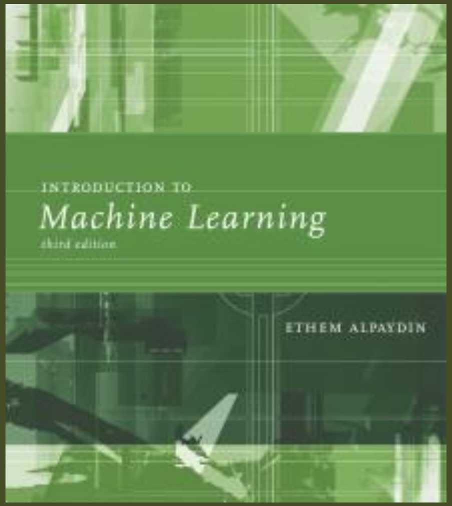


Lecture Slides for

INTRODUCTION TO MACHINE LEARNING

3RD EDITION

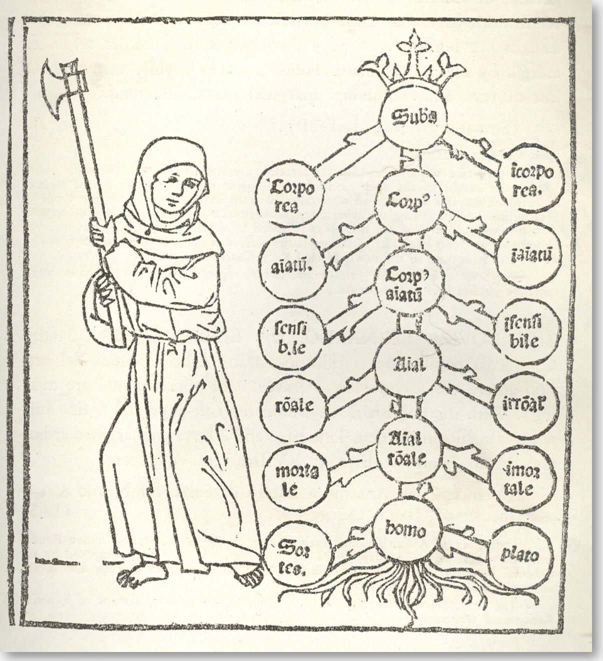
ETHEM ALPAYDIN

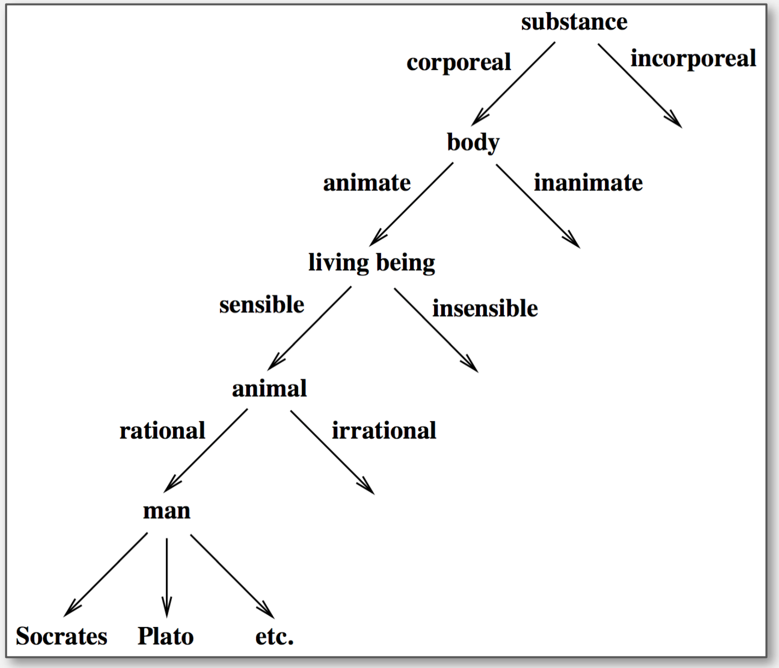
alpaydin@boun.edu.tr
http://www.cmpe.boun.edu.tr/~ethem/i2ml3e

CHAPTER 9:

DECISION TREES

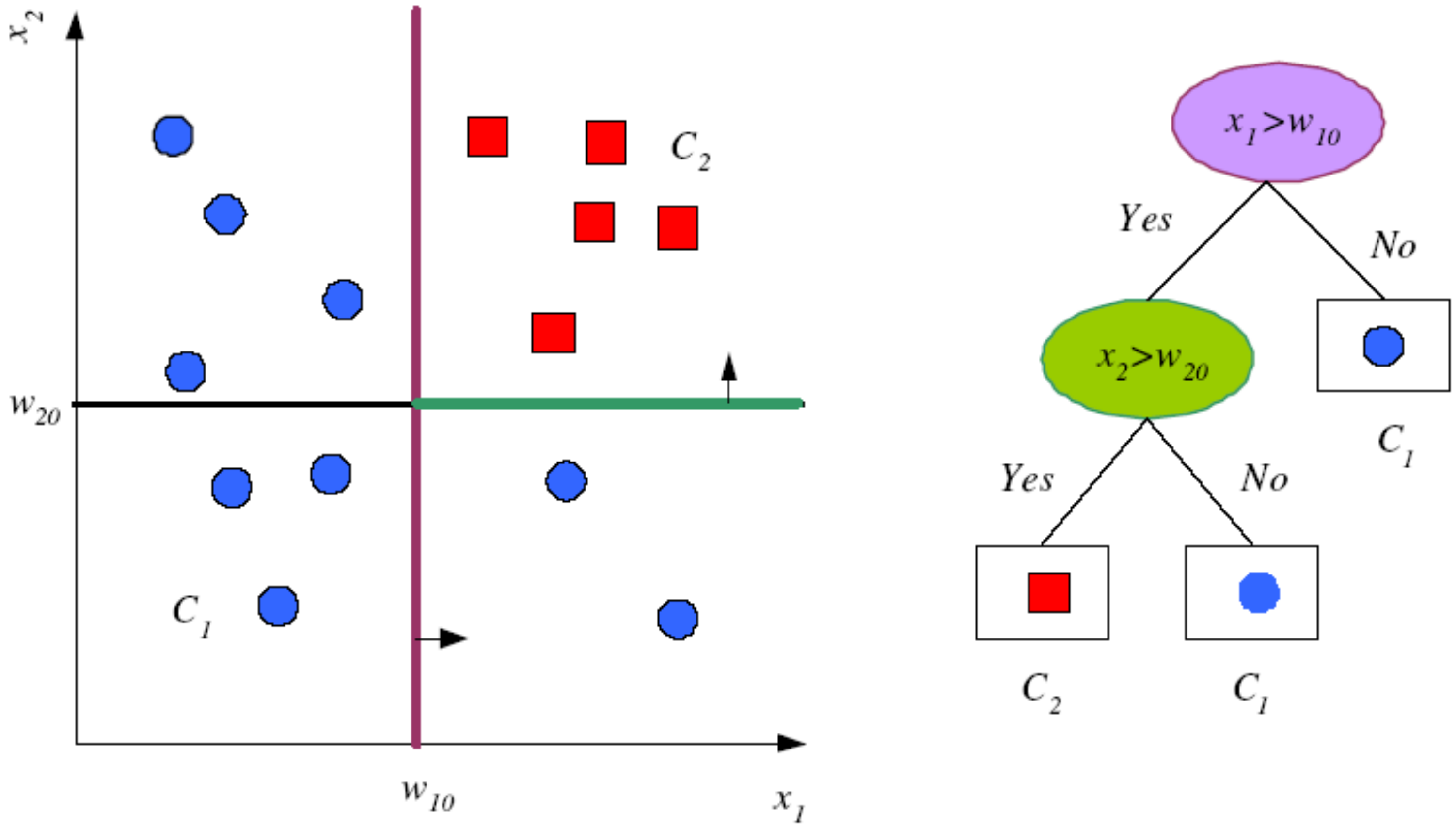
TREE OF PORPHYRY





- Porphyry of Tyre (c.234 – c.305 CE): Platonic philosopher
- Concept definition by dichotomy (< Plato)

Tree Uses Nodes and Leaves



SUITABLE PROBLEMS FOR DECISION TREE LEARNING

- Instances are represented by attribute-value pairs
- Target function has discrete output values
- Disjunctive descriptions may be required
- Training data may contain errors
- Training data may contain missing attribute values

(Mitchell, 1997)

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)

SHANNON INFORMATION
(VERY BRIEFLY!)

- Information varies directly with surprise
- Information varies inversely with probability
- Information is additive
- $\therefore$ The information content of a message is proportional to the negative log of its probability

$$I\{s\} = -\lg \Pr\{s\}$$

ENTROPY

- Suppose have source S of symbols from ensemble $\{s_1, s_2, \ldots, s_N\}$
- Average information per symbol:

$$\sum_{k=1}^{N} \Pr\{s_k\} I\{s_k\} = \sum_{k=1}^{N} \Pr\{s_k\}\left(-\lg \Pr\{s_k\}\right)$$

- This is the *entropy* of the source:

$$H\{S\} = -\sum_{k=1}^{N} \Pr\{s_k\} \lg \Pr\{s_k\}$$

MAXIMUM AND MINIMUM ENTROPY

- Maximum entropy is achieved when all signals are equally likely

  No ability to guess; maximum surprise

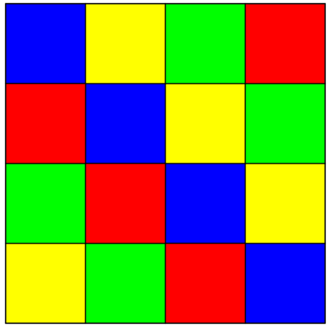
  $H_{max} = \lg N$

- Minimum entropy occurs when one symbol is certain and the others are impossible

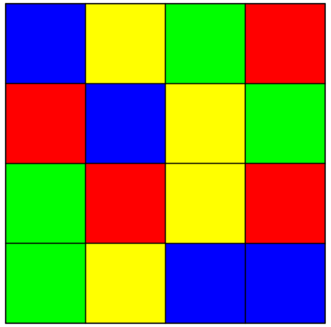
  No uncertainty; no surprise

  $H_{min} = 0$

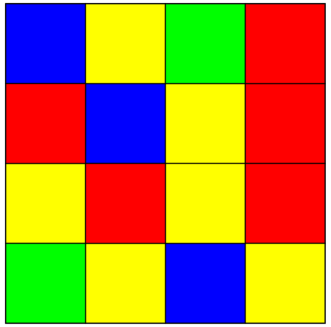
ENTROPY EXAMPLES



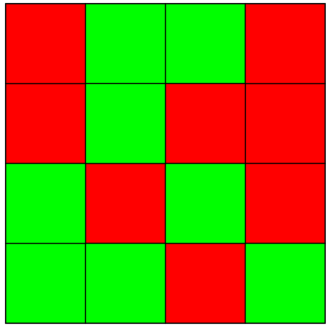
H = 2.0 bits



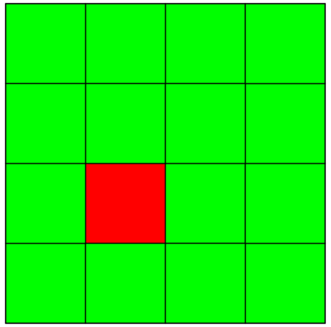
H = 2.0 bits



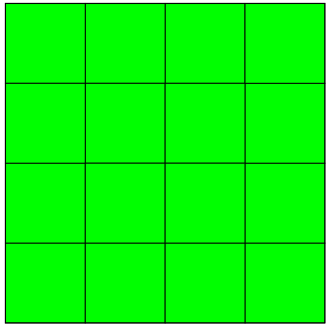
H = 1.9 bits



H = 1.0 bits



H = 0.3 bits



H = 0.0 bits

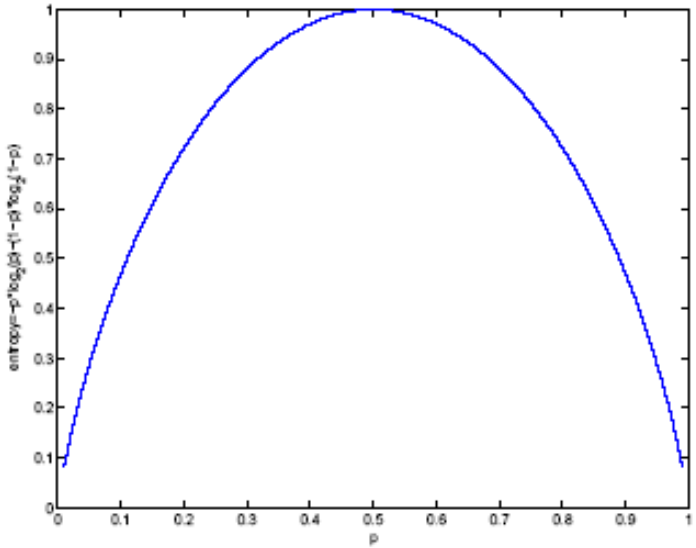
Classification Trees (ID3, CART, C4.5)

- 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 entropy

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



SOME IMPURITY MEASURES

- Requirements for 2-class impurity measure:
  $\varphi(p, 1-p) \geq 0$
  $\varphi(1/2\,, 1/2) \geq \varphi(p, 1-p)$ for any $p \in [0,1]$
  $\varphi(0,1) = 0 = \varphi(1,0)$
  $\varphi(p, 1-p)$ is increasing in p on $[0, 1/2]$ and decreasing in p on $[1/2\,, 1]$
- Entropy: $\varphi(p, 1-p) = -p\log_2 p - (1-p)\log_2(1-p)$
- Gini index: $\varphi(p, 1-p) = 2p(1-p)$
- Misclassification error: $\varphi(p, 1-p) = 1 - \max(p, 1-p)$

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 minimizes impurity (among all variables — and split positions for numeric variables) $\Rightarrow$ maximum decrease of impurity (greedy)

(edited by BJM)

NUMERIC ATTRIBUTE

- N_m data points reach node m
- There are $N_m - 1$ possible split points
- Best split will always be between values belonging to different classes
- Test halfway points between them to see which leads to highest purity (e.g. minimum entropy)

```
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*:

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

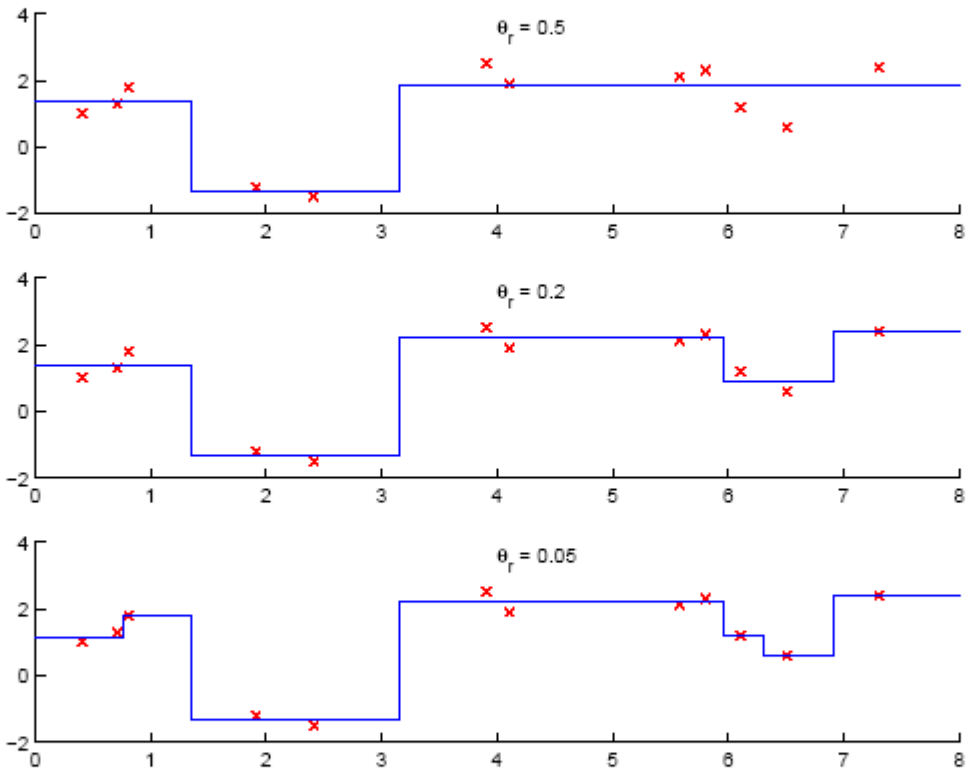
$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\mathbf{x}^t\right) \qquad g_m = \frac{\sum_t b_m\left(\mathbf{x}^t\right) r^t}{\sum_t b_m\left(\mathbf{x}^t\right)}$$

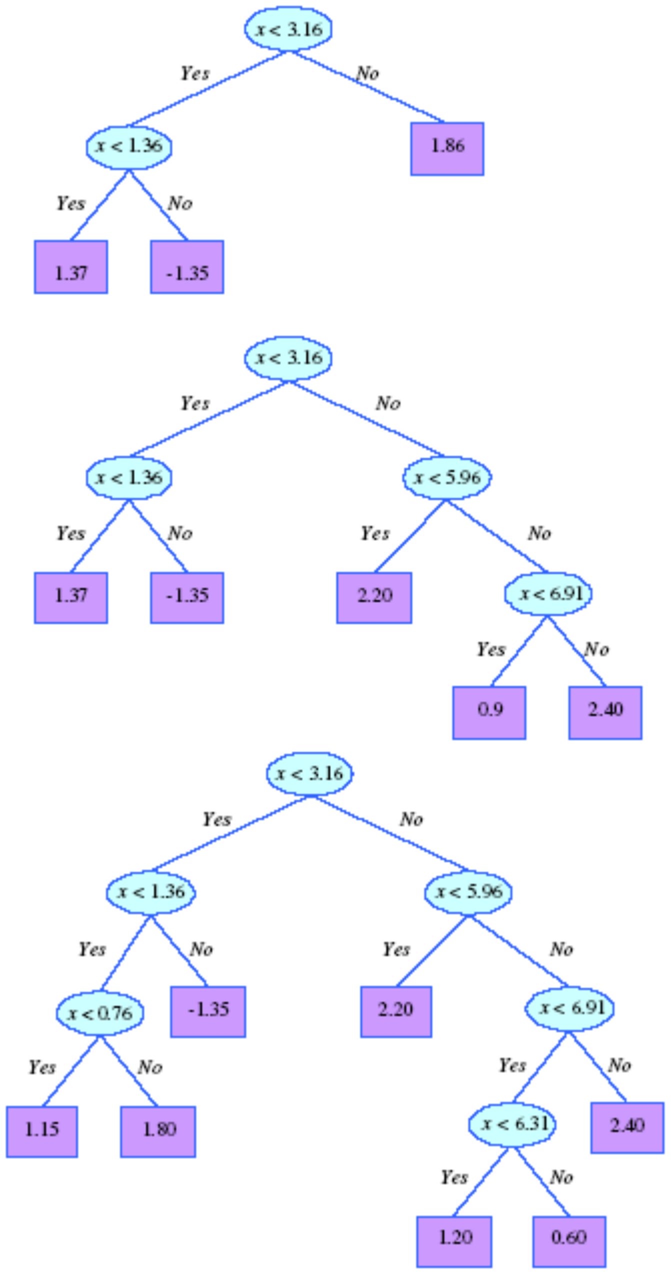
- After splitting:

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

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

Model Selection in Trees



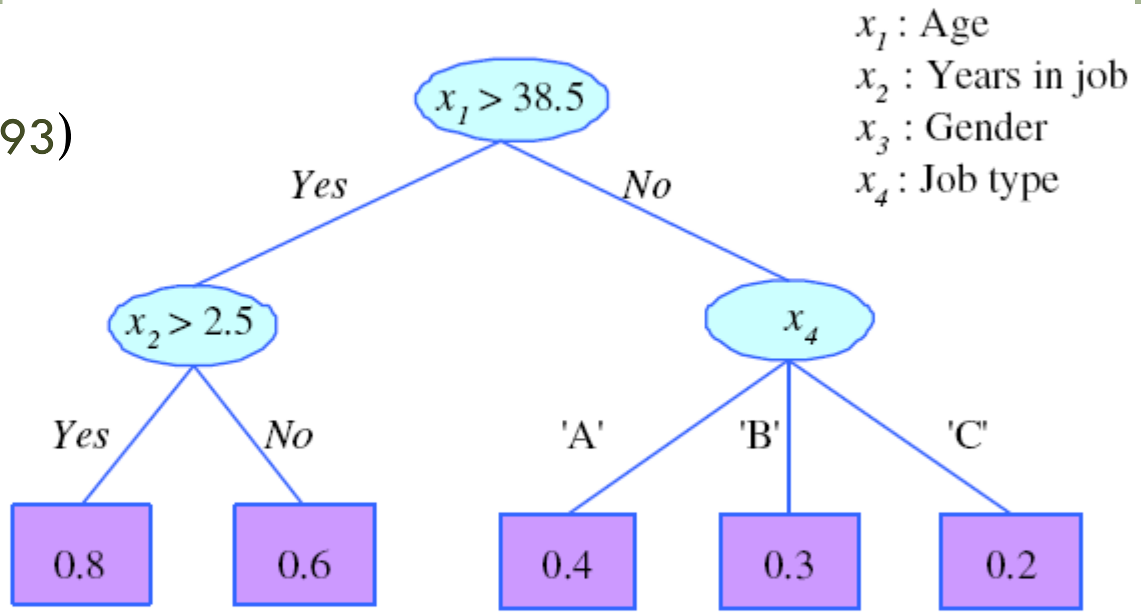


Pruning Trees

- 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

RULE POSTPRUNING

1. Infer DT from training set, until training data is fit as well as possible and allowing overfitting to occur
2. Convert learned DT into equivalent rule base (following paths from root to leaves)
3. Prune (generalize) each rule by removing any preconditions that result in improved accuracy on pruning set
4. Sort pruned rules by estimated accuracy, and consider them in this sequence when classifying instances

Learning Rules

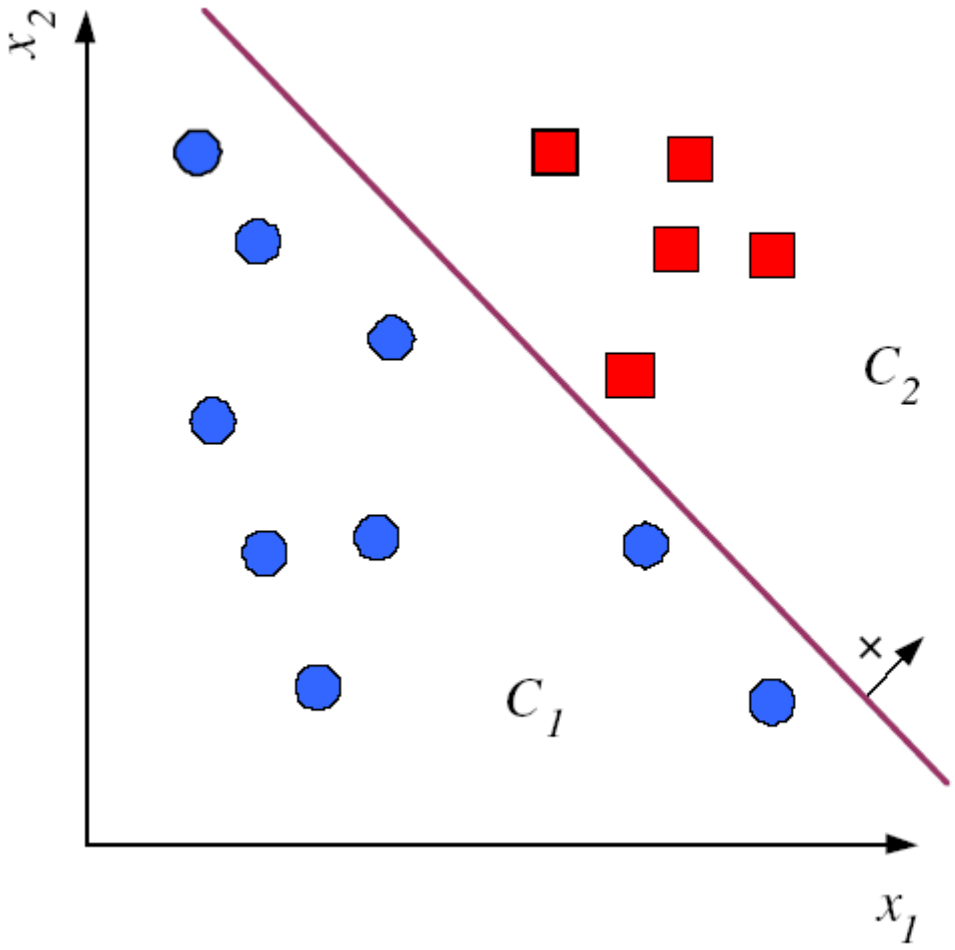
- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: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

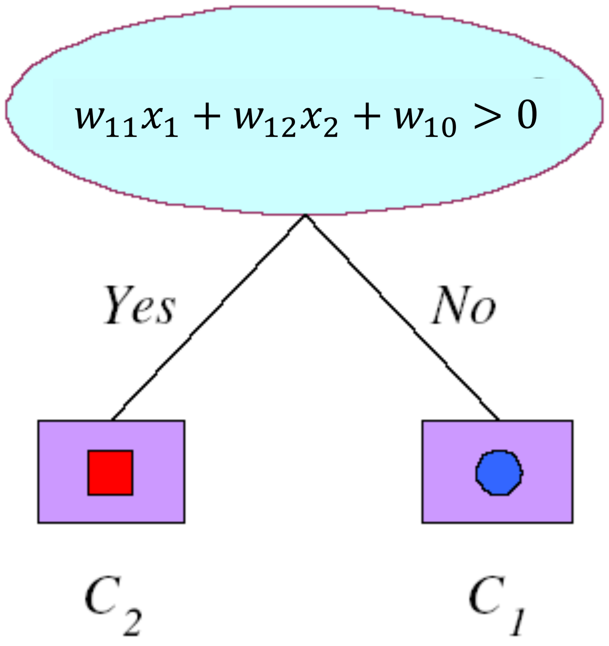
(edited by BJM)

```
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





DECISION TREES VS. INSTANCE-BASED METHODS

- Each leaf node corresponds to a "bin," except:
  - bins need not be the same size (as in Parzen windows)
  - bins need not contain equal numbers of training instances (as in kNN)
- Bin divisions are not based only on similarity in input space, but also on supervised output information
  - entropy or mean square error also used
- Thanks to tree structure, leaf ("bin") is found much faster with smaller number of comparisons
- Decision tree (once constructed) does not store whole training set but only structure of tree, the parameters of the decision nodes, and the output values in leaves
  - implies space complexity is much less than instance-based nonparametric methods (which store all training examples)

(Mitchell, 1997)

FURTHER OBSERVATIONS ON DECISION TREES AND RELATED METHODS

- Considers given features of instances (not some alternative representation)
- Considers the features one at a time
- Sequentially divides the feature space
- Produces explicit rules
- A plausible model of sequential, deliberative, rule-based decision making
- But not of ordinary, real-time cognition and recognition, which is holistic and considers many microfeatures simultaneously

READ ALPAYDIN CH. 10