

# **BASIC DECISION TREE INDUCTION ALGORITHM**

Lecture Notes on Learning (5)  
cse352

Professor Anita Wasilewska  
Stony Brook University

# Decision Tree Algorithms

## Short History

- **Late 1970s - ID3 (Iterative Dichotomiser)** by **J. Ross Quinlan**.
- This work expanded on earlier work on concept learning system, described by **E. B. Hunt, J. Marin, and P. T. Stone**.
- **Early 1980 - C4.5 a successor of ID3** by **Quinlan**.
- **C4.5** later became a **benchmark** to which newer supervised learning algorithms, are often compared.
- **In 1984**, a group of statisticians (**L. Breinman, J. Friedman, R. Olshen, and C. Stone**) published the book **“Classification and Regression Trees(CART) “**.
-

# Decision Tree Algorithms

## Short History

- The “**Classification and Regression Trees (CART)**” book described a generation of binary decision trees .
- **ID3,C4.5** and **CART** were invented **independently** of one another yet follow a similar approach for learning decision trees from training tuples.
- These **two cornerstone algorithms** spawned a **flurry of work on decision tree induction.**

# Decision Tree Algorithms

## General Description

- **ID3, C4.5**, and **CART** adopt a greedy (i.e., non-backtracking) approach.
- In this approach decision trees are constructed in a **top-down recursive divide-and conquer** manner.
- **Most algorithms for decision tree induction also follow such a top-down approach.**
- All of the algorithms start with a **training set** of tuples and their associated class labels (classification data table).
- **The training set is recursively partitioned into smaller subsets as the tree is being built.**

# BASIC Decision Tree Algorithm

## General Description

- **A Basic Decision Tree Algorithm** presented here is as published in J.Han, M. Kamber book “Data Mining, Concepts and Techniques”, 2006 (second Edition)
- The algorithm may appear long, but is quite straightforward.
- Basic Algorithm strategy is as follows.
- The algorithm is called with three parameters: ***D***, ***attribute\_list***, and ***Attribute\_selection\_method***.
- We refer to **D** as a **data partition**.
- Initially, **D** is the **complete set of training tuples and their associated class labels** (input training data).

# Basic Decision Tree Algorithms

## General Description

- The parameter ***attribute\_list*** is a list of attributes describing the tuples.
- ***Attribute\_selection\_method*** specifies a heuristic procedure for selecting the attribute that “best” discriminates the given tuples according to class.
- ***Attribute\_selection\_method*** procedure employs an attribute selection measure, such as **Information Gain** or the **Gini Index**
- Whether the tree is strictly binary is generally driven by the attribute selection measure.

# Basic Decision Tree Algorithms

## General Description

- Some **attribute selection measures**, like the **Gini Index** enforce the resulting tree to be **binary**.
- Others, like the **Information Gain**, do not.
- They, as **Information Gain** does, allow multi-way splits (i.e. two or more branches to be grown from a node). In this case the branches represent all the (discrete) values of the nodes attributes

# Basic Decision Tree Algorithms

## General Description

- The tree starts as a single node  $N$ .
- The node  $N$  represents the training tuples in  $D$  (training data table)
- This is the step 1 in the algorithm.
- If the tuples in  $D$  are all of the same class, then node  $N$  becomes a leaf and is labeled with that class .
- These are the steps 2 and 3 in the algorithm.
  
- The steps 4 and 5 in the algorithm are terminating conditions.
- All of the other terminating conditions are explained at the end of the algorithm.

# Basic Decision Tree Algorithms

## General Description

- **Otherwise**, the algorithm calls *attribute\_selection\_method* to determine the **splitting criterion**.
- The **splitting criterion** tells us which attributes to test at node N in order to determine the “best” way to separate or partition the tuples in **D** into individual classes (sub-tables) called **partitions**.
- This is the **step 1** in the algorithm.
- The **splitting criterion** also tells us which **branches to grow from node N** with respect to the outcomes of the chosen test.
- More specifically, the *splitting criterion* indicates the **splitting attribute** and may also indicate either **a split-point** or **a splitting subset**.

# Basic Decision Tree Algorithms

## General Description

- 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 of the tuples in it belong to the same class.
- In other words, if we were to 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.**
-

# Basic Decision Tree Algorithms

## General Description

- The node  $N$  is labeled with the splitting criterion, which 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
- (step 10 and 11).
- There are three possible scenarios, as illustrated in figure 6.4 in the handout.

# Basic Decision Tree Algorithms

## General Description

- Let  $A$  be the **splitting attribute**.
- $A$  has distinct values (attribute values)
- $\{a_1, a_2, \dots, a_v\}$
- The values  $\{a_1, a_2, \dots, a_v\}$  of the attribute  $A$  are **based on the training data** within the run of the algorithm
- This is the **step 7** in the algorithm.
- We have the following cases depending of the **TYPE** of the values of the split attribute  $A$ .

# Basic Decision Tree Algorithms

## General Description

### 1. **A is discrete-valued:**

- In this case, the outcomes of **the test at node N** correspond directly to the known (in training set) values of A.
- A branch is created for **each value  $a_j$**  of the attribute A
- The branch is labeled with that value  **$a_j$** .
- **There are as many branches the number of values of A in the training data.**
- **Partition  $D_j$**  is the subset of class-labeled tuples in D **having value  $a_j$  of A.**
- **Partition  $D_j$**  is a sub-table of the table at the node N.
- Because all of the tuples in a given partition have the same value for A, then A need not be considered in any future partitioning of the tuples.
- Therefore the attribute A **it is removed** from ***attribute\_list***
- Theses are the steps **8 and 9** in the algorithm.

# Basic Decision Tree Algorithms

## General Description

- **A is continuous-valued.**
- In this case, **the test at node N** has two possible outcomes, corresponding to the conditions
- $A \leq \textit{split\_point}$  and  $A > \textit{split\_point}$
- The *split\_point* is the split-point returned by *Attribute\_selection\_method* .
- In practice, the split-point is often taken as the midpoint of two known adjacent values of A and therefore may not actually be a pre-existing value of A from the training data.
- **Two branches are grown from N and labeled**
- $A \leq \textit{split\_point}$  and  $A > \textit{split\_point}$
- The tuples (table at the node N) are **partitioned** in two sets (sub-tables) **D1 and D2**.
- **D1** holds the subset of class-labeled tuples in **D** for which  $A \leq \textit{split\_point}$ , while **D2** holds the rest.

# Basic Decision Tree Algorithms

## General Description

- A is discrete-value and a binary tree must be produced (as described by the attribute selection measure or algorithm being used): The test at node N is of the form “ $A \geq SA$ ?”. 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  $a_j$  of A and if  $a_j \geq SA$ , 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  $D_1$  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  $D_2$  corresponds to the subset of class-labeled tuples from D that do not satisfy the test.
- 
- The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition,  $D_j$ , of D (step 14).
-

# Basic Decision Tree Algorithms

## General Description

- **TERMINATING CONDITIONS**
- The recursive partitioning **stops** only when any one of the following terminating conditions is true.
- **1. All of the tuples** in partition D (represented at node N) **belong to the same class** (step 2 and 3), or
- **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).
- **Majority voting** involves converting node N into a leaf and labeling it with **the most common class in D which is a set of training tuples and their associated class labels**. Alternatively, the class distribution of the node tuples may be stored.
- **3. There are no tuples for a given branch**, that is, a partition  $D_j$  is empty.
- In this case, a leaf is created with **the majority class in D**.
- The decision tree is returned
- This is the **step 14** of the algorithm.

# Basic Decision Tree Algorithm

- 
- Algorithm: *Geneate\_decision\_tree*
- 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 sub-trees for each partition
  - (11) Let **D<sub>j</sub>** be the set of a data tuples in D satisfying outcome **j**; // a partition
  - (12) If **D<sub>j</sub>** is empty then
  - (13) Attach a leaf labeled with the **majority class in D** to node N;
  - (15) Else attach the node returned by **Geneate\_decision\_tree** (D<sub>j</sub>, attribute list) to node N;
  - (16) Return N;
-