frac{90}{100} \left[ \frac{22}{90} \log_2 \frac{22}{90} + \frac{68}{90} \log_2 \frac{68}{90} \right] \\ &= -0.1 \times (-0.7219) - 0.9 \times (-0.8024) = 0.7944\end{aligned}$$

⇒ **Smaller entropy** (vs larger information gain) is better. ∴ Split 2 is preferred. II is wrong.

Remark: SRM Answer is different because they use natural log.

For Split 1, there are 8+18=26 errors while for Split 2 there are 2+22 = 24 errors. With fewer errors, Split 2 is preferred.

**Answer:** (E)

## Case Study 2

**Example 8:** (Deployment of a decision tree.  
SRM-02-18, Q33)

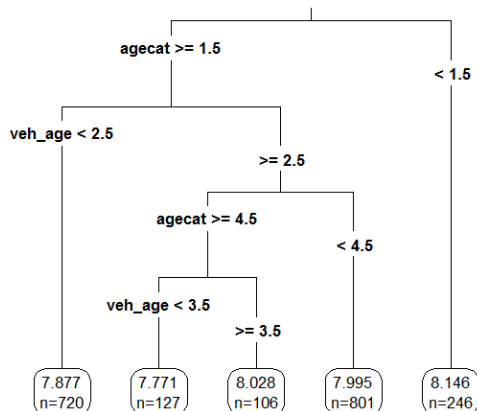
The regression tree shown below was produced from a dataset of auto claim payments. Age Category (agecat: 1, 2, 3, 4, 5, 6) and Vehicle Age (veh\_age: 1, 2, 3, 4) are both predictor variables, and log of claim amount (LCA) is the dependent variable.

Consider three autos I, II, III:

- i. An Auto in Age Category 1 and Vehicle Age 4
- ii. An Auto in Age Category 5 and Vehicle Age 5
- iii. An Auto in Age Category 5 and Vehicle Age 3

# Case Study 2 (cont)

## Example 8: (cont)



## Case Study 2 (cont)




### Example 8: (cont)

Rank the estimated LCA of Autos I, II, and III.

- (A)  $LCA(I) < LCA(II) < LCA(III)$
- (B)  $LCA(I) < LCA(III) < LCA(II)$
- (C)  $LCA(II) < LCA(I) < LCA(III)$
- (D)  $LCA(II) < LCA(III) < LCA(I)$
- (E)  $LCA(III) < LCA(II) < LCA(I)$

**Answer:** (E) Do you know why?




# References I

-  Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen, *Classification and regression trees*, Chapman and Hall/CRC, 1984, ISBN 9780412048418.
-  Leo Breiman, *Random forests*, Mach. Learn. **45** (2001), no. 1, 5–32.
-  Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, 2nd ed., John Wiley & Sons, Inc., 2006.

# References II

-  Darrel Hankerson, Greg A. Harris, and Jr. Peter D. Johnson, *Introduction to information theory and data compression*, 2nd ed., CRC Press LLC, 2003.
-  Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An introduction to statistical learning: with applications in r*, Springer Texts in Statistics, Springer Science+Business Media New York, 2013.
-  G. V. Kass, *An exploratory technique for investigating large quantities of categorical data*, Applied Statistics **29** (1980), no. 2, 119–127.

# References III

-  David J. C. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, 2003.
-  Tom M. Mitchell, *Machine learning*, MIT Press, 1997.
-  R. Messenger and L. Mandell, *A modal search technique for predictive nominal scale multivariate analysis*, Journal of the American Statistical Association **67** (1972), no. 340, 768–772.

# References IV



J. R. Quinlan, *Induction of decision trees*, Machine Learning **1** (1986), 81–106,  
<https://doi.org/10.1007/BF00116251>.



# Outline

- 1 Methods of Classification
  - Terminologies and Impurity Measures
  - ID3 and Its Generalisations
  - CART Decision Trees
  - Other Decision Tree Algorithms
- 2 Results Interpretation
- 3 Models Comparison
- 4 Case Study
- 5 Lab Practice on Classification Method

# Lab Practice on Classification Method 4

s44\_tree.R