Topic 5: Decision Trees

5.1 Methods for Expressing Attribute Test Conditions 106

5.2 "Impurity" Measures for Categorical Output 106

5.2.1 Gini Impurity (Index) 106

5.2.2 Entropy and Information Gain 108

5.2.3 Intrinsic Information and Gain Ratio 111

5.3 Variance Reduction for Numeric Output 117

5.4 Various Implementations of Decision Tree Algorithms 118

5.4.1 ID3 Algorithm 119

5.4.2 C4.5 and C5.0 125

5.4.3 CART 127

5.4.4 Other Implementations: CHAID 131

5.5 Practical Considerations 132

5.5.1 Interpretability 132

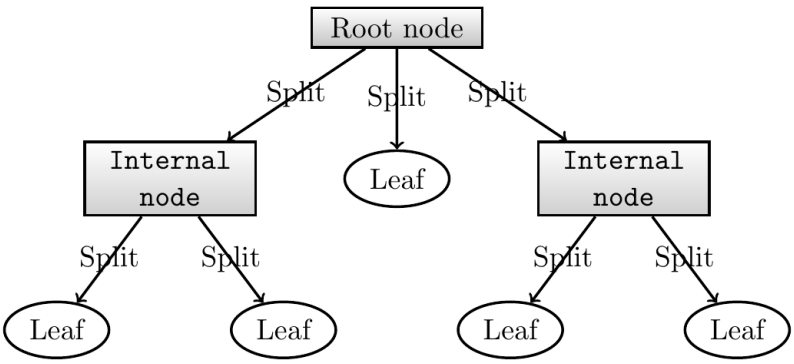
5.5.2 Underfitting and Overfitting 132

5.5.3 Hyperparameter Tuning — Tree Pruning 133

5.5.4 Complexity 135

A *decision tree (learning)* is a non-parametric supervised learning method used for classification and regression.

The learning process involves the **segmenting** of the "predictor space" into a number of "simple regions" [James et al., 2013, Chapter 8]. In the simplest cases, each node (root or internal) is a predicate involving an attribute/a "predictor", each branch represents the outcome of the predicate test, and each **leaf node** represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.



The goal in building a decision tree is to have a **compact** tree with **mostly pure leaves**. Since find the most compact takes an extremely long time, we will be using information theory or statistical rules to find an acceptable tree.

5.1 Methods for Expressing Attribute Test Conditions

Decision tree construction algorithms usually split the tabular data based on attribute test condition for different attribute types.

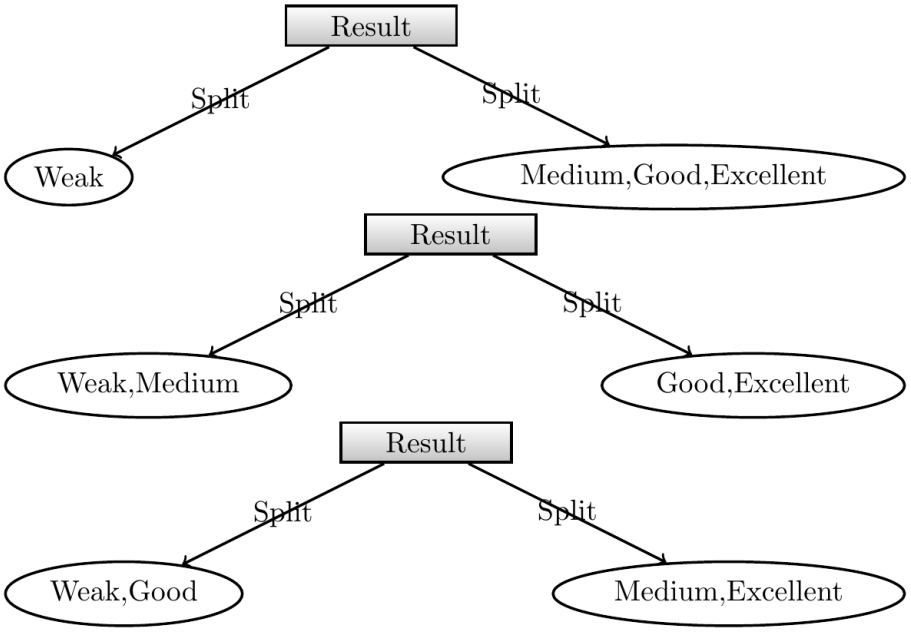
Binary Attributes: The test condition can be expressed with only two outputs.

Nominal Attributes: Since a nominal attribute can have many values, its test condition can be expressed in two ways:

1. ***Binary split***: Divides values into two subsets.
2. ***Multi-way split*** / ***L-way split***: Use as many partitions as distinct values.

In theory, binary splits are more flexible than L-way splits, but L-way splits could give more interpretable trees. The L-way split will be preferred when there are a reasonable number of levels for an attribute. When the number of levels is large, a binary split will be preferred because there are generally fewer splits to consider.

Ordinal Attributes: They can also produce binary or multiway splits **without violating the order property of the attribute values**:



Continuous Attributes: For continuous attributes, the test conditions can be expressed as a comprison test $\boxed{A < v}$ or $\boxed{A \geq v}$ with binary split, or range query with outcomes of the form $\boxed{v_i \leq A < v_{i+1}}$ for $i = 1, ..., k$.

5.2 "Impurity" Measures for Categorical Output

One of the most popular techniques to split data in decision trees is based on **impurity metrics**, which measure the homogeneity of the target variable within the subsets. We introduce them following `https://en.wikipedia.org/wiki/Decision_tree_learning`.

5.2.1 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 output classes:

$$G(S) = \sum_{j=1}^{K} p_j(1 - p_j) = 1 - \sum_{j=1}^{K} p_j^2, \quad p_j = \frac{|S_j|}{|S|} \tag{5.1}$$

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 $S = S_1 \cup \cdots \cup S_K$.

The Gini impurity, $G(S)$, characterises the probability of mis-classification of an object being selected at random from one of K classes according to probabilities $(p_1, p_2, \cdots, p_K)$ and is randomly assigned to a class using the same distribution.

Gini impurity will be maximised when observations are heterogeneous, i.e. **impure**. For example, when all classes are **uniformly distributed**, i.e. $p_1 = \cdots = p_K = \frac{1}{K}$, 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

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

The ***Gini impurity of a tree*** is defined as

$$G^T(\underbrace{\{S_1, \cdots, S_L\}}_{S^A}) = \sum_{i=1}^{L} \frac{|S_i|}{|S|} G(S_i). \tag{5.2}$$

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

Example 5.2.1. Given a table below which consists the results whether to play tennis (output/label) based on the features Outlook, Temperature, Humidity and Wind [Mitchell, 1997, §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". As an illustration, the calculations of Gini impurity for the splitting based on Wind and Humidity are given below.

Illustration: Let S_{Weak} and S_{Strong} be the split of Wind is "Weak" and "String" 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. The steps to find Gini impurity of Wind are:

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

- $\mathbb{P}(S_{Weak,Yes}) = \frac{6}{8}; \quad \mathbb{P}(S_{Weak,No}) = \frac{2}{8} \Rightarrow G(S_{Weak}) = 1 - (\frac{6}{8})^2 - (\frac{2}{8})^2 = 0.375$
- $\mathbb{P}(S_{Strong,Yes}) = \frac{3}{6}; \quad \mathbb{P}(S_{Strong,No}) = \frac{3}{6} \Rightarrow G(S_{Strong}) = 1 - (\frac{3}{6})^2 - (\frac{3}{6})^2 = 0.5$
- $G(S^{Wind}) = \frac{8}{14}(0.375) + \frac{6}{14}(0.5) = 0.4286$

Gini impurity of Humidity:

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

Try to check that $G(S^{Temperature}) = 0.4405$, $G(S^{Outlook}) = 0.3429$.

5.2.2 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 `https://en.wikipedia.org/wiki/Entropy` and information

content from information theory [Cover and Thomas, 2006]:

$$H(S) = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{5.3}$$

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

The entropy (5.3) is related to the KL-Divergence $KL(p||q)$. Although KL-Divergence is not a metric because it is not symmetric, i.e., $KL(p||q) \neq KL(q||p)$, but it can be used to compute "closeness".

Let $q_1, \ldots, q_K$ be the uniform label/distribution. i.e. $q_k = \frac{1}{K}$, $k = 1, ..., K$. The KL-Divergence,

$$0 \leq KL(p||q) = \sum_{k=1}^{K} p_k \log_2 \frac{p_k}{q_k} = \sum_{k=1}^{K} p_k \log_2(Kp_k) = \log_2 K \sum_k p_k + \sum_k p_k \log_2(p_k).$$

Since $\sum_k p_k = 1$, we have

$$KL(p||q) = \log_2 K + \sum_k p_k \log_2(p_k) \Rightarrow \max_p KL(p||q) = \max_p \sum_k p_k \log_2(p_k). \tag{5.4}$$

By substituting (5.3) into (5.4), we obtain the maximisation of "impurity" is equivalent to the minimisation of entropy:

$$\max_p KL(p||q) = \max_p(-H(S)) = \min_p H(S).$$

The ***entropy over tree*** is

$$H^T(\underbrace{\{S_1, \cdots, S_L\}}_{S^A}) = \sum_{i=1}^{L} \frac{|S_i|}{|S|} H(S_i). \tag{5.5}$$

The ***information gain*** measures how well *a given attribute separates the training data S according to their response variable classification*, i.e. how much information an attribute gives to the classification:

$$IG(S^A) = H(S) - \sum_{i \in Values(A)} \frac{|S_i|}{|S|} H(S_i) \tag{5.6}$$

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

Features that perfectly partition should give **maximal information gain** or **minimal entropy** while unrelated features should give no information.

Example 5.2.2. Let S be the data in Example 5.2.1 with the target variable "Play". Compute the **information gain** of all features (note: Day is not a feature). As an illustration, the calculations for $\text{IG}(S_{\text{Wind}})$ and $\text{IG}(S_{\text{Humidity}})$ are shown.

Illustration: The **overall entropy (of the target output) before splitting**:

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

Remark: Most calculators do not have $\log_2 x$, to calculate it, we normally use $\ln x / \ln 2$.

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

- $\mathbb{P}(S_{Weak,Yes}) = \frac{6}{8}$; $\mathbb{P}(S_{Weak,No}) = \frac{2}{8} \Rightarrow H(S_{Weak}) = -(\frac{6}{8}\log_2\frac{6}{8} + \frac{2}{8}\log_2\frac{2}{8}) = 0.8113$

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

Calculate the information gain **after splitting on** Humidity:

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

The following blank box is for exercises: try to use the steps above to show that the entropy and information gain of the split of S on "Temperature" and "Outlook". [Ans: IG(Temperature)=0.0292, IG(Outlook)=0.2467]

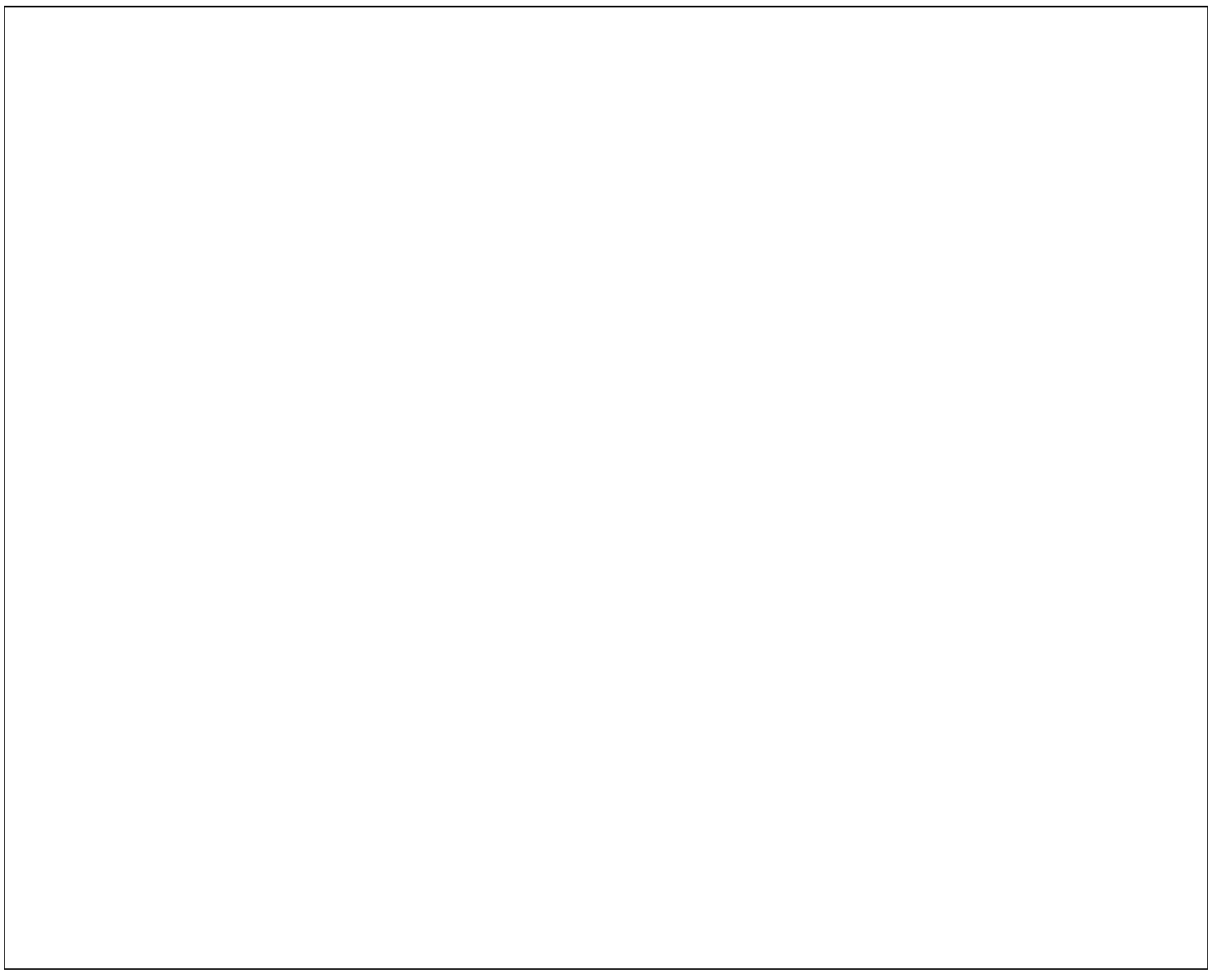
Example 5.2.3 (May 2022 Semester Final Exam, Q4(b))**.** The data in Table 4.1 from a study of low birth weight infants in a neonatal intensive care unit is used to examine the development of bronchopulmonary dysplasia (BPD), a chronic lung disease, in a sample of 10 infants weighing less than 1750 grams. The response variable `bpd` is binary (0 denotes 'no' and 1 denotes 'yes'), denoting whether an infant develops BPD by day 28 of life. The predictors are listed below:

- `brthwght`: birth weight in number of grams;
- `gestage`: gestational age in number of weeks;
- `toxemia`: a condition in pregnancy characterised by abrupt hypertension, albuminuria and edema. This is a binary variable with 0 = no, 1 = yes.

Calculate the **information gain** of a decision tree if the data in Table 4.1 is splitted based on the predictor `brthwght` with a cut-off 1145. (5 marks)

brthwght	gestage	toxemia	bpd
850	30	0	0
1400	30	0	0
1720	30	0	0
1150	32	0	0
1610	34	0	0
1230	32	1	0
820	26	0	1
840	30	0	1
1060	30	0	1
1140	34	0	0

Table 4.1: Training data for a study of low birth weight infants.



Remark 5.2.4. The actual splitting of continous attributes requires the cut-off to be selected from the midpoints between two adjacent sorted values rather than fixing a value as shown in Example 5.2.3.

5.2.3 Intrinsic Information and Gain Ratio

There is a natural bias in information gain measure that **favours** attribute with large number of values. This is because subsets are more likely to be pure if there are a large number of values in the attribute and hence information gain is biased toward choosing attributes with a large number of values.

Example 5.2.5. Consider Example 5.2.2 and use Day as a predictor to calculate the information gain.

Solution:

- For $i = 1, 2, 6, 8, 14$, $\mathbb{P}(S_{D_i,Yes}) = 0$, $\mathbb{P}(S_{D_i,No}) = 1 \Rightarrow H(S_{D_i}) = -(0 \times \log_2 0 + 1 \times \log_2 1) = 0$
- For i=2,3,45,7,9,10,11,12,13, $\mathbb{P}(S_{D_i,Yes}) = 1$, $\mathbb{P}(S_{D_i,No}) = 0 \Rightarrow H(S_{D_i}) = -(1 \times \log_2 1 + 0 \times \log_2 0) = 0$
- $IG(S^{Day}) = 0.9403 - \frac{1}{14} \times 0 - \cdots - \frac{1}{14} \times 0 = 0.9403.$

Remark: 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 the bias on highly branching predictors** by normalising information gain by the intrinsic information of a split. The ***intrinsic information*** of the attribute A, is the entropy of the "splitting factor":

$$H(A) = -\sum_{i=1}^{K} \frac{|A_i|}{|A|} \log_2 \frac{|A_i|}{|A|}. \tag{5.7}$$

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

$$R(S^A) = \frac{IG(S^A)}{H(A)} \tag{5.8}$$

where $IG(S^A)$ is the information gain (5.6) in the previous section. The predictor with the maximum gain ratio is selected as the splitting predictor and the algorithm is similar to the information gain.

Example 5.2.6 (Final Exam Jan 2019, Q2(b))**.** Gain ratio is a modification of information gain. Discuss on why and how the modification is done on information gain. (2 marks)

Solution: Gain ratio is a modification of information gain that **reduces its bias on highly branching predictors**. It takes into account the number and size of branches when choosing a predictor. This is done by normalising information gain by the intrinsic information of a split, which is defined as the information need to determine the branch to which an observation belongs.

Example 5.2.7. Consider the data in Example 5.2.1, compute the **gain ratio** of the output "Play" against other features (excluding Day). As a illustration, the calculations of $R(S^{Wind})$ and $R(S^{Humidity})$ are shown below.

Illustration: The first step is the same as the first step in Example 5.2.2:

$$\mathbb{P}(S_{Yes}) = \frac{9}{14}, \quad \mathbb{P}(S_{No}) = \frac{5}{14} \Rightarrow H(S) = 0.9403$$

The second step is the same as the second step in Example 5.2.2 but with extra calculations to find the gain ratio of Wind:

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

Calculate the gain ratio of Humidity:

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

Now try to use the steps above to check that $R(S^{Temperature}) = 0.0188$ and $R(S^{Outlook}) = 0.1564$.

Example 5.2.8 (Final Exam Jan 2019, Q2(a))**.** State definition for the following terms:

(i) Entropy (1 mark)
(ii) Information gain (1 mark)
(iii) Intrinsic information (1 mark)
(iv) Gain ratio (1 mark)
(v) Gini impurity (1 mark)

Solution:

(i) Entropy characterizes the impurity in a collection of data. [1 mark]
(ii) Information gain measures how well a given split separates the data according to their response variable classification. [1 mark]
(iii) Intrinsic information represents the potential information generated by splitting the data into different partitions. [1 mark]
(iv) Gain ratio is the normalisation of information gain by intrinsic information. [1 mark]
(v) Gini impurity measures total variance across the classes. [1 mark]

Example 5.2.9 (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. You are considering 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.

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.

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).

Solution:

$$\begin{aligned}
\text{Gini(Split 1)} &= \frac{20}{100}\left[1-(\frac{12}{20})^2-(\frac{8}{20})^2\right]+\frac{80}{100}\left[1-(\frac{18}{80})^2-(\frac{62}{80})^2\right] \\
&= 0.2\times 0.48+0.8\times 0.34875 = 0.375 \\
\text{Gini(Split 2)} &= \frac{10}{100}\left[1-(\frac{8}{10})^2-(\frac{2}{10})^2\right]+\frac{90}{100}\left[1-(\frac{22}{90})^2-(\frac{68}{90})^2\right] \\
&= 0.1\times 0.32+0.9\times 0.3693827 = 0.3644
\end{aligned}$$

$\Rightarrow$ **Lower Gini impurity** is better. $\therefore$ Split 2 is preferred. I is wrong.

$$\begin{aligned}
\text{H(Split 1)} &= -\frac{20}{100}\left[\frac{12}{20}\log_2\frac{12}{20}+\frac{8}{20}\log_2\frac{8}{20}\right]-\frac{80}{100}\left[\frac{18}{80}\log_2\frac{18}{80}+\frac{62}{80}\log_2\frac{62}{80}\right] \\
&= -0.2\times(-0.9710)-0.8\times(-0.7692) = 0.8096 \\
\text{H(Split 2)} &= -\frac{10}{100}\left[\frac{8}{10}\log_2\frac{2}{10}+\frac{2}{10}\log_2\frac{2}{10}\right]-\frac{90}{100}\left[\frac{22}{90}\log_2\frac{22}{90}+\frac{68}{90}\log_2\frac{68}{90}\right] \\
&= -0.1\times(-0.7219)-0.9\times(-0.8024) = 0.7944
\end{aligned}$$

$\Rightarrow$ **Smaller entropy** (vs larger information gain) is better. $\therefore$ 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)

Example 5.2.10 (Final Exam Jan 2019, Q2(d))**.** A classification tree is being constructed to predict if an insurance policy will lapse. A random sample of 1000 policies contains 320 that lapsed. You are considering two splits:

Split 1: One node has 260 observations with 140 lapses and one node has 740 observations with 180 lapses.

Split 2: One node has 120 observations with 70 lapses and one node has 880 observations with 250 lapses.

Determine which split is preferred based on

(i) Gini impurity (4 marks)

(ii) Information gain (4 marks)

(iii) Gain ratio (3 marks)

(iv) Classification errors (3 marks)

Solution: The breakdown of classification error for Split 1 is given below.

- Node A: $[140+, 120-]$; Node B: $[180+, 560-]$
- Node A will predict $+$, hence error $= 120$
- Node B will predict $-$, hence error $= 180$
- Total classification error $= \dfrac{120 + 180}{1000} = 0.3$

The breakdown of classification error for Split 2 is given below.

- Node A: $[70+, 50-]$; Node B: $[250+, 630-]$
- Node A will predict $+$, hence error $= 50$
- Node B will predict $-$, hence error $= 250$
- Total classification error $= \dfrac{50 + 250}{1000} = 0.3$

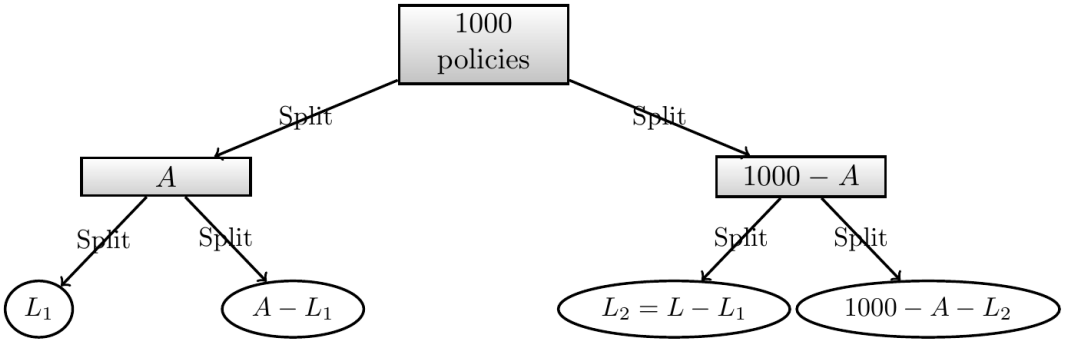
A lower classification error is preferred, but both of them give the same classification error, so either one is fine.

Not in the question but assumption: If we have a "split 3" given below:

- Node A: $[70+, 20-]$; Node B: $[250+, 660-]$
- Node A will predict $+$, hence error $= 20$
- Node B will predict $-$, hence error $= 250$
- Total classification error $= \dfrac{20 + 250}{1000} = 0.27$

Split 3 is preferred due to lower classification error.

In general, if we have A observations, L_1 lapses on the left, L_2 lapses on the right, $L_1 + L_2 = L$. In this example, $L = 320$.



Total classification error is

$$\frac{(A - L_1) + L_2}{1000} = \frac{A - L_1 + (L - L_1)}{1000} = \frac{A + L - 2L_1}{1000}$$

5.3 Variance Reduction for Numeric Output

Variance reduction [Breiman et al., 1984] is often employed in **regression tree**, i.e. the target variable is continuous. It can be regarded as the 'information gain' for Gaussian model. The variance reduction of an attribute A is defined as the total reduction of the variance of the target variable Y due to the split at this node:

$$I_V(A) = \frac{1}{|S|^2}\sum_{i\in S}\sum_{j\in S}\frac{1}{2}(y_i - y_j)^2 - \left(\frac{1}{|S_t|^2}\sum_{i\in S_t}\sum_{j\in S_t}\frac{1}{2}(y_i - y_j)^2 + \frac{1}{|S_f|^2}\sum_{i\in S_f}\sum_{j\in S_f}\frac{1}{2}(y_i - y_j)^2\right) \tag{5.9}$$

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. Each of the above summands are indeed variance estimates, though, written in a form without directly referring to the mean.

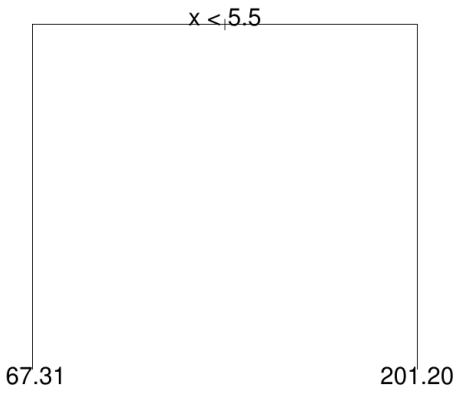
Example 5.3.1. Consider the data from Exercise 1.7.1.

y	23.82	47.16	66.66	88.39	110.54
x	1	2	3	4	5
y	131.1	174.15	214.72	233.9	252.14
x	6	8	10	11	12

Note that both R's `tree` and `rpart` uses deviance of the following form rather than (5.9):

$$deviance(y) = \frac{1}{2|S|}\sum_{i\in S}\sum_{j\in S}(y_i - y_j)^2.$$

The tree generated by `tree` with the leaves the means of the y_i after the split is shown below:



and the deviances

before the split: $\frac{1}{2\times 10}\sum_{i,j}(y_i - y_j)^2$;

after the split: $\frac{1}{2\times 5}\sum_{x<5.5}(y_i - y_j)^2 = 4611.024$ and $\frac{1}{2\times 5}\sum_{x>5.5}(y_i - y_j)^2 = 9492.676$

5.4 Various Implementations of Decision Tree Algorithms

The variations in the data splitting lead to many different decision tree construction algorithms. An evolution of various decision tree algorithms is shown below [Loh, 2014], [Charpentier, CRC Press, Section 4.4].

1st generation: The introduction classification tree through node splitting.

- THAID [Messenger and Mandell, 1972]
- CHAID [Kass, 1980]
- ID3 [Quinlan, 1986]

2nd generation: The introduction of tree-prunning, cross-validation, gain ratio.

- CART [Breiman et al., 1984]
- M5 (Quinlan, 1992)* (R's Cubist::cubist: Rule- and Instance-Based Regression Modelling)
- C4.5 (Quinlan, 1993)
- FACT (Loh and Vanichsetakul, 1988)

3rd generation: The elimination of selection bias and the use of more enhanced split rules.

- QUEST (Loh and Shih, 1997)
- CRUISE (Kim and Loh, 2001, 2003)
- Bayesian CART (Chipman et al., 1998; Denison et al., 1998)*

4th generation: Ensemble methods are used to improve accuracy tree-based models.

- GUIDE (Loh, 2002, 2009; Loh and Zheng, 2013; Loh et al., 2015)
- CTREE (Hothorn et al., 2006)
- MOB (Zeileis et al., 2008)*
- Random forest [Breiman, 2001]
- TARGET (Fan and Gray, 2005; Gray and Fan, 2008)
- BART (Chipman et al., 2010)

It is impossible to cover all classification trees. In this section, we hope explore the simplest classification tree algorithm — ID3, and then touch a little bit on C4.5 and CART. A comparison table below summarises their differences.

Aspect	**Original ID3**	**C4.5**	**CART**
Features	Discrete features only; cannot handle numeric features	continuous and discrete features	continuous and discrete features
missing values	Not supported	Handles missing attributes by ignoring them in Gain Ratio or IG	Surrogate variables are used (see rpart documentation)
splitting criteria	Multi-way split which max. IG / min. entropy	Multi-way split which maximises gain ratio	uses Gini impurity to perform **binary split**
shape	short and wide tree	short and wide tree	taller tree
pruning	no pruning, prone to overfitting	performs post-pruning (bottom-up pruning)	prunes the tree to a size with the lowest cross-validation estimate of error

5.4.1 ID3 Algorithm

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan [Quinlan, 1986] for classification problems. The training observations are sorted to the corresponding internal nodes. If the nodes only contain observations from a single class, 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.

The following example demonstrates the ID3 algorithm for the handling of categorical features.

Example 5.4.1 (Final Exam May 2019, Q5(b))**.** A decision tree model is built to detect whether an online transaction is a fraud transaction (Yes/No), based on three variables – credit card association, transaction amount and country. Table Q5(b) – i to Q5(b) – iii shows the number of observations for different variables with different levels, for fraud and not fraud cases respectively.

Credit Card Associations	Fraud = Yes	Fraud = No
Amex	11	20
Master	34	47
Visa	25	33

Table Q5(b) – i

Transaction Amount	Fraud = Yes	Fraud = No
<1000	52	21
≥1000	18	79

Table Q5(b) – ii

Country	Fraud = Yes	Fraud = No
A	25	32
B	17	46
C	28	22

Table Q5(b) – iii

Construct a decision tree with leaves purity of at least 70% by using gain ratio. State the prediction and respective purity for each terminal node. (15 marks)

Solution: Let S be the whole table of Fraud, Fraud=Yes be positive (+) class and Fraud=No be negative (-) class.

To answer this question, we need to recall the Gain ratio, Equation (5.8) on Page 112:

$$R(S^A) = IG(S^A)/H(A).$$

Total, $S = [70+, 100-]$ $\Rightarrow H(S) = -\left[\frac{70}{170}\log_2\frac{70}{170} + \frac{100}{170}\log_2\frac{100}{170}\right] = 0.9774$

For X_1 = Credit card association [Amex=A; Master=M; Visa=V]:

$A = [11+, 20-]$ $\Rightarrow H(Fraud|X_1 = A) = -\left[\frac{11}{31}\log_2\frac{11}{31} + \frac{20}{31}\log_2\frac{20}{31}\right] = 0.9383$

$M = [34+, 47-]$ $\Rightarrow H(S_M) = 0.9813$

$V = [25+, 33-]$ $\Rightarrow H(S_V) = 0.9862$

$IG(S^{X_1}) = 0.9774 - \left[\frac{31}{170}\times 0.9383 + \frac{81}{170}\times 0.9813 + \frac{58}{170}\times 0.9862\right] = 0.0023$

$H(X_1) = -\left[\frac{31}{170}\log_2\frac{31}{170} + \frac{81}{170}\log_2\frac{81}{170} + \frac{58}{170}\log_2\frac{58}{170}\right] = 1.4866$

$R(S^{X_1}) = \dfrac{0.0023}{1.4866} = 0.0015$

For X_2 = Transaction amount [$< 1000 = L$; $\geq 1000 = H$]:

$L = [52+, 21-]$ $\Rightarrow H(S_L) = 0.8657$

$H = [18+, 79-]$ $\Rightarrow H(S_H) = 0.6921$

$IG(S^{X_2}) = 0.9774 - \left[\frac{73}{170} \times 0.8657 + \frac{97}{170}0.6921\right] = 0.2108$
$H(X_2) = -\left[\frac{73}{170}\log_2\frac{73}{170} + \frac{97}{170}\log_2\frac{97}{170}\right] = 0.9856$
$R(S^{X_2}) = 0.2139$

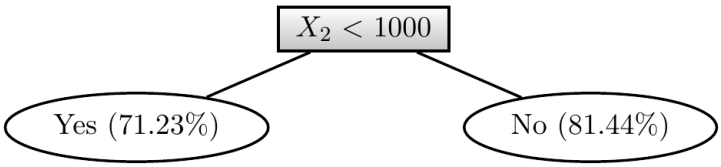
For X_3 = Country $[A; B; C]$:
$A = [25+, 32-] \quad \Rightarrow H(S_A) = 0.9891$
$B = [17+, 46-] \quad \Rightarrow H(S_B) = 0.8412$
$C = [28+, 22-] \quad \Rightarrow H(S_C) = 0.9896$
$IG(S^{X_3}) = 0.0430$
$H(X_3) = -\left[\frac{57}{170}\log_2\frac{57}{170} + \frac{63}{170}\log_2\frac{63}{170} + \frac{50}{170}\log_2\frac{50}{170}\right] = 1.5786$
$R(S^{X_3}) = 0.0272$

The transaction amount, X_2, has the highest gain ratio, hence it should be chosen as the root node.

For $X_2 = L(< 1000)$: prediction=Yes; purity = $\frac{52}{52+21} = 0.7123$ (the "purity" is more than 70%)

For $X_2 = H(\geq 1000)$: prediction=No; purity = $\frac{79}{18+79} = 0.8144$ (the "purity" is more than 70%)

Therefore, we have the classification tree below.



Example 5.4.2. You are a team member from the data team of ClickMe, an online platform selling electronic products. Your company would like to do target marketing. You were asked to build a model to predict whether the customer will buy the product based on their demographic information such as age, race, gender, and income. The data collected is shown in the table below.

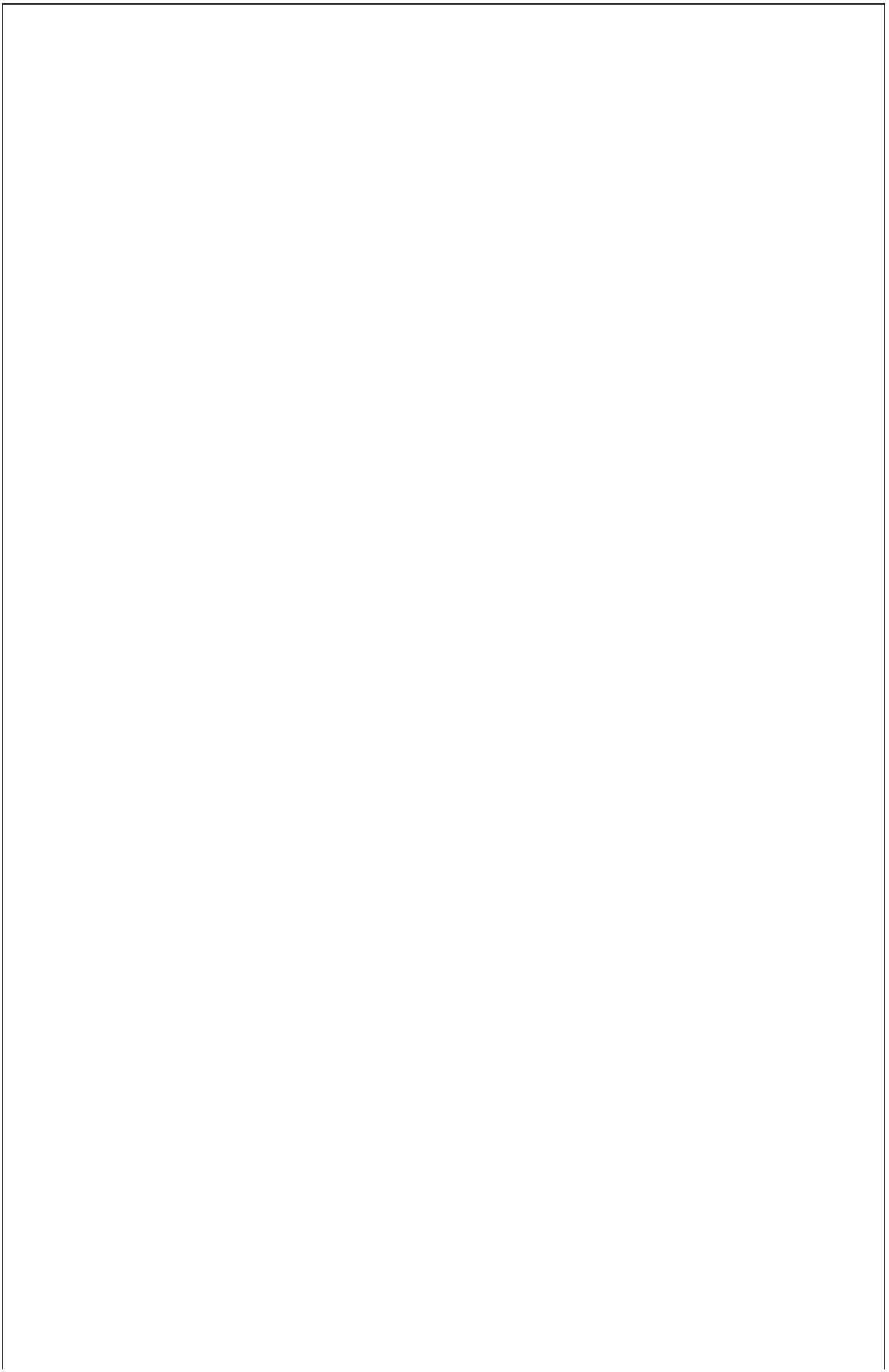
Obs	Age	Race	Gender	Income	Buy
1	52	Malay	Male	11500	Not Buy
2	22	Chinese	Male	6500	Buy
3	30	Indian	Male	8000	Buy
4	26	Indian	Female	8500	Buy
5	27	Malay	Female	6500	Buy
6	32	Malay	Female	6000	Not Buy
7	33	Malay	Male	9500	Not Buy
8	50	Indian	Female	4000	Not Buy
9	31	Chinese	Male	10500	Buy
10	27	Malay	Male	10000	Buy
11	25	Indian	Female	5500	Not Buy
12	40	Indian	Female	3000	Not Buy
13	48	Malay	Female	8000	Not Buy
14	46	Chinese	Female	7000	Buy
15	51	Indian	Female	3000	Not Buy
16	42	Indian	Male	5000	Buy

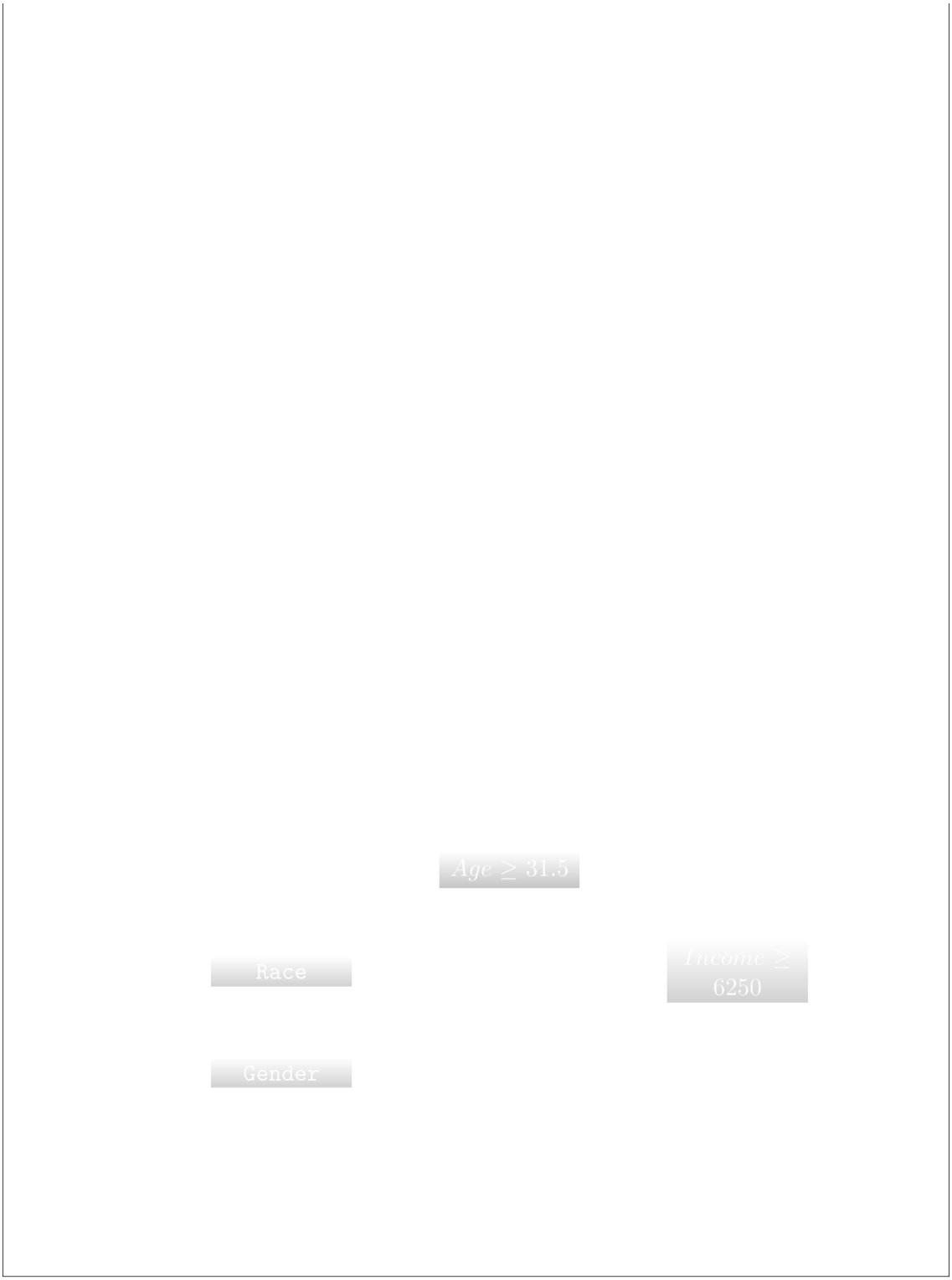
Additional Information:

- Use L-way split for qualitative predictors
- Split for *Age* is 31.5

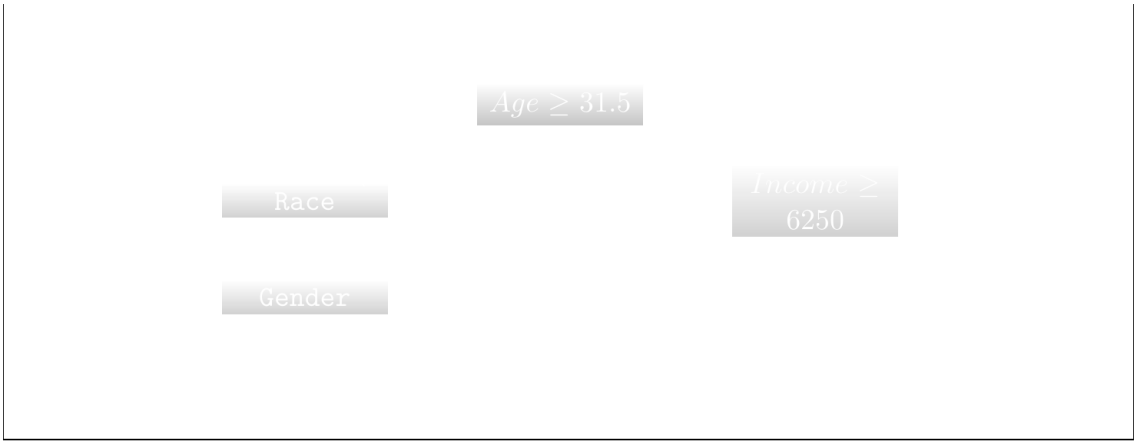
- Split for *Income* is 6250

(a) Construct a decision tree with pure leaves by using information gain as criterion.





(b) Construct a decision tree with pure leaves by using Gini impurity as criterion.



Example 5.4.3. Continue Example 5.2.2 with the data in Example 5.2.1 and construct the full "playing tennis" ID3 decision tree.

Solution: ID3 selects the parent root using the highest information gain. From Example 5.2.2, the highest information gain is 0.2467, which is "Outlook".

This leads to the table to be split into 3 sub-tables.

When "Outlook" = Overcast,

Day	Outlook	Temperature	Humidity	Wind	Play
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

The target "Play" is pure. There is no need to further branch.

When "Outlook" = Rain,

Day	Outlook	Temperature	Humidity	Wind	Play
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

The target "Play" is not pure, so we need to calculate the information gain (?) for features other than "Outlook":

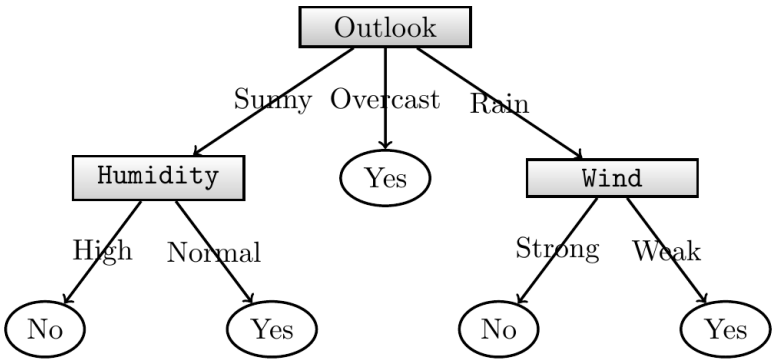
- $H(S^{Temperature}) = 0.9509775$; $IG(S^{Temperature}) = 0.01997309$;
- $H(S^{Humidity}) = 0.9509775$; $IG(S^{Humidity}) = 0.01997309$
- $H(S^{Wind}) = 0$; $IG(S^{Wind}) = 0.9709506$ ✓

When "Outlook" = Sunny,

Day	Outlook	Temperature	Humidity	Wind	Play
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

The target "Play" is not pure, so we need to calculate the entropy (less to calculate than information gain) for features other than "Outlook":

- $H(S^{Temperature}) = 0.4$; $IG(S^{Temperature}) = 0.5709506$
- $H(S^{Humidity}) = 0$; $IG(S^{Humidity}) = 0.9709506$ ✓
- $H(S^{Wind}) = 0.9509775$; $IG(S^{Wind}) = 0.01997309$



Remark 5.4.4. The decision tree classifies whether a morning is suitable to play tennis. An observation is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf, Yes or No. For example, an observation with {Outlook=Sunny; Temperature=Hot; Humidity=High; Wind=Strong} would be sorted down the left most branch of the tree and would hence be classified as a negative class, i.e. the tree predicts not to play tennis.

Remark 5.4.5. `https://www.r-bloggers.com/id3-classification-using-data-tree/` has an implementation of ID3 in R based on the `data.tree` library.

5.4.2 C4.5 and C5.0

The ID3 algorithm is restricted to inputs which are all categorical. It is possible to introduce a threshold to turn the numeric features to a binary categorical data [Mitchell, 1997, §3.7.2].

The suggestions for the improvement of ID3 algorithm lead to `https://en.wikipedia.org/wiki/C4.5_algorithm` which grows an initial tree using divide-and-conquer algorithm as follows:

1. If all cases in S belong to the same class or S is small, the tree is a leaf labelled with the most frequent class in S.

2. Otherwise, choose a single attribute X_i with two or more outcomes based on "gain ratio" as the root of the tree with one branch for each outcome of the measure, partition S into corresponding subsets $S_1, S_2, \cdots, S_L$ if X_i has L distinct class. If X_i is ordered, the node is split into two children nodes "$S_1 = X_i < c$" and "$S_2 = X_i \geq c$" where c needs to be estimated. Apply the same procedure recursively to each subset.

The initial tree above is then pruned to avoid overfitting. The pruning algorithm is based on a pessimistic estimate of the error rate associated with a set of n cases. Pruning is carried out from the leaves to the root based on binomial confidence limits.

Implementations of C4.5 are available at `https://www.rulequest.com/Personal/`, Weka, `https://github.com/barisesmer/C4.5` and `https://sefiks.com/2018/05/13/a-step-by-step-c4-5-decision-tree-example/`

C4.5 converts the trained trees into sets of if-then rules. These accuracy of each rule is

then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule's precondition if the accuracy of the rule improves without it.

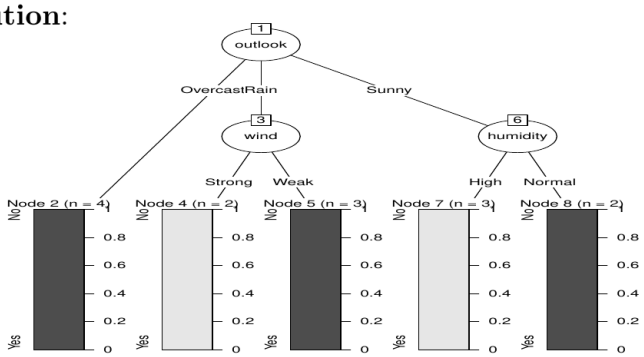
C4.5 is superseded in 1997 by a commercial system See5/C5.0 according to `https://rulequest.com/see5-comparison.html`) and CART (which is implemented in an R package `rpart`).

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 for classification problems.

The `C50` library implements C5.0 decision trees and rule-based models for pattern recognition extending C4.5. The company `https://www.rulequest.com` owns the pattern of C5.0 and provides `Cubist` (Rule- And Instance-Based Regression Modelling) and `outliertree` (Explainable outlier detection through decision tree conditioning based on `GritBot`) for statistical inference with tree models.

Example 5.4.6 (Categorical Features)**.** Use C5.0 to construct the ID3 tree for the tennis data in Example 5.2.1.

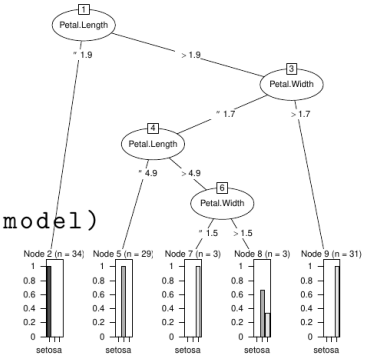
Solution:



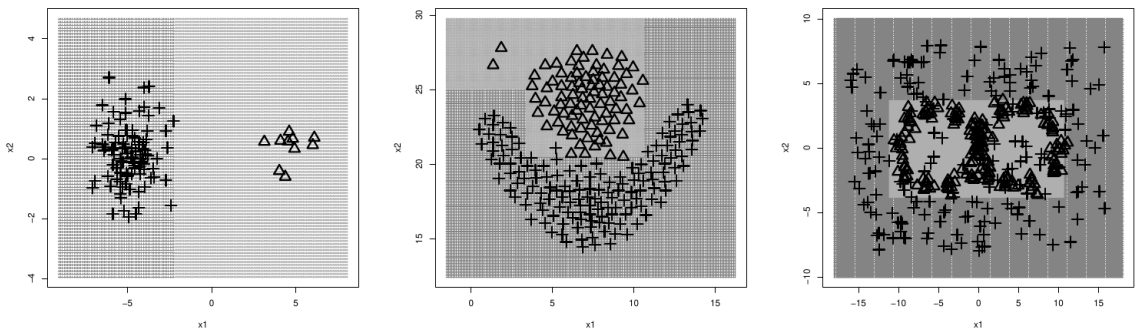
Example 5.4.7 (Numeric Features)**.** Use C5.0 to analyse the iris flower data.

Solution:

```
library(C50)
set.seed(1)
idx = sample(nrow(iris), 100)
iris.train = iris[ idx, ]
iris.test  = iris[-idx, ]
model=C5.0(Species~., data=iris.train); plot(model)
yhat =predict(object=model, iris.test,
       type="class")
print(table(yhat, iris.test$Species))
```



Example 5.4.8 (Decision Boundaries of C5.0)**.** The decision boundaries of the C5.0 for completely separable data, milely nonlinear flame data and strongly nonlinear data are shown below.



□

5.4.3 CART

Classification And Regression Tree (CART) is a binary tree generation algorithm using Gini index as its splitting criteria for classification and deviance for regression. CART handles missing values by surrogating tests to approximate outcomes.

CART grows an initial tree using binary splitting as follows [Breiman et al., 1984]:

1. Find each predictor's best split.
   - For each continuous and ordinal predictor, sort its values from the smallest to the largest (of K different values). For the sorted predictor, go through each value from top to examine each candidate split point (call it v, if $x \leq v$, the case goes to the left child node, otherwise, goes to the right.) to determine the best. The best split point is the one that maximise the splitting criterion (usually the Gini impurity) the most when the node is split according to it.
   - For each nominal predictor X of I categories, examine each $2^I - 1$ possible subset of categories (call it A, if $x \in A$, the case goes to the left child node, otherwise, goes to the right.) to find the best split.
2. Find the node's best split: Among the best splits found in step 1, choose the one that maximises the splitting criterion.
3. Split the node using its best split found in step 2 if the stopping rules are not satisfied.

In R, CART is supported by `tree`, `rpart`, `party` and `partykit`.

The `tree` library provides a limited functions of constructing, plotting and pruning trees. The `rpart`, which stands for "recursive partitioning" for classification (method="class"), regression (method="anova") and survival (method="exp") trees, implements most of the functionality of Breiman et al. [1984]. The `rpart.plot` or `partykit::plot.party` can be used to draw nice diagrams for `rpart`.

```
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, ...)

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

The `ctree()` from `party` is designed based on continuous input(s) X_i, however, it supports nominal, ordinal, numeric, censored as well as multivariate response variables Y. It implements the conditional inference trees which embed tree-structured regression models:

1. Test the global null hypothesis of independence between X_i and Y;
2. Perform binary split in the selected X_i;
3. Recursively repeat steps 1. & 2.

The `ctree()` from `partykit` is a reimplementation of (most of) `party::ctree` which has similar options and treats nominal variables as ordered variables.

```
ctree(formula, data, subset, weights, na.action=na.pass, offset,
```

```
  cluster, control = ctree_control(...), ytrafo = NULL,
  converged = NULL, scores = NULL, doFit = TRUE, ...)
```

`ctree` avoids certain variable selection bias of rpart (which tends to select variables that have many possible splits or many missing values) and uses a (permutation-based) significance test procedure in order to select variables instead of selecting the variable that maximises the impurity (e.g. Gini coefficient).

In contrast to rpart and ctree which are based on impurity measures or statistics on the individual input against the output, the **model-based recursive partitioning** yields a decision tree with fitted parametric models (such as GLM) associated with each terminal node.

```
party::mob(formula, weights, data = list(), na.action=na.omit,
  model=glinearModel, control=mob_control(), ...)

partykit::mob(formula, data, subset, na.action, weights, offset,
  cluster, fit, control=mob_control(), ...)
```

In Python, CART for classification is implemented as `DecisionTreeClassifier` in `sklearn.tree`, which is capable of both binary classification with labels $\{-1, 1\}$ and multiclass classification with labels $\{0, \cdots, K-1\}$. CART for regression is implemented as `DecisionTreeRegressor` which does not support categorical variables for now (if the input is categorical, it will have to be converted to number).

```
DecisionTreeClassifier(*, criterion='gini', splitter='best',
    max_depth=None, min_samples_split=2, min_samples_leaf=1,
    min_weight_fraction_leaf=0.0, max_features=None,
    random_state=None, max_leaf_nodes=None,
    min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

Example 5.4.9. Use CART to construct a decision tree for the tennis data in Example 5.2.1.

Solution: We can either use

(a) `tree` library:

```
1 library(tree)
2 tennis.data = read.csv("playtennis2.csv",row.names=1,
3   stringsAsFactor=T,skip=1)
4 t.m = tree(Play~., 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)
```

The text/logic representation of `tree::tree`:

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 14 18.250 Yes ( 0.3571 0.6429 )
   2) outlook: Rain,Sunny 10 13.860 No ( 0.5000 0.5000 )
     4) humidity: High 5  5.004 No ( 0.8000 0.2000 )
       8) outlook: Rain 2  2.773 Yes ( 0.5000 0.5000 )
        16) wind: Strong 1  0.000 No ( 1.0000 0.0000 ) *
        17) wind: Weak 1  0.000 Yes ( 0.0000 1.0000 ) *
       9) outlook: Sunny 3  0.000 No ( 1.0000 0.0000 ) *
     5) humidity: Normal 5  5.004 Yes ( 0.2000 0.8000 )
      10) wind: Strong 2  2.773 No ( 0.5000 0.5000 )
```

```
        20) outlook: Rain 1  0.000 No ( 1.0000 0.0000 ) *
        21) outlook: Sunny 1  0.000 Yes ( 0.0000 1.0000 ) *
      11) wind: Weak 3  0.000 Yes ( 0.0000 1.0000 ) *
   3) outlook: Overcast 4  0.000 Yes ( 0.0000 1.0000 ) *
```

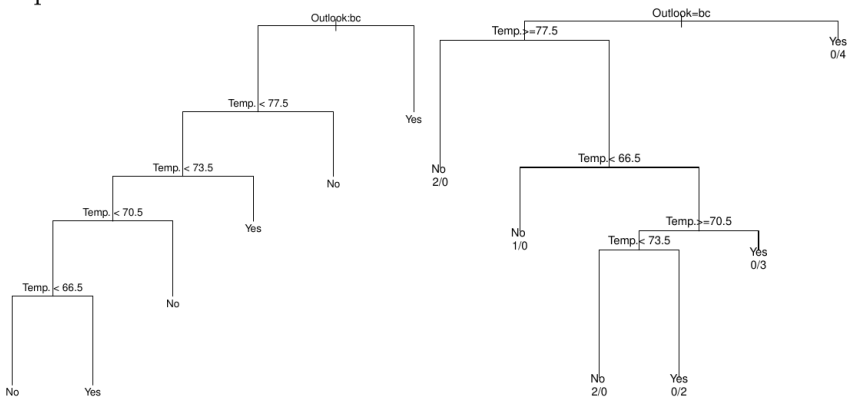
or (b) the `rpart` library:

```
library(rpart)
tennis.data = read.csv("playtennis2.csv",row.names=1,skip=1)
#rpart can handle strings (no need to convert to factor)
dt = rpart(Play ~ ., data=tennis.data,
     control=rpart.control(minsplit=2, minbucket=1, cp=0.001))
print(dt)
plot(dt, margin=0.05); text(dt, use.n=TRUE, cex=0.8)
```

The text/logic representation of `rpart::rpart`:

```
node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 14 5 Yes (0.3571429 0.6428571)
   2) outlook=Rain,Sunny 10 5 No (0.5000000 0.5000000)
     4) humidity=High 5 1 No (0.8000000 0.2000000)
       8) outlook=Sunny 3 0 No (1.0000000 0.0000000) *
       9) outlook=Rain 2 1 No (0.5000000 0.5000000)
        18) wind=Strong 1 0 No (1.0000000 0.0000000) *
        19) wind=Weak 1 0 Yes (0.0000000 1.0000000) *
     5) humidity=Normal 5 1 Yes (0.2000000 0.8000000)
      10) wind=Strong 2 1 No (0.5000000 0.5000000)
        20) outlook=Rain 1 0 No (1.0000000 0.0000000) *
        21) outlook=Sunny 1 0 Yes (0.0000000 1.0000000) *
      11) wind=Weak 3 0 Yes (0.0000000 1.0000000) *
   3) outlook=Overcast 4 0 Yes (0.0000000 1.0000000) *
```

Graphical representation of CART trees:



(a) tree (b) rpart

So far, we have only mentioned the tree construction part of various algorithms, in practice, tree pruning criteria, cross-validation (Section 6.4.1) are incorporated into a tree-algorithm such as CART [James et al., 2013, Chapter 8]:

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. Apply cost complexity pruning to the large tree in order to obtain a sequence of best sub-trees, as a function of α. E.g. for regression tree, the cost function $C_\alpha(T) = \sum_{i=1}^{|T|} \sum_{\boldsymbol{x}_k \in R_m} (y_k - \widehat{c}_m)^2 + \alpha|T|$ where T is a sub-tree of the initial tree T_0.

3. Use K-fold cross-validation to choose α. That is, for each $k = 1, \cdots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

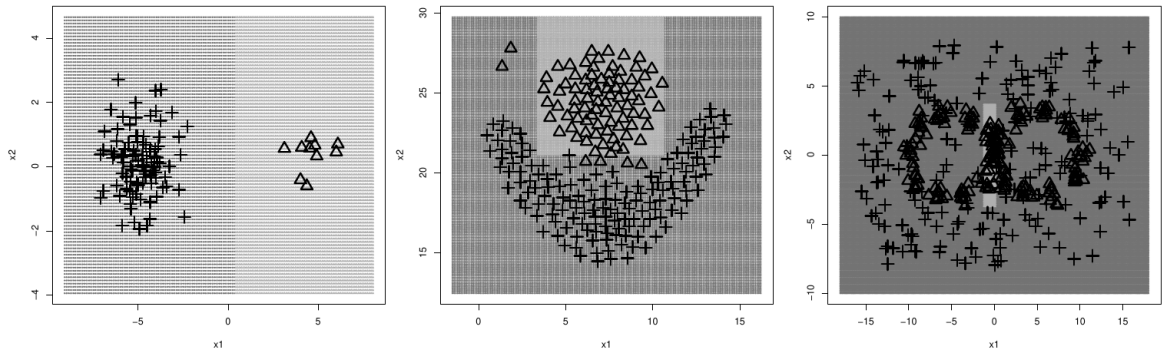
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimise the average error.

4. Return the subtree from Step 2 that corresponds to the chose value of α.

The leaves of CART partition the input space into rectilinear regions. In each partition, the predicted response class is the class with the highest probability. In particular, we can "visualise" the partitioning of decision tree in 2D which are illustrated below.

Example 5.4.10 (Decision Boundaries of CART 'tree')**.** The decision boundaries of the CART for completely separable data, milely nonlinear flame data and strongly nonlinear data are shown below.

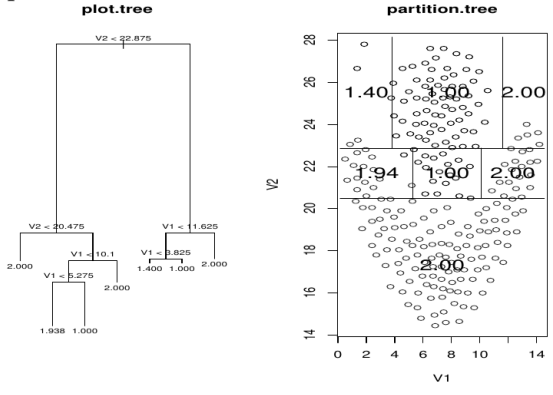


It is a surprise that CART perform so badly with nonlinear data! Only the middle is identified to be Δ. □

Example 5.4.11 (CART for Regression Flame Data)**.** Training the CART `tree` on the "flame" data with the output as numeric would lead to a regression tree with the following text representation:

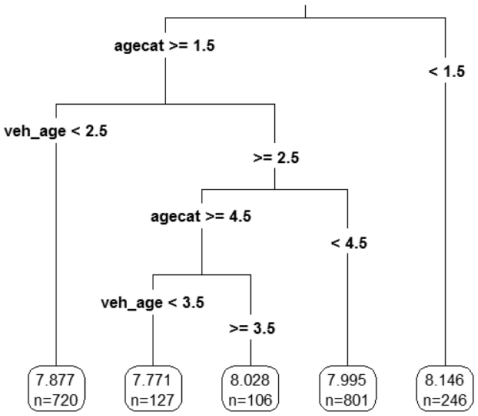
```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 240 55.4600 1.638
   2) V2 < 22.875 165 16.8100 1.885
     4) V2 < 20.475 113  0.0000 2.000 *
     5) V2 > 20.475 52 12.0600 1.635
      10) V1 < 10.1 34  8.3820 1.441
        20) V1 < 5.275 16  0.9375 1.938 *
        21) V1 > 5.275 18  0.0000 1.000 *
      11) V1 > 10.1 18  0.0000 2.000 *
   3) V2 > 22.875 75  6.3470 1.093
     6) V1 < 11.625 70  1.9430 1.029
      12) V1 < 3.825 5  1.2000 1.400 *
      13) V1 > 3.825 65  0.0000 1.000 *
     7) V1 > 11.625 5  0.0000 2.000 *
```

The 2D feature space can be recursively partitioned as shown below.



Example 5.4.12 (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

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)

Solution:
LCA(I) = 8.146;
LCA(II) = 8.028;
LCA(III) = 7.771
Answer: (E)

More examples of regression trees can be found in [Lantz, 2015, Chapter 6], `https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html`, and `https://www.python-course.eu/Regression_Trees.php`.

5.4.4 Other Implementations: CHAID

Chi-Square Automation Interaction Detection (CHAID) was developed as an early Decision Tree based on the 1963 model of AID tree. As opposed to CHAID, it does not substitute the missing values with the equally reducing values. All the missing values are taken as a single class which facilitates merging with another class. For finding the significant variable, we make use of the χ^2 test. This is only valid for qualitative or discrete variables. We create CHAID in the following steps (according to `https://data-flair.training/blogs/r-decision-trees/`):

1. The cross-tabulation of categories is carried out by χ^2 which groups them in k-categories. Furthermore, the response variable for X holds 3 categories. The categories of X cross-tabulates them with the k categories of the response variable for every X having at least 3 categories.
2. We need to repeat the above step until all the pairs of categories have a significant χ^2, or until no more than two categories exist. If there is an existence of frequency below the minimum value, the collaboration is carried out with a category that is closest to χ^2.
3. The groupings are performed to the variables into their corresponding classes. If there are any missing values present then it is assumed as a category. After the completion, CHAD merges it with another category.

4. We now obtain the probability of χ^2 of the best table. We multiply this with Bonferroni correction, obtain the number of possibilities to cluster m categories into g groups with this coefficient. Furthermore, its product by the probability that is associated with χ^2 prevents evaluation of the significance of multiple values.
5. With CHAID, we select the most significant variable for χ^2. This variable possesses the lowest probability. If it is below the given limit, then we perform division of the node into child nodes that are equal to the categories of variables required for grouping.

CHAID trees are wider than deeper. Furthermore, there is no pruning function available for it. Also, the construction halts when the largest tree is created. It is currently available in commercial software only [Loh, 2014]. Other implementions are also not open source and not free for general use.

5.5 Practical Considerations

Decision trees are bad in many nonlinear data but can be extremely useful if the features are not terribly complex. This section discusses a benefit of decision tree and the practical condiderations to consider when applying decision tree for predictive modelling.

5.5.1 Interpretability

Decision trees can be displayed graphically, hence are *easily interpreted and explained*. They closely mirror human's decision-making process than other classification methods. They can easily handle qualitative predictors without creating dummy variables. Decision trees can handle missing values in predictors and are robust to error.

Example 5.5.1 (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.

(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 James et al. [2013, Section 8.1]. The statement that trees are easier to explain than linear regression methods 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)

5.5.2 Underfitting and Overfitting

The training and testing error rates are large when the size of the tree is very small — **model underfitting**.

When the tree becomes large, its training error rate is becoming smaller but the testing error rate is getting large (rather than smaller) — **model overfitting**

- Decision trees tend to **overfit** on data with a large number of features. Getting the right ratio of samples to number of features is important, since a tree with few samples in high dimensional space is very likely to overfit.
- Consider performing dimensionality reduction (Chapter 8, PCA, ICA, or Feature selection) beforehand to give your tree a better chance of finding features that are discriminative.
- Understanding the decision tree structure will help in gaining more insights about how the decision tree makes predictions, which is important for understanding the important features in the data.
- Visualise your tree as you are training, e.g. use a maximum depth of 3, to get a feel for how the tree is fitting to your data, and then increase the depth.
- Remember that the number of samples required to populate the tree doubles for each additional level the tree grows to.
- Balance your dataset before training to prevent the tree from being biased toward the classes that are dominant. Class balancing can be done by sampling an equal number of samples from each class, or preferably by normalising the sum of the sample weights for each class to the same value.
- If the samples are weighted, it will be easier to optimise the tree structure using weight-based pre-pruning criterion.
- In Scikit-learn, all decision trees use `np.float32` arrays internally. If training data is not in this format, a copy of the dataset will be made.
- If the input matrix X is very sparse, it is recommended to convert to sparse `csc_matrix` before calling fit. Training time can be orders of magnitude faster for a sparse matrix input compared to a dense matrix when features have zero values in most of the samples.

5.5.3 Hyperparameter Tuning — Tree Pruning

We need to decide how **large** should the tree be:

- How many **branches** / **leaves** should we have?
- How **tall** (depth) should the tree be?

Two ways to control tree size:

- **Pre-pruning**: Does not allow the tree to grow beyond the limit
- **Post-pruning**: Let the tree grow to a large tree and then trim the tree down.

Reasons to control tree size:

- A decision tree may produce good predictions on the training set, but it is likely to **overfit** the data, leading to poor test set performance because "noise" is being fitted.
- A large tree may useless in decision making.

The theory behind tree pruning is so complex that the usual exam questions associated with it is "philosophical" questions shown below.

Example 5.5.2 (Final Exam Jan 2019, Q2(c))**.** A decision tree might be overfitted to the training set if the resulting tree is too complex. What are the approaches that can be used to avoid building a complex tree? Explain the approaches. (4 marks)

Solution: The two approaches are:

- **Pre-pruning** that stop growing the tree earlier, before it perfectly classifies the training set.
- **Post-pruning** that allows the tree to perfectly classify training set, and post prune the tree.

Example 5.5.3 (SRM-02-18, Q25)**.** Determine which of the following statements concerning decision tree pruning is/are true.

I. The recursive binary splitting method can lead to overfitting the data.

II. A tree with more splits tends to have lower variance.

III. When using the cost complexity pruning method, $\alpha = 0$ results in a very large tree.

(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:

I is true because the method optimises with respect to the training set, but may perform poorly on the test set.

II is false because additional splits tends to increase variance due to adding to the complexity of the model.

III is true because in this case only the training error is measured.

Answer: (C)

Example 5.5.4 (Final Exam May 2019, Q2(b))**.** A basic decision tree has been constructed to predict the promotion of an employee. Figure Q2(b) shows the tree formed.

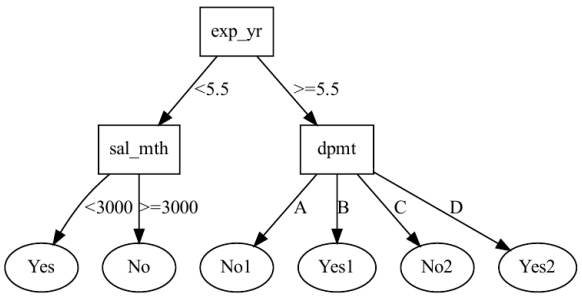


Figure Q2(b)

(i) Compare and discuss the importance of variables based on the results from Table Q2(a) in Example 3.1.12 and Figure Q2(b) in this example. (3 marks)

(ii) State two disadvantages of a single decision tree and approaches to overcome the stated disadvantages. (4 marks)

5.5.4 Complexity

Let n be the number of samples and p be the number of features (which are used throughout the lecture). According to `https://scikit-learn.org/stable/modules/tree.html`, the 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. Assuming that the subtrees remain approximately balanced, the cost at each node consists of searching through $O(p)$ to find the feature that offers the largest reduction in entropy. This has a cost of $O(p \times n \log(n))$ at each node, leading to a total cost over the entire trees (by summing the cost at each node) of $O(p \times n^2 \log(n))$.

Scikit-learn offers a more efficient implementation for the construction of decision trees. A naive implementation in Scikit-learn would recompute the class label histograms (for classification) or the means (for regression) at for each new split point along a given feature. Presorting the feature over all relevant samples, and retaining a running label count, will reduce the complexity at each node to $O(p \log(n))$, which results in a total cost of $O(p \times n \log(n))$. This is an option for all tree based algorithms. By default it is turned on for gradient boosting, where in general it makes training faster, but turned off for all other algorithms as it tends to slow down training when training deep trees.