Lecture 7 - Decision Tree Learning (1)

Prof. Xiaowei Huang

https://cgi.csc.liv.ac.uk/~xiaowei/ai.html

(Attendance Code: **357226**)

Decision Tree up to now,

- Decision tree representation

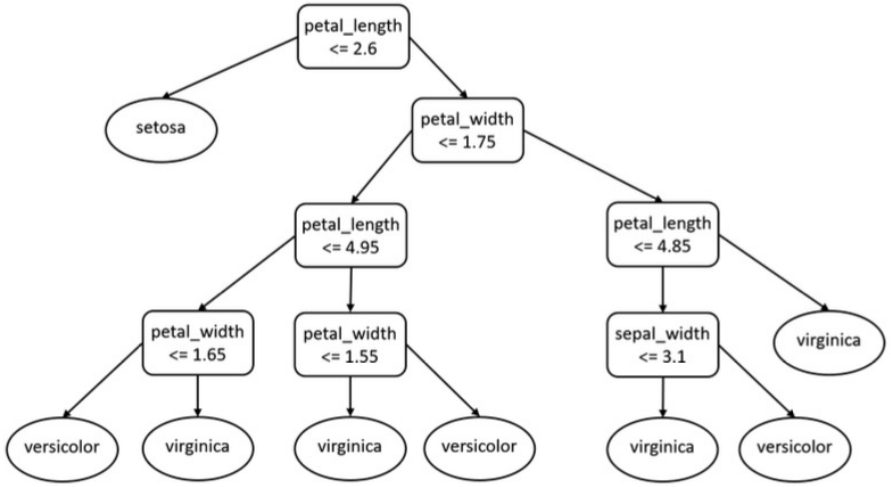


Fig. 5.1: A decision tree for **iris** dataset [67]

Today's Topics

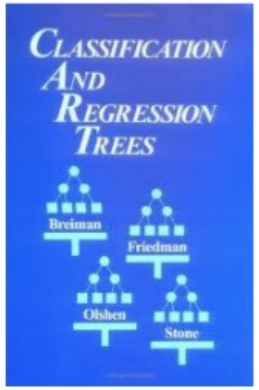
A general top-down algorithm

How to do splitting on numeric features

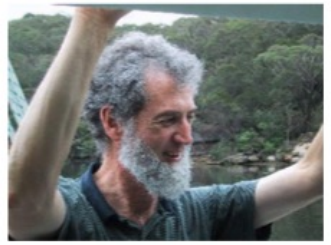
Principle of Decision Tree Learning

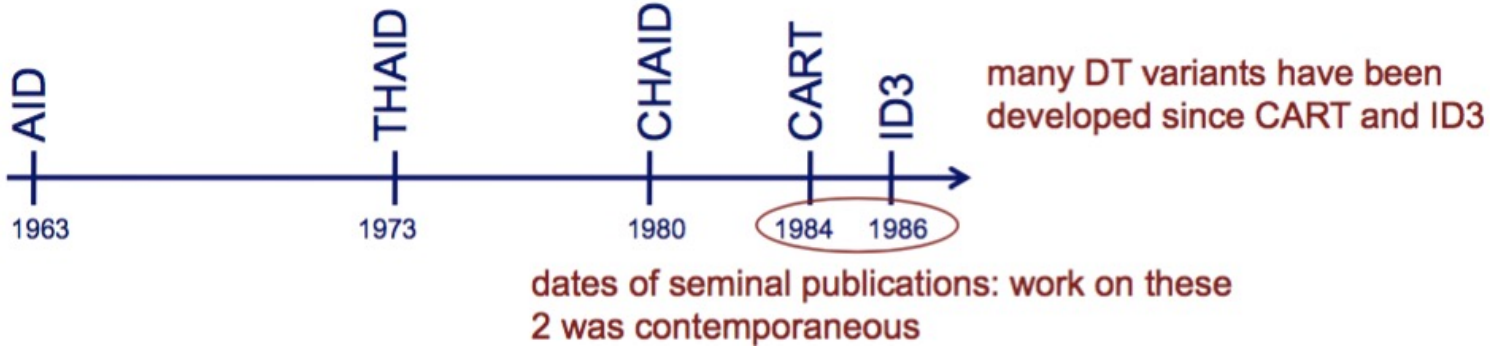
History of decision tree learning

CART developed by Leo Breiman, Jerome Friedman, Charles Olshen, R.A. Stone



ID3, C4.5, C5.0 developed by Ross Quinlan





Decision tree learning algorithms

This is what we will discuss in this module

- **ID3**, or Iterative Dichotomizer
  - one property is tested on each node of the tree
  - maximizing information gain
- CART, or Classification and Regression Trees
  - binary trees
  - splits are selected using the towing criteria

based on Gini Index

- C4.5, improved version on ID3

Decision tree learning algorithms

	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible to outliers
CART	Towing Criteria	Handles both Categorical & Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical & Numeric value	Handle missing values.	Error Based pruning is used	Susceptible to outliers

Top-down decision tree learning

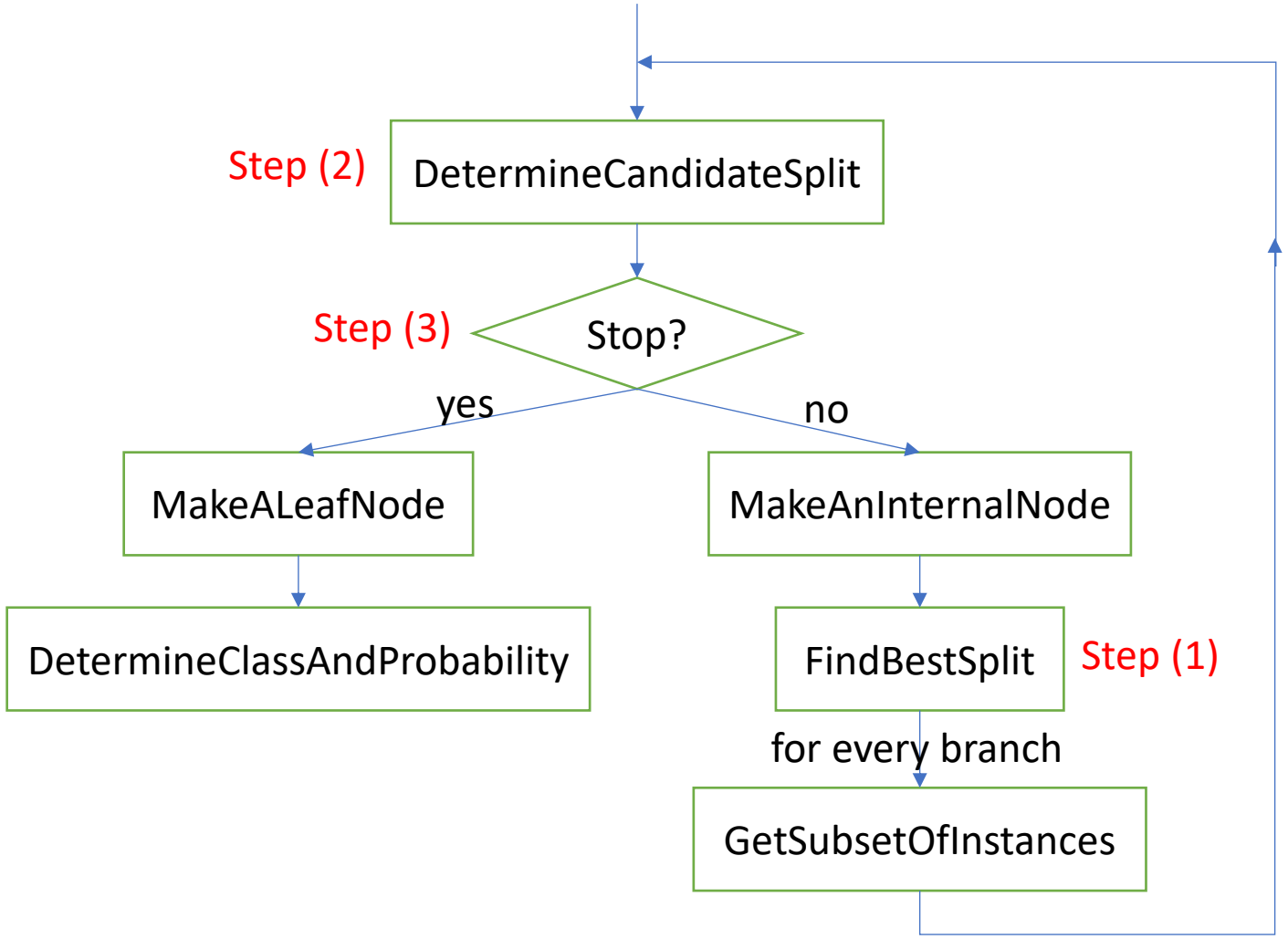
```
1   MakeSubtree(set of training instances D)
2       C = DetermineCandidateSplits(D)
3       if stopping criteria met
4           make a leaf node N
5           determine class label/probabilities for N
6       else
7           make an internal node N
8           S = FindBestSplit(D, C)
9           for each outcome k of S
10              D_k = subset of instances that have outcome k
11              k^th child of N = MakeSubtree(D_k)
12      return subtree rooted at N
```

Step (2)

Step (3)

Step (1)

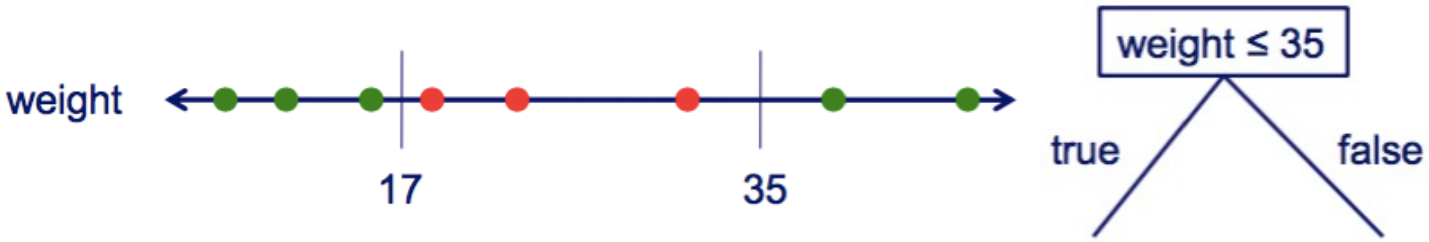
Top-down decision tree learning



Step(1): **FindBestSplit** (for a given feature)

Candidate splits on numeric features

- Given a set of training instances D and a specific feature X_i
  - sort the values of X_i in D
  - evaluate split thresholds in intervals between instances of different classes



- Could use midpoint of each considered interval as the threshold
- C4.5 instead picks the largest value of X_i in the entire training set that does not exceed the midpoint

Candidate splits on numeric features (in more detail)

```
1  // Run this subroutine for each numeric feature at each node of DT induction
2  DetermineCandidateNumericSplits(set of training instances D, feature X_i)
3      C = {}          // initialize set of candidate splits for feature X_i
4      S = partition instances in D into sets s_1 ... s_V where the instances in each
           set have the same value for X_i
5      let v_j denote the value of X_i for set s_j
6      sort the sets in S using v_j as the key for each s_j
7      for each pair of adjacent sets s_j, s_{j+1} in sorted S
8          if s_j and s_{j+1} contain a pair of instances with different class labels
9              // assume we're using midpoints for splits
10             add candidate split X_i ≤ (v_j + v_{j+1})/2 to C
11     return C
```

D= {(0,b),(1,a),(1,c),(2,b),(3,b)}

s_1={0,b)}
s_2={(1,a),(1,c)}
s_3={(2,b)}
s_4={(3,b)}

v_1=0; v_2=1;
v_3=2; v_4=3

a split between s_1 and s_2

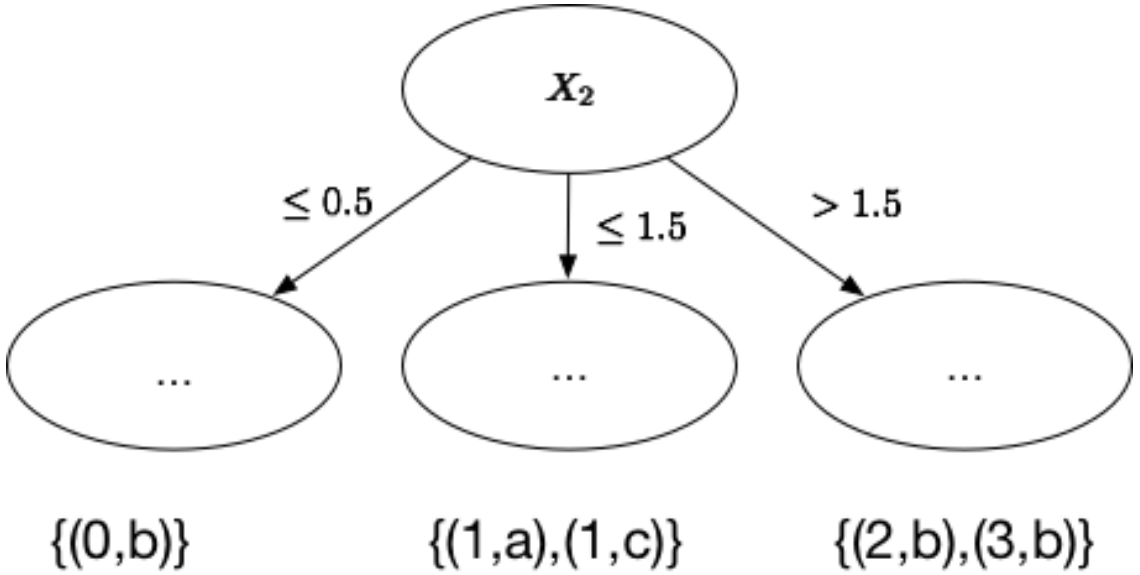
a split between s_2 and s_3

Example

- (1,a),(2,b),(1,c),(0,b),(3,b) for (value of a feature, label)

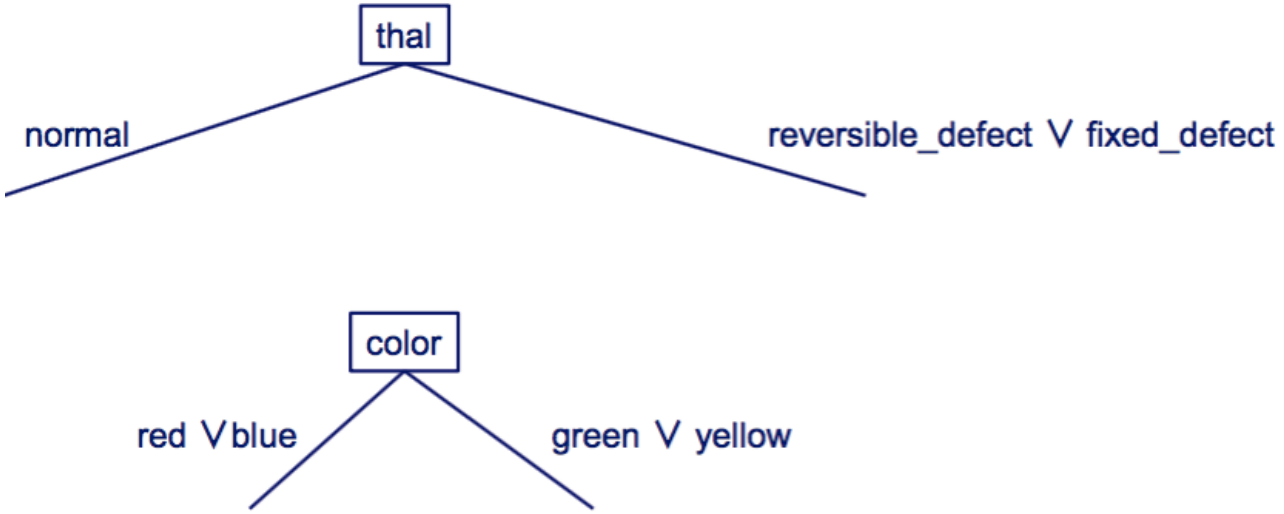
- {(0,b)}, {(1,a),(1,c)}, {(2,b)}, {(3,b)}

- C = { X_i <= 0.5 }
- C = {X_i <= 0.5, X_i <= 1.5 }
- C = {X_i <= 0.5, X_i <= 1.5, X_i <= 2.5 }



Candidate splits

- instead of using k-way splits for k-valued features, could require binary splits on all discrete features (CART does this)



Step (2): **DetermineCandidateSplits** (i.e., select the best feature among a set of features)

Finding the best split

- How should we select the best feature to split on at each step?

- Key hypothesis: the simplest tree that classifies the training instances accurately will work well on previously unseen instances

Occam’s razor

- Attributed to 14th century William of Ockham

- “when you have two competing theories that make exactly the same predictions, the simpler one is the better”

- A thousand years earlier, I said, “We consider it a good principle to explain the phenomena by the simplest hypothesis possible.”

Occam's razor and decision trees

- Why is Occam's razor a reasonable heuristic for decision tree learning?
  - there are fewer short models (i.e. small trees) than long ones
  - a short model is unlikely to fit the training data well by chance
  - a long model is more likely to fit the training data well coincidentally

Finding the best splits

- Can we find and return the smallest possible decision tree that accurately classifies the training set?

  This is an NP-hard problem

  [Hyafil & Rivest, *Information Processing Letters, 1976]*

- Instead, we'll use an information-theoretic heuristics to greedily choose splits

Recap: Expected Value (Finite Case)

- Let X be a random variable with a finite number of finite outcomes x_1, x_2, …, x_k occurring with probability p_1, p_2, …, p_k, respectively. The expectation of X is defined as

$$E[X] = p_1x_1 + p_2x_2 + \ldots + p_kx_k$$

- Expectation is a weighted average

Expected Value Example

- Let X represent the outcome of a roll of a fair six-sided die
- Possible values for X include {1,2,3,4,5,6}
- Probability of them are {1/6, 1/6, 1/6, 1/6, 1/6, 1/6}
- The expected value is $\mathrm{E}[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5$