BASIC DECISION TREE INDUCTION FULL ALGORITHM

cse537
Artificial Intelligence

Professor Anita Wasilewska
Stony Brook University
Decision Tree Algorithms

Short History

• Late 1970s - **ID3 (Interactive 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 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. a non-backtracking) approach

• It 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
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 Algorithm

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 Algorithm
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
- They allow for 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 Algorithm
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**

• Theses are the **steps 2 and 3** in the **algorithm**

• The **steps 4 and 5** in the **algorithm** are **terminating conditions**
• All of the **terminating conditions** are explained at the **end** of the **algorithm**
Basic Decision Tree Algorithm
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 6** 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 Algorithm
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 Algorithm
General Description

• The node $N$ is labeled with the splitting criterion, which serves as a test at the node
• This is 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
• These are steps 10 and 11

• There are three possible scenarios, as illustrated in figure 6.4 on your handout
Basic Decision Tree Algorithm
General Description

• Let $A$ be the **splitting attribute**
• $A$ has distinct values (attribute values)
• $a_1, a_2, \ldots, a_v$
• The values $a_1, a_2, \ldots, a_v$ of the attribute $A$ are based on the training data for the run of the algorithm
• This is the **step 7** in the algorithm

• We have the following **cases** depending on the **TYPE** of the values of the split attribute $A$
Basic Decision Tree Algorithm
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
Basic Decision Tree Algorithm
General Description

2. A is continuous-valued

• In this case, the test at node N has two possible outcomes, corresponding to the conditions

• $A \leq \text{split}\_\text{point}$ and $A > \text{split}\_\text{point}$

• 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 $A$

• Therefore the **split-point** may not actually be a pre-existing value of $A$ from the **training data**
Basic Decision Tree Algorithm
General Description

• Two branches are grown from N and labeled $A \leq \text{split\_point}$ and $A > \text{split\_point}$

• The tuples (table at the node N) are partitioned sub-tables D1 and D2

• D1 holds the subset of class-labeled tuples in D for which $A \leq \text{split\_point}$

• D2 holds the rest
Basic Decision Tree Algorithms
General Description

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

- The test at node N is of the form “A?SA?”
- SA is the splitting subset for A
- SA is returned by attribute_selection_method as part of the splitting criterion
- SA is a subset of the known values of A
- IF a given tuple has value aj of A and aj belongs to SA , THEN the test at node N is satisfied
Basic Decision Tree Algorithms

General Description

• **Two branches** are grown from **N**
• 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**

• **The algorithm** uses the same process **recursively** to form a **decision tree** for the tuples **at each** resulting partition, **Dj** of **D**
• This is **step 14**
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)
Basic Decision Tree Algorithms
General Description

- **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 the set of training tuples** \( D \)

- The **decision tree** is returned

- This is the **step 15** 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 `Dj` be the set of a 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`;
  - (15) else attach the node returned by `Geneate_decision_tree` `(Dj, attribute_list)` to node `N`;
  - (16) return `N`;
Attribute Selection Measure: Information Gain (ID3/C4.5)

- 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 $D$:

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

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

$$\text{Info}_A(D) = \sum_{j=1}^{v} \left| \frac{D_j}{D} \right| \times I(D_j)$$

- **Information gained** by branching on attribute $A$:

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$
Computing Information-Gain for Continuous-Value 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 \(a_{i+1}\)
  - 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)

  \[
  \text{SplitInfo}_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)
  \]

  \[= \text{GainRatio}(A) = \frac{\text{Gain}(A)}{\text{SplitInfo}(A)}\]

• **Ex.**\[\text{SplitInfo}_A(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 0.926\]

  \[= \text{gain\_ratio}(\text{income}) = \frac{0.029}{0.926} = 0.031\]

• The attribute with the **maximum gain ratio** is **selected** as the splitting attribute.
Gini index (CART, IBM IntelligentMiner)

• If a data set \( D \) contains examples from \( n \) classes, gini index, \( gini(D) \) is defined as

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

where \( p_j \) is the relative frequency of class \( j \) in \( D \)

• 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

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

• Reduction in Impurity:

\[
\Delta gini(A) = gini(D) - gini_A(D)
\]

• The attribute provides the smallest \( gini_{\text{split}}(D) \) (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)

- Ex. \( D \) has 9 tuples in \( \text{buys\_computer} = \text{"yes"} \) and 5 in \( \text{"no"} \)

\[
gini(D) = 1 - \left( \frac{9}{14} \right)^2 - \left( \frac{5}{14} \right)^2 = 0.459
\]

- Suppose the attribute \( \text{income} \) partitions \( D \) into 10 in \( D_1 \): \{low, medium\} and 4 in \( D_2 \)

\[
gini_{\text{income} \in \{\text{low, medium}\}}(D) = \left( \frac{10}{14} \right) \text{Gini}(D_1) + \left( \frac{4}{14} \right) \text{Gini}(D_1)
\]

\[
= \frac{10}{14} \left(1 - \left( \frac{6}{10} \right)^2 - \left( \frac{4}{10} \right)^2\right) + \frac{4}{14} \left(1 - \left( \frac{1}{4} \right)^2 - \left( \frac{3}{4} \right)^2\right)
\]

\[
= 0.450
\]

but \( gini_{\{\text{medium, high}\}} \) is 0.30 and thus the best since it is the lowest

- Case: All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes
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
Other Attribute Selection Measures

- **CHAID**: a popular decision tree algorithm, measure based on $\chi^2$ test for independence
- **C-SEP**: performs better than info. gain and gini index in certain cases
- **G-statistics**: has a close approximation to $\chi^2$ distribution
- **MDL** (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree as the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- **Multivariate splits** (partition based on multiple variable combinations)
  - **CART**: finds multivariate splits based on a linear comb. of attrs.
- **Which attribute selection measure is the best?**
  - Most give good results, **none is significantly superior** than others
Overfitting and Tree Pruning

- **Overfitting**: An induced tree may **overfit** the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples

- Two approaches to avoid overfitting
  - **Prepruning**: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - **Postpruning**: Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the “best pruned tree”
Enhancements to Basic Decision Tree Induction

• Allow for continuous-valued attributes
  – Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals

• Handle missing attribute values
  – Assign the most common value of the attribute
  – Assign probability to each of the possible values

• Attribute construction
  – Create new attributes based on existing ones that are sparsely represented
  – This reduces fragmentation, repetition, and replication
Classification in Large Databases

- **Classification**—a classical problem extensively studied by statisticians and machine learning researchers

- **Scalability**: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed

- **Why decision tree induction in data mining?**
  - relatively **faster learning speed** (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use **SQL queries** for accessing databases
  - comparable **classification accuracy** with other methods
Scalable Decision Tree Induction Methods

- **SLIQ** (EDBT’ 96 — Mehta et al.)
  - Builds an index for each attribute and only class list and the current attribute list reside in memory
- **SPRINT** (VLDB’ 96 — J. Shafer et al.)
  - Constructs an attribute list data structure
- **PUBLIC** (VLDB’ 98 — Rastogi & Shim)
  - Integrates tree splitting and tree pruning: stop growing the tree earlier
- **RainForest** (VLDB’ 98 — Gehrke, Ramakrishnan & Ganti)
  - Builds an AVC-list (attribute, value, class label)
- **BOAT** (PODS’ 99 — Gehrke, Ganti, Ramakrishnan & Loh)
  - Uses bootstrapping to create several small samples