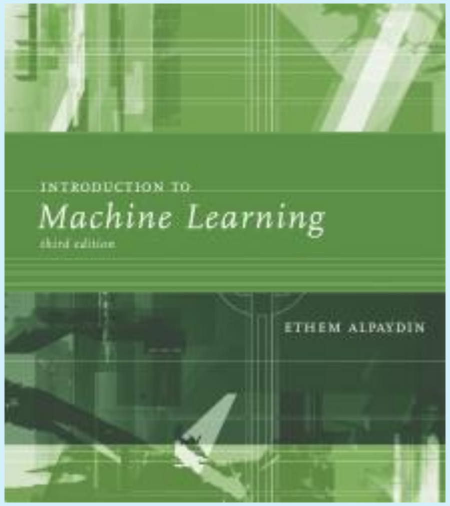


Lecture Slides for

INTRODUCTION TO MACHINE LEARNING

3RD EDITION

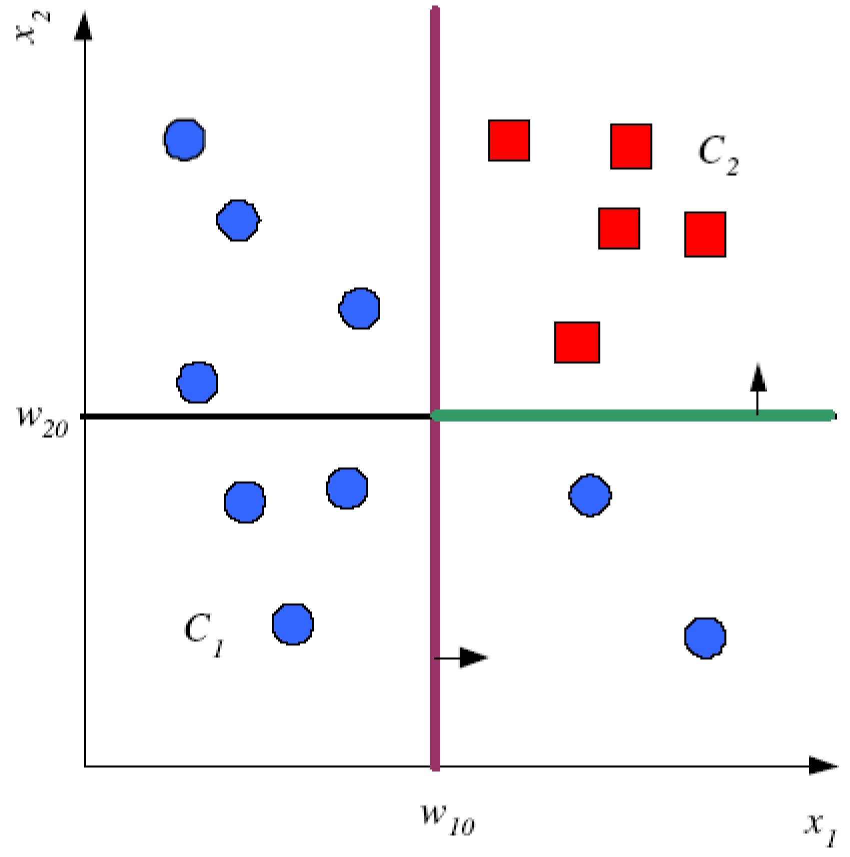
ETHEM ALPAYDIN

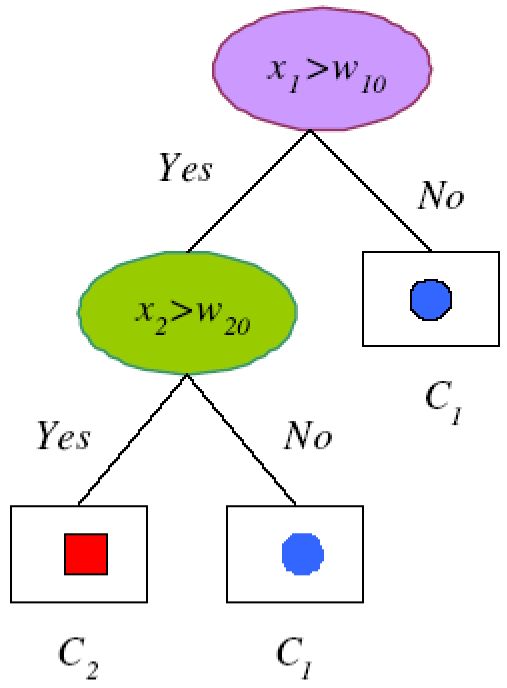
alpaydin@boun.edu.tr
http://www.cmpe.boun.edu.tr/~ethem/i2ml3e

CHAPTER 9:

Decision Trees

Tree 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)

Side Discussion “Greedy Algorithms”

- Fast and therefore attractive to solve NP-hard and other problems with high complexity. Later decisions are made in the context of decision selected early dramatically reducing the size of the search space.
- They do not backtrack: if they make a bad decision (based on local criteria), they never revise the decision.
- They are not guaranteed to find the optimal solutions, and sometimes can get deceived and find really bad solutions.
- In spite of what is said above, a lot successful and popular algorithms in Computer Science are greedy algorithms.
- Greedy algorithms are particularly popular in AI and Operations Research.

Popular Greedy Algorithms: Decision Tree Induction,…

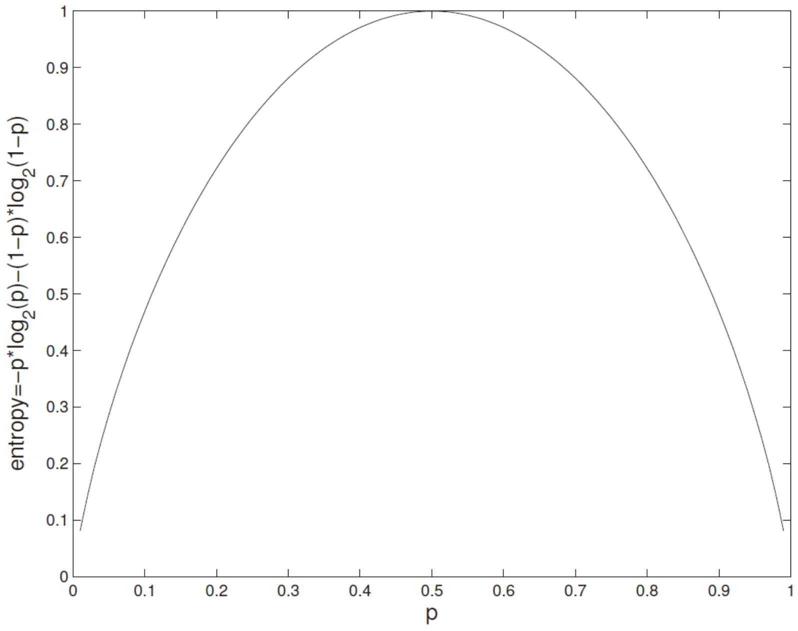
Classification Trees (ID3,CART,C4.5)

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

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

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

$$I_m = -\sum_{i=1}^{K} p^i_m \log_2 p^i_m$$



$$\text{Entropy} = -p\log_2 p - (1-p)\log_2(1-p)$$

For a two-class problem $p^1 \equiv p$ and $p^2 = 1 - p$, $\phi(p,1-p)$ is a nonnegative function

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

Examples are

1. Entropy

$$\phi(p,1-p) = -p\log_2 p - (1-p)\log_2 (1-p)$$

2. Gini index

$$\phi(p,1-p) = 2p(1-p)$$

3. Misclassification error

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

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

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

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

Tree Induction

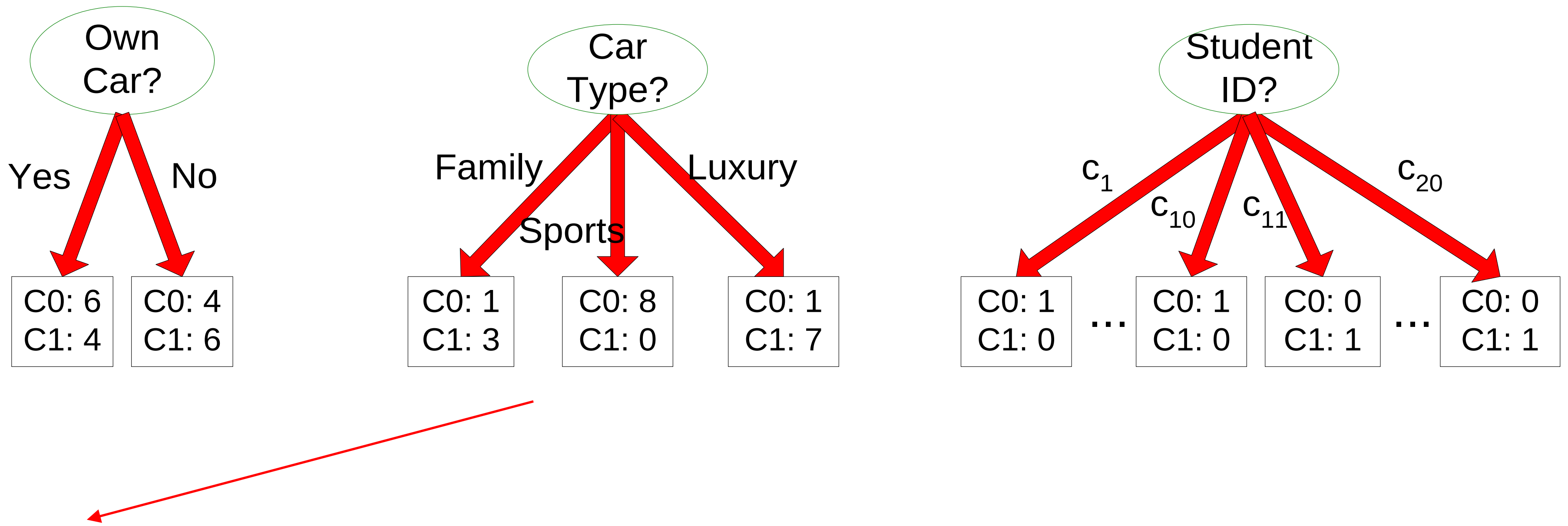
- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

How to determine the Best Split?

Before Splitting: 10 records of class 0, 10 records of class 1

Before: E(1/2,1/2)



After: 4/20*E(1/4,3/4) + 8/20*E(1,0) + 8/20*E(1/8,7/8)

Gain: Before-After

Pick Test that has the highest gain!

Remark: E stands for Gini, Entropy (H), Impurity (1-max$_c$(P(c)), Gain-ratio

Splitting Continuous Attributes

- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), clustering, or supervised clustering.
  - Binary Decision: (A < *v*) or (A ≥ *v*)
    - consider all possible splits and finds the best cut *v*

Classification tree construction.

$$I_m = -\sum_{i=1}^{K} p_m^i \log_2 p_m^i$$

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p_{mj}^i \log_2 p_{mj}^i$$

```
GenerateTree($\mathcal{X}$)
    If NodeEntropy($\mathcal{X}$)$< \theta_I$ /* eq. 9.3
        Create leaf labelled by majority class in $\mathcal{X}$
        Return
    $i \leftarrow$ SplitAttribute($\mathcal{X}$)
    For each branch of $\boldsymbol{x}_i$
        Find $\mathcal{X}_i$ falling in branch
        GenerateTree($\mathcal{X}_i$)
SplitAttribute($\mathcal{X}$)
    MinEnt$\leftarrow$ MAX
    For all attributes $i = 1, \ldots, d$
        If $\boldsymbol{x}_i$ is discrete with $n$ values
            Split $\mathcal{X}$ into $\mathcal{X}_1, \ldots, \mathcal{X}_n$ by $\boldsymbol{x}_i$
            e $\leftarrow$ SplitEntropy($\mathcal{X}_1, \ldots, \mathcal{X}_n$) /* eq. 9.8 */
            If e<MinEnt MinEnt $\leftarrow$ e; bestf $\leftarrow$ i
        Else /* $\boldsymbol{x}_i$ is numeric */
            For all possible splits
                Split $\mathcal{X}$ into $\mathcal{X}_1, \mathcal{X}_2$ on $\boldsymbol{x}_i$
                e$\leftarrow$SplitEntropy($\mathcal{X}_1, \mathcal{X}_2$)
                If e<MinEnt MinEnt $\leftarrow$ e; bestf $\leftarrow$ i
    Return bestf
```

Stopping Criteria for Tree Induction

1. Grow entire tree
   - Stop expanding a node when all the records belong to the same class
   - Stop expanding a node when all the records have the same attribute values
2. Pre-pruning (do not grow complete tree)
   1. Stop when only *x* examples are left (pre-pruning)
   2. … other pre-pruning strategies

How to Address Over-fitting in Decision Trees

The most popular approach: Post-pruning

- Grow decision tree to its entirety
- Trim the nodes of the decision tree in a bottom-up fashion
- If generalization error improves after trimming, replace sub-tree by a leaf node.
- Class label of leaf node is determined from majority class of instances in the sub-tree

Advantages Decision Tree Based Classification

- Inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Okay for noisy data
- Can handle both continuous and symbolic attributes
- Accuracy is comparable to other classification techniques for many simple data sets
- Decent average performance over many datasets
- Kind of a standard—if you want to show that your “new” classification technique really “improves the world” $\rightarrow$ compare its performance against decision trees (e.g. C 5.0) using 10-fold cross-validation
- Does not need distance functions; only the order of attribute values is important for classification: 0.1,0.2,0.3 and 0.331,0.332, 0.333 is the same for a decision tree learner.

Disadvantages Decision Tree Based Classification

- Relies on rectangular approximation that might not be good for some dataset
- Selecting good learning algorithm parameters (e.g. degree of pruning) is non-trivial
- Ensemble techniques, support vector machines, and *k-nn* might obtain higher accuracies for a specific dataset.
- More recently, forests (ensembles of decision trees) have gained some popularity.

Regression Trees

- Error at node *m*:

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

If at a node, the error is acceptable, that is, $E_m < \theta_r$, then a leaf node is created and it stores the g_m value.

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

MSE from the estimated value — estimated value in node *m*

- After splitting:

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

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

Regression Trees

- The drop in error for any split is given as the difference between E_m (the mean square error from the estimated value) and E'_m (the error after the split).
- We look for the split such that this drop is maximum or, equivalently, where E'_m takes its minimum.
- The code given in figure 9.3 (slide 11) can be adapted to training a regression tree by replacing entropy calculations with mean square error and class labels with averages.

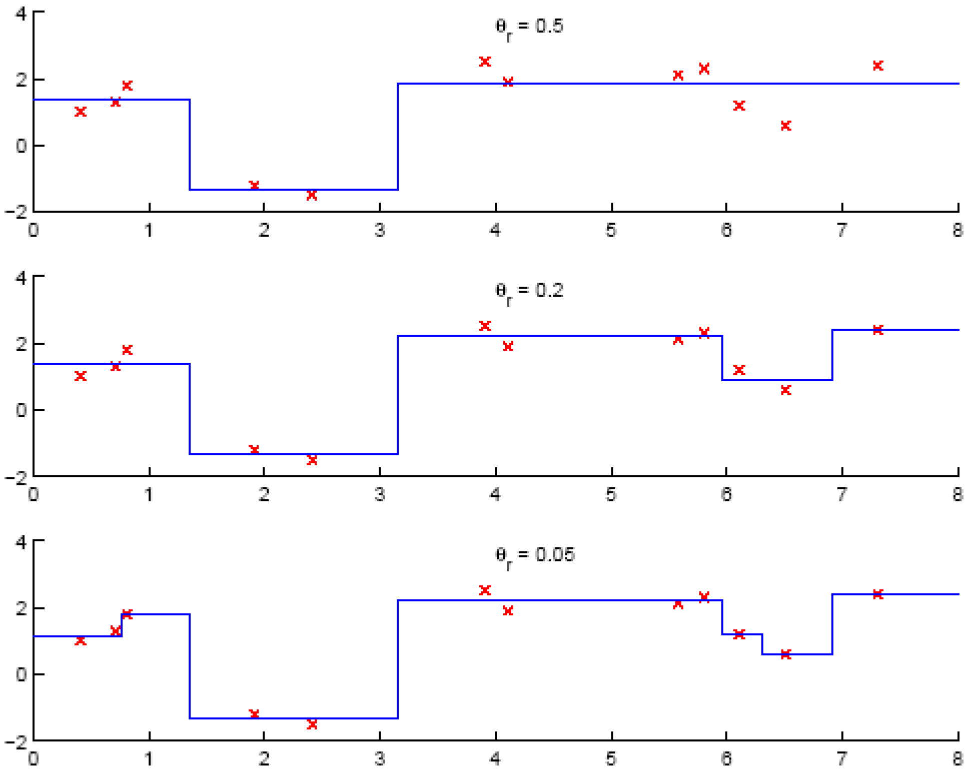
Regression Trees

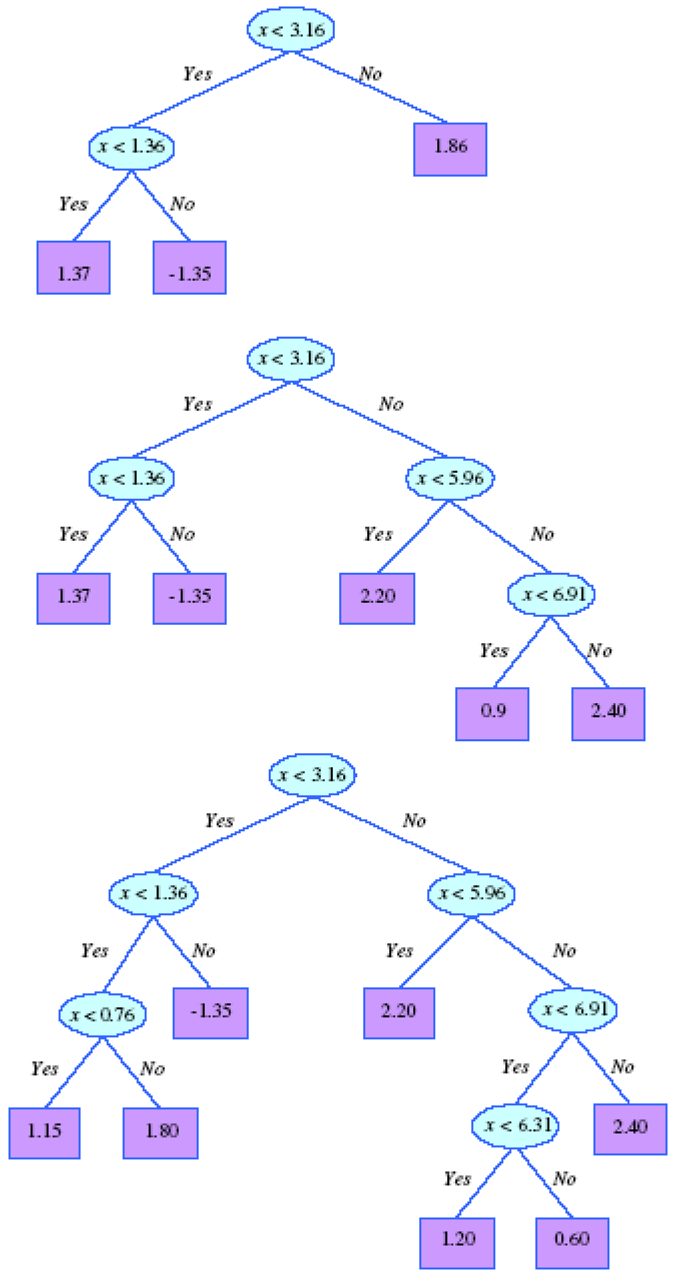
- Worst Possible Error:

$$E_m = \max_j \max_t \left| r^t - g_{mj} \right| b_{mj}\left(\mathbf{x}^t\right)$$

- we can guarantee that the error for any instance is never larger than a given threshold.
- The acceptable error threshold is the complexity parameter; when it is small, we generate large trees and risk overfitting; when it is large, we underfit and smooth too much.
- Linear regression fit over the instances choosing the leaf: $g_m(\mathbf{x}) = \mathbf{w}_m^T \mathbf{x} + w_{m0}$

Model Selection in Trees



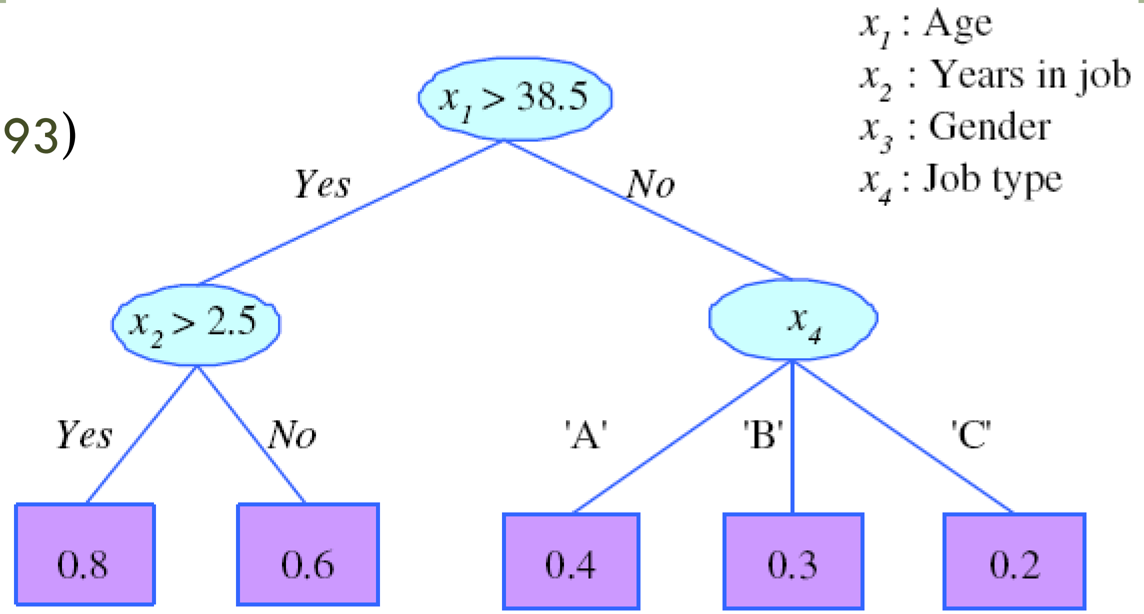


Pruning Trees

- 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 from Data

- 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
- Rule Induction Algorithm: IREP, Ripper
- Rules are added to explain positive examples such that if an instance is not covered by any rule, then it is classified as negative.

Learning Rules

- One of the most ***expressive*** and ***human readable*** representations for learned hypotheses is sets of ***production rules*** (***if-then*** rules).
- Rules can be derived from other representations (e.g., decision trees) or they can be learned ***directly***. Here, we are concentrating on the direct method.
- An important aspect of direct rule-learning algorithms is that they can learn sets of ***first-order rules*** which have much more representational power than the ***propositional*** rules that can be derived from decision trees.
- Rule Learning also allows the incorporation of background knowledge into the process.
- Learning rules is also useful for the data mining task of association rules mining.

Propositional versus First-Order Logic

- ***Propositional Logic*** does not include variables and thus cannot express general relations among the values of the attributes.
- ***Example 1:*** in Propositional logic, you can write: ***<u>IF</u> (Father$_1$=Bob) ^ (Name$_2$=Bob)^ (Female$_1$=True) <u>THEN</u> Daughter$_{1,2}$=True.***

  This rule applies only to a specific family!
- ***Example 2:*** In First-Order logic, you can write: ***<u>IF</u> Father(y,x) ^ Female(y), <u>THEN</u> Daughter(x,y)***

  This rule (which you cannot write in Propositional Logic) applies to any family!

Learning Propositional Rules: Sequential Covering Algorithms

- The algorithm is called a ***sequential covering algorithm*** because it sequentially learns a set of rules that together cover the whole set of positive examples.
- It has the advantage of reducing the problem of learning a disjunctive set of rules to a sequence of simpler problems, each requiring that a single conjunctive rule be learned.
- The final set of rules is sorted so that the most accurate rules are considered first at classification time.
- However, because it does not backtrack, this algorithm is not guaranteed to find the smallest or best set of rules → Learn-one-rule must be very effective!

RIPPER

- Here are two kinds of loop in the Ripper algorithm:
  - Outer loop: adding one rule at a time to the rule base
  - Inner loop: adding one condition at a time to the current rule
    - Conditions are added to the rule to maximize an information gain measure.
    - Conditions are added to the rule until it covers no negative example.

Ripper Algorithm

- In Ripper, conditions are added to the rule to maximize an information gain measure

$$Gain(R',R) = s \cdot (\log_2 \frac{N'_+}{N'} - \log_2 \frac{N_+}{N})$$

  - R : the original rule
  - R' : the candidate rule after adding a condition
  - N (N'): the number of instances that are covered by R (R')
  - N_+ (N'_+): the number of true positives in R (R')
  - s : the number of true positives in R and R' (after adding the condition)

until it covers no negative example.

Prunning by maximizing RVM

$$rvm(R) = \frac{p-n}{p+n} \boxed{\approx 1}$$

Rule value metric

p and n : the number of true and false positives respectively.

```
procedure IREP(Pos,Neg)
begin
    Ruleset := $\emptyset$
    while Pos$\neq \emptyset$ do
        /* grow and prune a new rule */
        split (Pos,Neg) into (GrowPos,GrowNeg)
          and (PrunePos,PruneNeg)
        Rule := GrowRule(GrowPos,GrowNeg)
        Rule := PruneRule(Rule,PrunePos,PruneNeg)
        if the error rate of Rule on
          (PrunePos,PruneNeg) exceeds 50% then
            return Ruleset
        else
            add Rule to Ruleset
            remove examples covered by Rule
              from (Pos,Neg)
        endif
    endwhile
    return Ruleset
end
```

```
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
```

$O(N\log^2 N)$

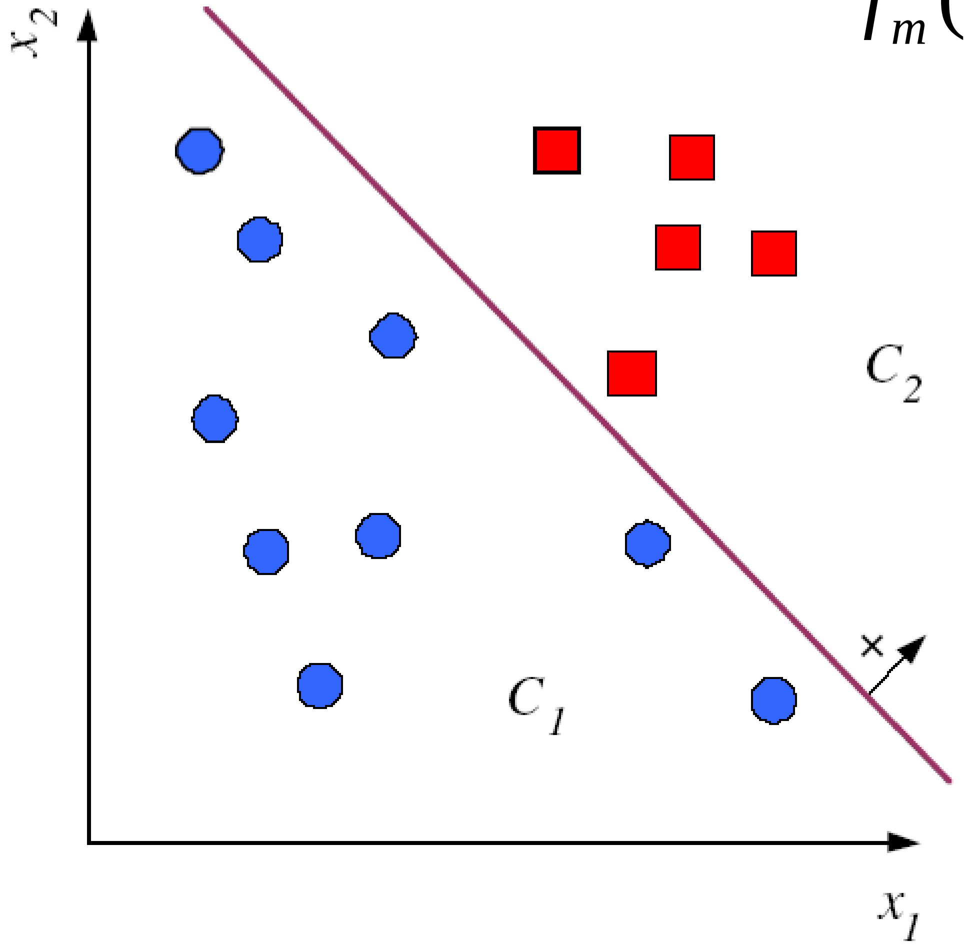
DL: description length of the rule base

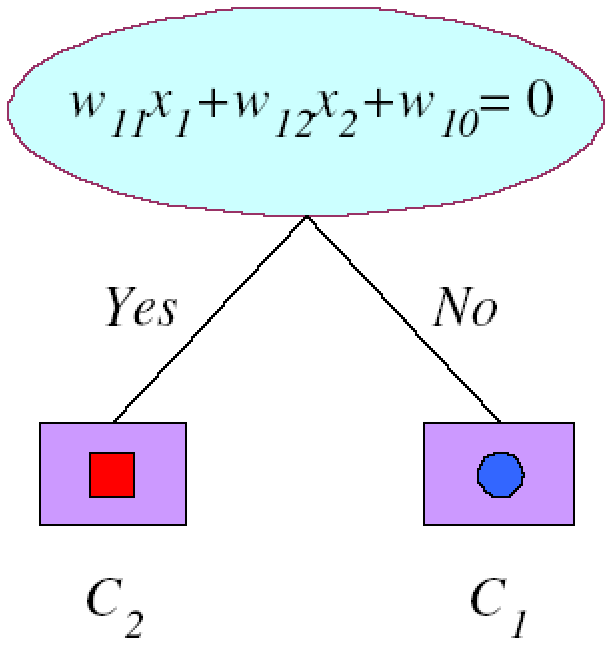
The description length of a rule base
= (the sum of the description lengths of all the rules in the rule base)
+ (the description of the instances not covered by the rule base)

```
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

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





Multivariate Trees

- $f_m(\mathbf{x}) : \mathbf{w}_m^T\mathbf{x} + w_{m0} > 0$ defines a hyperplane with arbitrary orientation.
- Leaf nodes define polyhedra in the input space.
- In a univariate node there are *d* possible orientations ($\mathbf{w}_\mathrm{m}$) and $N_\mathrm{m} - 1$ possible thresholds ($-w_{\mathrm{m}0}$), making an exhaustive search possible.
- In a multivariate node, there are $2^d \binom{N_m}{d}$ possible hyperplanes and an exhaustive search is no longer practical.

Multivariate Trees

- Linear multivariate nodes are more flexible.
- Nonlinear multivariate nodes are even more flexible.

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

- Multilayer perceptron has been proposed.
- Sphere node is also possible.

$$f_m(\mathbf{x}) : \|\mathbf{x} - \mathbf{c}_m\| \le \alpha_m$$

where $\mathbf{c}_\mathrm{m}$ is the center and α_m is the radius.