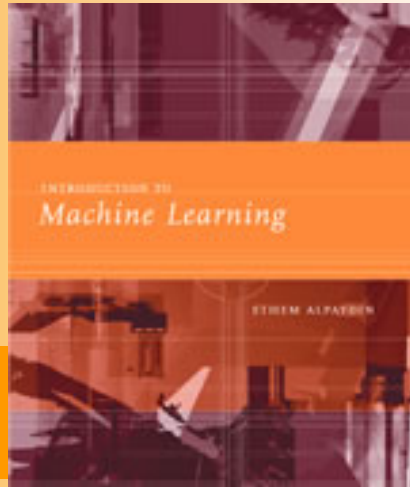




Lecture Slides for

INTRODUCTION TO

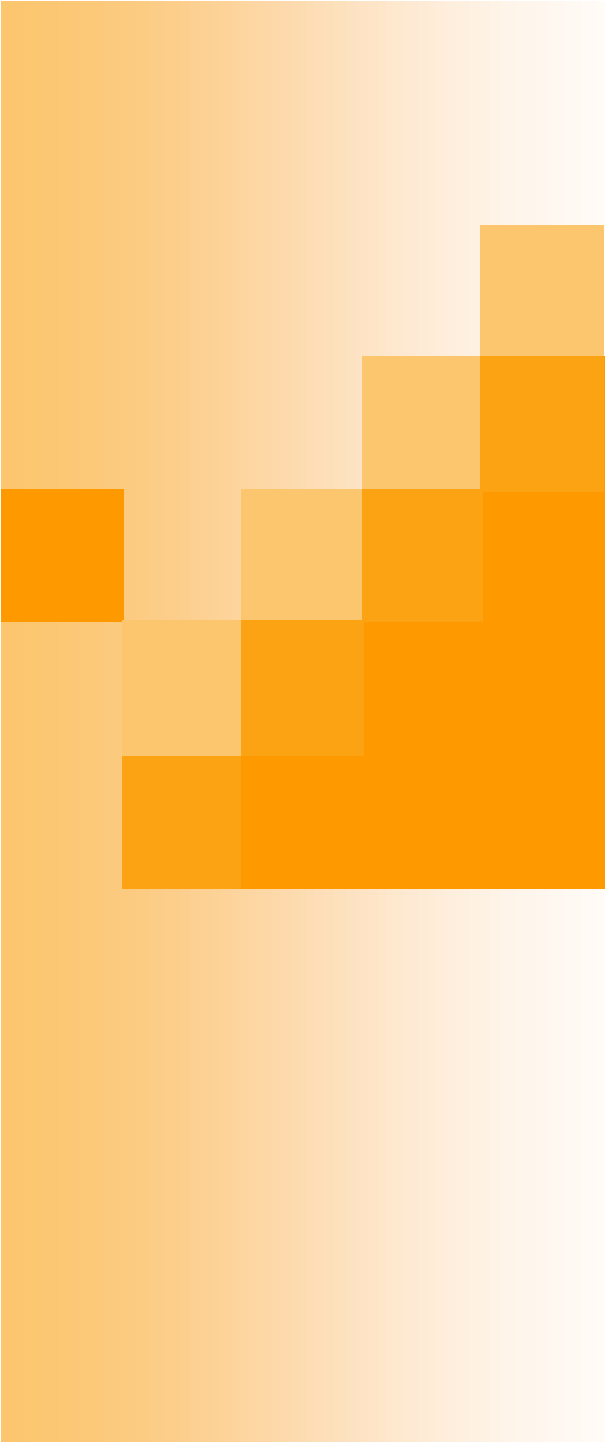
Machine Learning

ETHEM ALPAYDIN

© The MIT Press, 2004

alpaydin@boun.edu.tr

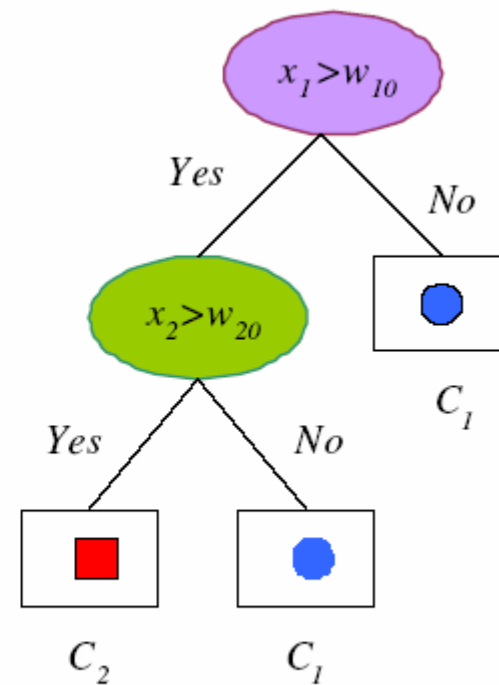
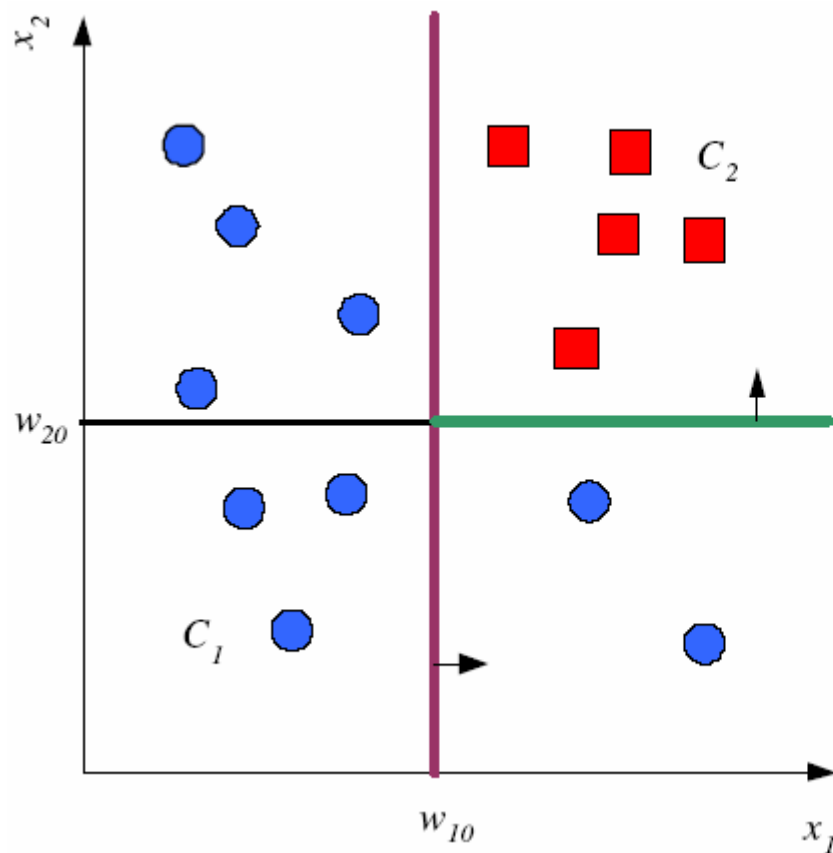
<http://www.cmpe.boun.edu.tr/~ethem/i2ml>



CHAPTER 9:

Decision 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, $\mathbf{x}$
- 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)

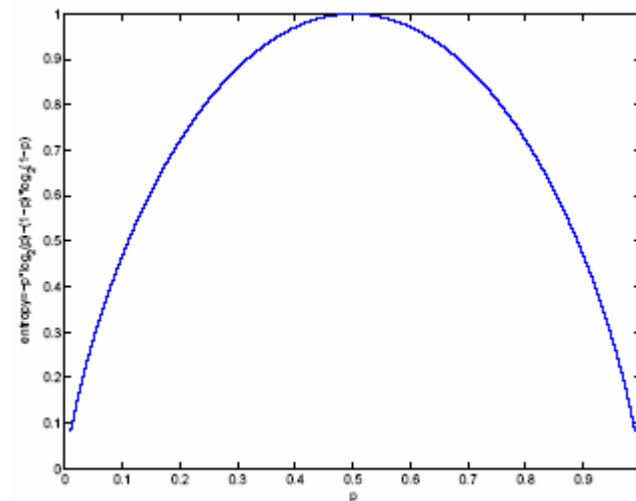
Classification Trees (ID3, CART, C4.5)

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

$$\hat{P}(C_i|\mathbf{x}, m) \equiv p_m^i = \frac{N_m^i}{N_m}$$

- Node m is **pure** if p_m^i is 0 or 1
- Measure of **impurity** is **entropy**

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





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}^i belong to C_i

$$\hat{P}(C_i | \mathbf{x}, m, j) \equiv p_{mj}^i = \frac{N_{mj}^i}{N_{mj}}$$

$$\mathcal{I}'_m = - \sum_{j=1}^n \frac{N_{mj}}{N_m} \sum_{i=1}^K p_{mj}^i \log p_{mj}^i$$

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

GenerateTree($\mathcal{X}$)

If NodeEntropy($\mathcal{X}$) $< \theta_I$ /* eq. 9.3

Create leaf labelled by majority class in $\mathcal{X}$

Return

$i \leftarrow \text{SplitAttribute}(\mathcal{X})$

For each branch of $\mathbf{x}_i$

Find $\mathcal{X}_i$ falling in branch

GenerateTree($\mathcal{X}_i$)

SplitAttribute($\mathcal{X}$)

MinEnt $\leftarrow$ MAX

For all attributes $i = 1, \dots, d$

If $\mathbf{x}_i$ is discrete with n values

Split $\mathcal{X}$ into $\mathcal{X}_1, \dots, \mathcal{X}_n$ by $\mathbf{x}_i$

$e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \dots, \mathcal{X}_n)$ /* eq. 9.8 */

If $e < \text{MinEnt}$ MinEnt $\leftarrow e$; bestf $\leftarrow i$

Else /* $\mathbf{x}_i$ is numeric */

For all possible splits

Split $\mathcal{X}$ into $\mathcal{X}_1, \mathcal{X}_2$ on $\mathbf{x}_i$

$e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \mathcal{X}_2)$

If $e < \text{MinEnt}$ MinEnt $\leftarrow e$; bestf $\leftarrow i$

Return bestf

Regression Trees

- Error at node m :

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

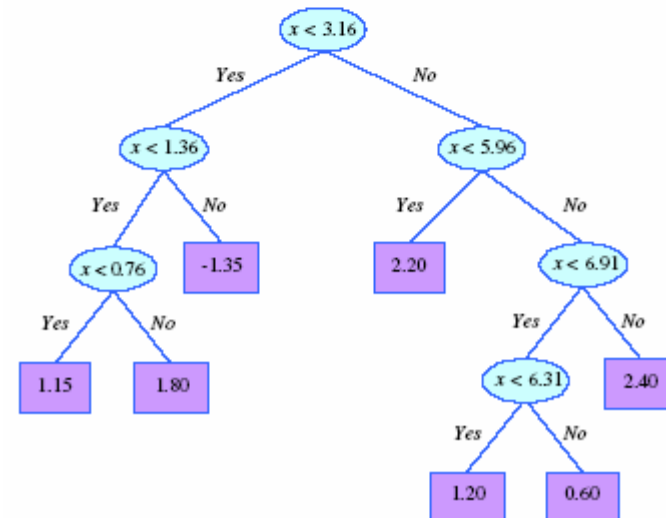
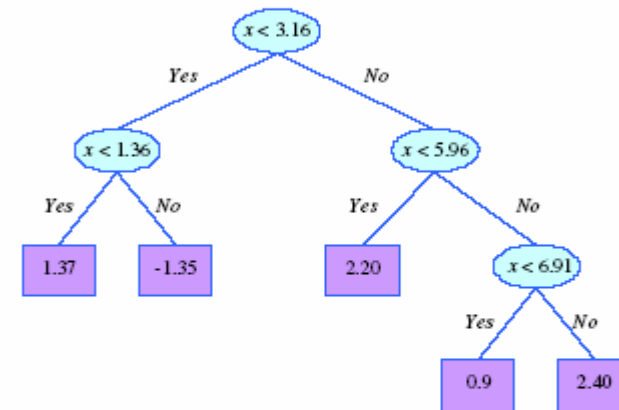
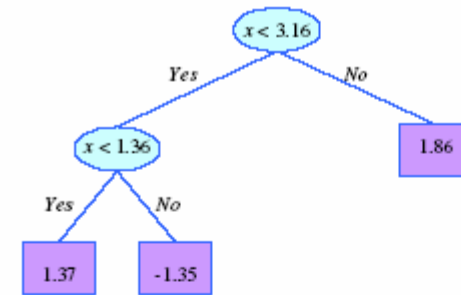
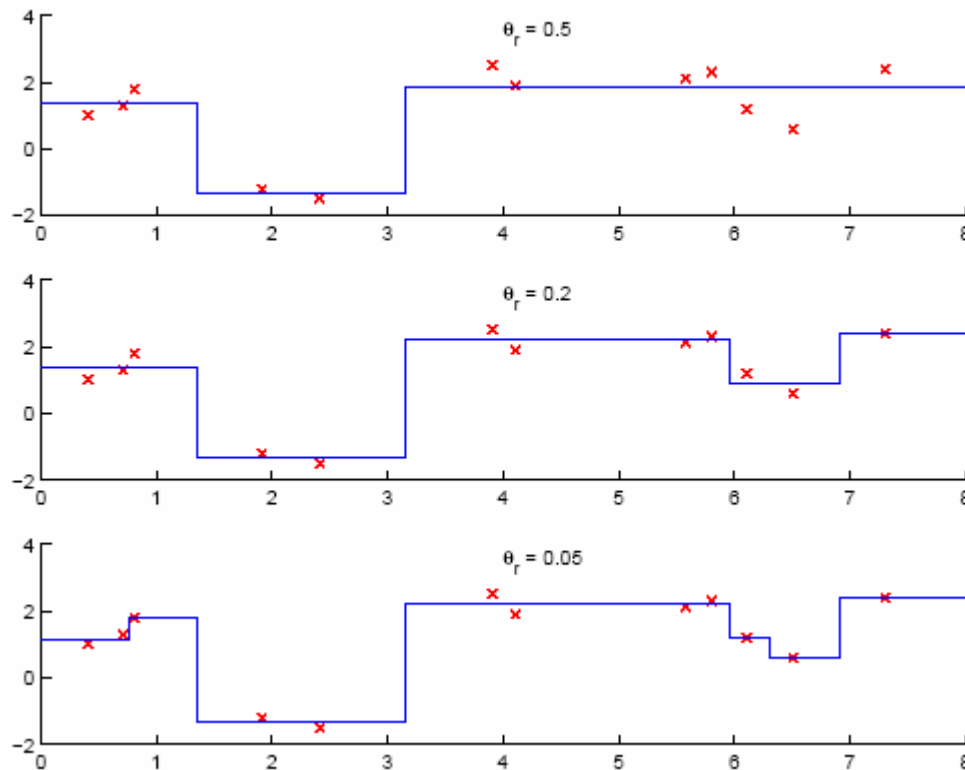
$$E_m = \frac{1}{N_m} \sum_t (r^t - g_m)^2 b_m(\mathbf{x}^t) \quad g_m = \frac{\sum_t b_m(\mathbf{x}^t) r^t}{\sum_t b_m(\mathbf{x}^t)}$$

- After splitting:

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

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

Model Selection in Trees:



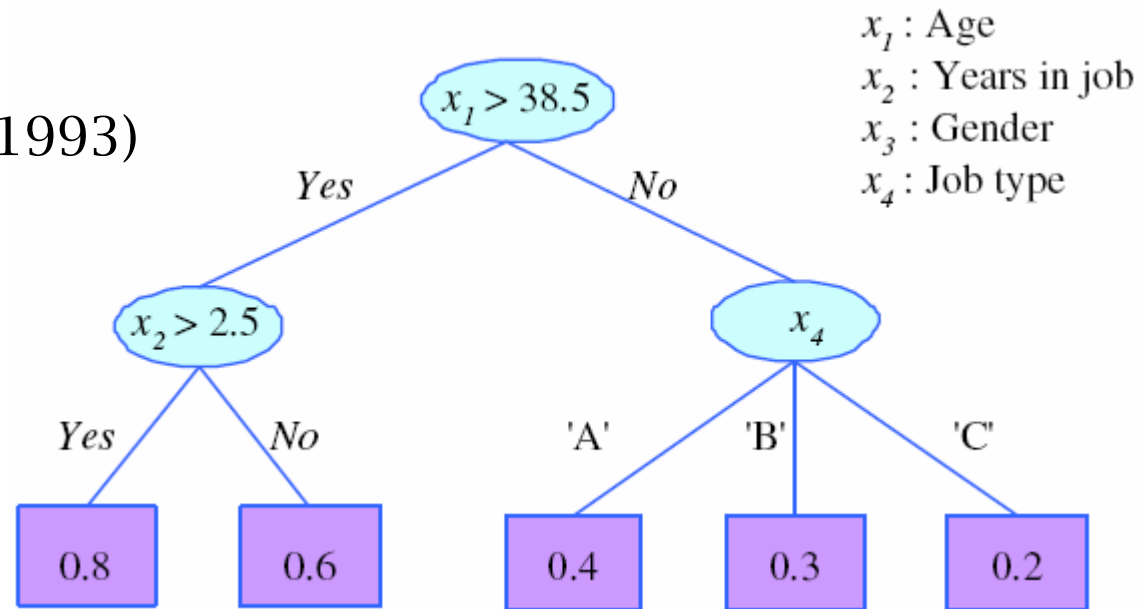


Pruning Trees

- 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)

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 $\leq$ 2.5) THEN $y = 0.6$
R3: IF (age $\leq$ 38.5) AND (job-type='A') THEN $y = 0.4$
R4: IF (age $\leq$ 38.5) AND (job-type='B') THEN $y = 0.3$
R5: IF (age $\leq$ 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)



```

Ripper(Pos, Neg, k)
  RuleSet  $\leftarrow$  LearnRuleSet(Pos, Neg)
  For  $k$  times
    RuleSet  $\leftarrow$  OptimizeRuleSet(RuleSet, Pos, Neg)
LearnRuleSet(Pos, Neg)
  RuleSet  $\leftarrow \emptyset$ 
  DL  $\leftarrow$  DescLen(RuleSet, Pos, Neg)
  Repeat
    Rule  $\leftarrow$  LearnRule(Pos, Neg)
    Add Rule to RuleSet
    DL'  $\leftarrow$  DescLen(RuleSet, Pos, Neg)
    If DL' > DL + 64
      PruneRuleSet(RuleSet, Pos, Neg)
      Return RuleSet
    If DL' < DL DL  $\leftarrow$  DL'
    Delete instances covered from Pos and Neg
  Until Pos =  $\emptyset$ 
  Return RuleSet

```



```

PruneRuleSet(RuleSet, Pos, Neg)
  For each Rule  $\in$  RuleSet in reverse order
    DL  $\leftarrow$  DescLen(RuleSet, Pos, Neg)
    DL'  $\leftarrow$  DescLen(RuleSet-Rule, Pos, Neg)
    IF DL' < DL Delete Rule from RuleSet
  Return RuleSet

OptimizeRuleSet(RuleSet, Pos, Neg)
  For each Rule  $\in$  RuleSet
    DL0  $\leftarrow$  DescLen(RuleSet, Pos, Neg)
    DL1  $\leftarrow$  DescLen(RuleSet-Rule+
      ReplaceRule(RuleSet, Pos, Neg), Pos, Neg)
    DL2  $\leftarrow$  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

