### Tree Uses Nodes, and Leaves

- **Internal decision nodes**
  - Univariate: Uses a single attribute, $x_j$
    - Numeric $x_j$: Binary split: $x_j > w_m$
    - Discrete $x_j$: $n$-way split for $n$ possible values
  - Multivariate: Uses all attributes, $x$

- **Leaves**
  - Classification: Class labels, or proportions
  - Regression: Numeric; $r$ average, or local fit

- **Learning is greedy**: find the best split recursively (Breiman et al, 1984; Quinlan, 1986, 1993)
High level

- Want small tree fitting data for good generalization
- Occam’s razor (simplest hypothesis consistent with data is best)
- Distinguish real world from all possible situations vs. Distinguish real world from all simpler situations
- $N$ data points, $d$ real valued attributes, $k$ nodes:
  - $\sim dN$ many different attribute < value tests
  - $(dN)^k$ many trees -- too many to search
  - Use greedy top-down search

Classification Trees (ID3, CART, C4.5)

- For node $m$, $N_m$ instances reach $m$, $N'_m$ belong to $C_i$
  \[ \hat{p}(C_i \mid x, m) = p'_m = \frac{N'_m}{N_m} \]
- Node $m$ is pure if $p'_m$ is 0 or 1
- Measure of impurity is entropy
  \[ I_m = -\sum_{i=1}^{k} p'_m \log p'_m \]

Best Split

- If node $m$ is pure, generate a leaf and stop, otherwise split and continue recursively
- Impurity after split: $N_{mj}$ of $N_m$ take branch $j$, $N'_{mj}$ belong to $C_i$
  \[ \hat{p}(C_i \mid x, m, j) = p''_{mj} = \frac{N'_{mj}}{N_{mj}} \]
  \[ I'_m = -\sum_{j=1}^{n} N_{mj} \log_2 p''_{mj} \]
- Find the variable and split that min impurity (among all variables -- and split positions for numeric variables)
Regression Trees

- Error at node $m$:
  \[ b_m(x) = \begin{cases} 
  1 & \text{if } x \in X_m : x \text{ reaches node } m \\
  0 & \text{otherwise}
  \end{cases} \]
  \[ F_m = \frac{1}{N_m} \sum_{x \in X_m} (y - b_m(x))^2 \quad g_m = \frac{\sum_{x \in X_m} b_m(x) y}{\sum_{x \in X_m} b_m(x)} \]

- After splitting:
  \[ b_{mj}(x) = \begin{cases} 
  1 & \text{if } x \in X_{mj} : x \text{ reaches node } m \text{ and branch } j \\
  0 & \text{otherwise}
  \end{cases} \]
  \[ F_m = \frac{1}{N_m} \sum_{j} \sum_{x \in X_{mj}} (y - g_{mj}) b_{mj}(x) \quad g_{mj} = \frac{\sum_{x \in X_{mj}} b_{mj}(x) y}{\sum_{x \in X_{mj}} b_{mj}(x)} \]

Pruning Trees

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

Model Selection in Trees:

Rule Extraction from Trees

C4.5 Rules
(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 $y = 0.2$
Learning Rules

- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered
- IREP (Fürnkranz and Widmer, 1994), Ripper (Cohen, 1995)

Multivariate Trees

Ripper(Pos, Neg, λ)

RuleSet ← LearnRuleSet(Pos, Neg)
For λ times
  RuleSet ← OptimizeRuleSet(RuleSet, Pos, Neg)
LearnRuleSet(Pos, Neg)
RuleSet ← Ø
DL ← DescLen(RuleSet, Pos, Neg)
Repeat
  Rule ← LearnRuleSet(Pos, Neg)
  Add Rule to RuleSet
  DL' ← DescLen(RuleSet, Pos, Neg)
  If DL' > DL + 64
    PruneRuleSet(RuleSet, Pos, Neg)
    Return RuleSet
  If DL' < DL
    DL ← DL'
    Delete instances covered from Pos and Neg
    Until Pos = Ø
    Return RuleSet

PruneRuleSet(RuleSet, Pos, Neg)

For each Rule ∈ RuleSet in reverse order
  DL ← DescLen(RuleSet, Pos, Neg)
  DL' ← DescLen(RuleSet-Rule, Pos, Neg)
  If DL' < DL Delete Rule from RuleSet
Return RuleSet

OptimizeRuleSet(RuleSet, Pos, Neg)

For each Rule ∈ RuleSet
  DL0 ← DescLen(RuleSet, Pos, Neg)
  DL1 ← DescLen(RuleSet-Rule+, RuleSet, Pos, Neg)
  DL2 ← DescLen(RuleSet-Rule+, RuleSet-Rule+, Pos, Neg)
  If DL1 = min(DL0, DL1, DL2)
    ReplaceRule(RuleSet, Rule, Pos, Neg)
    Delete Rule from RuleSet and add ReplaceRule(RuleSet, Rule, Pos, Neg)
  Else If DL2 < min(DL0, DL1, DL2)
    Delete Rule from RuleSet and add ReviseRule(RuleSet, Rule, Pos, Neg)
Return RuleSet
Random Forests

- Pick small random subset of features to try at each node rather than exhaustive search
- Build many trees and predict with most frequent prediction
- Subset saves time, robust against missing data
- Ensemble reduces variance - don’t need pruning
- Ho ‘95, Breiman ‘01