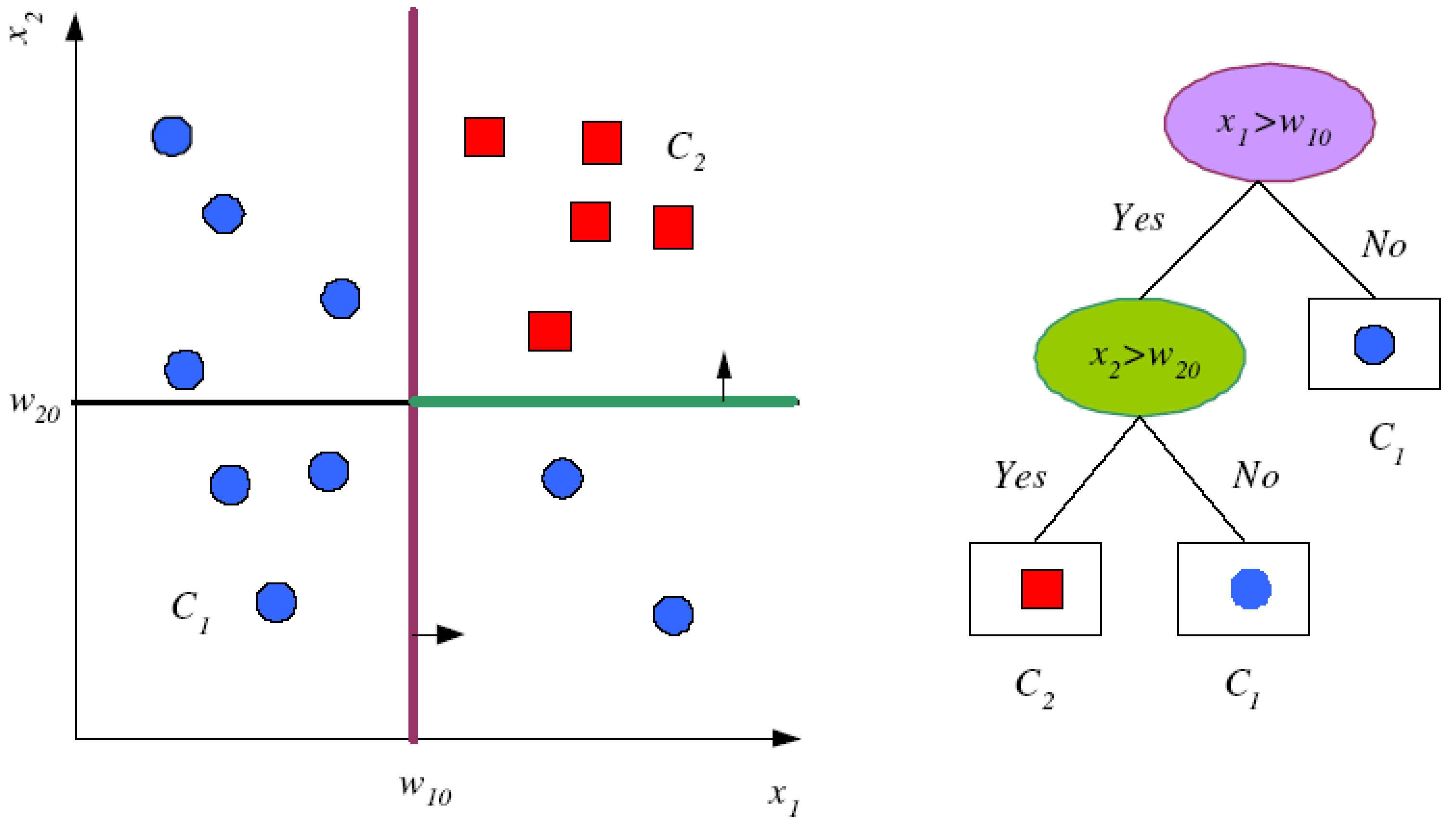
CHAPTER 9:

DECİSİON TREES

Tree Uses Nodes and Leaves



Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute, x_i
    - Numeric x_i : Binary split : $x_i > w_m$
    - Discrete x_i : n-way split for n possible values
  - Multivariate: Uses all attributes, $\boldsymbol{x}$
- 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)

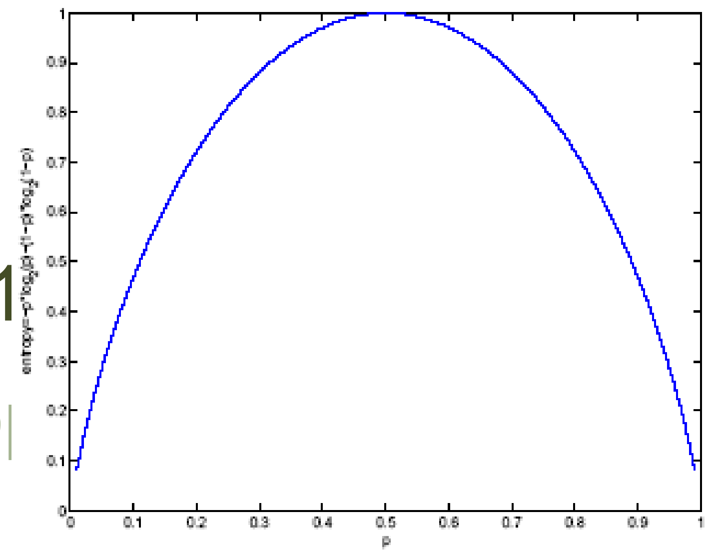
Classification Trees (ID3,CART,C4.5)

- For node m, N_m instances reach m, N^i_m belong to C_i

$$\hat{P}(C_i \mid \mathbf{x}, m) \equiv p^i_m = \frac{N^i_m}{N_m}$$

- Node m is pure if p^i_m is 0 or 1
- Measure of impurity is entro

$$\mathcal{I}_m = -\sum_{i=1}^{K} p^i_m \log_2 p^i_m$$



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^i_{mj} belong to C_i

$$\hat{P}(C_i \mid \mathbf{x}, m, j) \equiv p^i_{mj} = \frac{N^i_{mj}}{N_{mj}} \qquad \mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log_2 p^i_{mj}$$

- Find the variable and split that min impurity (among all variables -- and split positions for numeric variables)

```
GenerateTree(𝒳)
    If NodeEntropy(𝒳) < θ_I  /* eq. 9.3
        Create leaf labelled by majority class in 𝒳
        Return
    i ← SplitAttribute(𝒳)
    For each branch of x_i
        Find 𝒳_i falling in branch
        GenerateTree(𝒳_i)
SplitAttribute(𝒳)
    MinEnt ← MAX
    For all attributes i = 1, ..., d
        If x_i is discrete with n values
            Split 𝒳 into 𝒳_1, ..., 𝒳_n by x_i
            e ← SplitEntropy(𝒳_1, ..., 𝒳_n)  /* eq. 9.8 */
            If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
            For all possible splits
                Split 𝒳 into 𝒳_1, 𝒳_2 on x_i
                e ← SplitEntropy(𝒳_1, 𝒳_2)
                If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Regression Trees

- Error at node m:

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

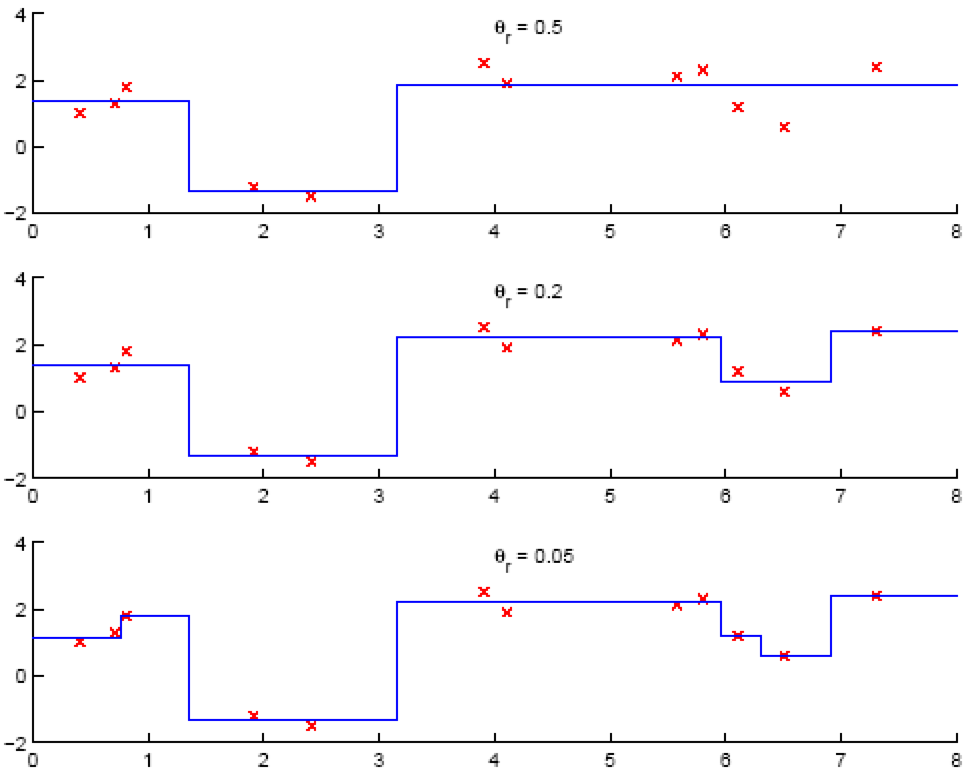
$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\mathbf{x}^t\right) \qquad g_m = \frac{\sum_t b_m\left(\mathbf{x}^t\right) r^t}{\sum_t b_m\left(\mathbf{x}^t\right)}$$

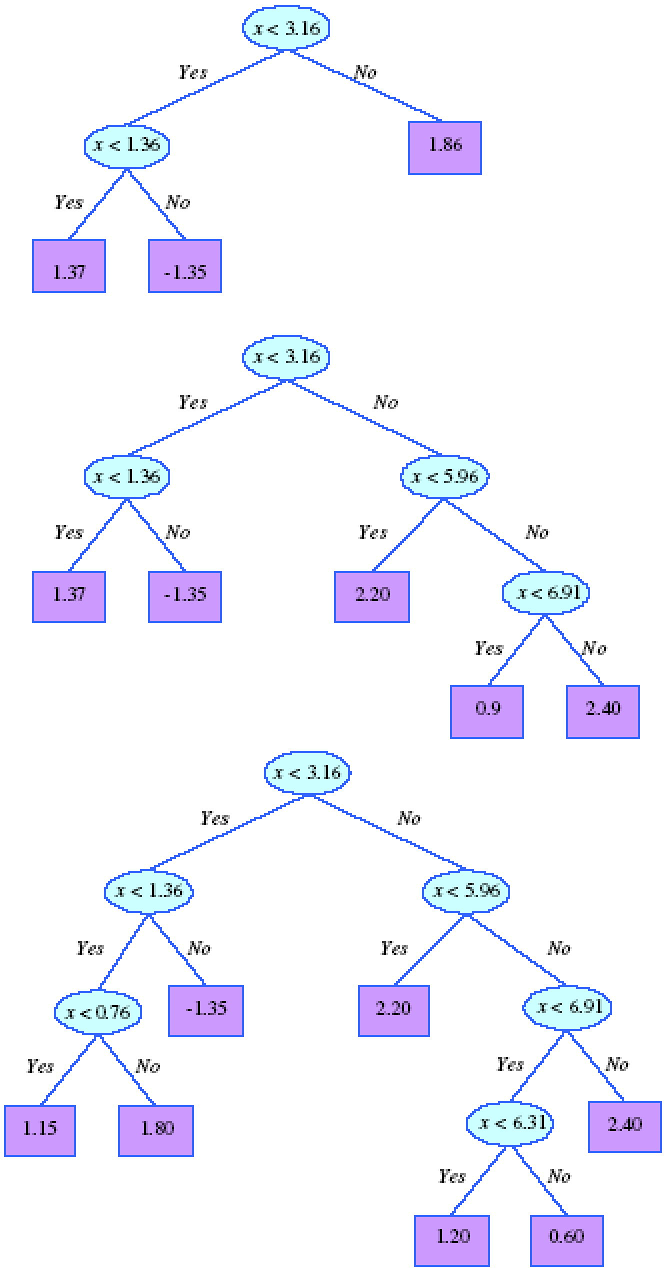
- After splitting:

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

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

Model Selection in Trees



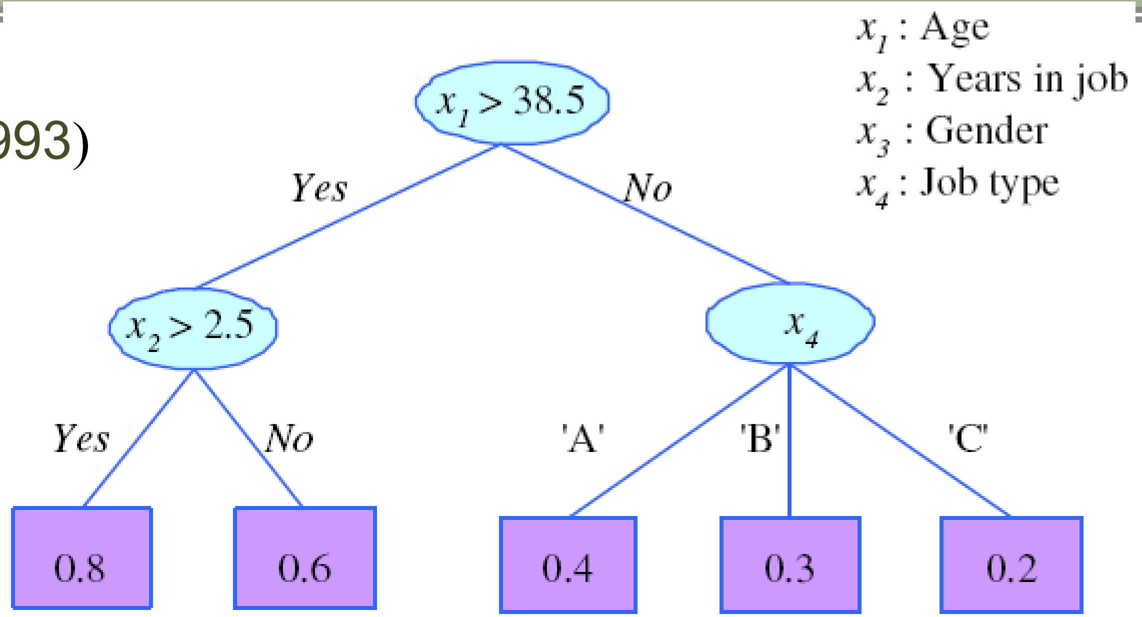


Pruning Trees

- 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

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 (Fürnkrantz and Widmer, 1994), Ripper (Cohen, 1995)

```
Ripper(Pos,Neg,k)
   RuleSet ← LearnRuleSet(Pos,Neg)
   For k times
      RuleSet ← OptimizeRuleSet(RuleSet,Pos,Neg)
LearnRuleSet(Pos,Neg)
   RuleSet ← ∅
   DL ← DescLen(RuleSet,Pos,Neg)
   Repeat
      Rule ← LearnRule(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 = ∅
   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+
        ReplaceRule(RuleSet,Pos,Neg),Pos,Neg)
      DL2 ← DescLen(RuleSet-Rule+
        ReviseRule(RuleSet,Rule,Pos,Neg),Pos,Neg)
      If DL1=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
            add ReplaceRule(RuleSet,Pos,Neg)
      Else If DL2=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
            add ReviseRule(RuleSet,Rule,Pos,Neg)
  Return RuleSet
```

Multivariate Trees