FUNDAMENTALS OF MACHINE LEARNING

DECISION TREES

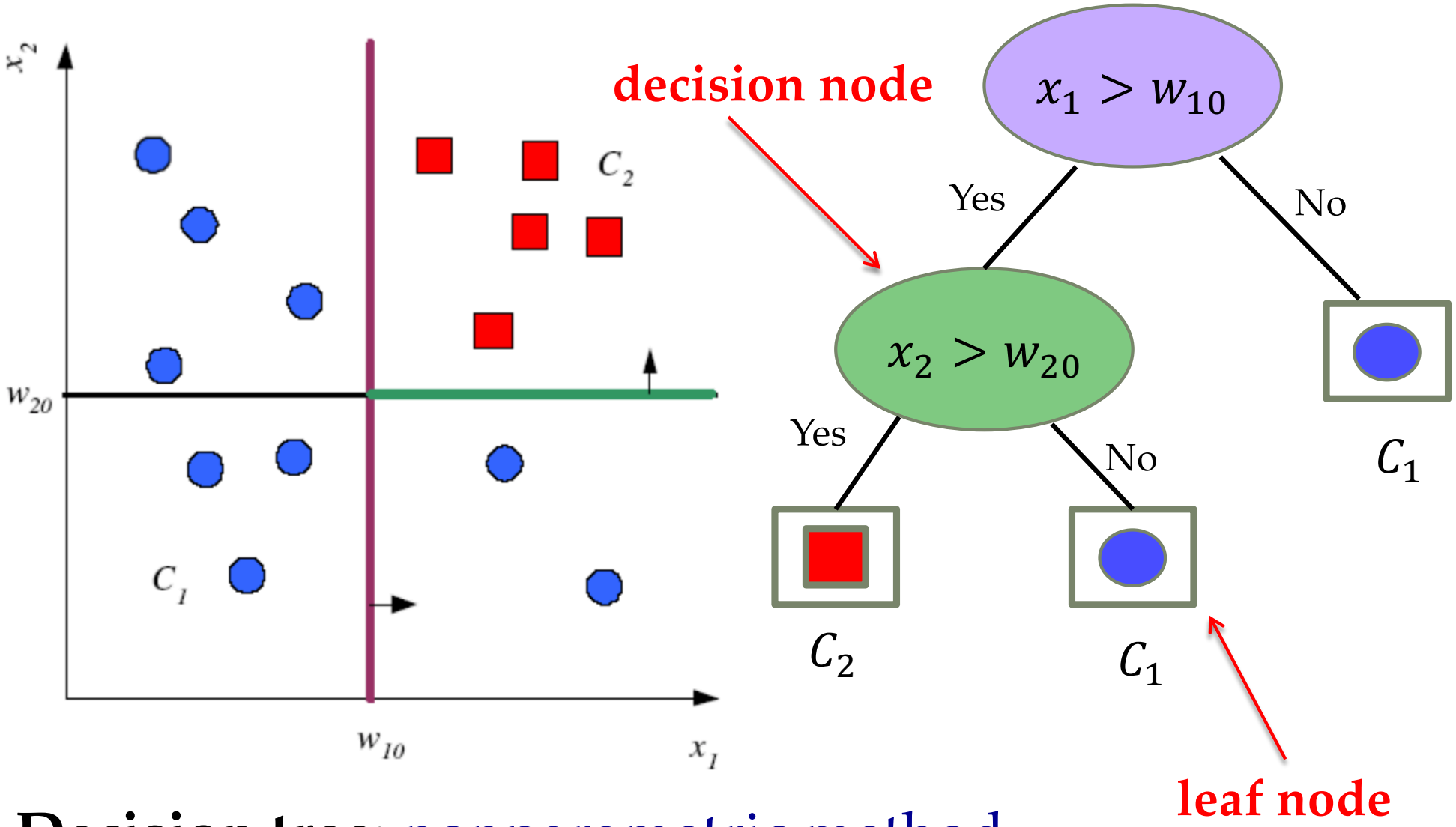
CSCI3320

Prof. John C.S. Lui, CSE Department, CUHK
Introduction to Machine Learning

Decision Tree

- An efficient ***non-parametric method*** for **supervised learning**, which is often used for classification and regression
- **Decision tree** is a ***hierarchical data structure*** which relies on **divide-and-conquer strategy**
- We will explore various learning algorithms to build the tree from a given labeled training sample
- Also will learn how to **convert a tree to a set of rules**
- **Parametric method**: define a model over whole input space, learn its parameters from all training data
- **Nonparametric method**: divide input space into regions (clusters) defined by some distance measure. For each input, the corresponding local model computed from the training data in that region is used.

Tree Uses Nodes and Leaves



Decision tree: nonparametric method

Divide and Conquer

- **Internal decision nodes**
  - **Univariate: Uses a single attribute (or single dimension),** x_i
    - **Numeric** x_i **:** Binary split : $x_i > w_m$
    - **Discrete** x_i **:** *n*-way split for *n* possible values (*example*)
  - **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**

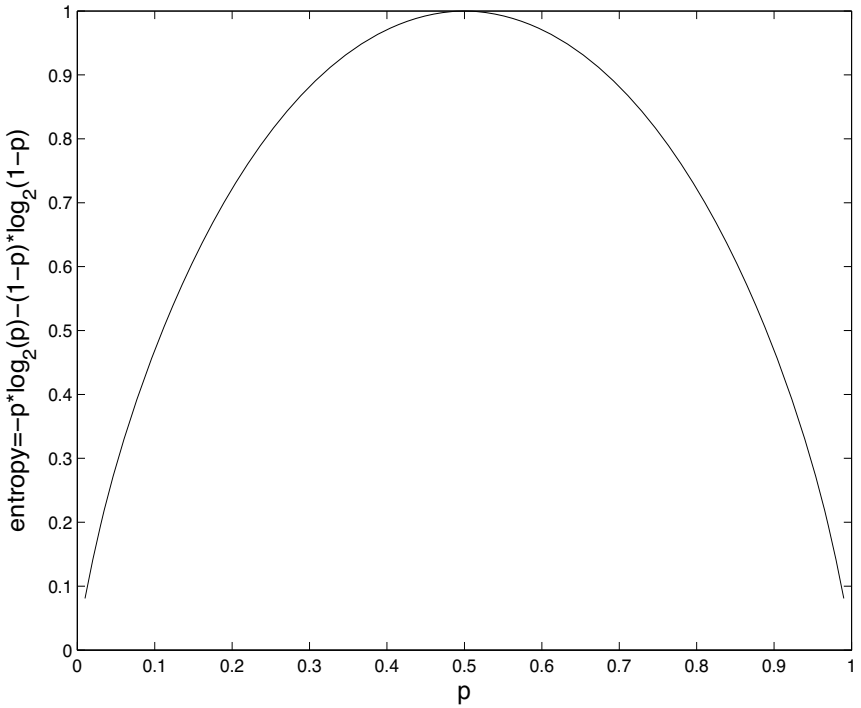
Univariate Classification Trees (ID3,CART,C4.5)

- For *classification tree*, the goodness of a split is measured by the ***impurity measure***.
- N=# of instances in the root, N_m=# of instances at node m, N_m^i= # of instances of N_m belongs to class C_i
- The probability of class C_i is

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

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

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



Explain entropy

Univariate Classification Trees (ID3,CART,C4.5)

- There are many possible *measures*.
- For a two class problem, we have $p^1 \equiv p$ and $p^2 = 1 - p$,
- A nonnegative function to measure impurity of a split

$$\phi(p, 1-p)$$

- The function needs to satisfy:

$$\phi(1/2, 1/2) \geq \phi(p, 1-p), \text{ for any } p \in [0,1].$$
$$\phi(0,1) = \phi(1,0) = 0.$$
$$\phi(p, 1-p) \text{ is increasing in } p \text{ on } [0, 1/2] \text{ and decreasing in } p \text{ on } [1/2, 1].$$

- **Possible functions**
  - **Entropy:** $\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2 (1-p)$
  - **Gini Index:** $\phi(p, 1-p) = 2p(1-p)$
  - **Misclassification error:** $\phi(p, 1-p) = 1 - \max(p, 1-p)$

Best Split for univariate classification tree

- If node *m* is pure, generate a leaf and stop, otherwise split and continue recursively
- Define impurity **after split**: N_{mj} of N_m take branch j. N^i_{mj} belong to C_i

explain

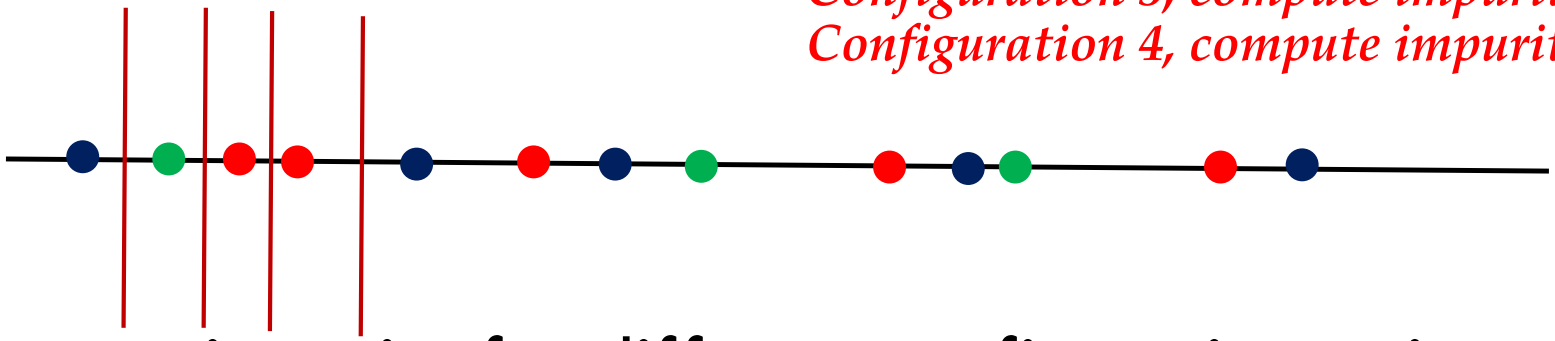
$$\hat{P}(C_i|\boldsymbol{x}, m, j) \equiv p^i_{mj} = \frac{N^i_{mj}}{N_{mj}} \qquad \mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log_2 p^i_{mj}$$

- For discrete attributes, there are *n* outcomes
- Find the variable and split that minimize impurity (among all variables -- and split positions for numeric variables)

How to do the splitting?

- Consider doing a splitting on feature x_i
- We have the following training instances

Configuration 1, compute impurity
Configuration 2, compute impurity
Configuration 3, compute impurity
Configuration 4, compute impurity….



- Computer impurity for different configurations using previous formula
- Choose the configuration which has the minimum
- Note: There are many ways to do splitting: exhaustive, binary search…etc

```
GenerateTree(X)
    If NodeEntropy(X)< θ_I /* equation 9.3 */ (or entropy)
      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) /* equation 9.8 */ (or I'_m)
        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
```

complexity parameter

Why is this a heuristic?

Univariate Regression Trees

- Construction is similar to classification tree, except replace entropy by some regression measures
- Let $\mathcal{X}_m$ be the set of instances which reached node *m*. We *also* define error (or variance) at node *m* as:

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

where $N_m = |\mathcal{X}_m| = \sum_t b_m(\boldsymbol{x}^t)$

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

$$g_m = \frac{\sum_t b_m(\boldsymbol{x}^t) r^t}{\sum_t b_m(\boldsymbol{x}^t)}$$

- If at node *m*, the error is acceptable (or $E_m < \theta_r$), then a leaf node is created and it stores the g_m value
- If the error is not acceptable, data reaching node *m* is split further to reduce the sum of errors in the branches

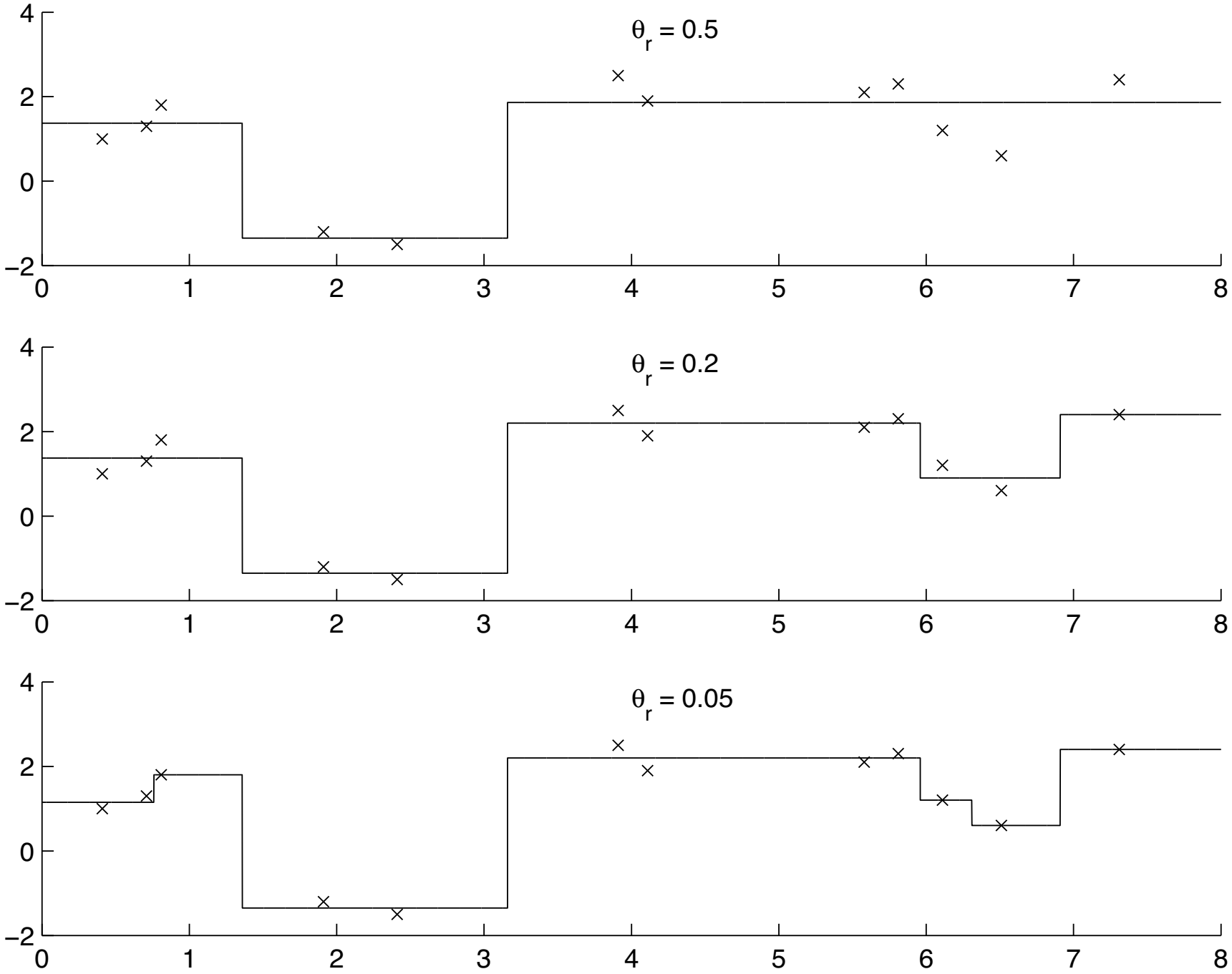
Univariate Regression Trees

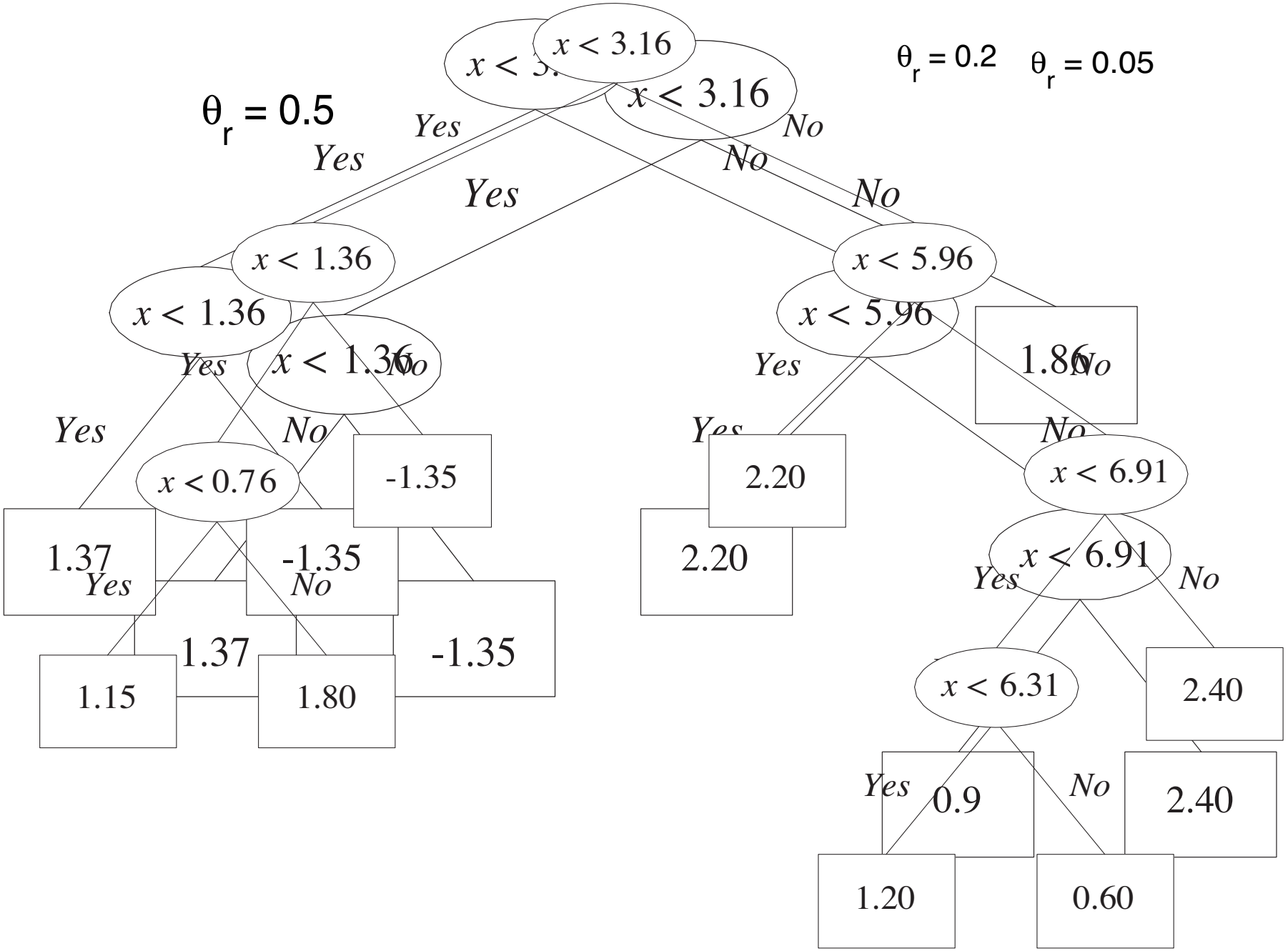
- To split at a node, we look for the attribute (and split threshold for a numeric attribute) that minimizes the error. Then continue recursively.
- Let $\mathcal{X}_{mj}$ be the subset of $\mathcal{X}_m$ taking branch j : $\cup_{j=1}^{n} \mathcal{X}_{mj} = \mathcal{X}_m$
- Define error as:

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

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

- Find lowest E'_m . For tree construction, replace entropy calculation with E'_m in the alg, and θ_I by θ_r


θr = 0.5
θr = 0.2
θr = 0.05
4
2
0
−2
0
1
2
3
4
5
6
7
8

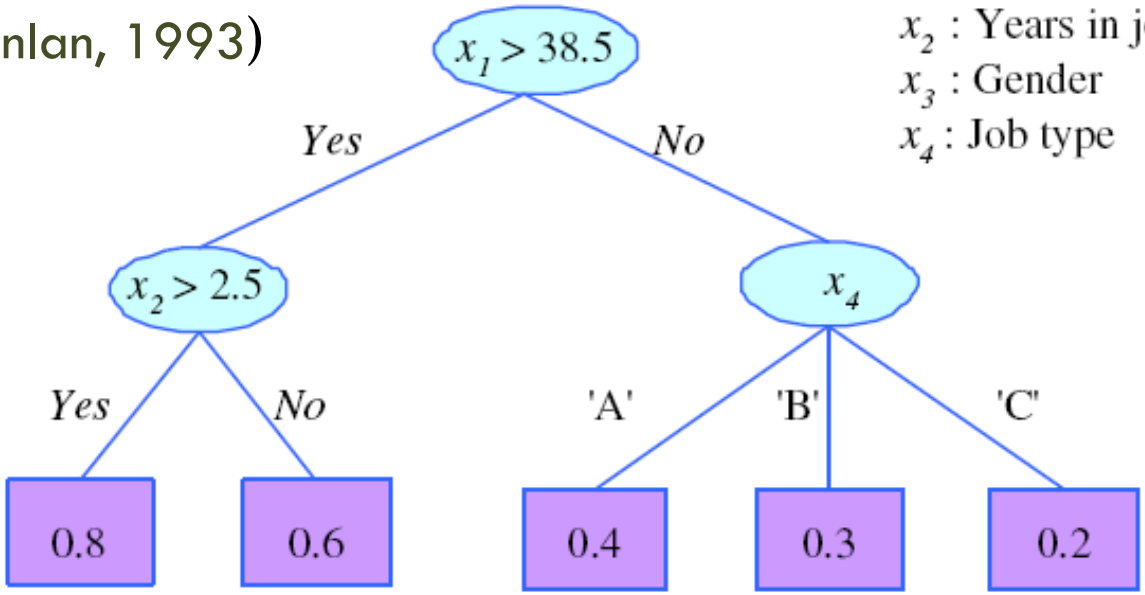

θr = 0.5
θr = 0.2
θr = 0.05
x < 3.16
x < 3.16
Yes
No
Yes
No
Yes
No
x < 1.36
x < 5.96
x < 1.36
x < 5.96
Yes
No
Yes
1.86
Yes
No
No
x < 0.76
-1.35
2.20
x < 6.91
1.37
-1.35
2.20
x < 6.91
Yes
No
Yes
No
1.37
x < 6.31
2.40
-1.35
1.15
1.80
Yes
No
0.9
2.40
1.20
0.60

Pruning Trees

- Remove sub-trees for better generalization (decrease variance)
  - **Pre-pruning**: Early stopping
  - **Post-pruning**: Grow the whole tree then prune subtrees that overfit on the pruning set
- Pre-pruning is faster, post-pruning is more accurate (requires a separate pruning set)

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



x_1 : Age
x_2 : Years in job
x_3 : Gender
x_4 : Job type

Use decision tree
As **feature extraction**.
In the figure, we
Don't need x_3

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 from Data Directly

- 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 conditions** on discrete or numeric ***attributes***
- Rule is said to cover an example if all terms of the rule evaluate to true for that example
- Once a rule is added to the rule base, all training samples covered by the rule are removed, the process continues
- **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 by Rule from Pos and Neg
    Until Pos = ∅
    Return RuleSet
```

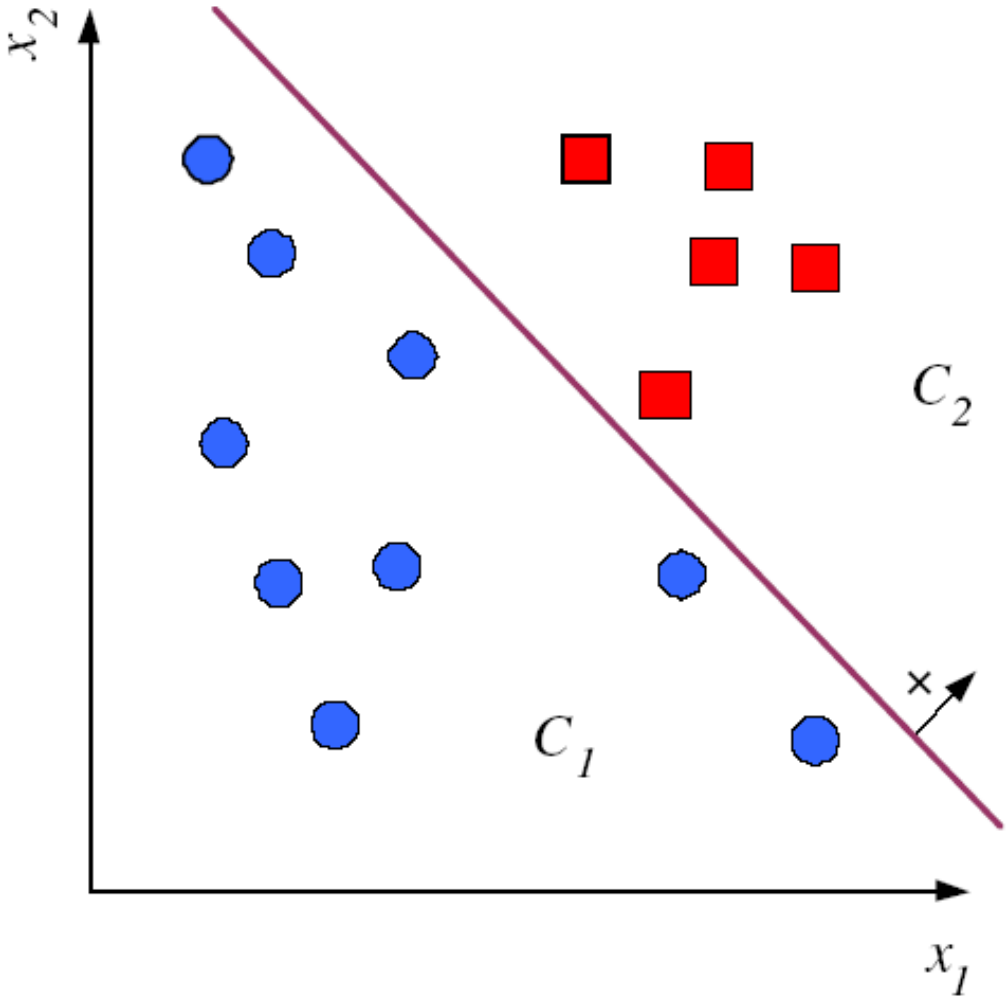
```
PruneRuleSet(RuleSet,Pos,Neg)
    For each Rule ∈ RuleSet in reverse order
        DL ← DescLen(RuleSet,Pos,Neg)
        DL' ← DescLen(RuleSet-Rule,Pos,Neg)
        IF DL'<DL Delete Rule from RuleSet
    Return RuleSet
OptimizeRuleSet(RuleSet,Pos,Neg)
    For each Rule ∈ RuleSet
        DL0 ← DescLen(RuleSet,Pos,Neg)
        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
```

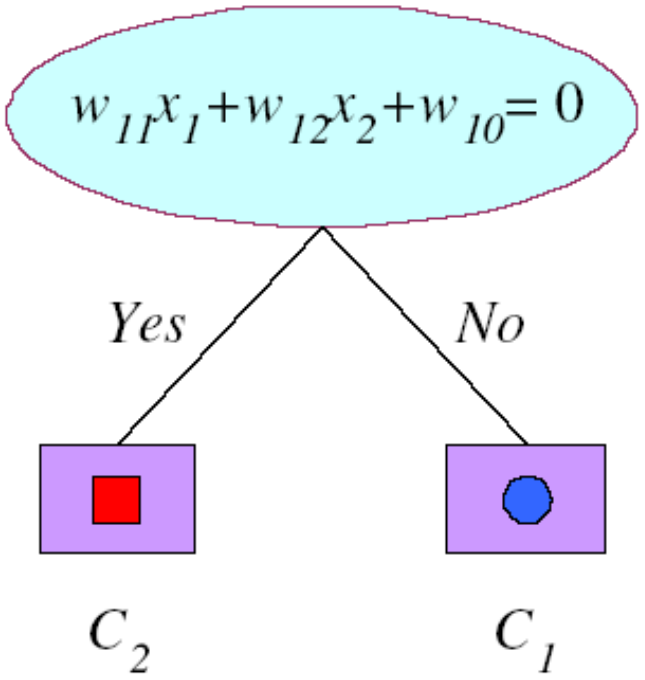
Multivariate Trees

- Unlike univariate tree, at a decision node, **ALL** input dimensions can be used
- When all inputs are numeric, a binary linear multivariate node is defined as: $f_m(\boldsymbol{x}) : \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0$
- The above equation defines a *hyperplane* (see figure)
- For discrete attributes should be represented by 0/1 dummy numeric variables
- Keep dividing until we have leaf nodes which are defined by a *polyhedra* in the input space

Multivariate Trees

Linear multivariate hyperplane





Multivariate Trees

- Instead of linear multivariate discriminant, we can use nonlinear multivariate discriminant at a node.
- Example: *quadratic multivariate discriminant*

$$f_m(\boldsymbol{x}) : \boldsymbol{x}^T \mathbf{W}_m \boldsymbol{x} + \boldsymbol{w}_m^T \boldsymbol{x} + w_{m0} > 0$$

- Example: *sphere node*

$$f_m(\boldsymbol{x}) : \|\boldsymbol{x} - \boldsymbol{c}_m\| \leq \alpha_m$$

- Many other discriminant methods, but note that these are all "heuristics"