Knowledge Discovery & Data Mining

— Classification: Decision Tree —

Instructor: Yong Zhuang

yong.zhuang@gvsu.edu

Recap: Supervised vs. Unsupervised Learning

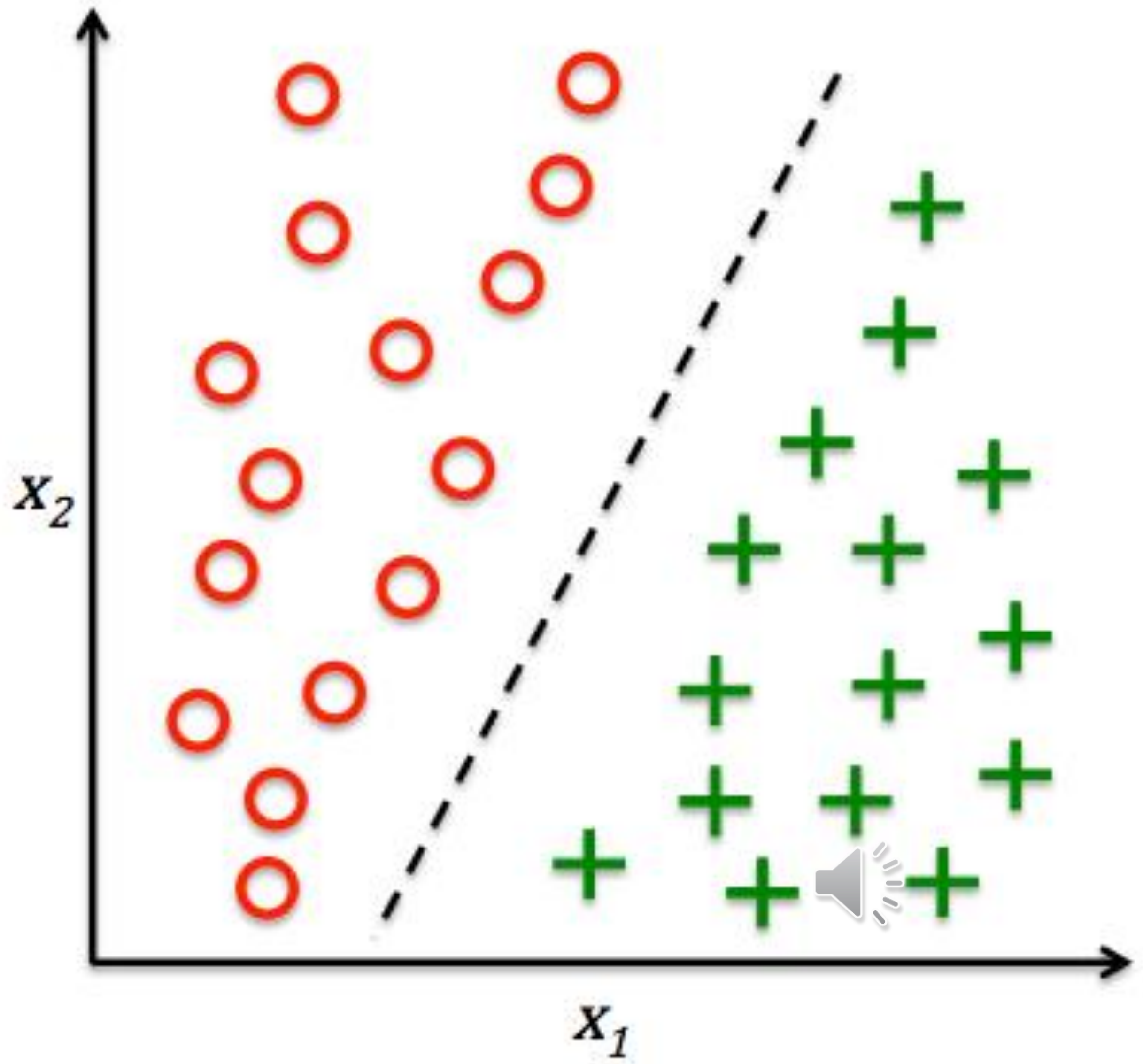
- Supervised Learning
  - Data: both the features, x, and a target, y, for each item in the dataset
  - Goal: ‘learn’ how to predict the target from the features, y = f(x)
  - Example: Regression and Classification
- Unsupervised Learning
  - Data: Only the features, x, for each item in the dataset
  - Goal: discover ‘interesting’ things about the dataset
  - Example: Clustering, Dimensionality reduction, Principal Component Analysis (PCA)

Outline

- Introduction to Classification
- Decision Tree
  - Decision Tree Algorithm
  - Attribute selection measures
    - Information gain
    - Gain ratio
    - Gini impurity
    - Other Attribute Selection Measures
- Regression tree
- Overfitting and Tree Pruning

What is classification?

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Regression
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical classification applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is



General approach to classification

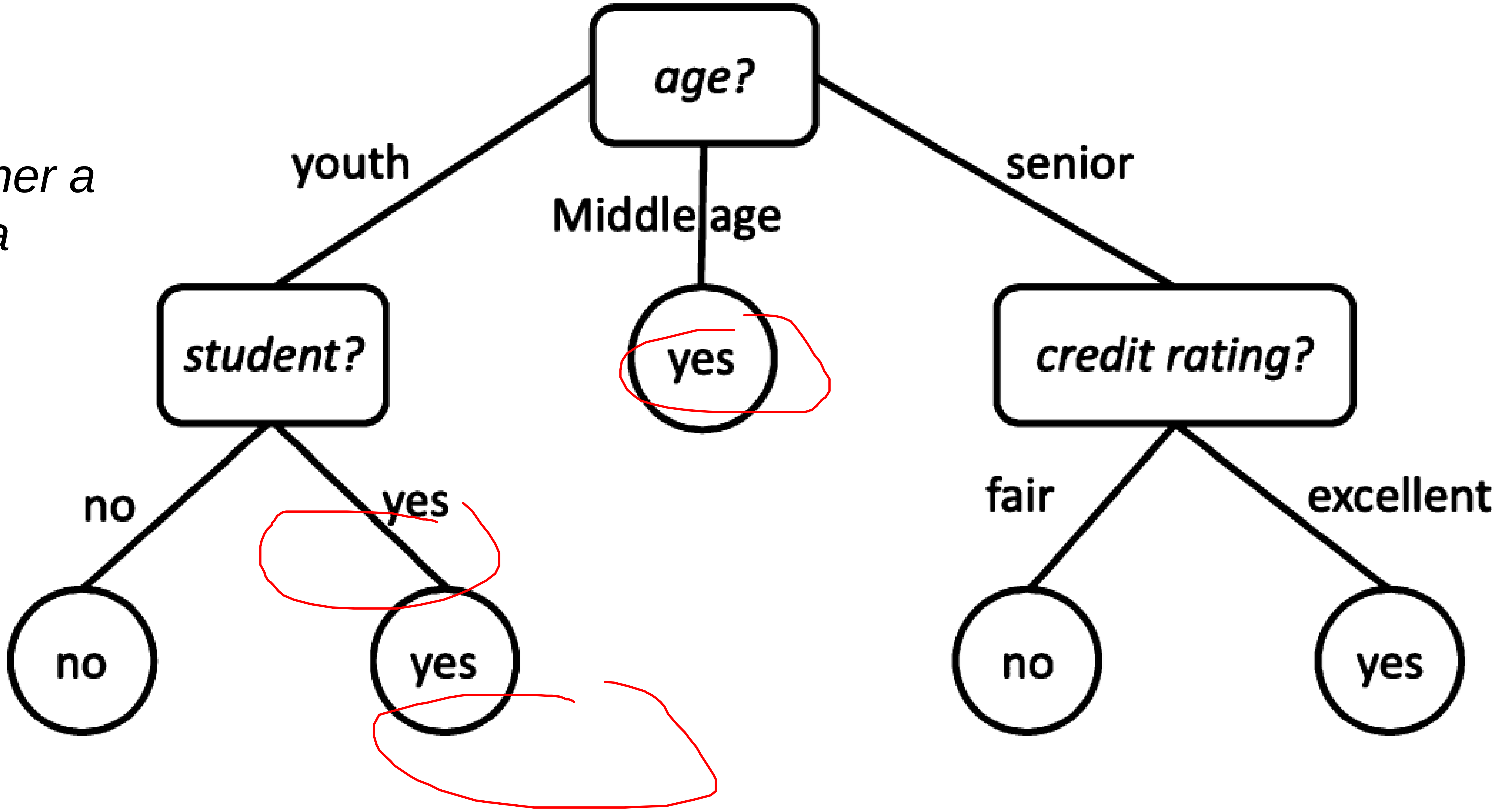
Data classification is a two-step process

- **Learning step:** (where a classification model is constructed)
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the **class label attribute**
  - The set of tuples used for model construction is **training set**
  - The model is represented as classification rules, tree-based models, or mathematical formula.
- **Classification step:** where the model is used to predict class labels for given data)
  - **Estimate accuracy** of the model
    - The known label of test sample is compared with the classified result from the model
    - **Accuracy** rate is the percentage of test set samples that are correctly classified by the model
    - **Test** set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to **classify new data**
- Note: If the test set is used to select models, it is called **validation (test) set**

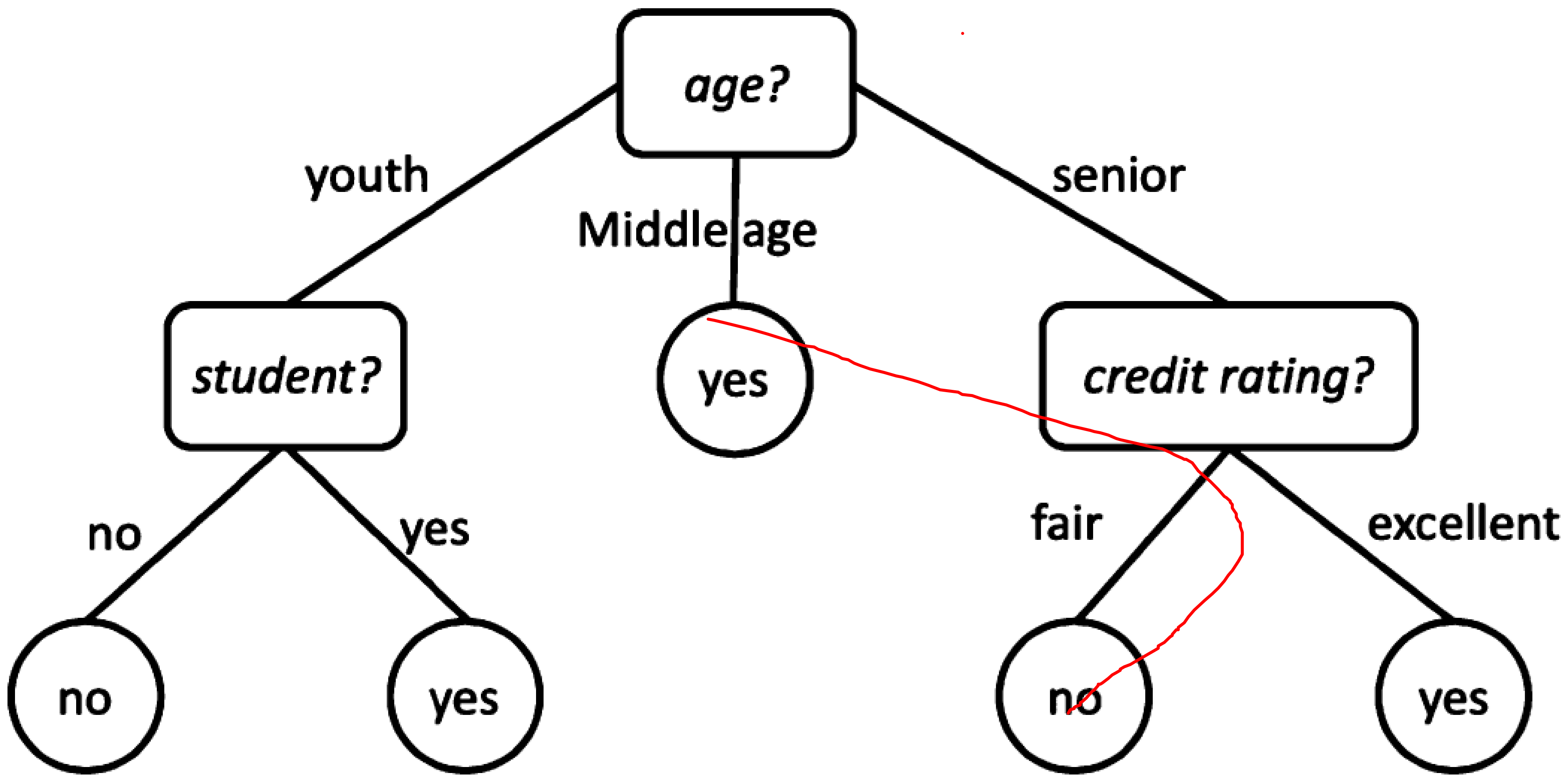
Decision Tree

Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each **internal node** (**non-leaf node**) denotes a test on an attribute, each branch represents an outcome of the test, and each **leaf node** (or **terminal node**) holds a class label. The topmost node in a tree is the **root node**.

A decision tree for the concept buys_computer indicates whether a customer is likely to purchase a computer.

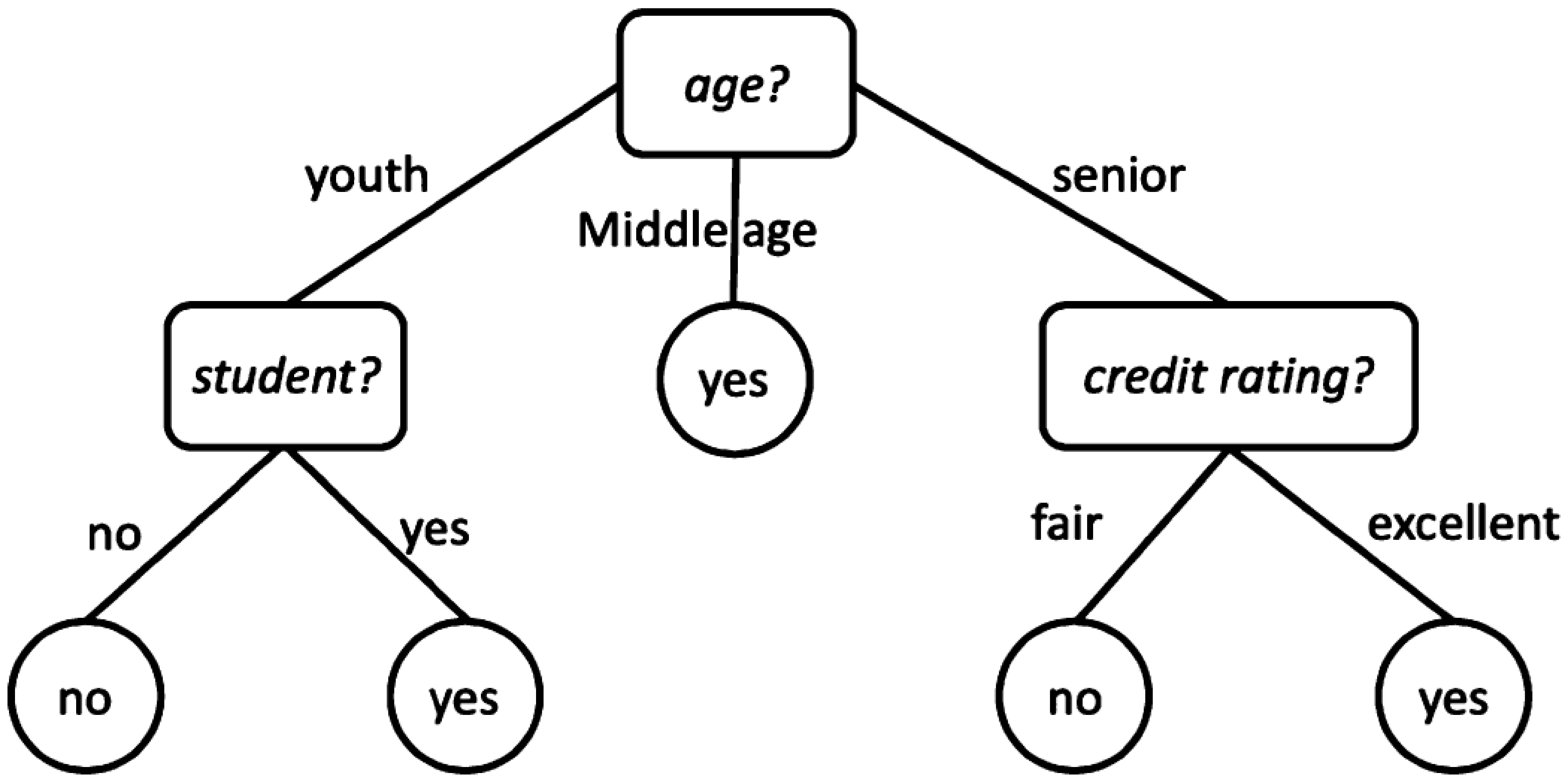


How are decision trees used for classification?



Given a tuple, X, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

Why are decision tree classifiers so popular?



- The construction of decision tree classifiers **does not require any domain knowledge** or parameter setting and therefore is appropriate for exploratory knowledge discovery.
- Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally **easy to understand**.
- The learning and classification steps of decision tree induction are **simple and fast**.
- In general, decision tree classifiers **have good accuracy**.
  - successful use may depend on the data at hand.
- Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

Evolution and Key Developments in Decision Tree Algorithms

- In the late 1970s and early 1980s, J. Ross Quinlan developed the **ID3** decision tree algorithm. It was built upon earlier work on concept learning by E. B. Hunt, J. Marin, and P. T. Stone.
- Quinlan later introduced **C4.5**, a successor to ID3. It is a benchmark for comparing new supervised learning algorithms.
- In 1984, L. Breiman, J. Friedman, R. Olshen, and C. Stone published "**Classification and Regression Trees (CART)**".
    - CART describes binary decision tree generation.
- ID3 and CART were developed independently but around the same time. Both ID3 and CART follow a similar approach for learning decision trees. Decision tree induction saw significant growth after these algorithms.
- ID3, C4.5, and CART use a greedy top-down approach. Most decision tree algorithms also follow a top-down method starting with a training set of tuples and class labels.

Algorithm

Initially, it is the complete set of training tuples and their associated class labels.

This procedure employs an attribute selection measure such as information gain or the Gini impurity. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini impurity, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3) return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5) return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and** multiway splits allowed **then** // not restricted to binary trees
(9) *attribute_list* $\leftarrow$ *attribute_list* $-$ *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
// partition the tuples and grow subtrees for each partition
(11) let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12) **if** D_j is empty **then**
(13) attach a leaf labeled with the majority class in D to node N;
(14) **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
endfor
(15) return N.

Algorithm

N Represents the training tuples in D.

The splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. A partition is pure if all the tuples in it belong to the same class. In other words, if we split up the tuples in D according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3)   return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5)   return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
    multiway splits allowed **then** // not restricted to binary trees
(9)   *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
    // partition the tuples and grow subtrees for each partition
(11)   let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12)   **if** D_j is empty **then**
(13)   attach a leaf labeled with the majority class in D to node N;
(14)   **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
    endfor
(15) return N.

Algorithm

Serves as a test at the node.

*A branch is grown from node N for each of the outcomes of the splitting criterion. The tuples in D are partitioned accordingly. There are **three possible scenarios**.*

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

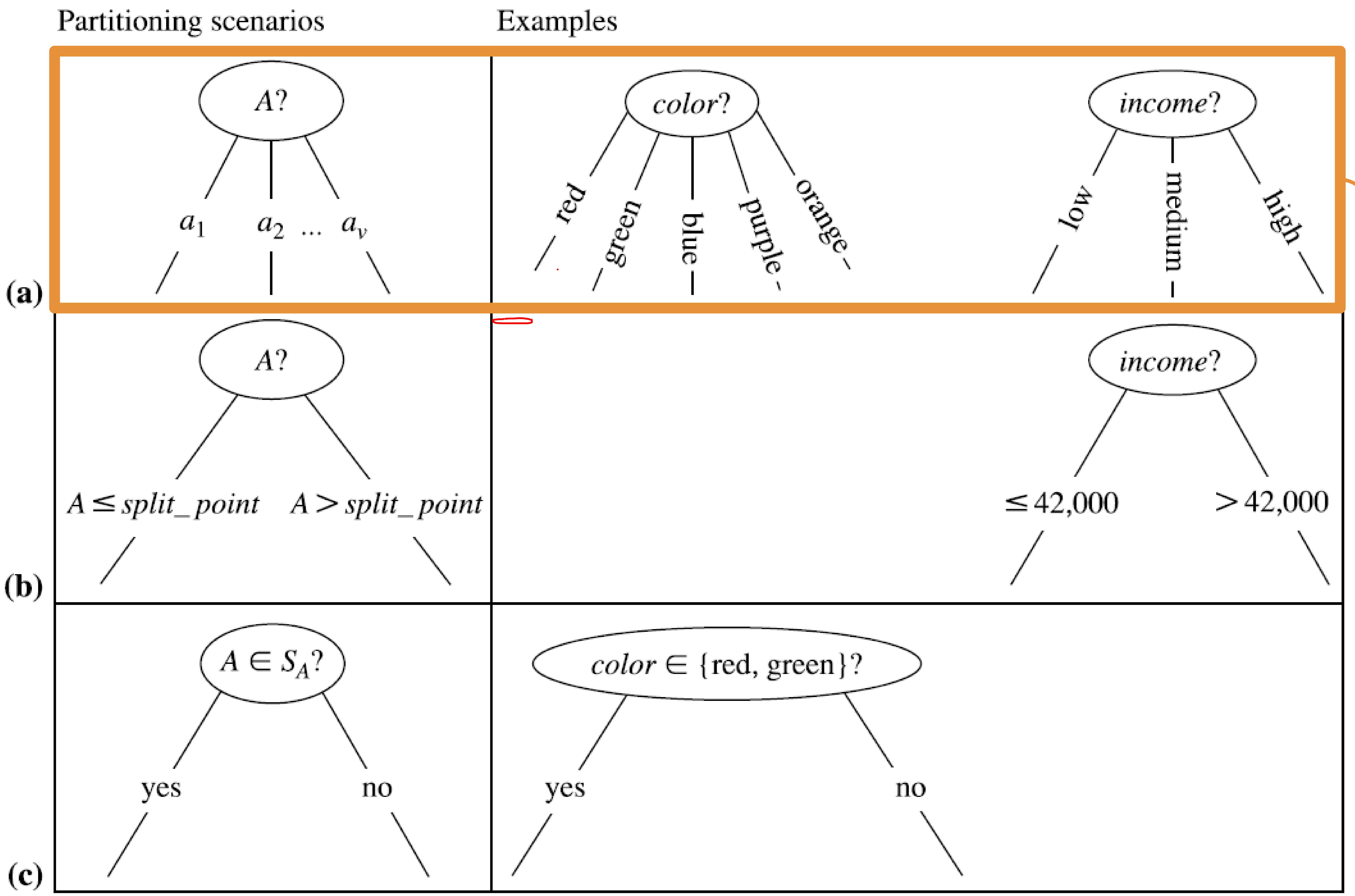
Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3)   return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5)   return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
    multiway splits allowed **then** // not restricted to binary trees
(9)   *attribute_list* ← *attribute_list* − *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
// partition the tuples and grow subtrees for each partition
(11)   let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12)   **if** D_j is empty **then**
(13)   attach a leaf labeled with the majority class in D to node N;
(14)   **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
endfor
(15) return N.

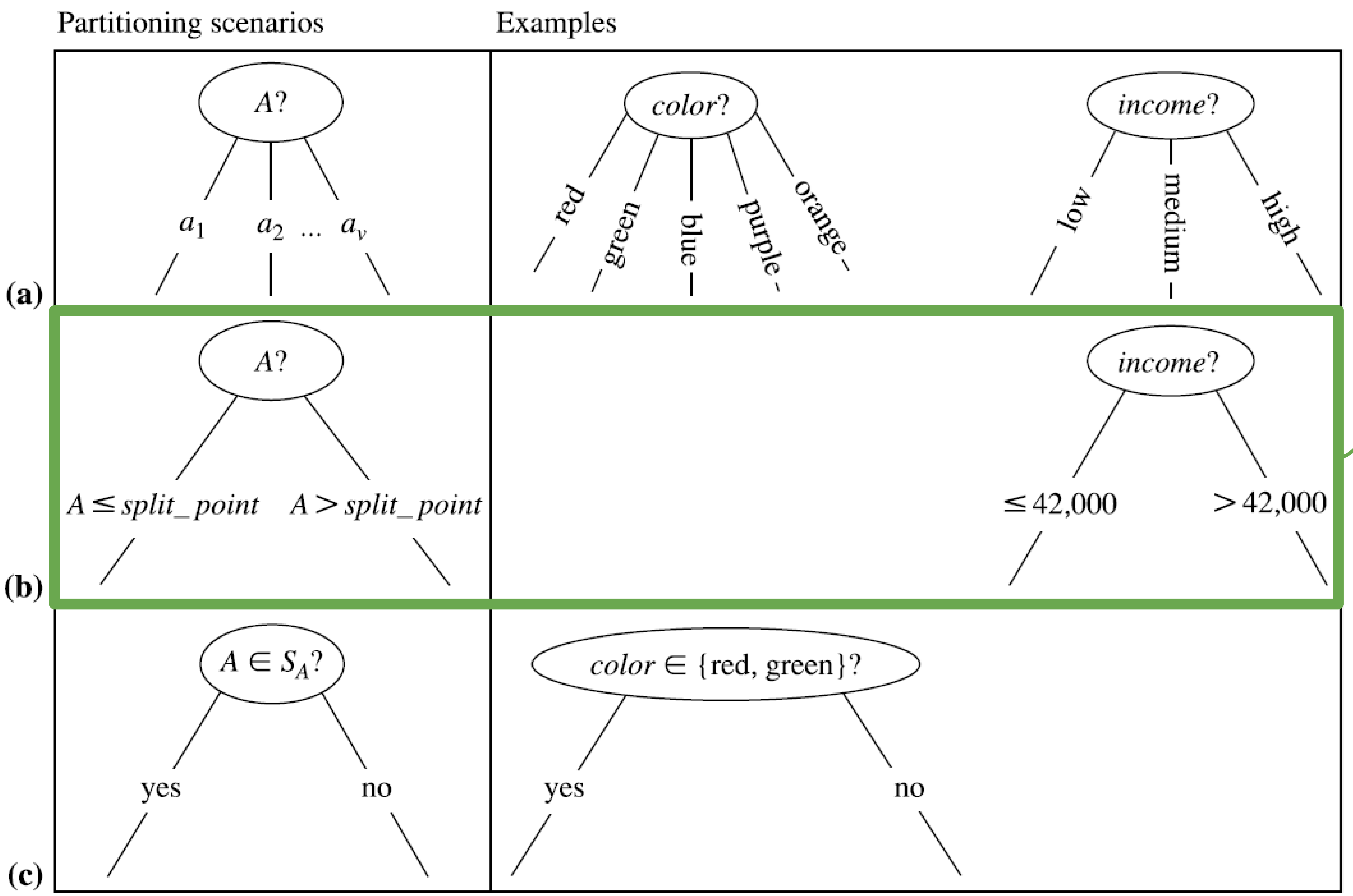
Three possible scenarios

Let A be the splitting attribute. A has v distinct values, $\{a_1, a_2, \ldots, a_v\}$, based on the training data.



A is discrete-valued: In this case, the outcomes of the test at node N directly correspond to the known values of A. A branch is created for each known value, aj, of A and labeled with that value. Partition Dj is the subset of class-labeled tuples in D having value aj of A. Because all the tuples in a given partition have the same value for A, A does not need to be considered in any future partitioning of the tuples. Therefore it is removed from attribute_list.

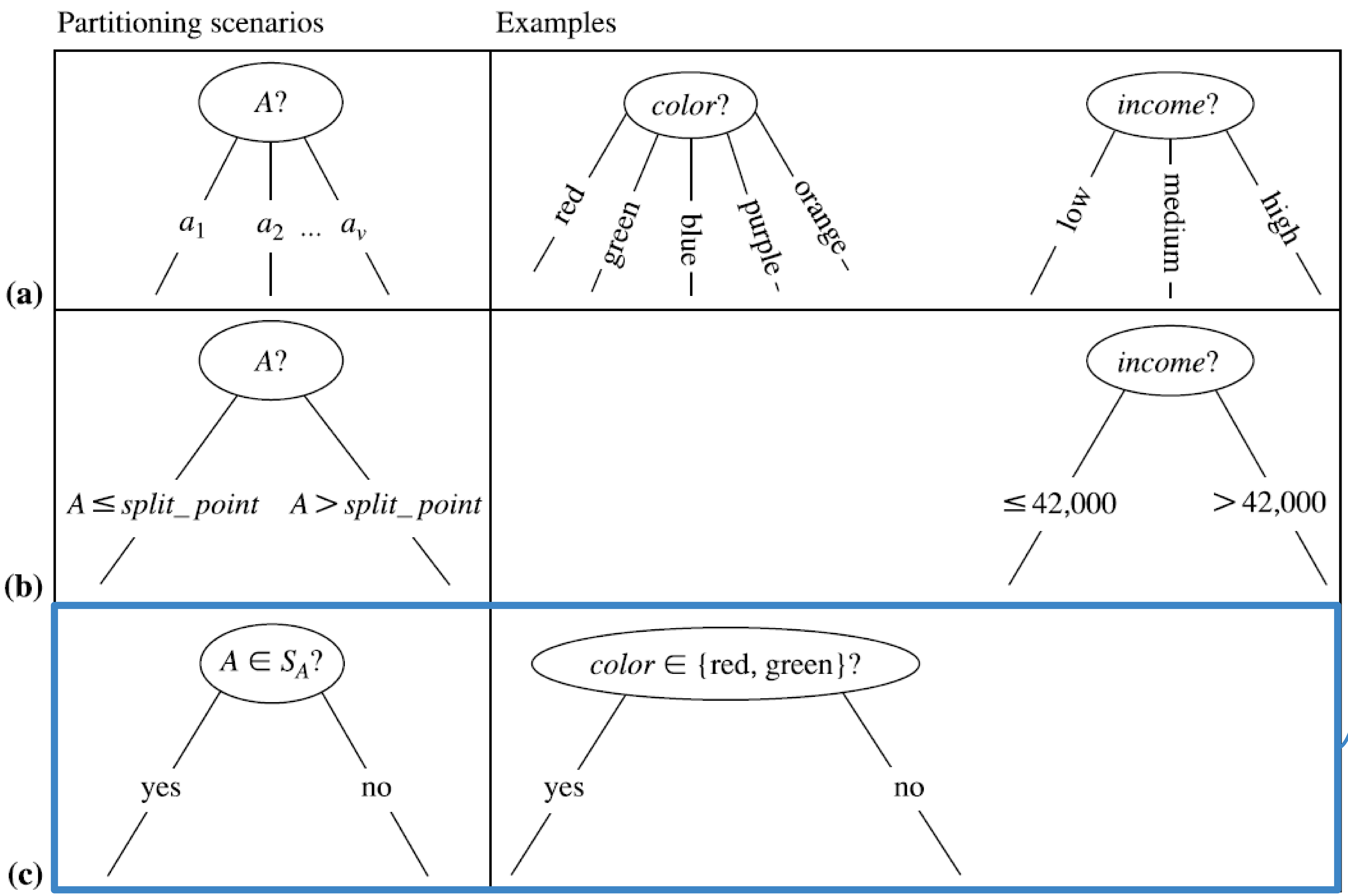
Three possible scenarios



A is continuous-valued: In this case, the test at node N has two possible outcomes, corresponding to the conditions A ≤ split_point and A>split_point, respectively, where split_point is the splitpoint returned by Attribute_selection_method as part of the splitting criterion. (In practice, the split-point, a, is often taken as the midpoint of two known adjacent values of A and therefore may not actually be a preexisting value of A from the training data.) Two branches are grown from N and labeled according to the previous outcomes. The tuples are partitioned such that D1 holds the subset of class-labeled tuples in D for which A ≤ split_point, while D2 holds the rest.

Three possible scenarios for partitioning

Let A be the splitting attribute. A has v distinct values, {a_1, a_2, . . . , a_v}, based on the training data.



A is discrete-valued and a binary tree must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node N is of the form "A ∈ SA?," where SA is the splitting subset for A, returned by Attribute_selection_method as part of the splitting criterion. It is a subset of the known values of A. If a given tuple has value aj of A, and if aj ∈ SA, then the test at node N is satisfied. Two branches are grown from N. By convention, the left branch out of N is labeled yes so that D1 corresponds to the subset of class-labeled tuples in D that satisfy the test. The right branch out of N is labeled no so that D2 corresponds to the subset of class-labeled tuples from D that do not satisfy the test.

Algorithm

Algorithm: Generate_decision_tree. Generate a decision tree from the training tuples of data partition, D.

Input:

- Data partition, D, which is a set of training tuples and their associated class labels;
- *attribute_list*, the set of candidate attributes;
- *Attribute_selection_method*, a procedure to determine the splitting criterion that "best" partitions the data tuples into individual classes. This criterion consists of a *splitting_attribute* and, possibly, either a *split-point* or *splitting subset*.

Output: A decision tree.

Method:

(1) create a node N;
(2) **if** tuples in D are all of the same class, C, **then**
(3)   return N as a leaf node labeled with the class C;
(4) **if** *attribute_list* is empty **then**
(5)   return N as a leaf node labeled with the majority class in D; // majority voting
(6) apply **Attribute_selection_method**(D, *attribute_list*) to **find** the "best" *splitting_criterion*;
(7) label node N with *splitting_criterion*;
(8) **if** *splitting_attribute* is discrete-valued **and**
    multiway splits allowed **then** // not restricted to binary trees
(9)   *attribute_list* $\leftarrow$ *attribute_list* $-$ *splitting_attribute*; // remove *splitting_attribute*
(10) **for each** outcome j of *splitting_criterion*
    // partition the tuples and grow subtrees for each partition
(11)   let D_j be the set of data tuples in D satisfying outcome j; // a partition
(12)   **if** D_j is empty **then**
(13)   attach a leaf labeled with the majority class in D to node N;
(14)   **else** attach the node returned by **Generate_decision_tree**(D_j, *attribute_list*) to node N;
endfor
(15) return N.

a recursive approach

All the tuples in partition D (represented at node N) belong to the same class

There are no remaining attributes on which the tuples may be further partitioned. In this case, majority voting is employed. This involves converting node N into a leaf and labeling it with the most common class in D. Alternatively, the class distribution of the node tuples may be stored.

There are no tuples for a given branch, that is, a partition Dj is empty. In this case, a leaf is created with the majority class in D.

Attribute selection measures

An **attribute selection measure** is a heuristic for selecting the splitting criterion that "best" separates a given data partition, D, of class-labeled training tuples into individual classes. If we were to split D into smaller partitions according to the outcomes of the splitting criterion, ideally, each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion is the one that most closely results in such a scenario. Attribute selection measures are also known as **splitting rules** because they determine how the tuples at a given node are to be split. It provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure is chosen as the splitting attribute for the given tuples.

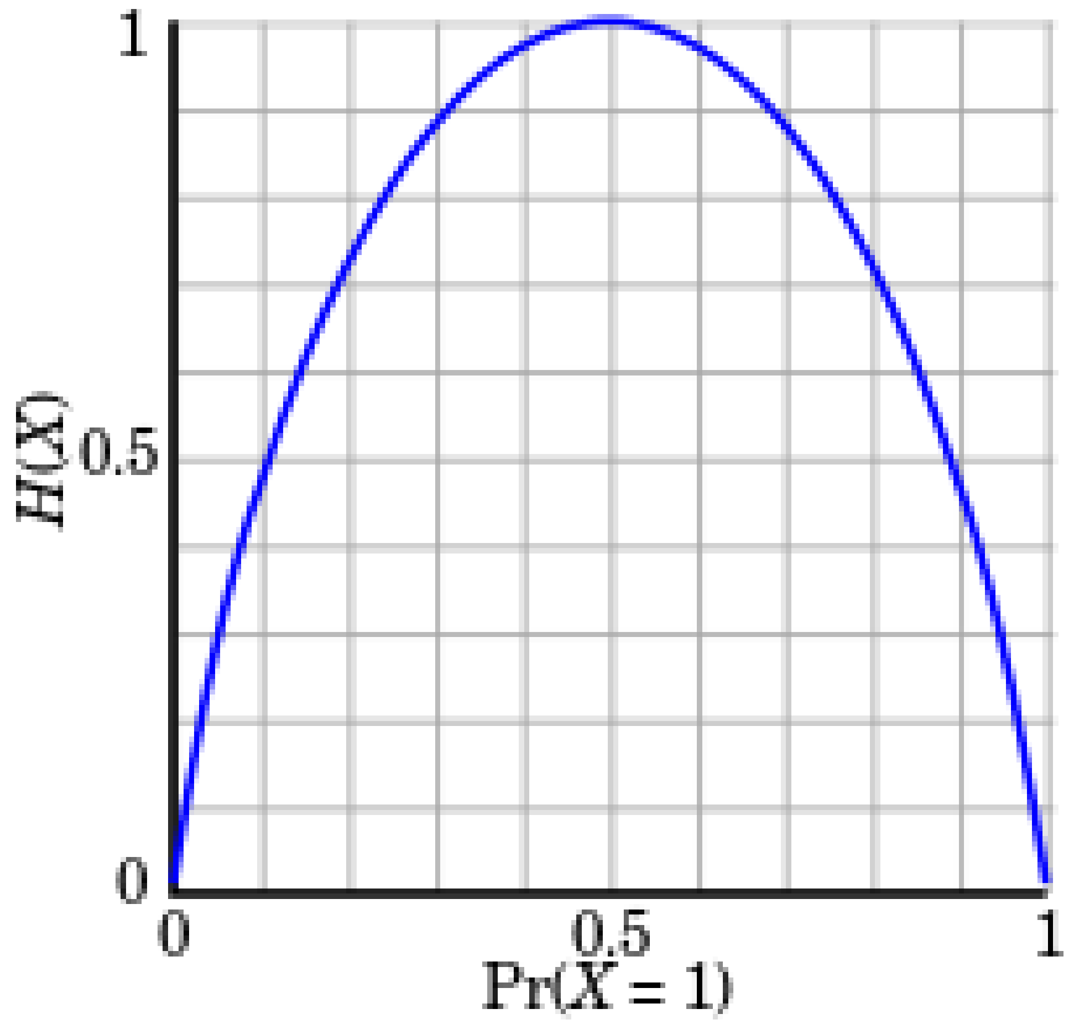
Brief Review of Entropy

- Entropy (Information theory):
  - A measure of uncertainty associated with a random variable.
  - Calculation: For a discrete random variable X taking n distinct values (x_1, ..., x_n),

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

Use the base 2 log function

  - Interpretation:
    - Higher entropy ➡ higher uncertainty,
    - lower entropy ➡ lower uncertainty.



Attribute selection measure: Information gain

ID3 uses **information gain** as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node N represent or hold the tuples of partition D. The attribute with the highest information gain is chosen as the splitting attribute for node N. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found. The expected information needed to classify a tuple in D is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

where p_i is the nonzero probability that an arbitrary tuple in D belongs to class C_i and is estimated by $|C_{i,D}| / |D|$.

Entropy of D

Information gain

Suppose we were to partition the tuples in D on some attribute A having *v* distinct values, $\{a_1, a_2, \ldots, a_v\}$, as observed from the training data. If A is discrete-valued, these values correspond directly to the *v* outcomes of a test on A. Attribute A can be used to split D into v partitions or subsets, $\{D_1, D_2, \ldots, D_v\}$, where D_j contains those tuples in D that have outcome aj of A. These partitions would correspond to the branches grown from node N. Ideally, we would like this partitioning to produce an exact classification of the tuples. we would like for each partition to be pure.

How much more information would we still need (after the partitioning) to arrive at an exact classification?

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

The term $\frac{|D_j|}{|D|}$ acts as the weight of the jth partition. $Info_A(D)$ is the expected information required to classify a tuple from D based on the partitioning by A. The smaller the expected information (still) required, the greater the purity of the partitions. $Info_A(D)$ is also known as the conditional entropy of D (conditioned on the attribute A).

Information gain

Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on A). That is,

$$Gain(A) = Info(D) - Info_A(D)$$

Gain(A) tells us how much would be gained by branching on A. It is the expected reduction in the information requirement caused by knowing the value of A. The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N.

Information gain

Example. The following table presents a training set, D, of class-labeled tuples randomly selected from the customer database of an electronics store.

buys_computer, has two distinct values (yes, no); two classes (m = 2).
C1 = yes, has 9 tuples.
C2 = no, has 5 tuples.

How to choose the (root) node N(splitting attribute) for the tuples in D?

Try Gain(age)

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Gain(A) = Info(D) - Info_A(D)$$

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Information gain

$$Gain(A) = Info(D) - Info_A(D)$$ = 0.940−0.694 = **0.246**

Example. The following table presents a training set, D, of class-labeled tuples randomly selected from the customer database of an electronics store.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$\boxed{Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940}$$

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$\begin{aligned} Info_{age}(D) = &\frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ &+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) \\ &+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Information gain

Example. The following table presents a training set, D, of class-labeled tuples randomly selected from the customer database of an electronics store.

- Gain(age)= **0.246**
- Gain(income)= **0.029**
- Gain(student)= **0.151**
- Gain(credit_rating)= **0.048**

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Information gain

If the attribute is a numeric attribute, how can we compute the information gain?

age?

youth

income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

middle_aged

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Leaf

senior

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

Information gain of numeric attribute

Suppose, we have an attribute A that is continuous-valued. We must determine the "best" split-point for A, where the split-point is a threshold on A

- We first sort the values of A in the increasing order. Typically, the midpoint between each pair of adjacent values is considered as a possible split-point. Therefore, given v values of A, (v −1) possible splits are evaluated. For example, the midpoint between the values ai and ai+1 of A is: $(a_i + a_{i+1})/2$
- For each possible split-point for A, we evaluate $Info_A(D)$, where the number of partitions is two. The point with the minimum expected information requirement for A is selected as the split_point for A.
  - D1 is the set of tuples in D satisfying A ≤ split_point,
  - D2 is the set of tuples in D satisfying A>split_point.

Weaknesses of information gain

The information gain measure prefers to select attributes having a large number of values, which is biased toward tests with many outcomes.

For example, consider an attribute that acts as a unique identifier, such as product_ID. A split on product_ID would result in a large number of partitions (as many as there are values), each one containing just one tuple. Because each partition is pure, the information required to classify data set D based on this partitioning would be Infoproduct_ID(D) = 0. Therefore the information gained by partitioning on this attribute is maximal. Clearly, such a partitioning is useless for classification.

How to solve this problem?

Gain ratio

C4.5, a successor of ID3, uses an extension to information gain known as gain ratio. It applies a kind of normalization to information gain using a “**split information**” value defined similarly to Info(D) as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$\Downarrow$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Gain ratio

Example. Computation of gain ratio for the attribute income.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Gain(income)= **0.029**

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Gain ratio

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right)$$
$$= 1.557.$$

Example. Computation of gain ratio for the attribute income.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

Gain(income)= **0.029**

GainRatio(income)=

0.029/1.557 = 0.019

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Gini impurity

The **Gini impurity**, often referred to as **Gini index(or Gini in short)**, is a measure of impurity or disorder. It is commonly used in decision tree algorithms like CART (Classification and Regression Trees) to determine which feature to split on at any given step in the tree. Given a set D (a data partition or a set of training tuples), the Gini impurity is calculated as:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

Where D is the training set, m is the number of unique classes, and p_i is the proportion of instances of class i in D.

If all elements in the set are of the same class (i.e., the set is pure), the Gini impurity will be 0. If the elements are distributed uniformly across different classes, the Gini impurity will be maximized. In the context of decision trees, a lower Gini impurity value indicates that a node is more "pure".

Gini impurity

Let's examine the scenario where attribute A possesses v distinct discrete values, represented as $\{a_1, a_2, \ldots, a_v\}$, present in dataset D. To identify the optimal binary split for A, we analyze all potential subsets formed using A's known values. Each subset, S_A, serves as a binary criterion for A and can be phrased as "Does A belong to S_A?". This criterion is met if A's value for a given tuple falls within S_A. With v potential values for A, there are 2^v possible subsets. Taking 'income' as an example with three possible categories: {low, medium, high}, we derive subsets like {low, medium}, {medium, high}, {low}, and so on. However, we omit the full set, {low, medium, high}, and the null set as they don't inherently define a split. Consequently, we have $(2^v - 2)/2$ viable ways to bifurcate dataset D based on a binary division of A.

Gini impurity

When considering a binary split, the goal is to compute a weighted sum of the impurity of each resulting partition. Specifically:

- Let's assume a binary split on attribute A divides dataset D into two subsets D_1 and D_2.
- The Gini impurity of D after this split is computed as:

$$Gini_A(D) = \frac{|D_1|}{|D|} \times Gini(D_1) + \frac{|D_2|}{|D|} \times Gini(D_2)$$

For discrete-valued attributes:

- Every possible binary split is evaluated.
- The split that minimizes the Gini impurity for that attribute is chosen as the best split.

Gini impurity

Continuous-valued Attributes and Split-Points

For continuous-valued attributes, the task is to evaluate every possible split-point. This approach mirrors the method previously outlined for information gain. Specifically:

- We sort the values of the attribute.
- For each pair of adjacent values, the midpoint serves as a potential split-point.

The optimal split-point is the one which minimizes the Gini impurity for the continuous-valued attribute in question. To elucidate:

- For a potential split-point of attribute A, subset D_1 comprises tuples in D where $A \leq \mathrm{split\_point}$.
- Conversely, D_2 contains tuples in D where $A > \mathrm{split\_point}$.

Gini impurity

Reduction in Impurity

The impurity reduction achieved by a binary split on either a discrete or continuous-valued attribute A is expressed as:

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Here, the goal is to:

- Maximize the impurity reduction or, equivalently,
- Minimize the Gini impurity.

The attribute that meets this criterion becomes the splitting attribute. Coupled with this attribute is either:

- Its splitting subset (for discrete-valued attributes) or,
- Split-point (for continuous-valued attributes).

This combination essentially defines the splitting criterion.

Gini impurity

Example. Consider the training data in the following table, denoted as D. In this data, nine tuples are classified under the category `buys_computer = yes`, while the remaining five are categorized as `buys_computer = no`. We initiate by creating a root node, N, representing the tuples in D. How to compute N using Gini impurity?

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

$$Gini_A(D) = \frac{|D_1|}{|D|} \times Gini(D_1) + \frac{|D_2|}{|D|} \times Gini(D_2)$$

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

Gini impurity

Example. Consider the training data in the following table, denoted as D. In this data, nine tuples are classified under the category `buys_computer = yes`, while the remaining five are categorized as `buys_computer = no`. We initiate by creating a root node, N, representing the tuples in D. How to compute N using Gini impurity?

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

$$Gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2,$$

Gini impurity

To identify the optimal splitting criterion for D, we evaluate the Gini impurity for each attribute.

Example. Consider the training data in the following table, denoted as D. In this data, nine tuples are classified under the category `buys_computer = yes`, while the remaining five are categorized as `buys_computer = no`. We initiate by creating a root node, N, representing the tuples in D. How to compute N using Gini impurity?

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Income Attribute:

- Subset {low, medium}:

$$Gini_{income\in\{low,medium\}}(D)$$
$$=\frac{10}{14}\left(1-\left(\frac{7}{10}\right)^2-\left(\frac{3}{10}\right)^2\right)+\frac{4}{14}\left(1-\left(\frac{2}{4}\right)^2-\left(\frac{2}{4}\right)^2\right)=0.443$$

- Other subsets:
  - $Gini_{income\in\{low,high\}}(D)$ = 0.458
  - $Gini_{income\in\{medium,high\}}(D)$ = 0.450

$$Gini_A(D)=\frac{|D_1|}{|D|}\times Gini(D_1)+\frac{|D_2|}{|D|}\times Gini(D_2)$$

Gini impurity

To identify the optimal splitting criterion for D, we evaluate the Gini impurity for each attribute.

Example. Consider the training data in the following table, denoted as D. In this data, nine tuples are classified under the category `buys_computer = yes`, while the remaining five are categorized as `buys_computer = no`. We initiate by creating a root node, N, representing the tuples in D. How to compute N using Gini impurity?

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- `income`: Best split is {low, medium} or {high} with a Gini impurity of 0.458
- `age`: Best split is {youth, senior} or {middle_aged} with a Gini impurity of 0.357. **win!!!!!!**
- `student`: Binary attributes with Gini impurity values of 0.367
- `credit_rating`: Binary attributes with Gini impurity values of 0.429

$$Gini_A(D) = \frac{|D_1|}{|D|} \times Gini(D_1) + \frac{|D_2|}{|D|} \times Gini(D_2)$$

Gini impurity vs. Information gain

Criteria	Information Gain	Gini Index
Purpose	Measures the impurity based on the average amount of information needed to identify the class label.	Quantifies the likelihood of mis-classification of a randomly chosen tuple based on class label distribution.
Theory	Information theory.	Based on mis-classification.
Split	Allows multiway split.	Always used for binary split.
Efficiency	Involves logarithm computation, making it slightly less efficient.	More computationally efficient
Outcome	Both measures often lead to very similar decision trees.	

- **Information Gain**: Tends to be biased towards attributes with many values.
- **Gain Ratio**: Often prefers splits where one partition is significantly smaller than the others.
- **Gini Index**:
  - Shows a bias towards multivalued attributes.
  - Encounters challenges when the number of classes is large.
  - Generally favors tests that yield equal-sized partitions and maintain purity in both partitions.

Regression tree

Regression tree is used to predict the continuous output value.

- Similar to a decision tree in that it also partitions the entire attribute space into multiple subregions, each corresponding to a leaf node.
- The main difference:
  - A leaf node holds a continuous value instead of a categorical value (i.e., class label) in a decision tree. The continuous value of a leaf node is learned during the training phase, which is set as the average output value of all training tuples fallen in the corresponding subregions. “Classification and Regression Trees” (CART) uses residual sum of squares (RSS) as the objective function to train a regression tree.

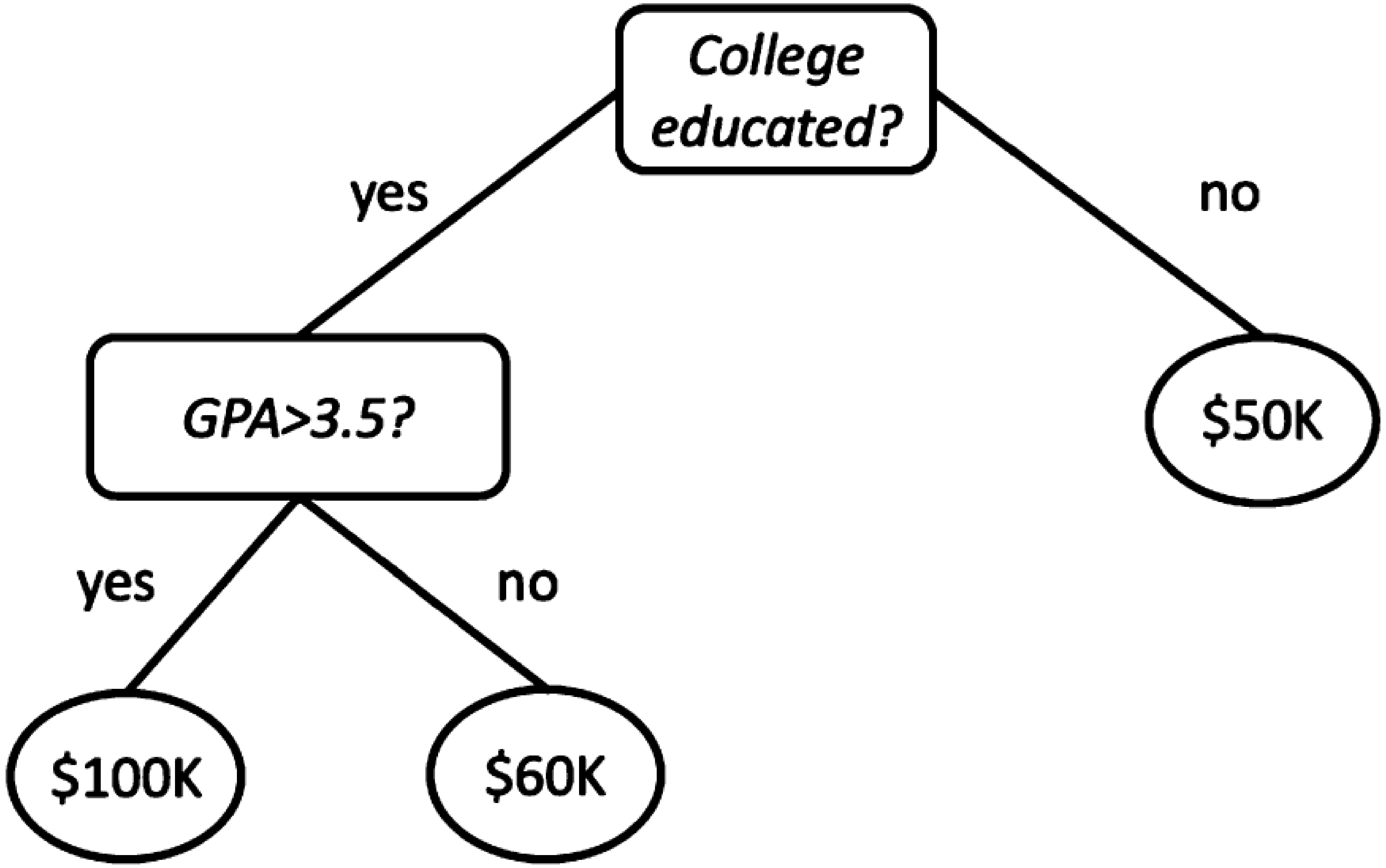
$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

where y_i is the actual output value of the ith training tuple, and $\hat{y}_i$ is the predicted output by the regression tree.

Regression tree

Example: A regression tree for predicting the average yearly income based on an individual's education.

$50K is the average yearly income of all training individuals who do not have a college degree; $60K is the average yearly income of all training individuals who have a college degree with a GPA less than or equal to 3.5; and $100K is the average yearly income of all training individuals who have a college degree with a GPA higher than 3.5. The leaf node values ($50K, $60K, and $100K) are used to predict the yearly income of any test individual who falls into the corresponding leaf nodes. categorical



Find the best split point use RSS

Example: Imagine a regression tree node containing five training tuples. Each tuple has a true output value denoted as y_i and a continuous attribute labeled as x_i where i ranges from 1 to 5. Our objective is to determine the optimal split point for attribute x_i to bifurcate the node into two distinct leaf nodes.

$$\text{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

attribute x_i	1	2	3	4	5
output y_i	10	12	8	20	22

Given five training tuples at a regression tree node, each with a true output value y_i and a continuous attribute $x_i (i = 1, ..., 5)$. We want to find the best split point for attribute x_i to split the tree node into two nodes (left node and right node).

Find the best split point use RSS

Example: Imagine a regression tree node containing five training tuples. Each tuple has a true output value denoted as y_i and a continuous attribute labeled as x_i where i ranges from 1 to 5. Our objective is to determine the optimal split point for attribute x_i to bifurcate the node into two distinct leaf nodes.

$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

Candidate split points: **[1.5, 2.5, 3.5, 4.5]**

considering the split point $x_i = 1.5$:

$$y_l = y_1 = 10$$

$$y_r = \frac{(y_2+y_3+y_4+y_5)}{4} = \frac{(12+8+20+22)}{4} = 15.5$$

$$RSS = (y_1 - y_l)^2 + (y_2 - y_r)^2 + (y_3 - y_r)^2 + (y_4 - y_r)^2 + (y_5 - y_r)^2 = 131$$

attribute x_i	1	2	3	4	5
output y_i	10	12	8	20	22

Given five training tuples at a regression tree node, each with a true output value y_i and a continuous attribute $x_i (i = 1, \ldots, 5)$. We want to find the best split point for attribute x_i to split the tree node into two nodes (left node and right node).

Find the best split point use RSS

Example: Imagine a regression tree node containing five training tuples. Each tuple has a true output value denoted as y_i and a continuous attribute labeled as x_i where i ranges from 1 to 5. Our objective is to determine the optimal split point for attribute x_i to bifurcate the node into two distinct leaf nodes.

Candidate split points: **[1.5, 2.5, 3.5, 4.5]**

$$\text{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

candidate split point x_i	1.5	2.5	3.5	4.5
predicted value of left leaf node y_l	10	11	10	12.5
predicted value of right leaf node y_r	15.5	16.7	21	22
RSS	131	116.67	10	83

attribute x_i	1	2	3	4	5
output y_i	10	12	8	20	22

Given five training tuples at a regression tree node, each with a true output value y_i and a continuous attribute x_i $(i = 1, \ldots, 5)$. We want to find the best split point for attribute x_i to split the tree node into two nodes (left node and right node).

Other Attribute Selection Measures

- **CHAID:** A widely used decision tree algorithm that employs the χ^2 test for independence.
- **C-SEP:** Outperforms Information Gain and Gini Index under specific scenarios.
- **G-statistic:** Offers an approximation closely aligned with the χ^2 distribution.
- **MDL (Minimal Description Length) Principle:** Operates on the belief that the simplest solution is often the best. It determines the optimal tree based on the one requiring the fewest bits for both:
  - Encoding the tree itself.
  - Encoding exceptions to the tree.
- **Multivariate Splits:** This involves partitioning based on combinations of multiple variables.
  - CART: Discovers multivariate splits using a linear combination of attributes.

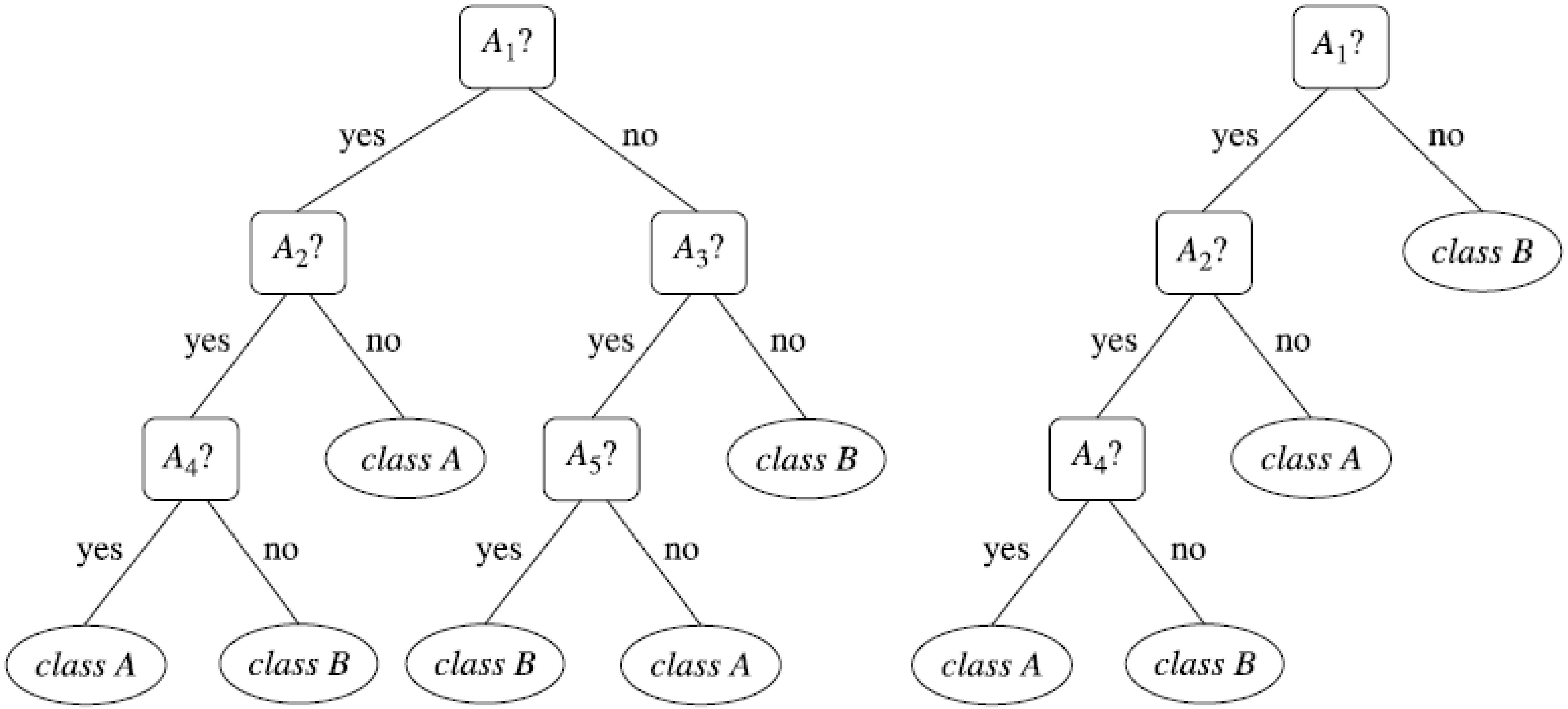
When considering which attribute selection measure dominates, it is important to note that most attribute selection measures provide reliable results. However, none are clearly better than the others.

Overfitting and Tree Pruning

Decision trees, when constructed, can sometimes become overly complex, capturing anomalies in the training data that result from noise or outliers. Such complexity can adversely affect the accuracy of predictions for new, unseen samples (overfitting). **Tree pruning** addresses this overfitting issue by eliminating branches that don't contribute significant power in predicting the outcome.

Benefits of Pruned Trees:

- Simplicity: They are smaller, simpler and easier to understand.
- Speed: Faster to execute due to reduced size.
- Accuracy: Generally outperform unpruned trees when classifying previously unseen data (new data).



How Does Tree Pruning Work?

Two approaches:

- **Prepruning**: The construction of the tree is halted early. This can be done by setting criteria like a minimum information gain or a specific level of statistical significance. If a potential split in the tree doesn't meet these criteria, the tree's growth is stopped at that point, turning the node into a leaf. This leaf will then either represent the most frequent class among the data at that node or show the probability distribution of the classes.
  - Difficult to choose an appropriate threshold:
  - High thresholds => oversimplified trees, low thresholds => very little simplification.
- **Postpruning**: Trimming subtrees from a fully developed tree. In this process, a subtree rooted at a particular node is removed, and the node is transformed into a **leaf**. This new leaf is then labeled with the dominant class label from the pruned subtree. **more common approach**

Summary

- Introduction to Classification
- Decision Tree
  - Decision Tree Algorithm
  - Attribute selection measures
    - Information gain
    - Gain ratio
    - Gini impurity
    - Other Attribute Selection Measures
- Regression tree
- Overfitting and Tree Pruning