Statistical Learning and Data Mining
CS 363D/ SSC 358

Lecture: Practical Issues in Classification

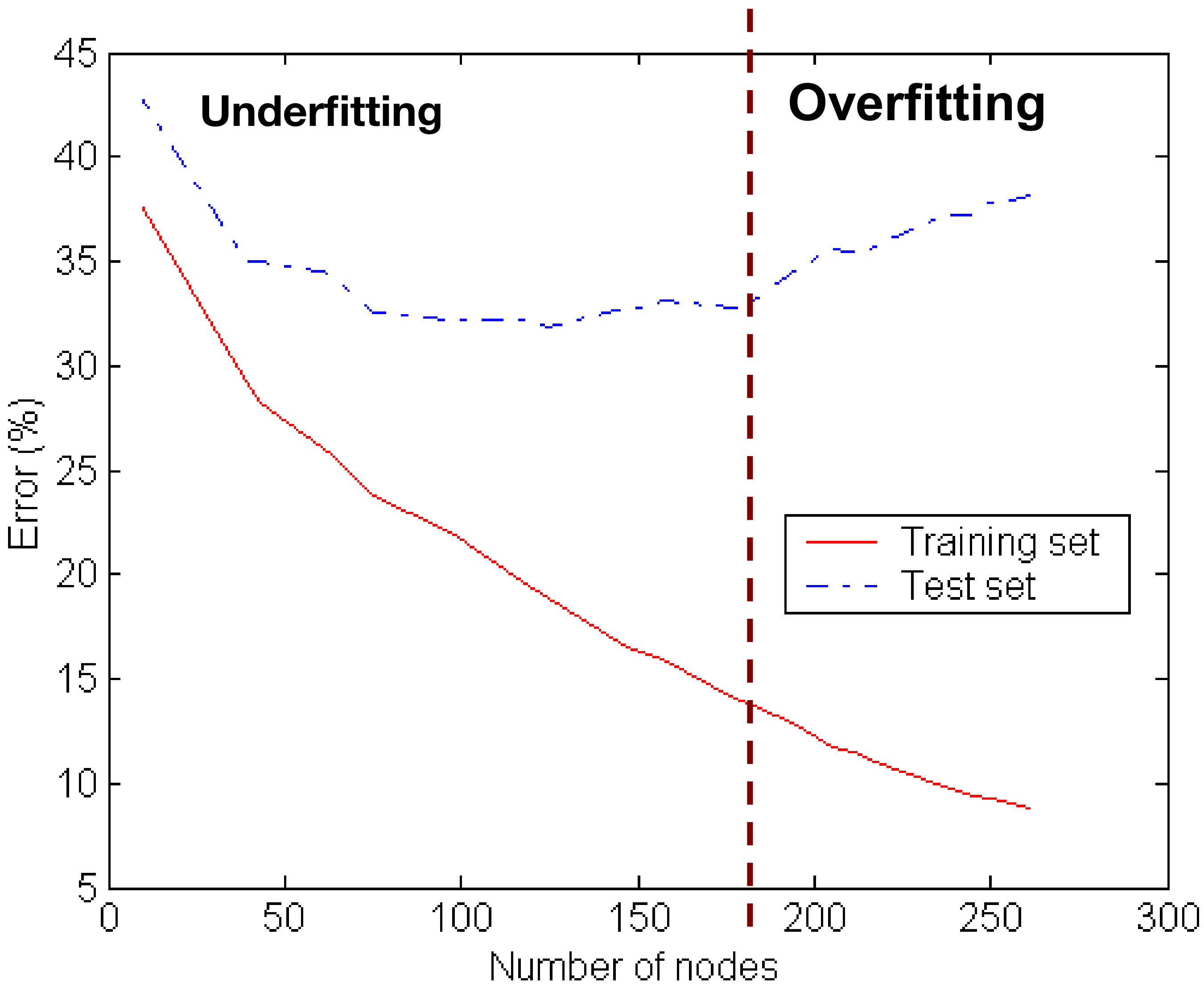
Prof. Pradeep Ravikumar
pradeepr@cs.utexas.edu

Adapted From: Pang-Ning Tan, Steinbach, Kumar

Practical Issues in Classification

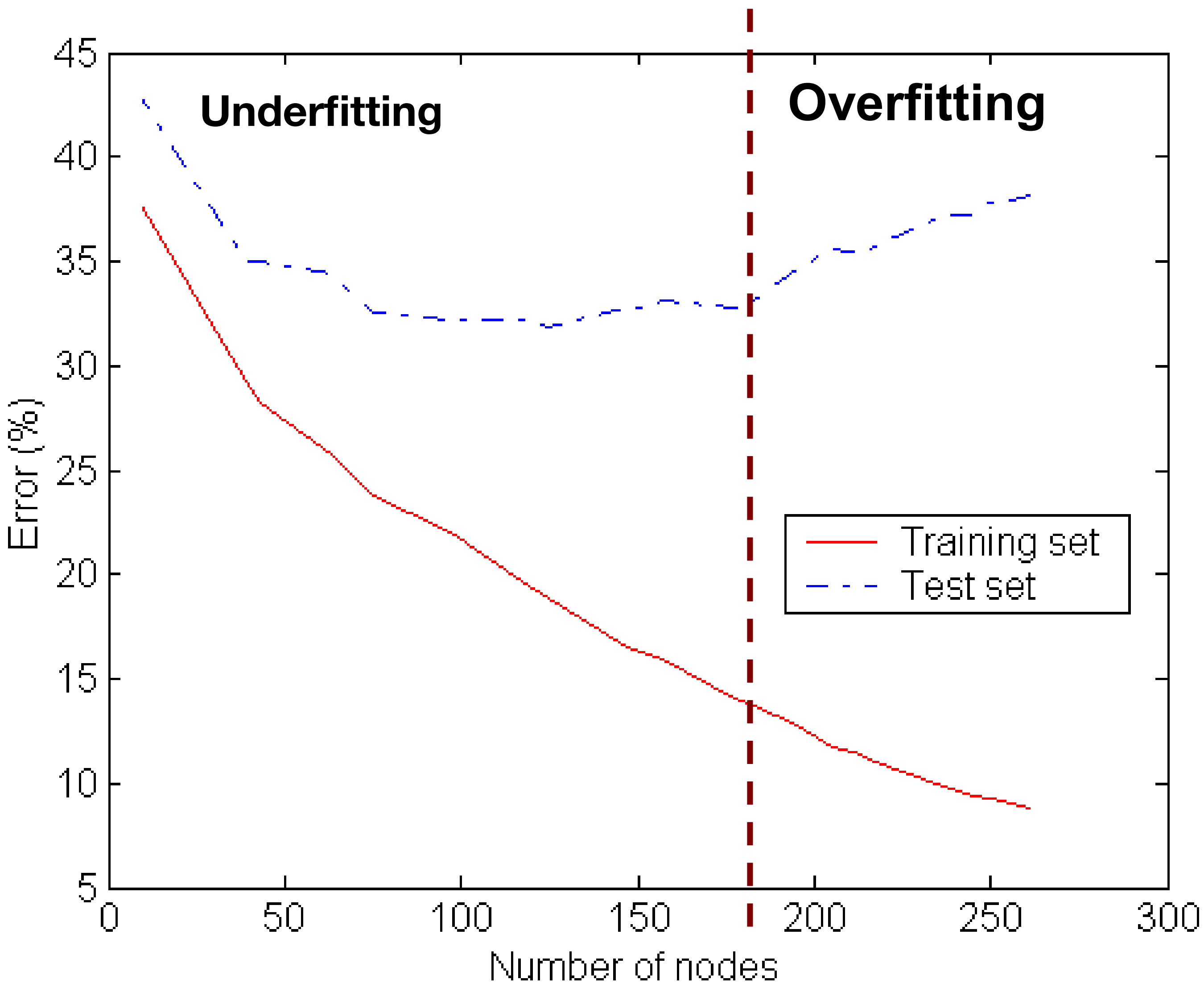
- Underfitting and Overfitting
- Missing Values
- Costs of Classification

Underfitting and Overfitting



Underfitting: when model is too simple, both training and test errors are large

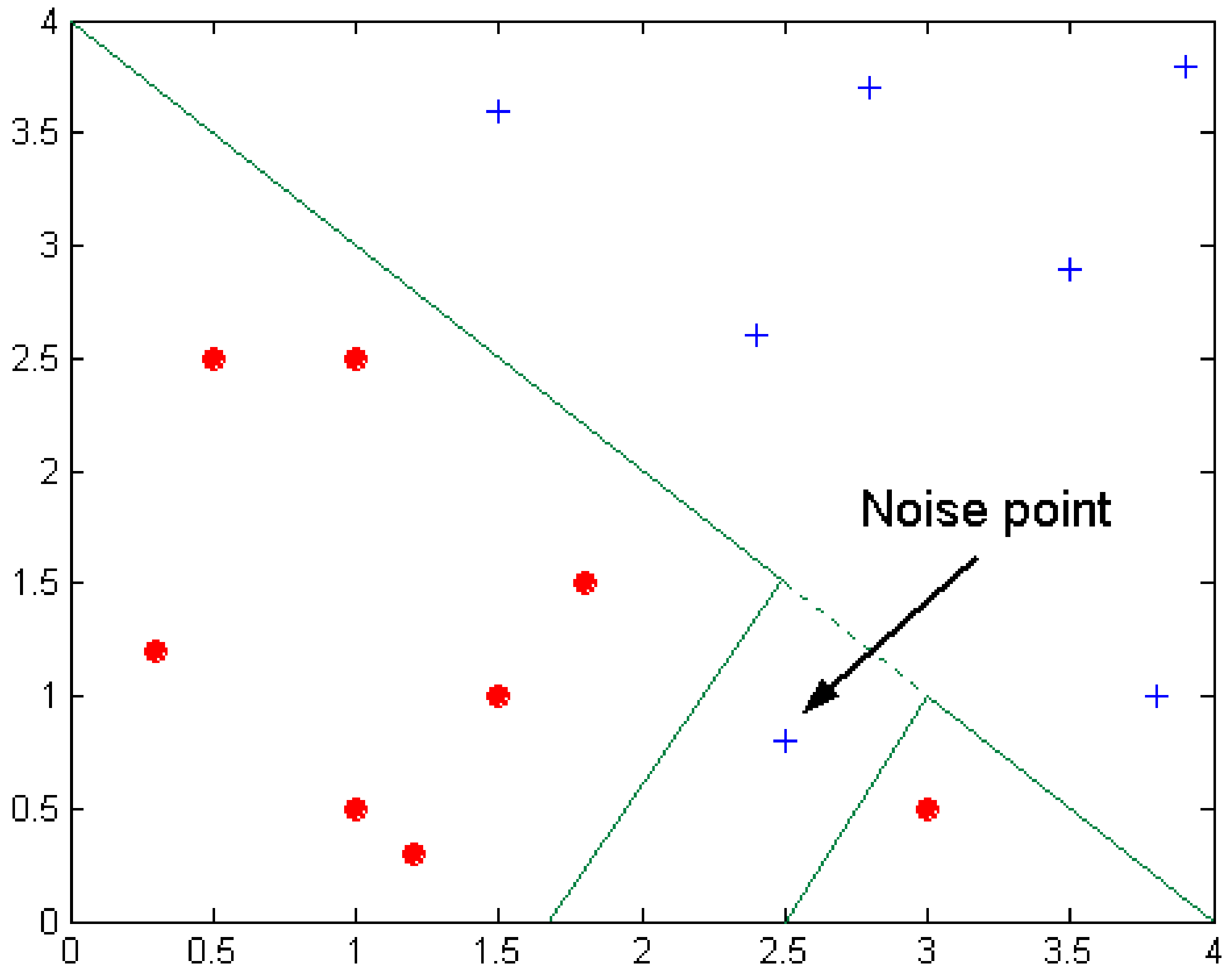
Underfitting and Overfitting



Underfitting: when model is too simple, both training and test errors are large

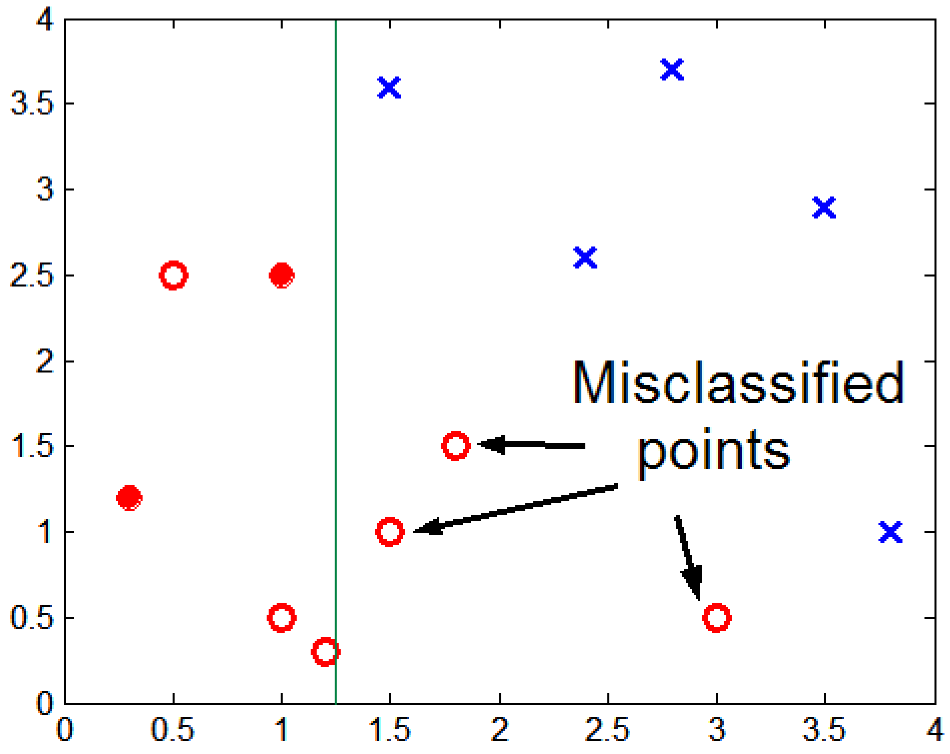
Overfitting: when model is too complex, training error small, test error large

Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples



Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records
- Need new ways for estimating errors

Generalization Error

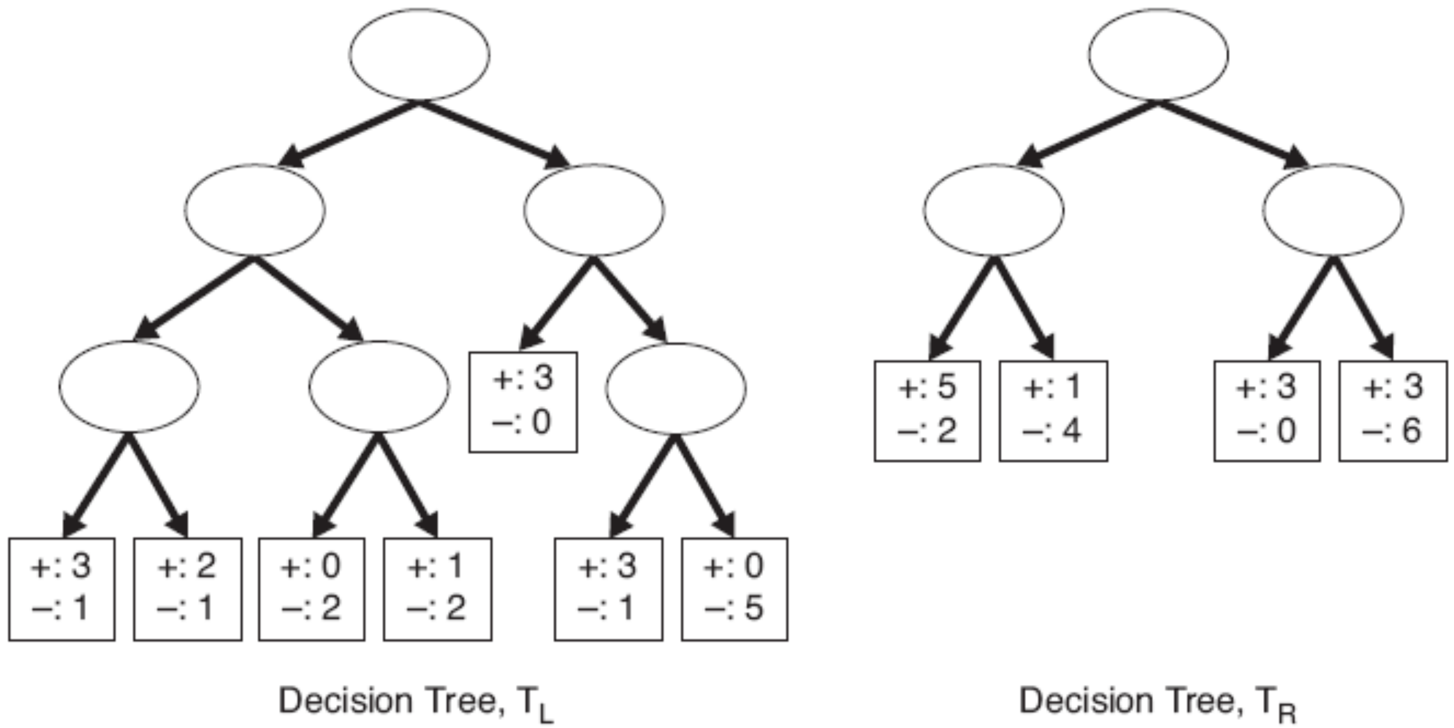
- The best way to find out if the tree is overfitting is to see if its **generalization error** is large.

- Generalization Error: Misclassification error on unseen data
  (“how well would the method generalize to unseen data?”)

- But we have access only to the training data during model building; how do we estimate the generalization error?

Generalization Error: Resubstitution Estimate

- Just use the training error



Example of two decision trees generated from the same training data.

$$e(T_L) = 4/24 = 0.167$$

$$e(T_R) = 6/24 = 0.25$$

Occam's Razor

- Given two models of similar generalization errors one should prefer the simpler model over the more complex model
- For complex models, there is a greater chance that it was fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

Ockham chooses a razor

Generalization Error: Pessimistic Estimate

- For each leaf node: e'(t) = (e(t)+0.5)
- Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
- For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):

  Training error = 10/1000 = 1%

  Generalization error = (10 + 30×0.5)/1000 = 2.5%

Add a "penalty" to no. of misclassified records at each leaf node

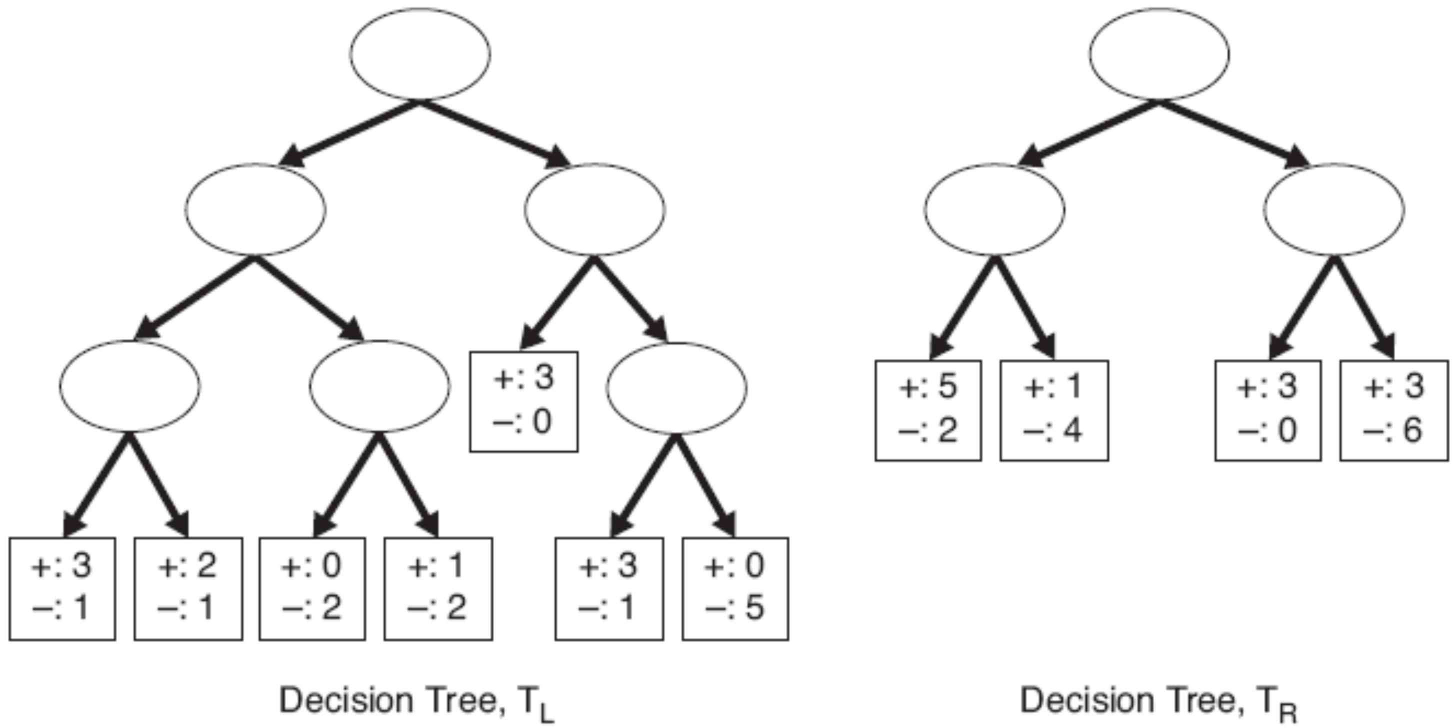
Generalization Error: Pessimistic Estimate

- For each leaf node: e'(t) = (e(t)+0.5)
- Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
- For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):

  Training error = 10/1000 = 1%

  Generalization error = (10 + 30×0.5)/1000 = 2.5%

Add a "penalty" to no. of misclassified records at each leaf node



Example of two decision trees generated from the same training data.

$$e_p(T_L) = \frac{4 + 7 \times 0.5}{24} = 0.3125$$

$$e_p(T_R) = \frac{6 + 4 \times 0.5}{24} = 0.3333$$

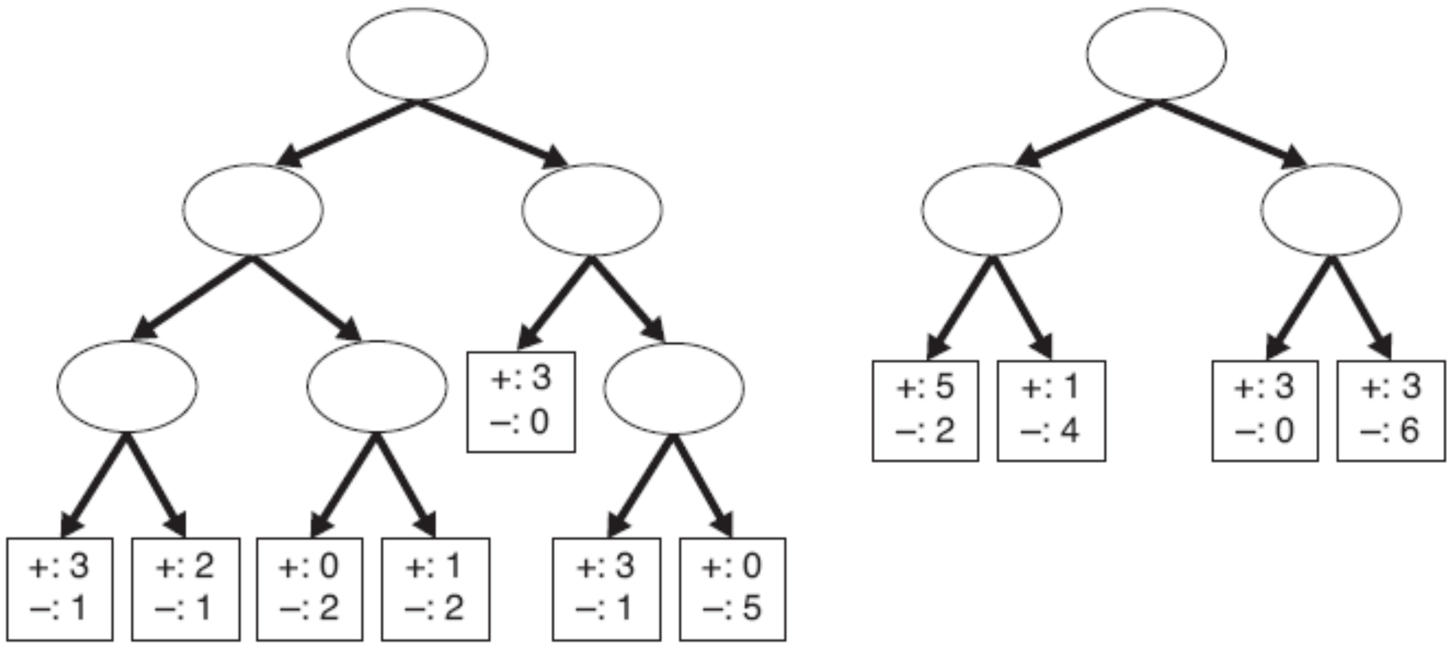
Generalization Error: Pessimistic Estimate

- For each leaf node: e'(t) = (e(t)+0.5)
- Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
- For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):

  Training error = 10/1000 = 1%

  Generalization error = (10 + 30×0.5)/1000 = 2.5%

Can be some other number e.g. 1



Decision Tree, T_L Decision Tree, T_R

Example of two decision trees generated from the same training data.

$$e_p(T_L) = \frac{4 + 7 \times 1}{24} = 0.458$$

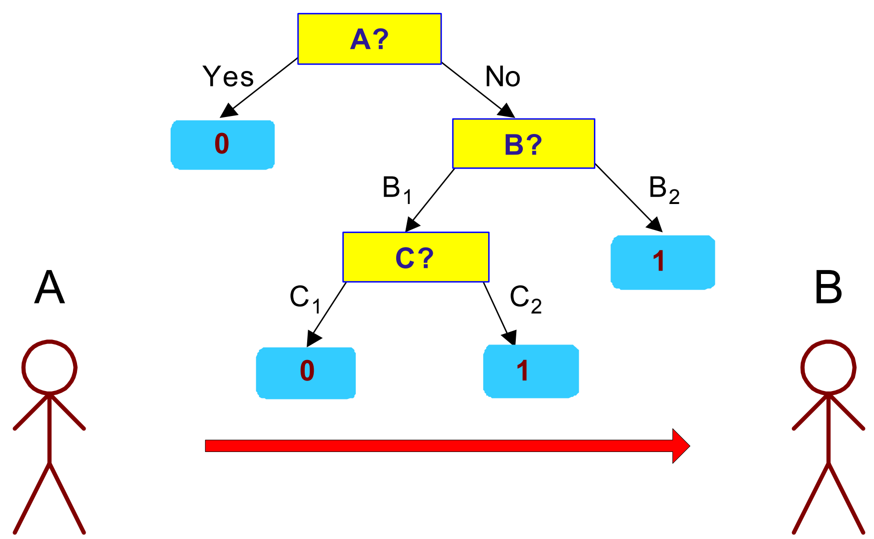
$$e_p(T_R) = \frac{6 + 4 \times 1}{24} = 0.417$$

Generalization Error: Validation Set

- Instead of using training set to compute generalization error (resubstitution estimate), split the training data into two components
  - one for model-building, and one, a “validation set,” for computing generalization error
  - choose one among a set of model choices (e.g. different sized trees) by comparing their errors on the validation set
  - Caveat: less training data available for model building

Generalization Error: Minimum Description Length

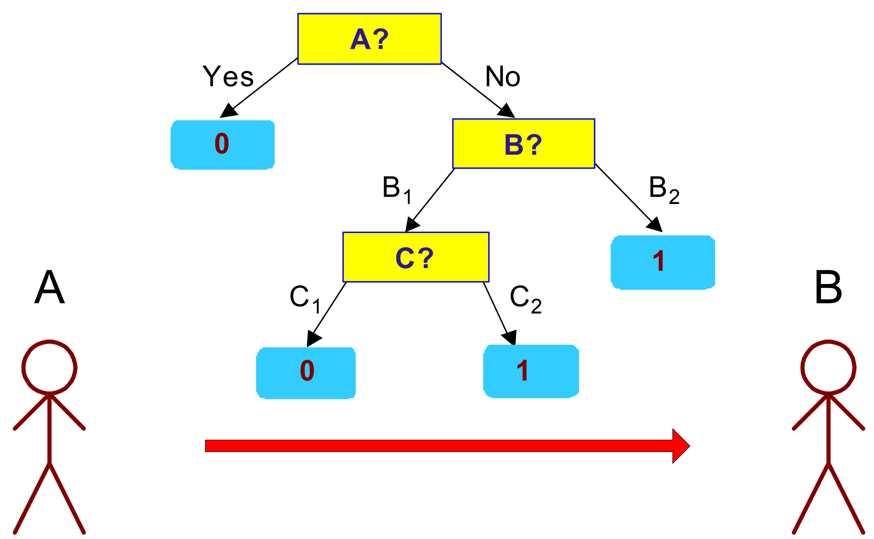
X	y
X_1	1
X_2	0
X_3	0
X_4	1
…	…
X_n	1



X	y
X_1	?
X_2	?
X_3	?
X_4	?
…	…
X_n	?

Generalization Error: Minimum Description Length

X	y
$\mathbf{X}_1$	1
$\mathbf{X}_2$	0
$\mathbf{X}_3$	0
$\mathbf{X}_4$	1
…	…
$\mathbf{X}_n$	1



X	y
$\mathbf{X}_1$	?
$\mathbf{X}_2$	?
$\mathbf{X}_3$	?
$\mathbf{X}_4$	?
…	…
$\mathbf{X}_n$	?

- Cost(Model,Data) = Cost(Data|Model) + Cost(Model)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- Cost(Data|Model) encodes the misclassification errors.
- Cost(Model) uses node encoding (number of children) plus splitting condition encoding.

How to Address Overfitting I

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree

How to Address Overfitting I

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same

How to Address Overfitting I

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

How to Address Overfitting II

- 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
  - Can use MDL for post-pruning

Example: Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

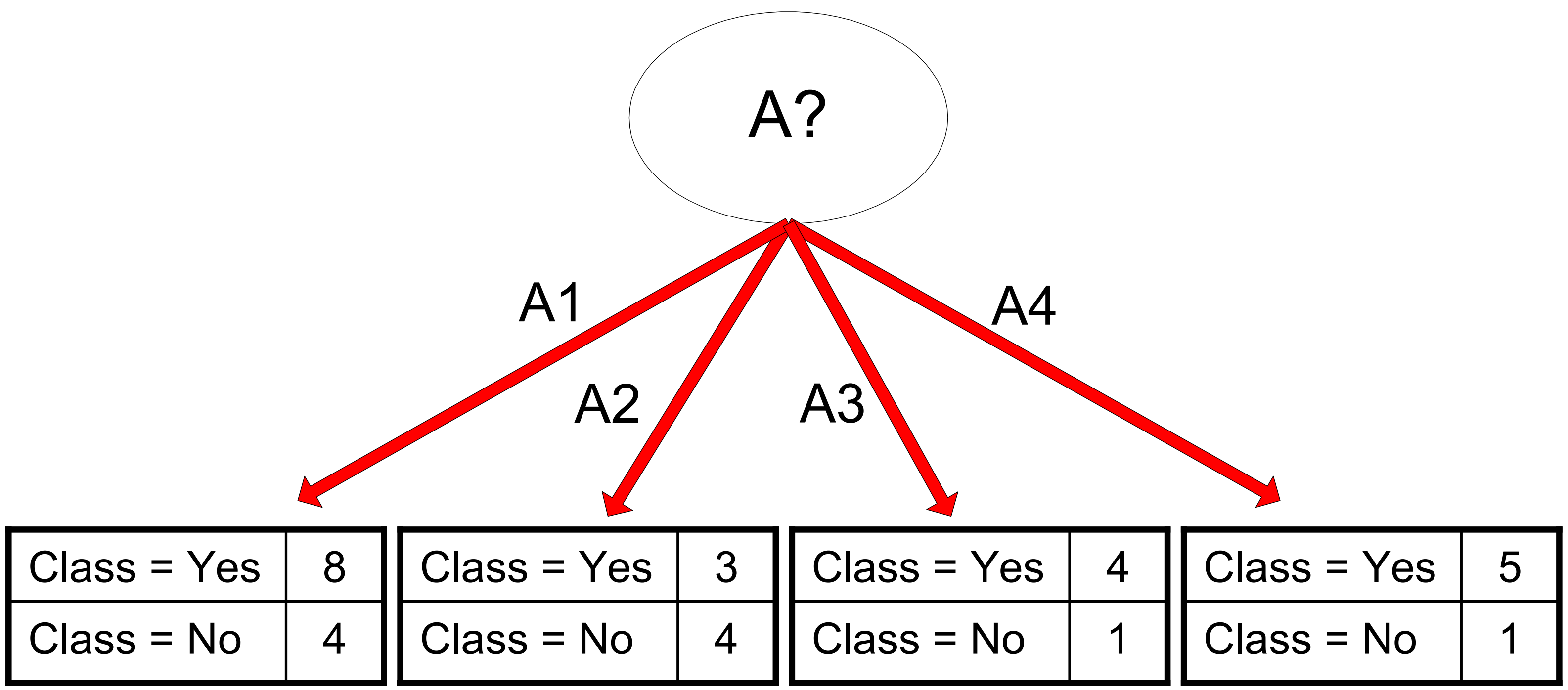
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Example: Post-Pruning

Training Error (Before splitting) = 10/30

Pessimistic error = (10 + 0.5)/30 = 10.5/30

Class = Yes	20
Class = No	10
Error = 10/30	



Example: Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

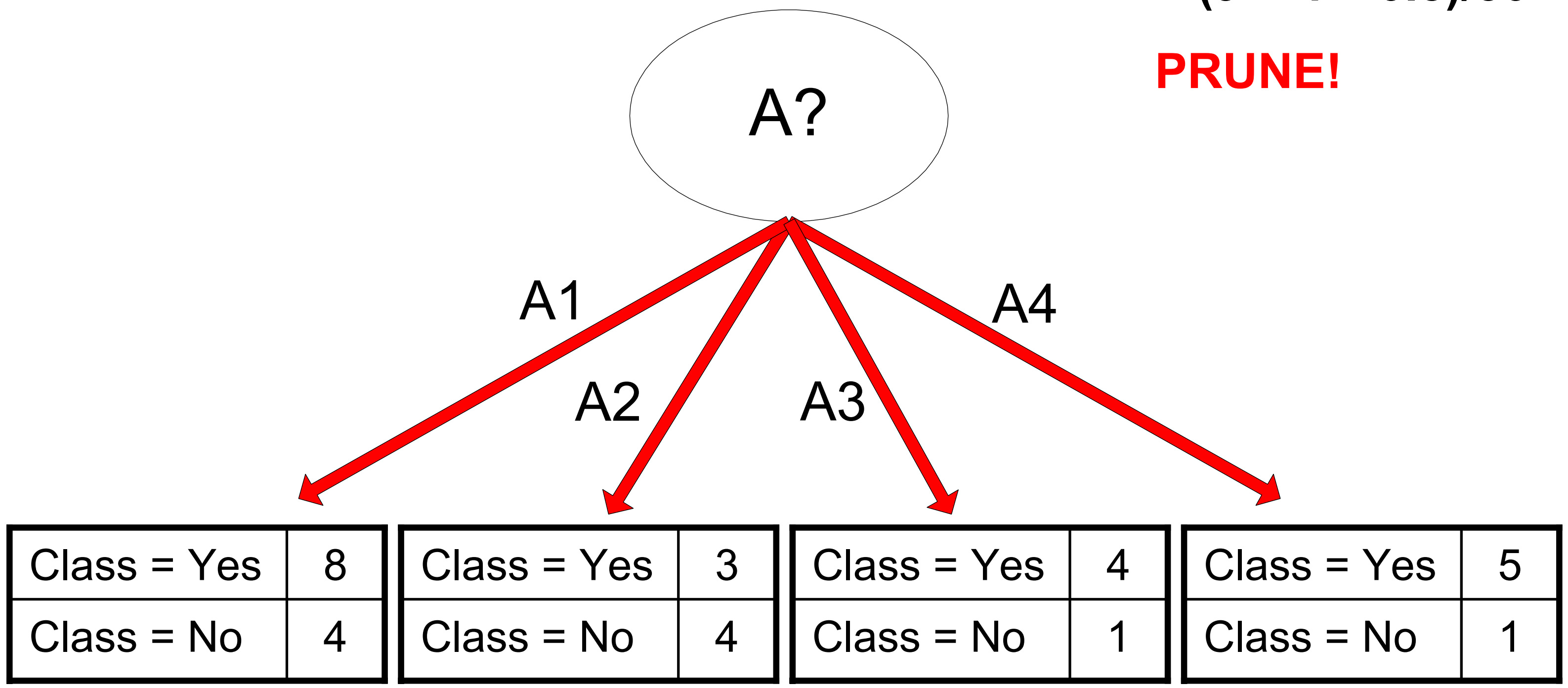
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 $\times$ 0.5)/30 = 11/30

PRUNE!



Example: Post-Pruning

- Optimistic error?

  Don't prune for both cases

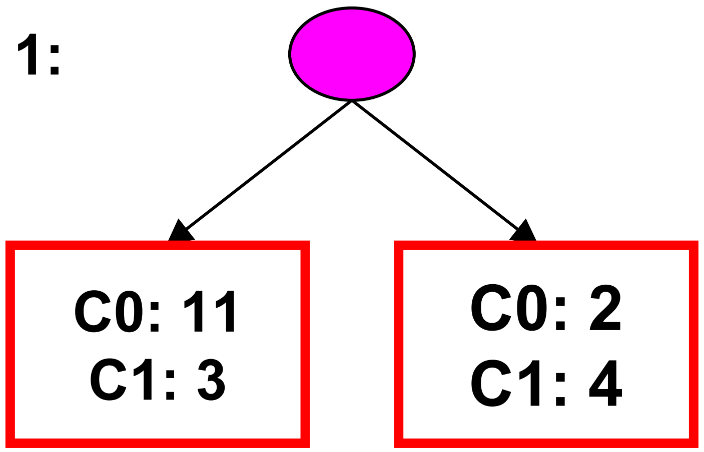
- Pessimistic error?

  Don't prune case 1, prune case 2

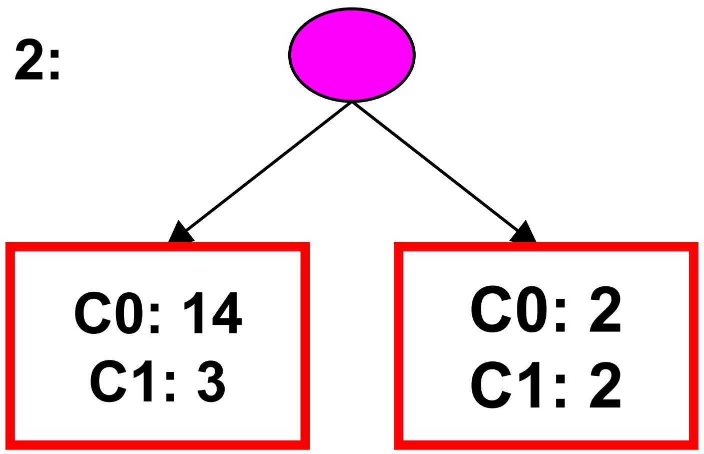
- Reduced error pruning?

  Depends on validation set

Case 1:



Case 2:



Handling Missing Attribute Values

- Missing values affect decision tree construction in three different ways:
  - Affects how impurity measures are computed
  - Affects how to distribute instance with missing value to child nodes
  - Affects how a test instance with missing value is classified

I. Computing Impurity Measure

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Class**
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	?	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)

= -0.3 log(0.3)-(0.7)log(0.7) = 0.8813

	Class = Yes	**Class = No**
Refund=Yes	**0**	**3**
Refund=No	**2**	**4**
Refund=?	**1**	**0**

I. Computing Impurity Measure

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	?	Single	90K	Yes

Missing value

Before Splitting:

Entropy(Parent)
= -0.3 log(0.3)-(0.7)log(0.7) = 0.8813

	Class = Yes	Class = No
Refund=Yes	0	3
Refund=No	2	4
Refund=?	1	0

Split on Refund:

Entropy(Refund=Yes) = 0

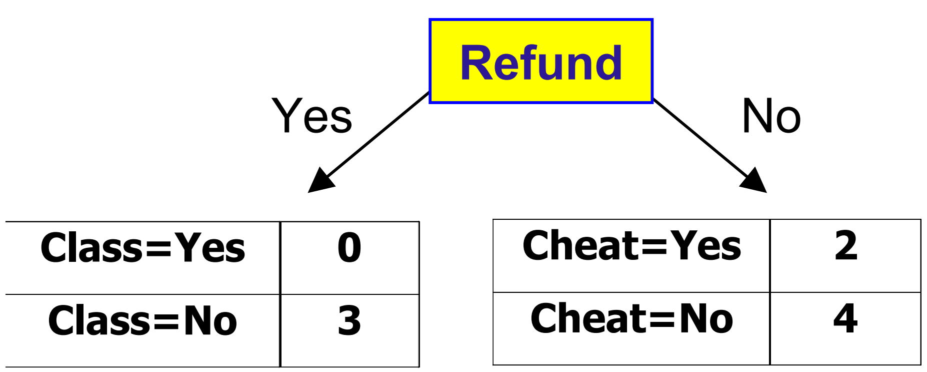
Entropy(Refund=No)
= -(2/6)log(2/6) – (4/6)log(4/6) = 0.9183

Entropy(Children)
= 0.3 (0) + 0.6 (0.9183) = 0.551

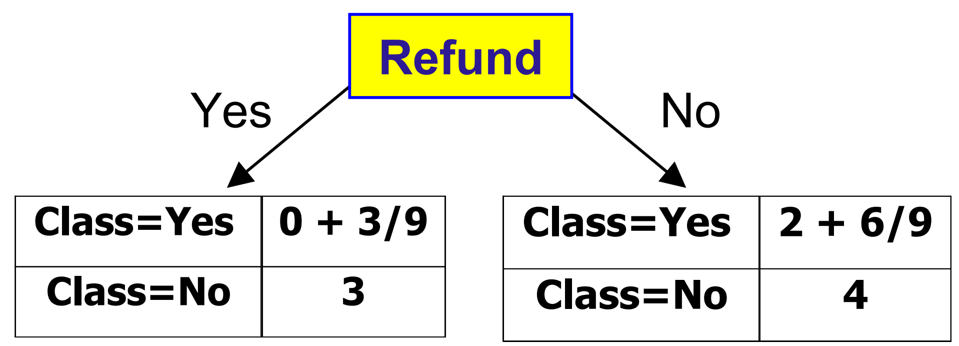
Gain = 0.9 x 0.8813 - 0.551 = 0.242

II. Distribute Instances

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No



Tid	Refund	Marital Status	Taxable Income	Class
10	?	Single	90K	Yes



Probability that Refund=Yes is 3/9

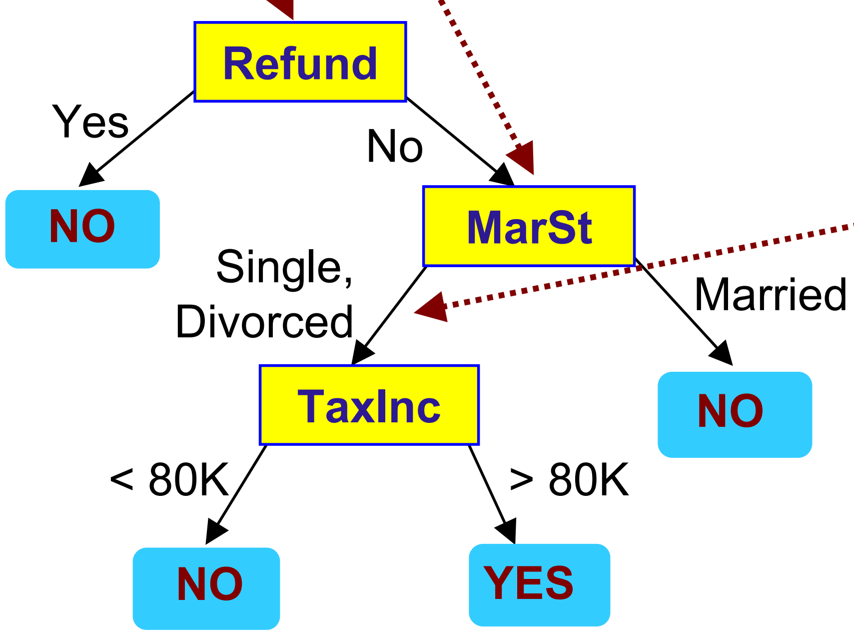
Probability that Refund=No is 6/9

Assign record to the left child with weight = 3/9 and to the right child with weight = 6/9

III. Classify Instances

New record:

Tid	**Refund**	**Marital Status**	**Taxable Income**	**Class**
11	No	?	85K	?



	Married	Single	Divorced	Total
Class=No	3	1	0	4
Class=Yes	6/9	1	1	2.67
Total	3.67	2	1	6.67

Probability that Marital Status = Married is 3.67/6.67

Probability that Marital Status ={Single,Divorced} is 3/6.67

Other Issues

- Data Fragmentation
- Search Strategy
- Expressiveness
- Tree Replication

Data Fragmentation

- Number of instances gets smaller as you traverse down the tree
- Number of instances at the leaf nodes could be too small to make any statistically significant decision

Data Fragmentation

- Number of instances gets smaller as you traverse down the tree
- Number of instances at the leaf nodes could be too small to make any statistically significant decision

Solution: Prune Early?

Search Strategy

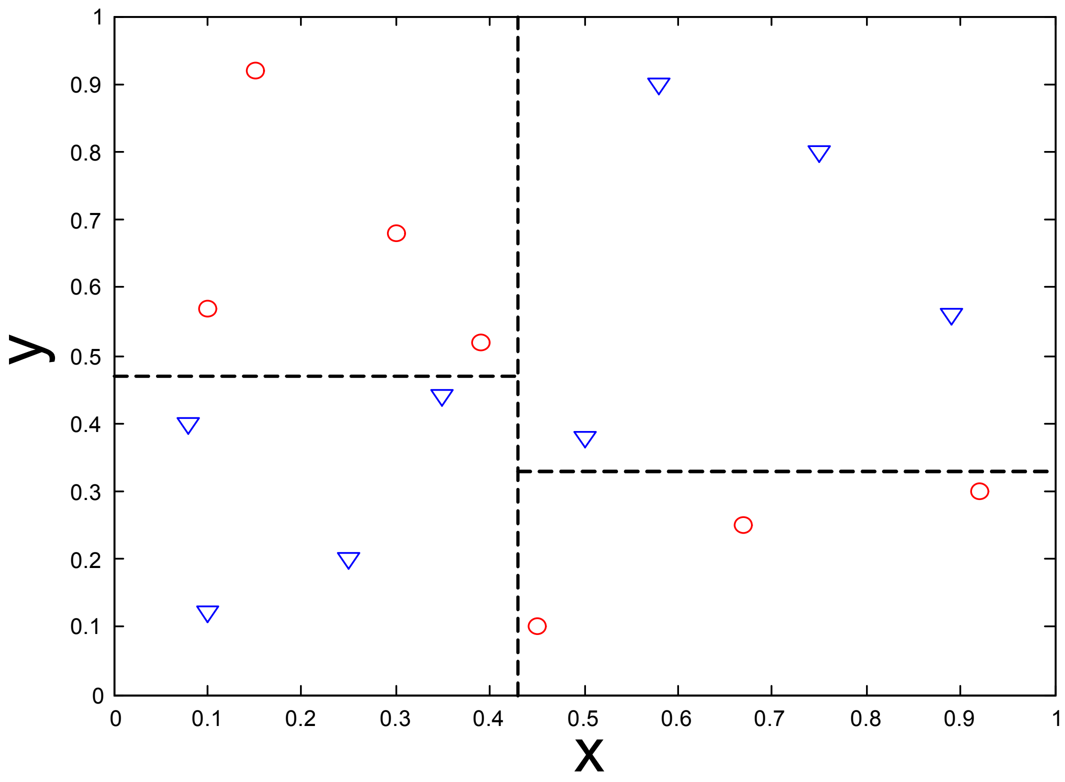
- Finding an optimal decision tree is NP-hard
- The algorithm presented so far uses a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
- Other strategies?
  - Bottom-up
  - Bi-directional

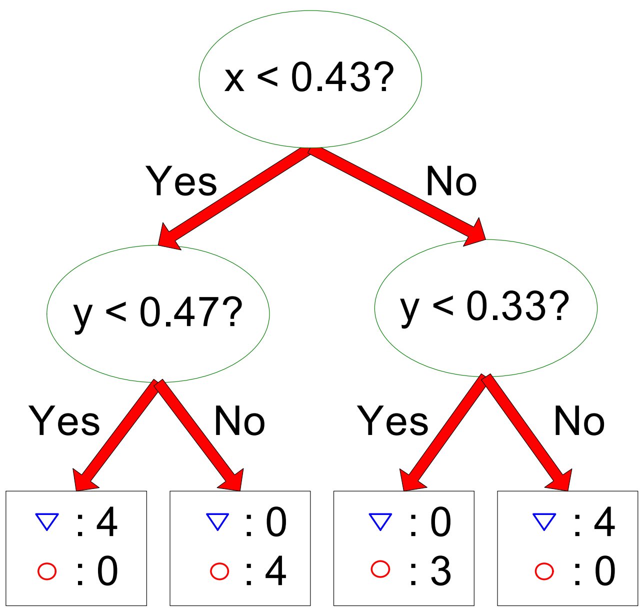
Expressiveness

- Decision tree provides expressive representation for learning discrete-valued function
  - But they do not generalize well to certain types of Boolean functions

- Not expressive enough for modeling continuous variables
  - Particularly when test condition involves only a single attribute at-a-time

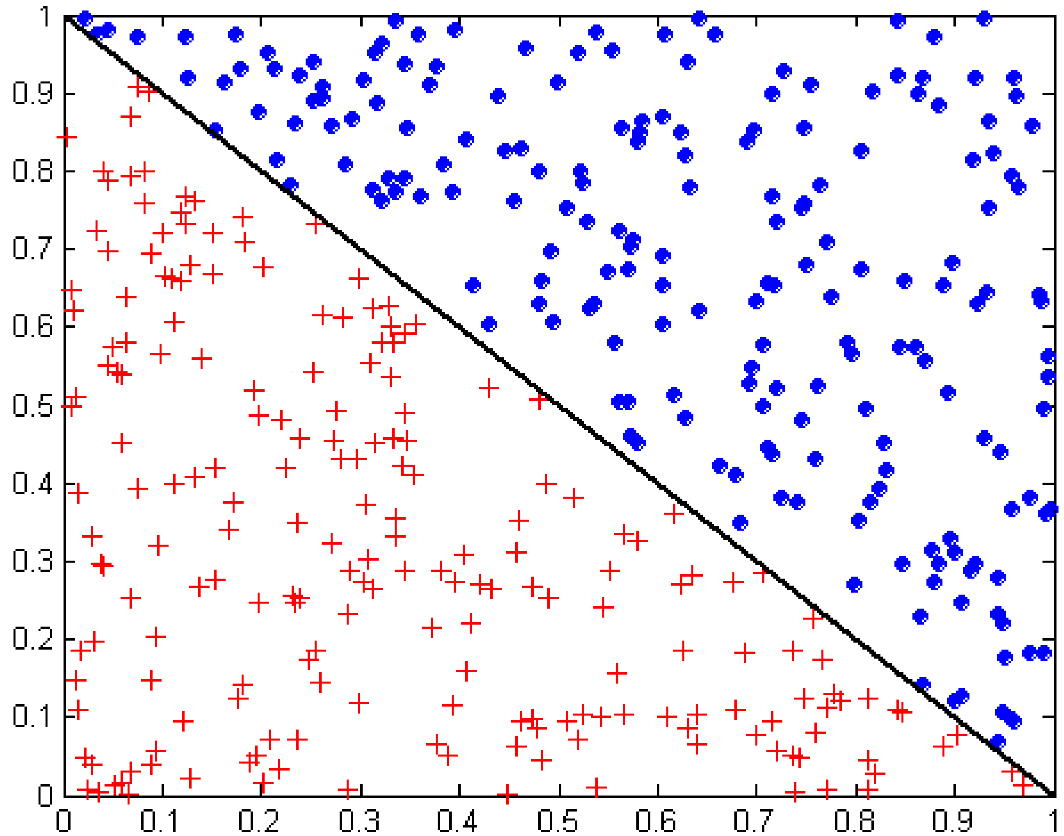
Decision Boundary

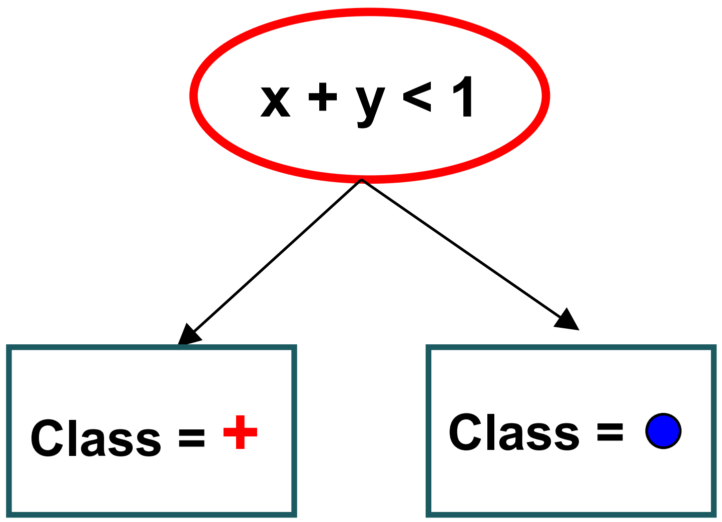




- **Border line between two neighboring regions of different classes is known as decision boundary**
- **Decision boundary is parallel to axes because test condition involves a single attribute at-a-time**

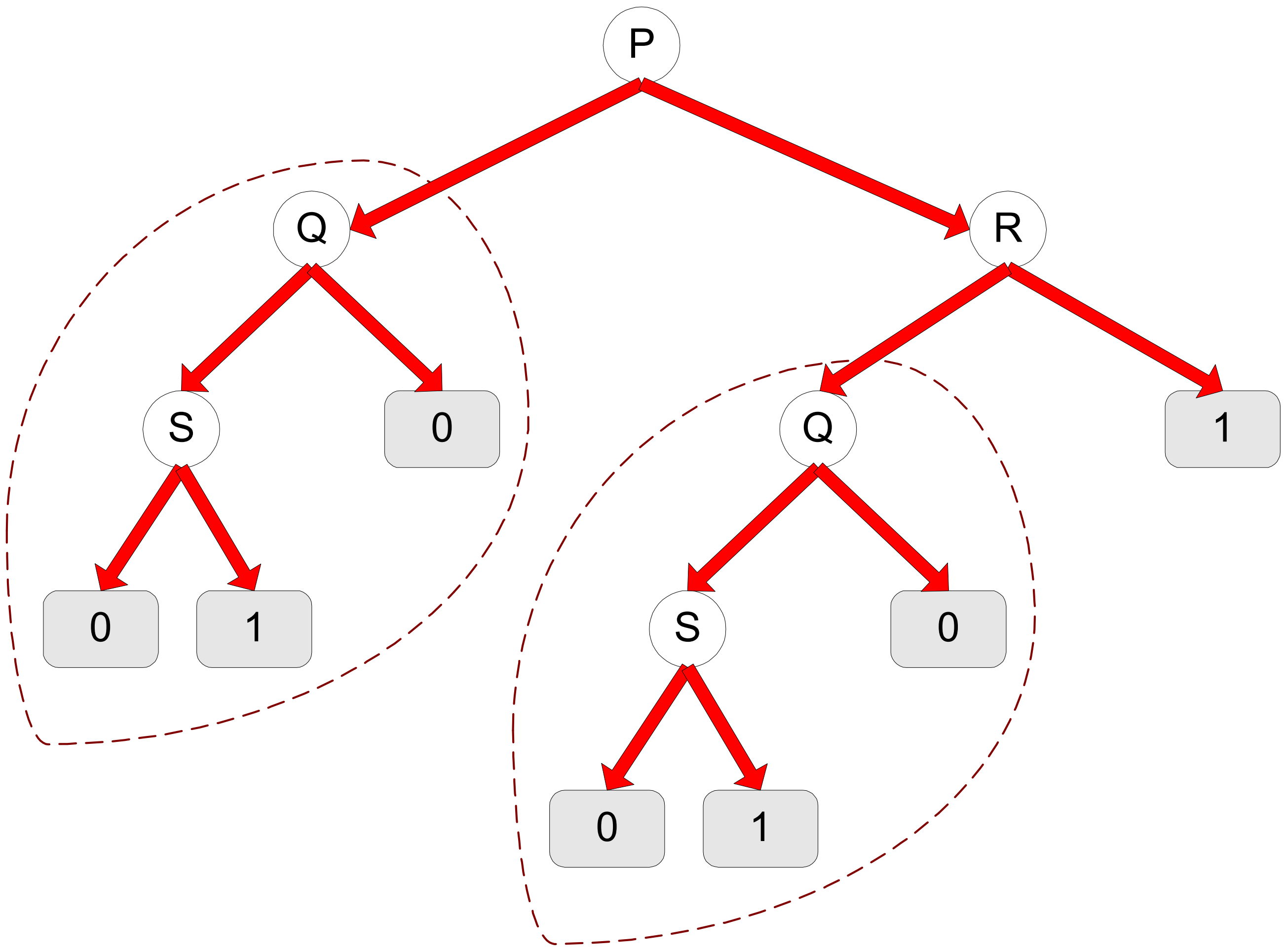
Oblique Decision Trees





- **Test condition may involve multiple attributes**
- **More expressive representation**
- **Finding optimal test condition is computationally expensive**

Tree Replication



- **Same subtree appears in multiple branches**