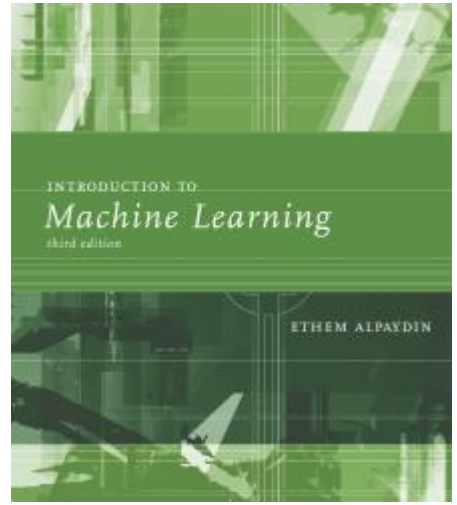


Decision Trees

Lecture notes by Ethem Alpaydın
Introduction to Machine Learning (Boğaziçi Üniversitesi)

Data Mining, Concepts and Techniques
Jiawei Han, Micheline Kamber, and Jian Pei

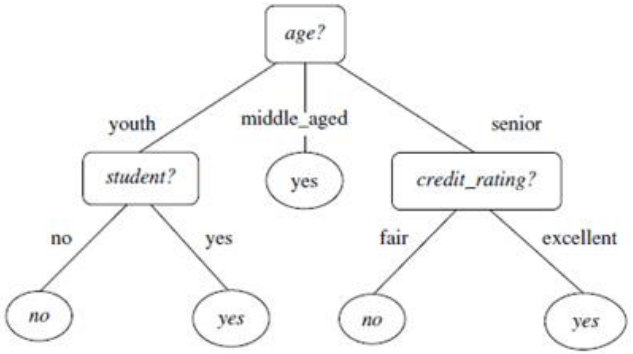
Decision Trees

- **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.

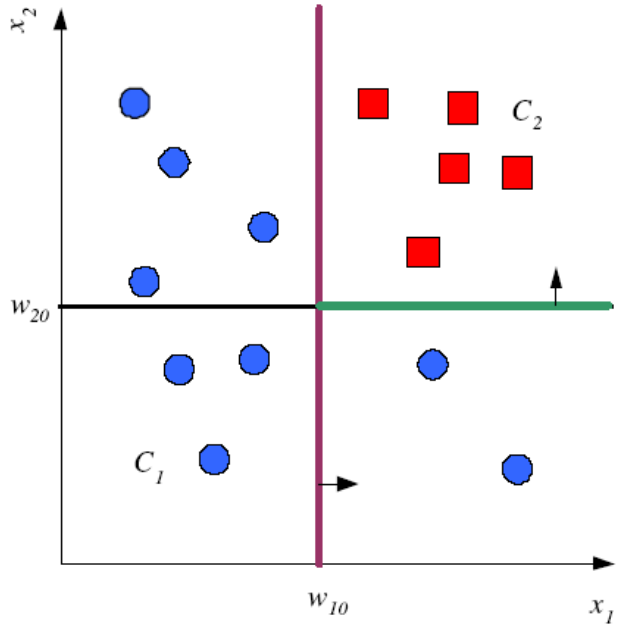
Decision Trees

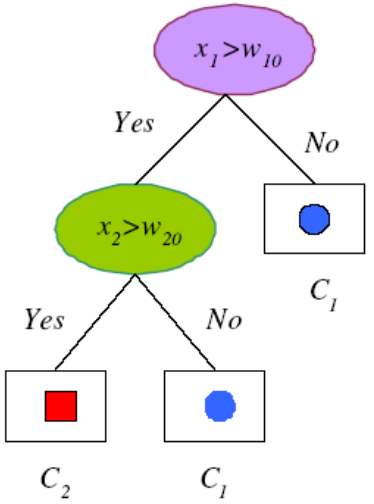
- The tree can be converted to a set of IF-THEN rules that are easily understandable.
- For this reason, decision trees are very popular and sometimes preferred over more accurate but less interpretable methods.
- Decision tree classifier have good accuracy.

Decision Trees



A decision tree for the concept *buys_computer*, indicating whether an *AllElectronics* customer is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = *yes* or *buys_computer* = *no*).





Tree Learning Algorithms

- Tree learning algorithms are greedy and, at each step, starting at the root with the complete training data, we look for the best split.
- This splits the training data into two or *n*, depending on whether the chosen attribute is numeric or discrete.
- We then continue splitting recursively with the corresponding subset until we do not need to split anymore, at which point a leaf node is created and labeled.

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

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 Dj be the set of data tuples in D satisfying outcome j; // a partition
(12)     if Dj 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(Dj, attribute_list) to node N;
     endfor
(15) return N;
```

Basic algorithm for inducing a decision tree from training tuples.

A basic decision tree algorithm

- The algorithm is called with three parameters: *D*, *attribute list*, and *Attribute_selection_method*. We refer to *D* as a data partition.
- The tree starts as a single node, *N*, representing the training tuples in *D* (step 1).
- If the tuples in *D* are all of the same class, then node *N* becomes a leaf and is labeled with that class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions.

A basic decision tree algorithm

- Otherwise, the algorithm calls *Attribute_selection_method* to determine the **splitting criterion**.
  - The splitting criterion tells us which attribute to test at node *N* by determining the “best” way to separate or partition the tuples in *D* into individual classes (step 6).
  - It also tells us which branches to grow from node *N* with respect to the outcomes of the chosen test.
  - The splitting criterion is determined so that, ideally, the result in partitions at each branch are as “pure” as possible.
  - A partition is **pure** if all the tuples in it belong to the same class.

A basic decision tree algorithm

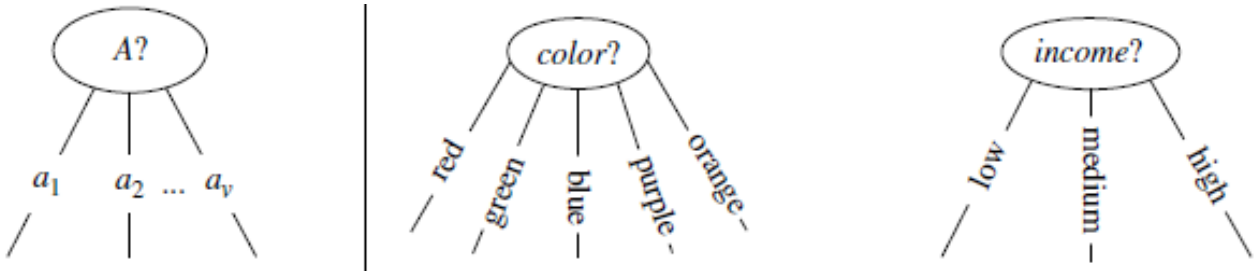
- The node *N* is labeled with the splitting criterion, which serves as a test at the node (step 7).
  - A branch is grown from node *N* for each of the outcomes of the splitting criterion.
  - The tuples in *D* are partitioned accordingly (steps 10 to 11).
  - There are three possible scenarios. Let *A* be the splitting attribute.
    - *A is discrete-valued*
    - *A is continuous-valued*
    - *A is discrete-valued* and a *binary tree* must be produced

A basic decision tree algorithm

A is discrete-valued:

In this case, the outcomes of the test at node *N* correspond directly to the known values of *A*.

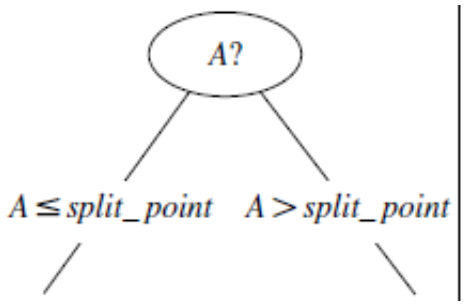
A branch is created for each known value a_j, of *A* and labeled with that value.

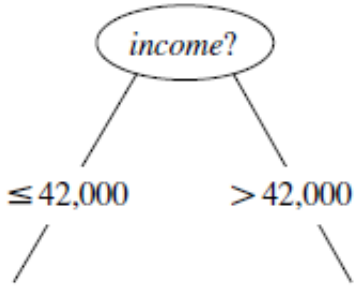


A basic decision tree algorithm

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 split-point returned by *Attribute_selection_method* as part of the splitting criterion

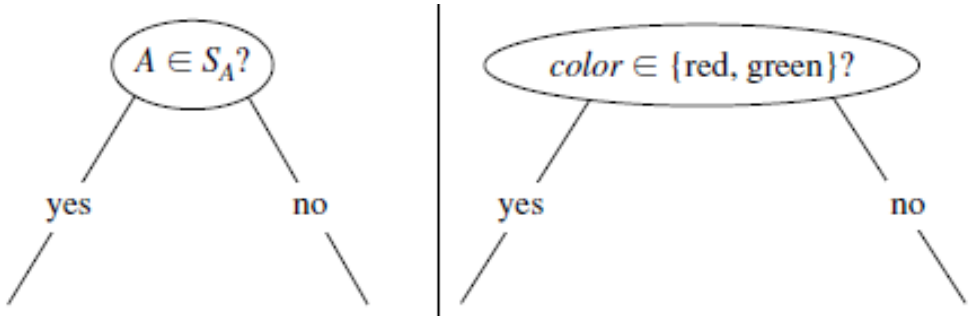




A basic decision tree algorithm

A is discrete-valued and a *binary tree* must be produced

The test at node *N* is of the form "$A \in S_A$?" where S_A is the splitting subset for *A*, returned by *Attribute_selection_method* as part of the splitting criterion.



A basic decision tree algorithm

- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, D_j , of *D.*

A basic decision tree algorithm

- The recursive partitioning stops only when any one of the following terminating conditions is true:
  1. All the tuples in partition *D* (represented at node *N*) belong to the same class (steps 2 and 3).
  2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node *N* into a leaf and labeling it with the most common class in *D*.
  3. There are no tuples for a given branch, that is, a partition D_j is empty (step 12). In this case, a leaf is created with the majority class in *D* (step 13).

A basic decision tree algorithm

- The resulting decision tree is returned

Attribute Selection Measure

- 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.
- Three popular attribute selection measures:
  - Information Gain (Entropy)
  - Gain Ratio
  - Gini Index

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Let node *N* represent or hold the tuples of partition *D*.
- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

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

- *Info(D)* is just the average amount of information needed to identify the class label of a tuple in D.

Attribute Selection Measure: Information Gain (ID3/C4.5)

- 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.
- 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 a_j of A. These partitions would correspond to the branches grown from node N.

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information needed (after using A to split D into v partitions) to classify D:

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

- $Info_A(D)$ 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.

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information gain is defined as the difference between the original information requirement and the new requirement
- Information gained by branching on attribute A

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

- In other words, 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.

Example

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

Example

$$Info(D) = -\sum_i p_i log p_i = -\frac{9}{14} log \frac{9}{14} - \frac{5}{14} log \frac{5}{14} = 0.940$$

Example

$Info(D) = -\sum_i p_i log p_i$

youth = 2Y, 3N; middle = 4Y,0N;senior=3Y,2N

$$Info_{Age}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} . Info(D_j)$$

$$= \frac{5}{14}\left(-\frac{2}{5}log\frac{2}{5} - \frac{3}{5}log\frac{3}{5}\right) + \frac{4}{14}\left(-\frac{4}{4}log\frac{4}{4} - \frac{0}{4}log\frac{0}{4}\right)$$

$$+ \frac{5}{14}\left(-\frac{3}{5}log\frac{3}{5} - \frac{2}{5}log\frac{2}{5}\right) = 0.694$$

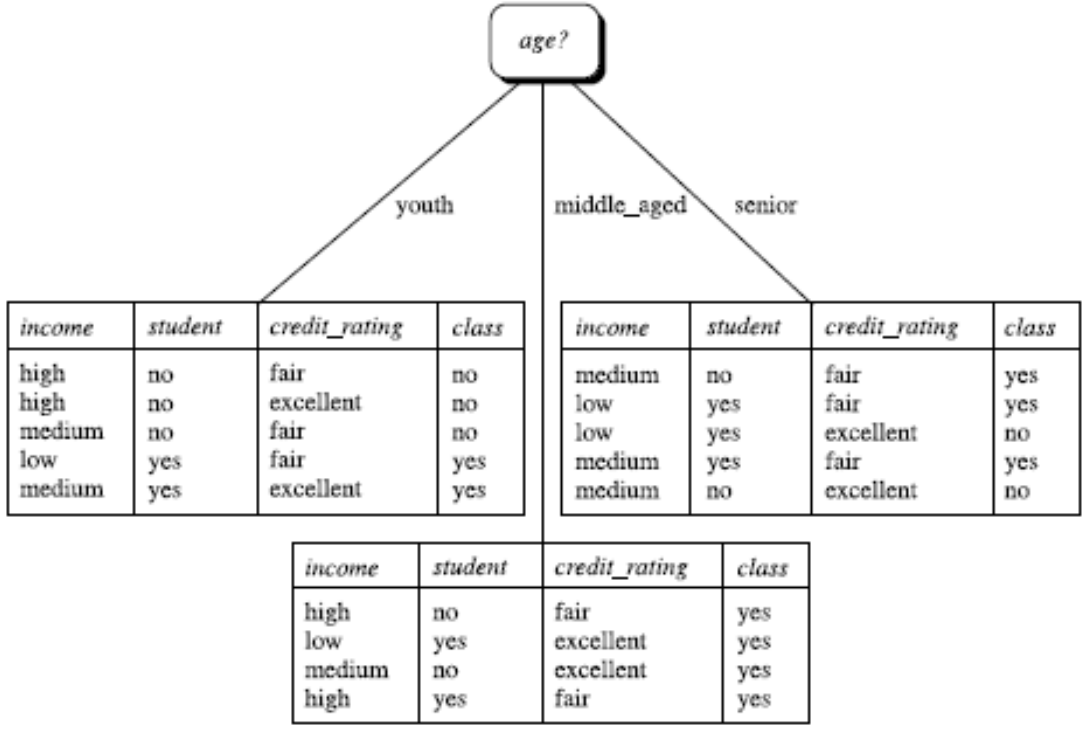
Example

$$Gain(Age) = Info(D) - Info_{Age}(D) = 0.940 - 0.694 = 0.246$$

Example

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

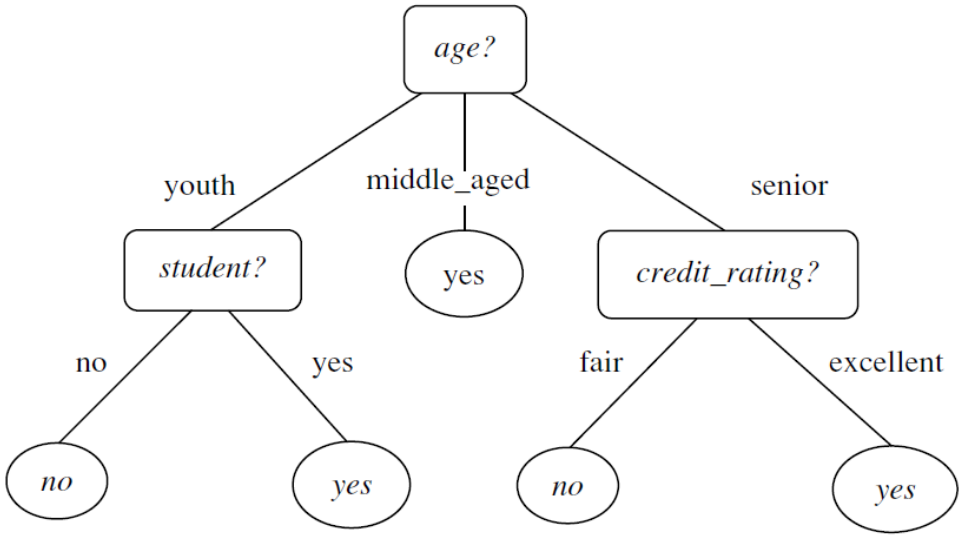


income	student	credit_rating	class
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

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

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

Example: Final Tree



Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - Therefore, given v values of A, then v - 1 possible splits are evaluated.
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

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

- Gain ratio: $GainRatio_A = \frac{Gain(A)}{SplitInfo_A(D)}$
- The attribute with the maximum gain ratio is selected as the splitting attribute

Gain Ratio for Attribute Selection (C4.5)

$$Gain(income) = 0.029$$

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

$$SplitInfo_{income}(D) = -\frac{4}{14} log \frac{4}{14} - \frac{6}{14} log \frac{6}{14} - \frac{4}{14} log \frac{4}{14} = 1.557$$

$$GainRatio_A = \frac{Gain(A)}{SplitInfo_A(D)} = \frac{0.029}{1.557} = 0.019$$

Gini Index (CART, IBM IntelligentMiner)

- The Gini index is used in CART.
- Using the notation previously described, the Gini index measures the impurity of *D*, a data partition or set of training tuples

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class *j* in *D*

- The Gini index considers a binary split for each attribute

Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

Gini Index (CART, IBM IntelligentMiner)

- For each attribute, each of the possible binary splits is considered.
- For a discrete-valued attribute, the subset that gives the minimum Gini index for that attribute is selected as its splitting subset.
- For continuous-valued attributes, each possible split-point must be considered. The strategy is similar to that described earlier for information gain, where the midpoint between each pair of (sorted) adjacent values is taken as a possible split-point.

Gini Index (CART, IBM IntelligentMiner)

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Gini Index (CART, IBM IntelligentMiner)

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

Consider the subset {low, medium} ($income \in \{low, medium\}$)

$$Gini_{income \in \{low,medium\}}(D) = \frac{10}{14} Gini(D_1) + \frac{4}{14} Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = 0.443$$

$$= Gini_{income \in \{high\}}(D)$$

$$Gini_{income \in \{low,high\}}(D) = 0.458 = Gini_{income \in \{medium\}}(D)$$

$$Gini_{income \in \{medium,high\}}(D) = 0.450 = Gini_{income \in \{low\}}(D)$$

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

Example#2

For this question, you're going to answer a couple questions regarding the dataset shown below. You'll be trying to determine whether Andrew finds a particular type of food appealing based on the food's temperature, taste, and size.

Appealing	Temperature	Taste	Size
No	Hot	Salty	Small
No	Cold	Sweet	Large
No	Cold	Sweet	Large
Yes	Cold	Sour	Small
Yes	H	Sour	Small
No	H	Salty	Large
Yes	H	Sour	Large
Yes	Cold	Sweet	Small
Yes	Cold	Sweet	Small
No	H	Salty	Large

1. Initial entropy of Appealing?
2. Let the root be Taste. Information gain of the root?
3. Draw the full decision tree?

Example#2

For this question, you're going to answer a couple questions regarding the dataset shown below. You'll be trying to determine whether Andrew finds a particular type of food appealing based on the food's temperature, taste, and size.

Appealing	Temperature	Taste	Size
No	Hot	Salty	Small
No	Cold	Sweet	Large
No	Cold	Sweet	Large
Yes	Cold	Sour	Small
Yes	H	Sour	Small
No	H	Salty	Large
Yes	H	Sour	Large
Yes	Cold	Sweet	Small
Yes	Cold	Sweet	Small
No	H	Salty	Large

1. Initial entropy of Appealing?

$$Info(D) = -\sum_i p_i log p_i = -\frac{5}{10} log \frac{5}{10} - \frac{5}{10} log \frac{5}{10} = 0.5 + 0.5 = 1$$

Example#2

For this question, you're going to answer a couple questions regarding the dataset shown below. You'll be trying to determine whether Andrew finds a particular type of food appealing based on the food's temperature, taste, and size.

Appealing	Temperature	Taste	Size
No	Hot	Salty	Small
No	Cold	Sweet	Large
No	Cold	Sweet	Large
Yes	Cold	Sour	Small
Yes	H	Sour	Small
No	H	Salty	Large
Yes	H	Sour	Large
Yes	Cold	Sweet	Small
Yes	Cold	Sweet	Small
No	H	Salty	Large

2. Let the root be Taste. Information gain of the root?

Taste
Salty =3N,0Y
Sweet=2N,2Y
Sour=0N,3Y

$$Gain(A) = Info(D) - Info_A(D)\ , Info(D) = -\sum_i p_i log p_i$$

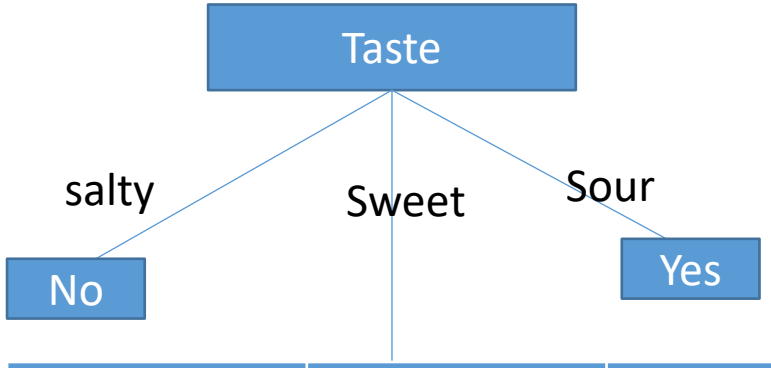
$$Info_{taste}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|}.Info(D_j)$$

$$= \frac{3}{10}\left(-\frac{3}{3}log\frac{3}{3} - \frac{0}{3}log\frac{0}{3}\right) + \frac{4}{10}\left(-\frac{2}{4}log\frac{2}{2} - \frac{2}{4}log\frac{2}{4}\right)$$

$$+ \frac{3}{10}\left(-\frac{3}{3}log\frac{3}{3} - \frac{0}{3}log\frac{0}{3}\right) = 0.4$$

$$Gain(taste) = Info(D) - Info_A(D) = 1 - 0.4 = 0.6$$

Example#2



Appealing	Temperature	Size
No	Cold	Large
No	Cold	Large
Yes	Cold	Small
Yes	Cold	Small

For this question, you're going to answer a couple questions regarding the dataset shown below. You'll be trying to determine whether Andrew finds a particular type of food appealing based on the food's temperature, taste, and size.

Appealing	Temperature	Taste	Size
No	Hot	Salty	Small
No	Cold	Sweet	Large
No	Cold	Sweet	Large
Yes	Cold	Sour	Small
Yes	H	Sour	Small
No	H	Salty	Large
Yes	H	Sour	Large
Yes	Cold	Sweet	Small
Yes	Cold	Sweet	Small
No	H	Salty	Large

Example#2

Appealing	Temperature	Size
No	Cold	Large
No	Cold	Large
Yes	Cold	Small
Yes	Cold	Small

$$Info(D) = -\sum_i p_i log p_i = -\frac{2}{4} log \frac{2}{4} - \frac{2}{4} log \frac{2}{4} = 1$$

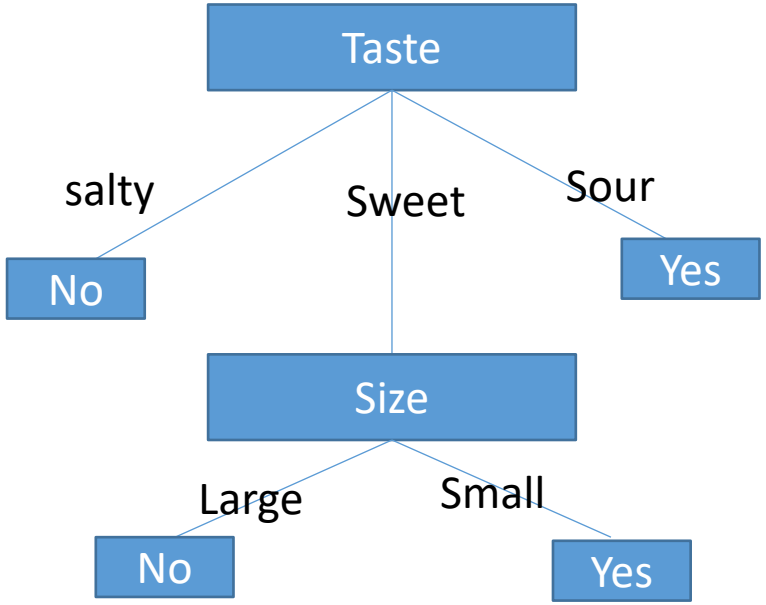
$$Gain(Size) = Info(D) - Info_{size}(D) = 1 - 0 = 1$$

$$Info_{size}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} . Info(D_j) = \frac{2}{4} . \left(-\frac{2}{2} \log \frac{2}{2} - \frac{0}{2} \log \frac{0}{2} \right) + \frac{2}{4} . \left(-\frac{0}{2} \log \frac{0}{2} - \frac{2}{2} \log \frac{2}{2} \right) = 0$$

$$Gain(Temp) = Info(D) - Info_{temp}(D) = 1 - 1 = 0$$

$$Info_{temp}(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} . Info(D_j) = \frac{4}{4} \left(-\frac{2}{4} log \frac{2}{4} - \frac{2}{4} log \frac{2}{4} \right) = 1$$

Example#2



Appealing	Temperature	Size
No	Cold	Large
No	Cold	Large
Yes	Cold	Small
Yes	Cold	Small

Regression Trees

- *A regression tree* is constructed in almost the same manner as a classification tree, except that the impurity measure that is appropriate for classification is replaced by a measure appropriate for regression.
- Let us say for node m, X_m is the subset of X reaching node m; namely, it is the set of all $\boldsymbol{x} \in X$ satisfying all the conditions in the decision nodes on the path from the root until node m.

Regression Trees

- Error at node m:

$$b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_m : \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\mathbf{x}^t\right)$$

- X_m is the subset of X reaching node m
- $N_m = |X_m| = \sum_t b_m(x^t)$

Regression Trees

- In a node, we use the mean (median if there is too much noise) of the required outputs of instances reaching the node

$$g_m = \frac{\sum_t b_m\left(\mathbf{x}^t\right) r^t}{\sum_t b_m\left(\mathbf{x}^t\right)}$$

- g_m estimated value in node m.

Regression Trees

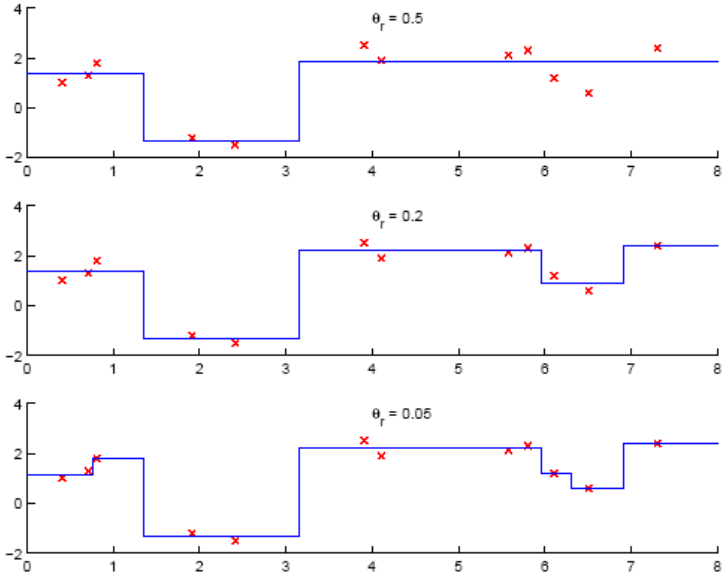
- If at a node, the error is acceptable, that is, $E_m < \theta_r$, then a leaf node is created and it stores the g_m value.
- If the error is not acceptable, data reaching node *m* is split further such that the sum of the errors in the branches is minimum.
- After splitting:

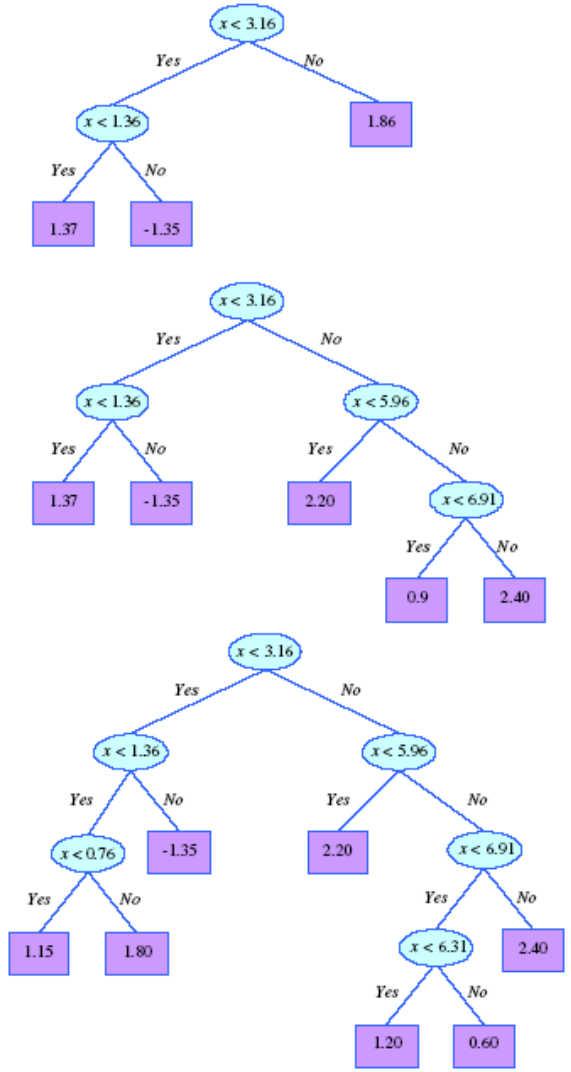
$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_{mj} : \mathbf{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

$$E'_m = \frac{1}{N_m} \sum_j \sum_t \left(r^t - g_{mj}\right)^2 b_{mj}\left(\mathbf{x}^t\right) \qquad g_{mj} = \frac{\sum_t b_{mj}\left(\mathbf{x}^t\right) r^t}{\sum_t b_{mj}\left(\mathbf{x}^t\right)}$$

Regression Trees

- The acceptable error threshold is the complexity parameter; when it is small, we generate large trees and risk overfitting; when it is large, we underfit and smooth too much



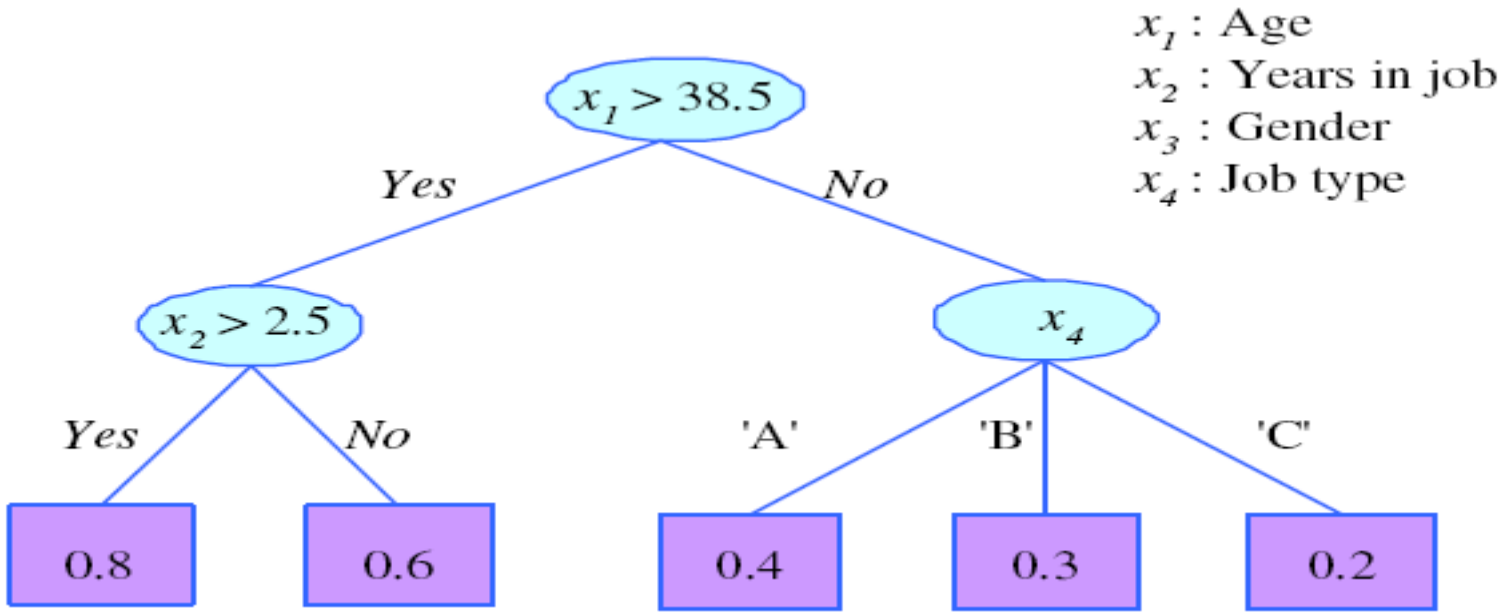


Pruning Trees

- Remove subtrees for better generalization (decrease variance)
  - Prepruning: Early stopping
  - Postpruning: Grow the whole tree then prune subtrees that overfit on the pruning set
- Prepruning is faster, postpruning is more accurate (requires a separate pruning set)

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2