

# Data Mining I

## Classification, Part 2

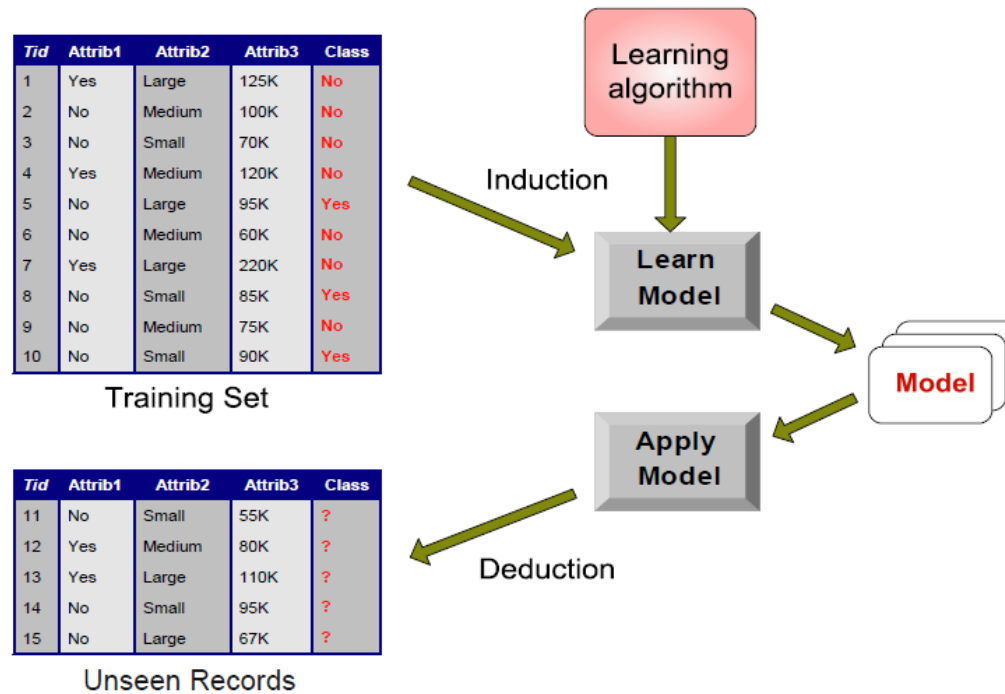


# Outline

1. What is Classification?
2. k Nearest Neighbors and Nearest Centroids
3. Naïve Bayes
4. Evaluating Classification
5. Decision Trees
6. The Overfitting Problem
7. Other Classification Approaches
8. Hyperparameter Tuning

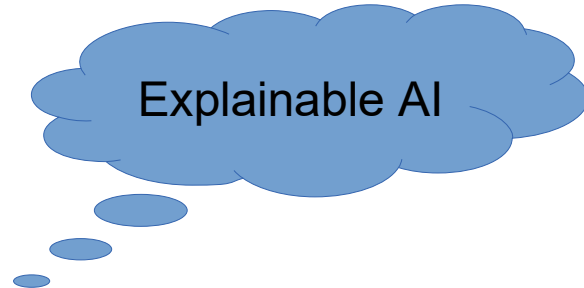
# Lazy vs. Eager Learning

- Both k-NN and Naïve Bayes are “lazy” methods
- They do not build an explicit model!
  - “learning” is only performed on demand for unseen records



# Today: Eager Learning

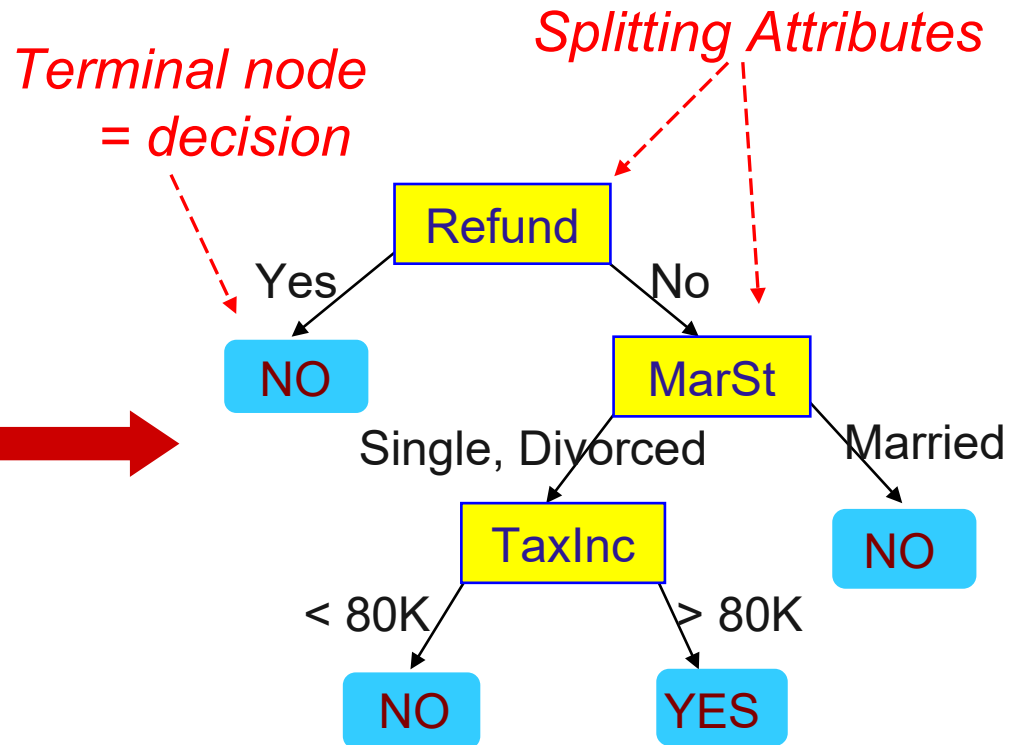
- Actually, we have two goals
  - classify unseen instances
  - learn a model
- Model
  - explains how to classify unseen instances
  - sometimes: interpretable by humans



# Decision Tree Classifiers

<i>Tid</i>	<i>Refund</i>	<i>Marital Status</i>	<i>Taxable Income</i>	<i>Cheat</i>
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	No	Single	90K	Yes

Training Data

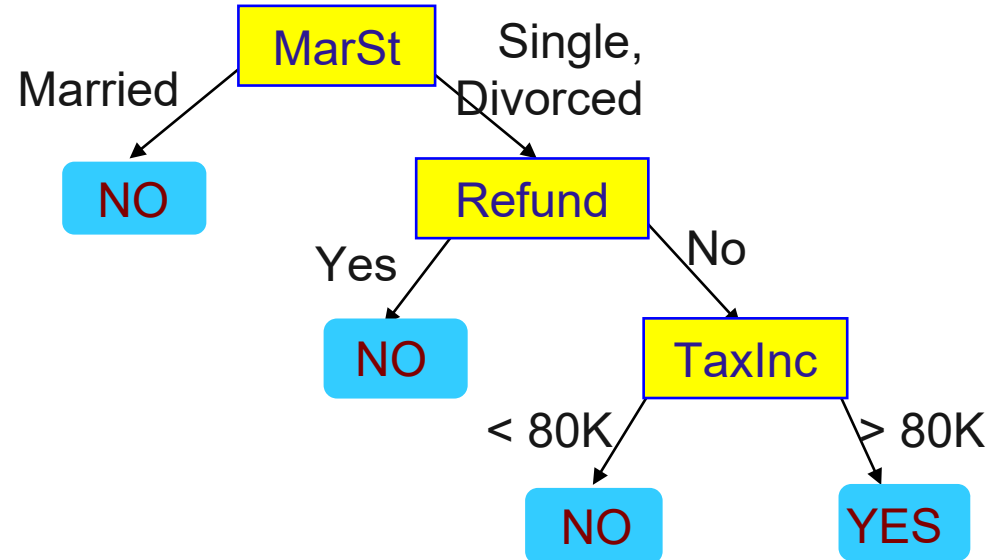


Model: Decision Tree

# Another Example of a Possible Decision Tree

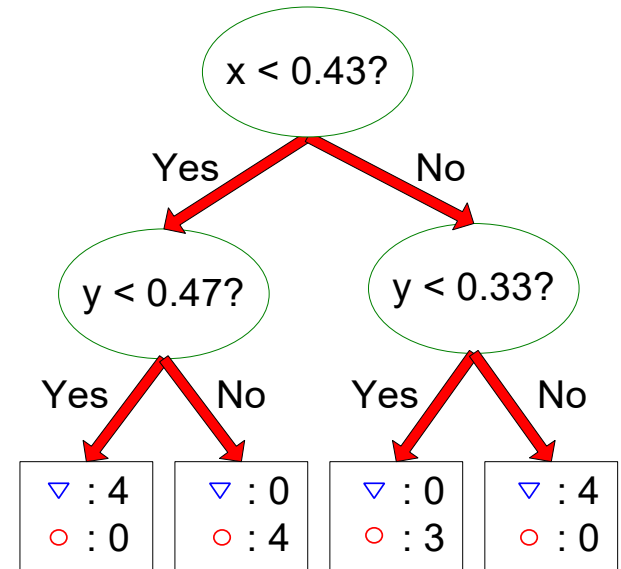
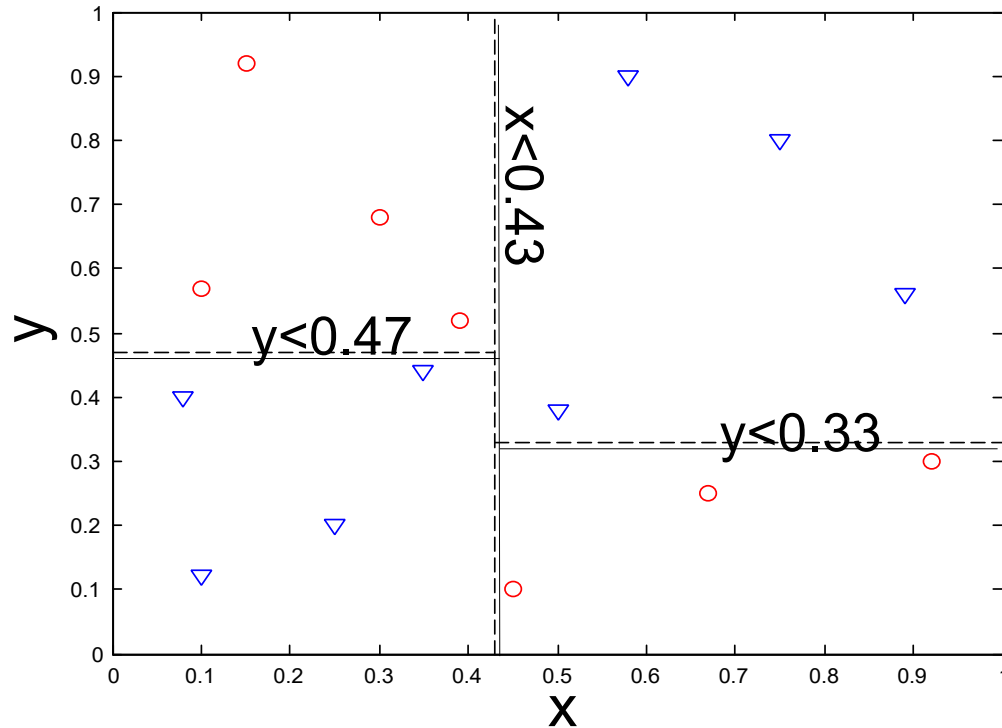
<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

categorical  
categorical  
continuous  
class



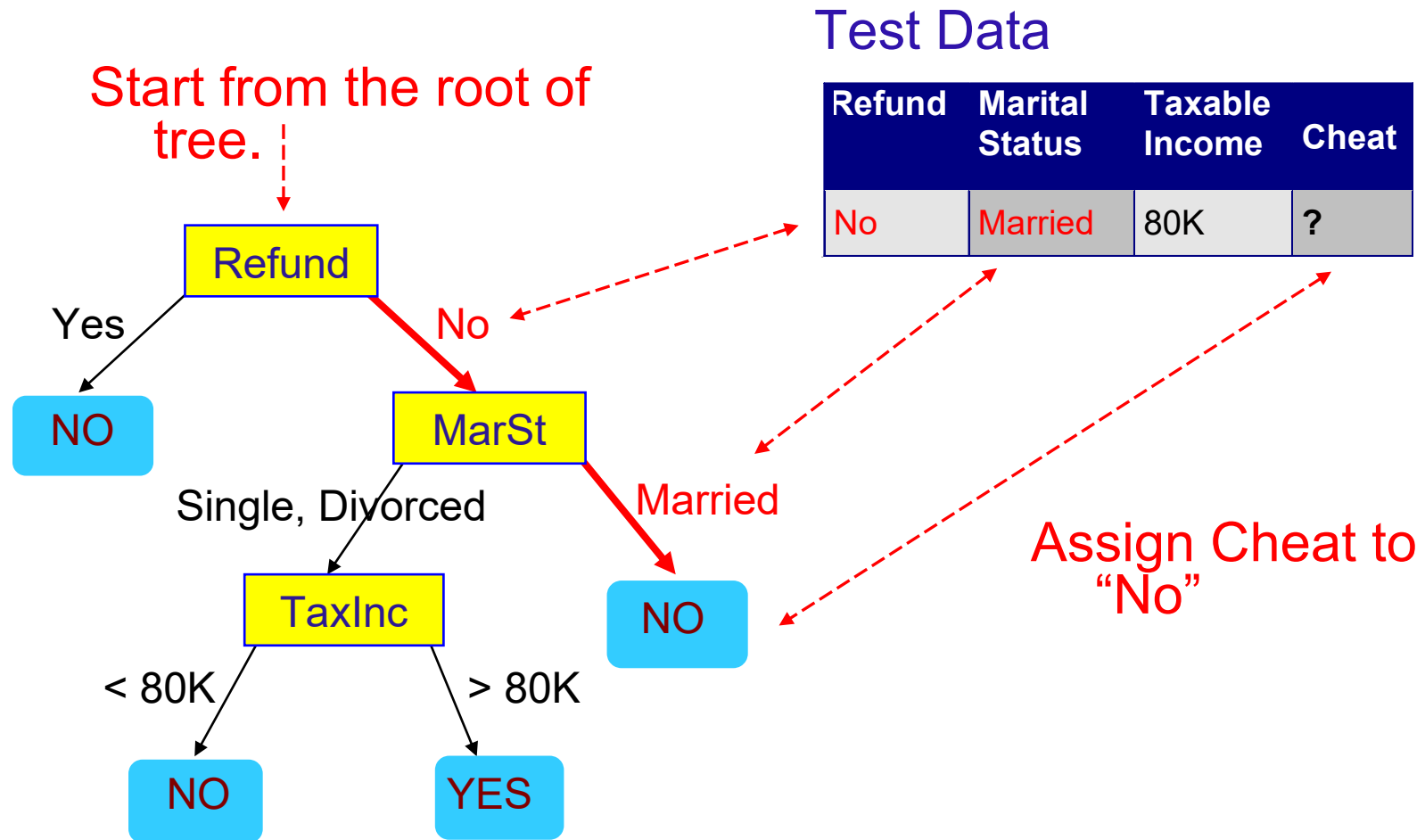
There can be more than one tree that fits the same data!

# 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

# Applying a Decision Tree to Test Data





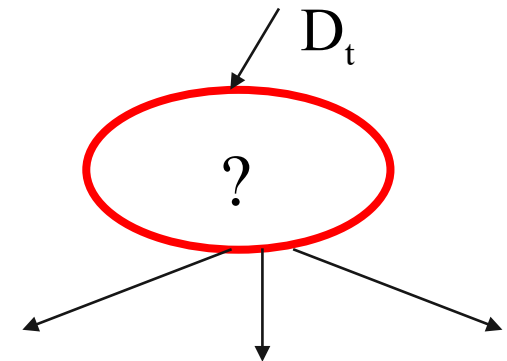
# Decision Tree Induction

- How to learn a decision Tree from test data?
- Finding an optimal decision tree is NP-hard
- Tree building algorithms use a greedy, top-down, recursive partitioning strategy to induce a reasonable solution
  - also known as: divide and conquer
- Many different algorithms have been proposed:
  - Hunt's Algorithm
  - ID3
  - CHAID
  - C4.5

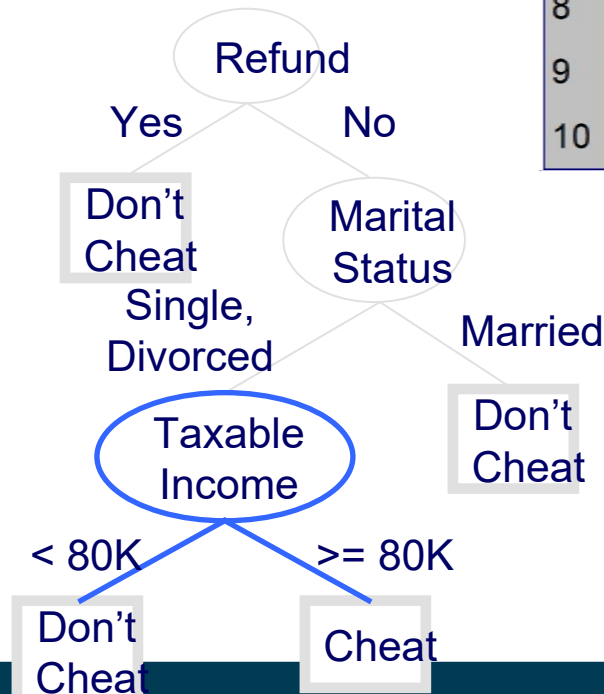
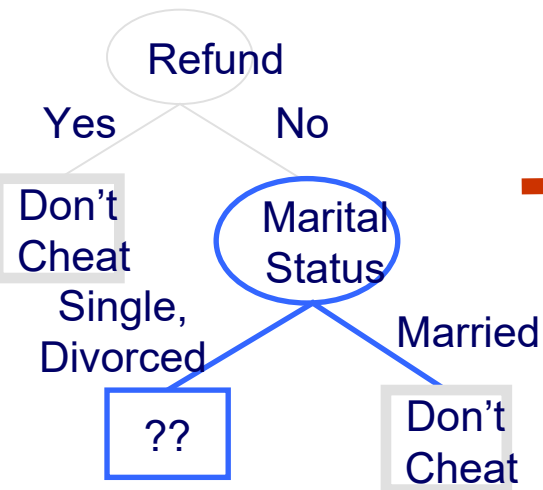
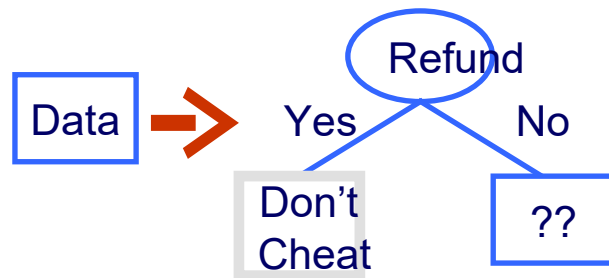
# General Structure of Hunt's Algorithm

- Let  $D_t$  be the set of training records that reach a node  $t$
- General Procedure:
  - If  $D_t$  contains only records that belong to the **same class**  $y_t$ , then  $t$  is a **leaf node** labeled as  $y_t$
  - If  $D_t$  contains records that belong to **more than one class**, use an **attribute test** to split the data into smaller subsets
  - **Recursively** apply the procedure to each subset

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
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	No	Single	90K	Yes



# Hunt's Algorithm



Tid	Refund	Marital Status	Taxable Income	Cheat
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	No	Single	90K	Yes

# Tree Induction Issues

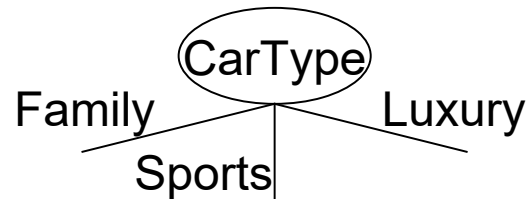
- Determine how to split the records
  - How to specify the attribute test condition?
  - How to determine the best split?
- Determine when to stop splitting

# How to Specify the Attribute Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous
- Depends on number of ways to split
  - 2-way split
  - Multi-way split

# Splitting Based on Nominal Attributes

- **Multi-way split:** Use as many partitions as distinct values

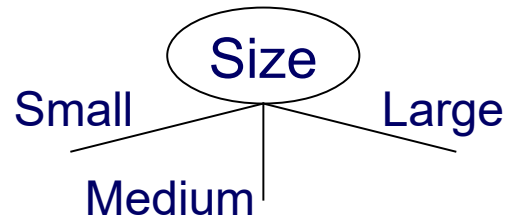


- **Binary split:** Divides values into two subsets.  
Need to find optimal partitioning

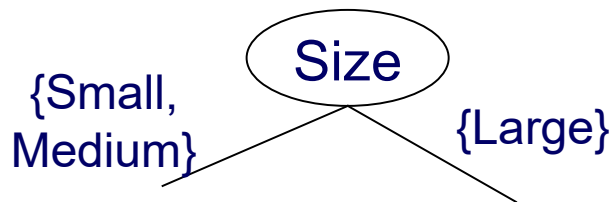


# Splitting Based on Ordinal Attributes

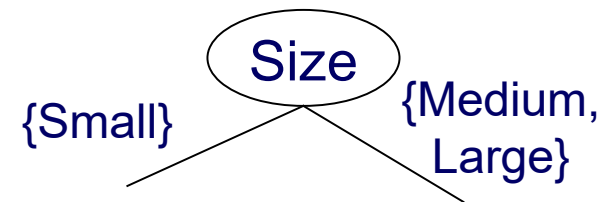
- **Multi-way split:** Use as many partitions as distinct values.



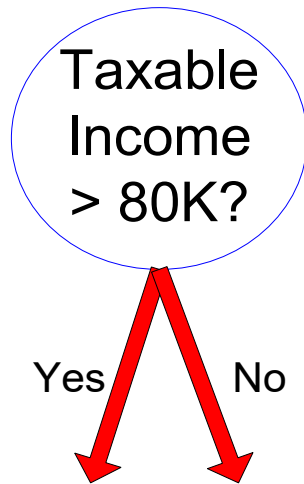
- **Binary split:** Divides values into two subsets, while keeping the order.  
Need to find optimal partitioning.



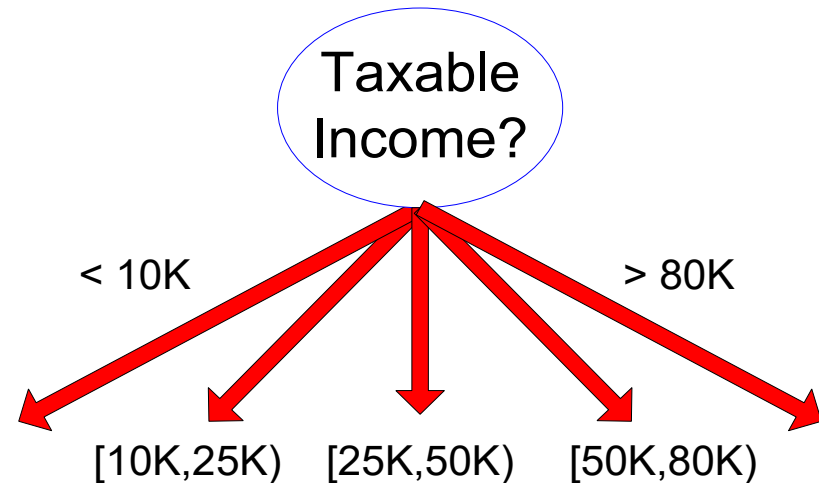
OR



# Splitting Based on Continuous Attributes



(i) Binary split



(ii) Multi-way split



# Splitting Based on Continuous Attributes

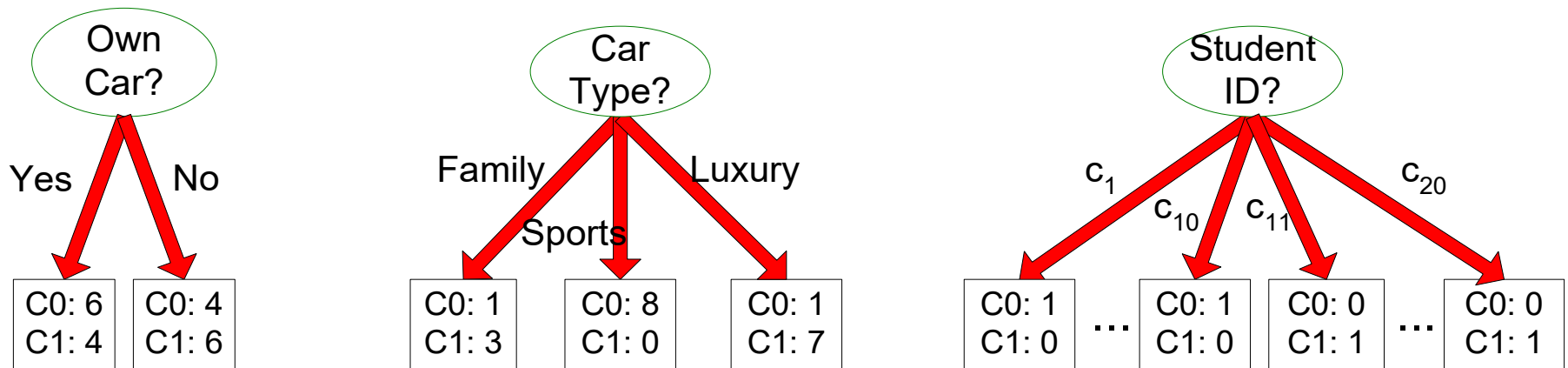
- Different ways of handling
  - **Discretization** to form an ordinal categorical attribute
    - equal-interval binning
    - equal-frequency binning
    - binning based on user-provided boundaries
  - **Binary Decision**:  $(A < v)$  or  $(A \geq v)$ 
    - usually sufficient in practice
    - consider all possible splits
    - find the best cut (i.e., the best  $v$ ) based on a purity measure (see later)
    - can be computationally expensive

# Discretization Example

- Attribute values (for one attribute e.g., age):
  - 0, 4, 12, 16, 16, 18, 24, 26, 28
- Equal-width binning – for bin width of e.g., 10:
  - Bin 1: 0, 4 [ $-\infty$ , 10) bin
  - Bin 2: 12, 16, 16, 18 [10, 20) bin
  - Bin 3: 24, 26, 28 [20,  $+\infty$ ) bin
    - $\infty$  denotes negative infinity,  $+\infty$  positive infinity
- Equal-frequency binning – for bin density of e.g., 3:
  - Bin 1: 0, 4, 12 [ $-\infty$ , 14) bin
  - Bin 2: 16, 16, 18 [14, 21) bin
  - Bin 3: 24, 26, 28 [21,  $+\infty$ ] bin

# How to determine the Best Split?

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



**Which test condition is the best?**

# How to determine the Best Split?

- Nodes with **homogeneous** class distribution are preferred
- Need a measure of node impurity:

C0: 5
C1: 5

Non-homogeneous,  
High degree of impurity

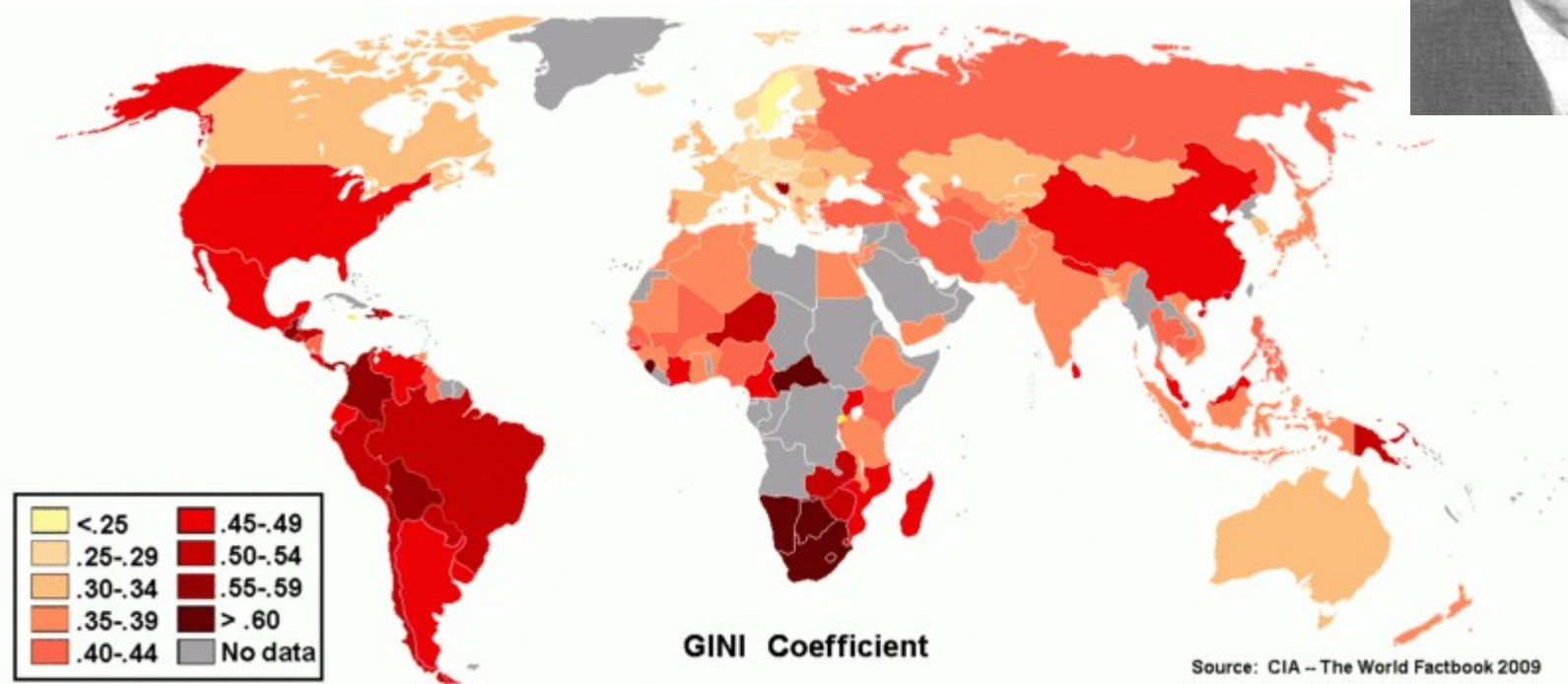
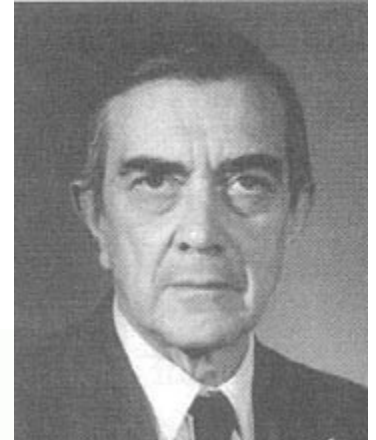
C0: 9
C1: 1

Homogeneous,  
Low degree of impurity

- Common measures of node impurity:
  - Gini Index
  - Entropy
  - Misclassification error

# Gini Index

- Named after Corrado Gini (1885-1965)
- Used to measure the distribution of income
  - 1: somebody gets everything
  - 0: everybody gets an equal share



# Measure of Impurity: GINI

- Gini-based purity measure for a given node  $t$  :

$$GINI(t) = 1 - \sum_j [p(j | t)]^2$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ ).

- Maximum ( $1 - 1/n_c$ ) when records are equally distributed among all classes, implying least interesting information
- Minimum (0.0) when all records belong to one class, implying most interesting information

C1	<b>0</b>
C2	<b>6</b>
<b>Gini=0.000</b>	

C1	<b>1</b>
C2	<b>5</b>
<b>Gini=0.278</b>	

C1	<b>2</b>
C2	<b>4</b>
<b>Gini=0.444</b>	

C1	<b>3</b>
C2	<b>3</b>
<b>Gini=0.500</b>	

# Splitting Based on GINI

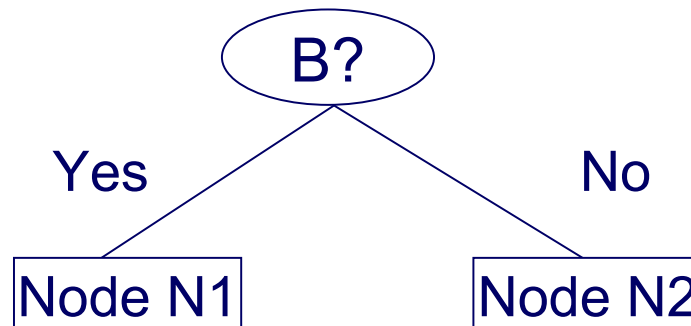
- When a node  $p$  is split into  $k$  partitions (children), the quality of split is computed as

$$GINI_{split} = \sum_{i=1}^k \frac{n_i}{n} GINI(i)$$

- where  $n_i$  = number of records at child  $i$ ,
  - $n$  = number of records at node  $p$ .
- Intuition:
  - The GINI index of each partition is weighted
  - according to the partition's size

# Binary Attributes: Computing GINI Index

- Splits into two partitions



	Parent
C1	6
C2	6
Gini = 0.500	

$$\begin{aligned} \text{Gini}(N1) &= 1 - (5/7)^2 - (2/7)^2 \\ &= 0.408 \end{aligned}$$

$$\begin{aligned} \text{Gini}(N2) &= 1 - (1/5)^2 - (4/5)^2 \\ &= 0.320 \end{aligned}$$

	N1	N2
C1	5	1
C2	2	4
Gini=0.371		

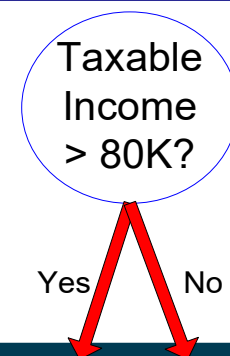
$$\begin{aligned} \text{Gini(Children)} &= 7/12 * 0.408 + \\ &\quad 5/12 * 0.320 \\ &= 0.371 \end{aligned}$$



# Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions,  $A < v$  and  $A \geq v$
- Simple method to choose best  $v$ 
  - For each  $v$ , scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work

Tid	Refund	Marital Status	Taxable Income	Cheat
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	No	Single	90K	Yes




# Continuous Attributes: Computing Gini Index

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	Taxable Income																		
	No	No	No	Yes	Yes	Yes	No	No	No	No									
	60	70	75	85	90	95	100	120	125	220									
Sorted Values →	55	65	72	80	87	92	97	110	122	172	230								
Split Positions →	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >	<= >								
Yes	0 3	0 3	0 3	0 3	1 2	2 1	3 0	3 0	3 0	3 0	3 0								
No	0 7	1 6	2 5	3 4	3 4	3 4	3 4	4 3	5 2	6 1	7 0								
Gini	0.420	0.400	0.375	0.343	0.417	0.400	<u>0.300</u>	0.343	0.375	0.400	0.420								

# Continuous Attributes: Computing Gini Index

- Note: it is enough to compute the GINI for those positions where the label changes!



Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No												
	Taxable Income																					
	60		70		75		85		90		95		100		120		125		220			
Sorted Values →	55		65		72		80		87		92		97		110		122		172		230	
Split Positions →	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		<u>0.300</u>		0.343		0.375		0.400		0.420	

# Alternative Splitting Criteria: Information Gain

- Entropy at a given node  $t$ :

$$Entropy(t) = -\sum_j p(j | t) \log_2 p(j | t)$$

(NOTE:  $p(j | t)$  is the relative frequency of class  $j$  at node  $t$ ).

- Measures homogeneity of a node
  - Maximum ( $\log nc$ ) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
- Entropy based computations are similar to the GINI index computations

# Splitting Based on Information Gain

- Information Gain:

$$GAIN_{split} = Entropy(p) - \left( \sum_{i=1}^k \frac{n_i}{n} Entropy(i) \right)$$

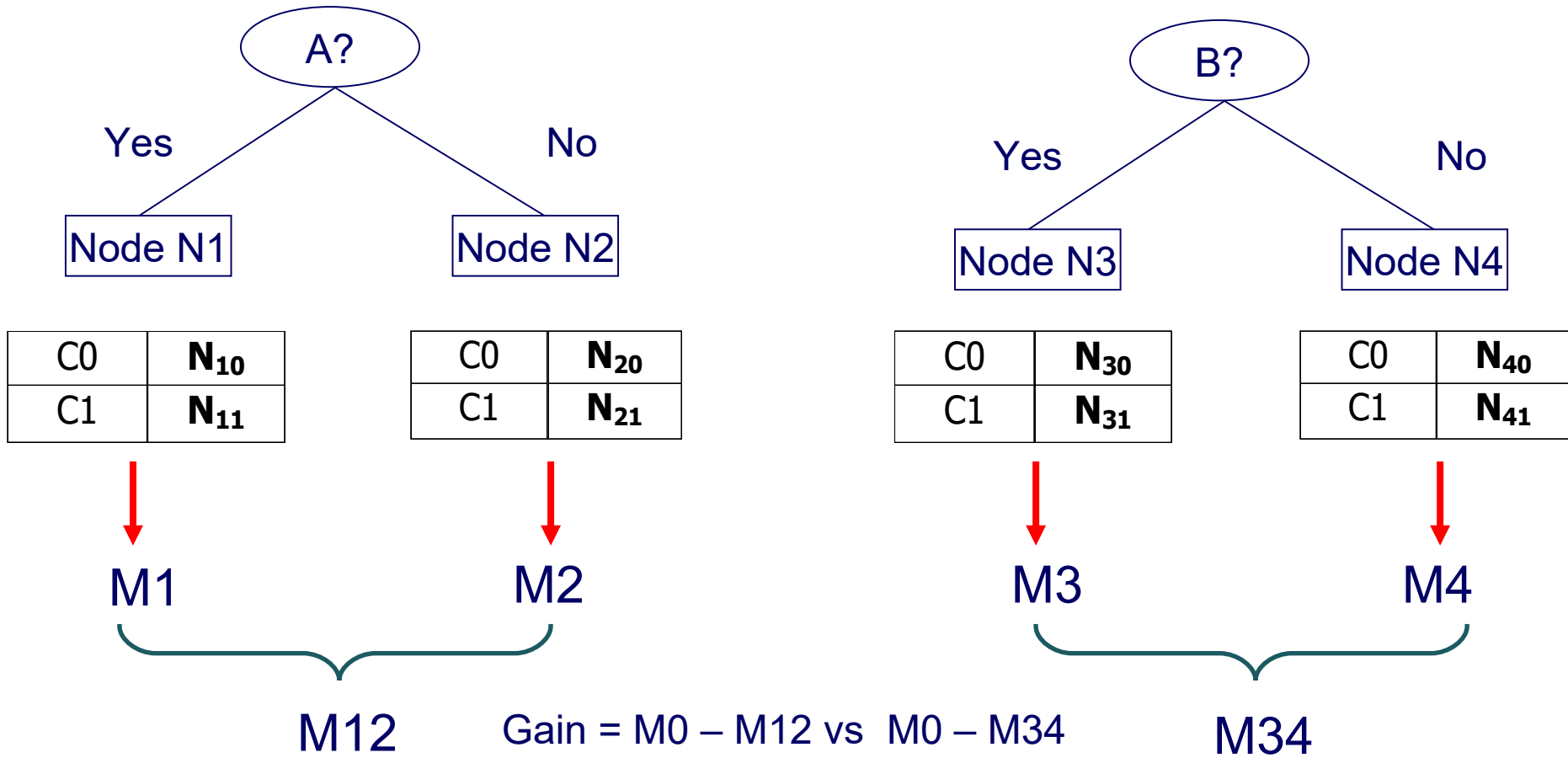
- Parent Node, p is split into k partitions;
- $n_i$  is number of records in partition i
- Measures reduction in entropy achieved because of the split
  - Choose the split that achieves most reduction (maximizes GAIN)
- Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure
  - e.g., split by ID attribute

# How to Find the Best Split

Before Splitting:

C0	$N_{00}$
C1	$N_{01}$

→ M0



# Discussion of Decision Trees

- Advantages:
  - Inexpensive to construct
  - Fast at classifying unknown records
  - Easy to interpret by humans for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets
- Disadvantages:
  - Decisions are based only on a single attribute at a time
  - Can only represent decision boundaries that are parallel to the axes

# Comparing Decision Trees and k-NN

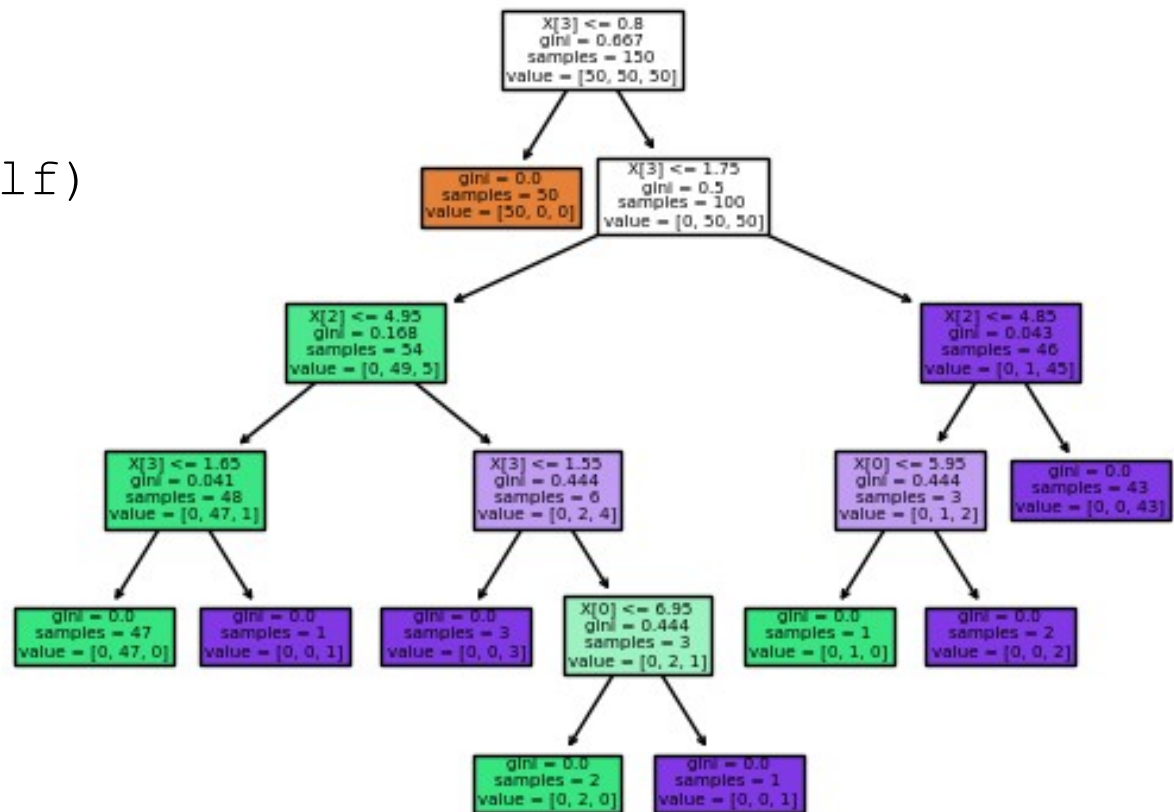
- Decision boundaries
  - k-NN: arbitrary
  - Decision trees: rectangular
- Sensitivity to scales
  - k-NN: needs normalization
  - Decision tree: does not require normalization (recap: Gini splitting)
- Runtime & memory
  - k-NN is cheap to train, but expensive for classification
  - decision tree is expensive to train, but cheap for classification



# Tree Induction in Python

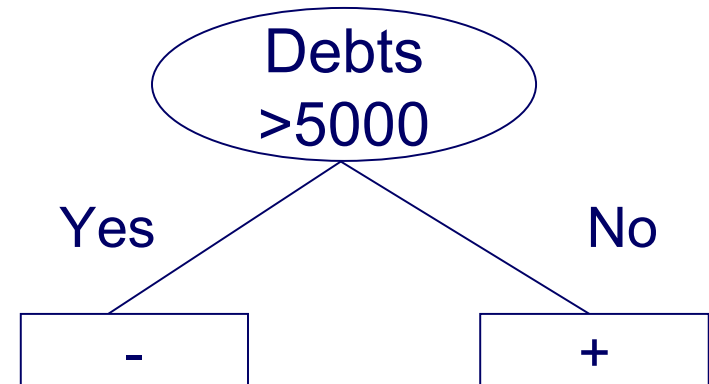
```
clf = DecisionTreeClassifier()  
clf = clf.fit(X,X_labels)
```

```
# Visualization  
tree.plot_tree(clf)
```



# Practical Issue: Overfitting

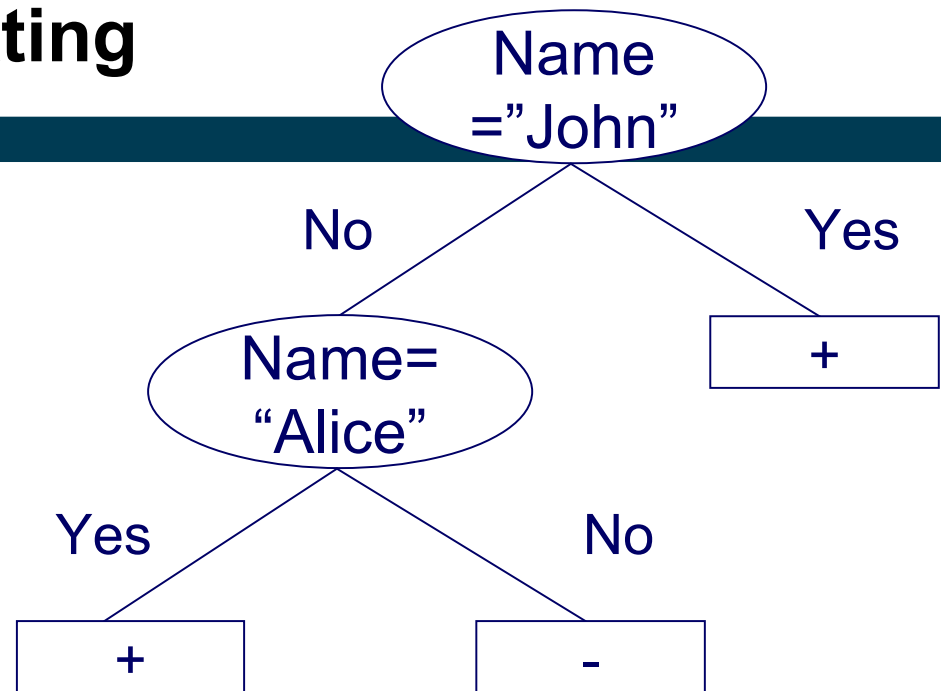
- Example: predict credit rating
  - possible decision tree:



Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

# Practical Issue: Overfitting

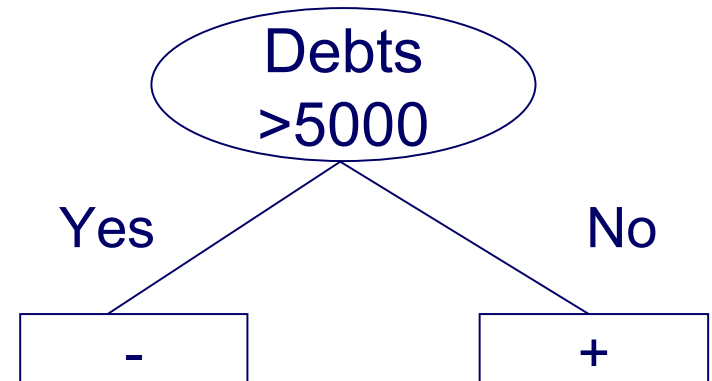
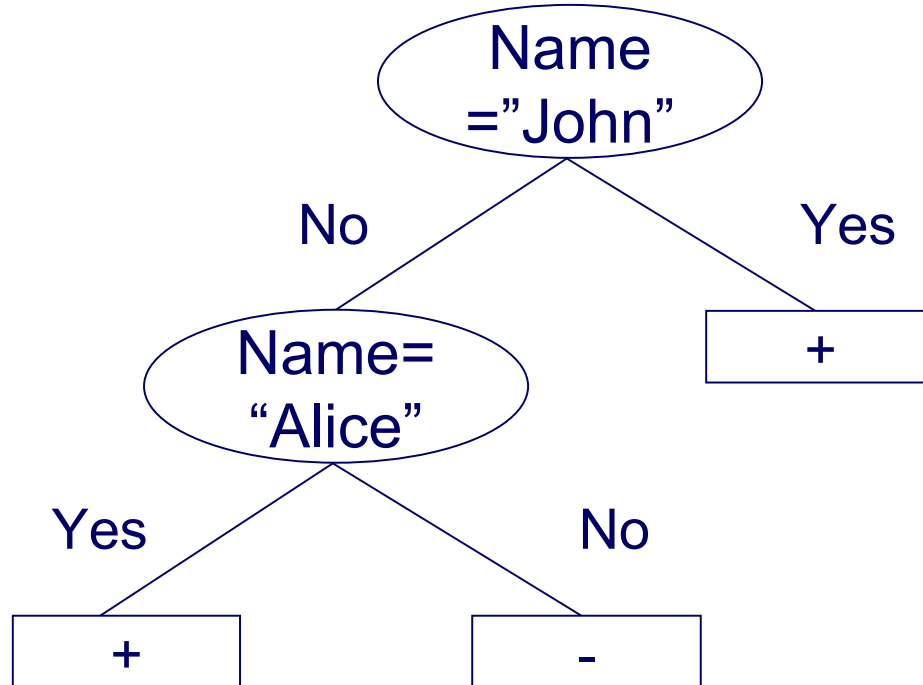
- Example: predict credit rating
  - alternative decision tree:



Name	Net Income	Job status	Debts	Rating
John	40000	employed	0	+
Mary	38000	employed	10000	-
Stephen	21000	self-employed	20000	-
Eric	2000	student	10000	-
Alice	35000	employed	4000	+

# Practical Issue: Overfitting

- Both trees seem equally good
  - Classify all instances in the training set correctly
  - Which one do you prefer?



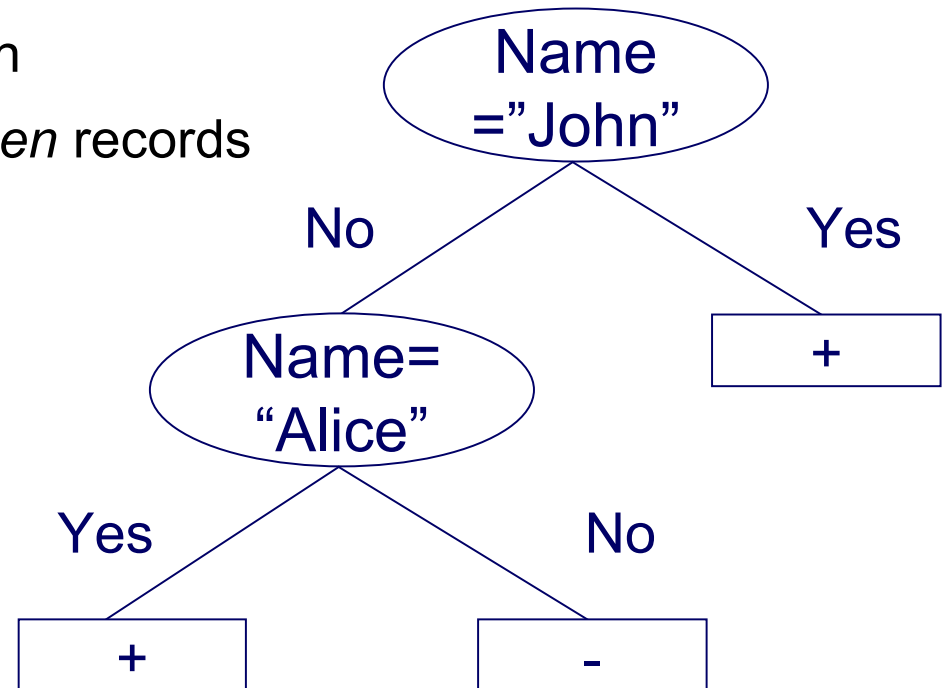
# Occam's Razor

- Named after William of Ockham (1287-1347)
- A fundamental principle of science
  - if you have two theories
  - that explain a phenomenon equally well
  - choose the simpler one
- Example:
  - phenomenon: the street is wet
  - theory 1: *it has rained*
  - theory 2: *a beer truck has had an accident, and beer has spilled. The truck has been towed, and magpies picked the glass pieces, so only the beer remains*



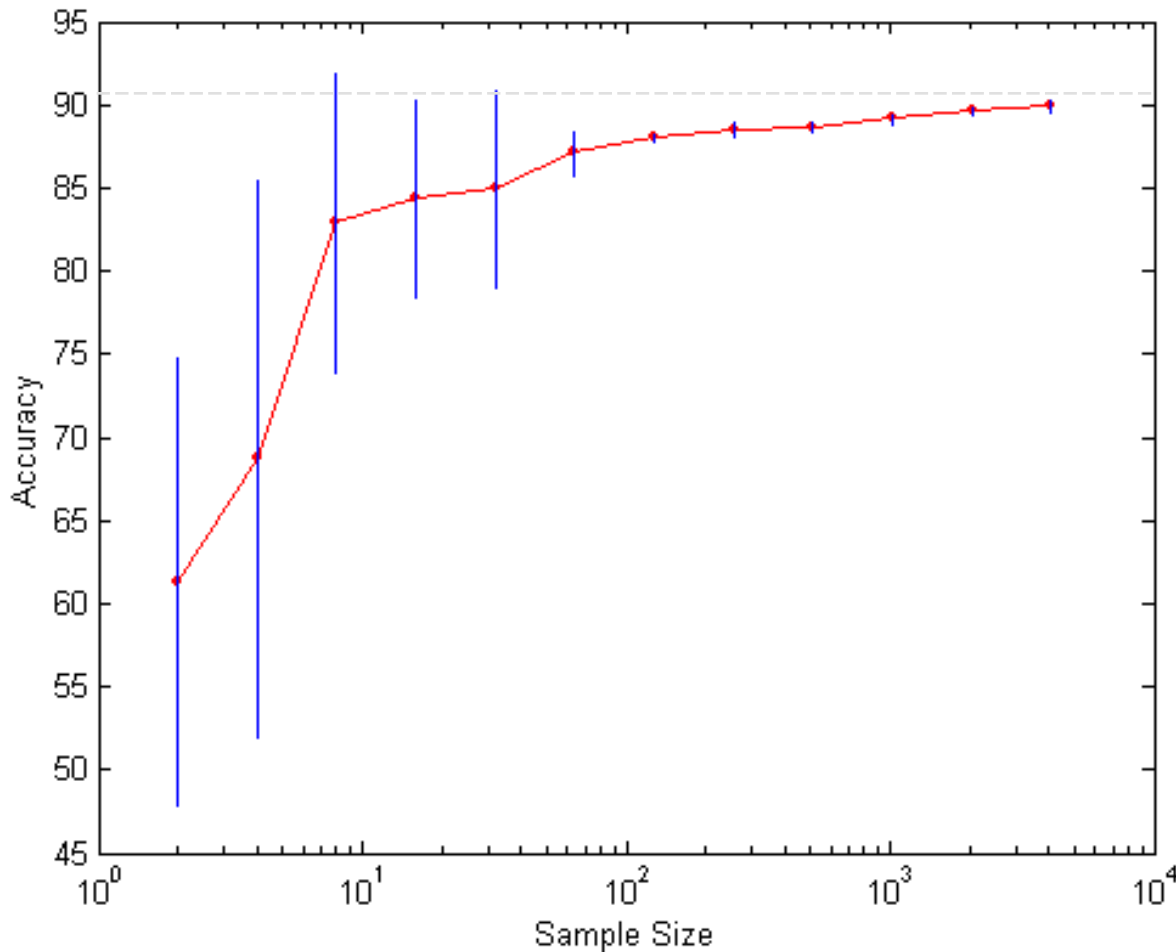
# Training and Testing Data

- Consider the decision tree again
- Our ultimate goal: classify *unseen* records



- Assume you measure the performance using the training data
- Conclusion:
  - We need separate data for testing

# Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Conclusion: Use as much data as possible for training
- At the same time: variation drops with larger evaluation sets
- Conclusion: use as much data as possible for evaluation

# Holdout Method



- The *holdout method* reserves a certain amount for testing and uses the remainder for training
- Typical: one third for testing, the rest for training
- applied when *lots of sample data* is available
- For unbalanced datasets, samples might not be representative
  - Few or none instances of some classes
- *Stratified sample*: *balances the data*
  - Make sure that each class is represented with approximately equal proportions in both subsets



# Leave One Out

- Iterate over all examples
  - train a model on all examples but the current one
  - evaluate on the current one
- Yields a very accurate estimate
- Uses as much data for training as possible
  - but is computationally infeasible in most cases
- Imagine: a dataset with a million instances
  - one minute to train a single model
  - Leave one out would take almost two years

# Cross-Validation

- Compromise of Leave One Out and decent runtime 
- *Cross-validation* avoids overlapping test sets 
  - First step: data is split into  $k$  subsets of equal size
  - Second step: each subset in turn is used for testing and the remainder for training
- This is called *k-fold cross-validation*
- The error estimates are averaged to yield an overall error estimate
- Frequently used value for  $k$  : 10
  - Why ten? Extensive experiments have shown that this is the good choice to get an accurate estimate
- Often the subsets are stratified before the cross-validation is performed

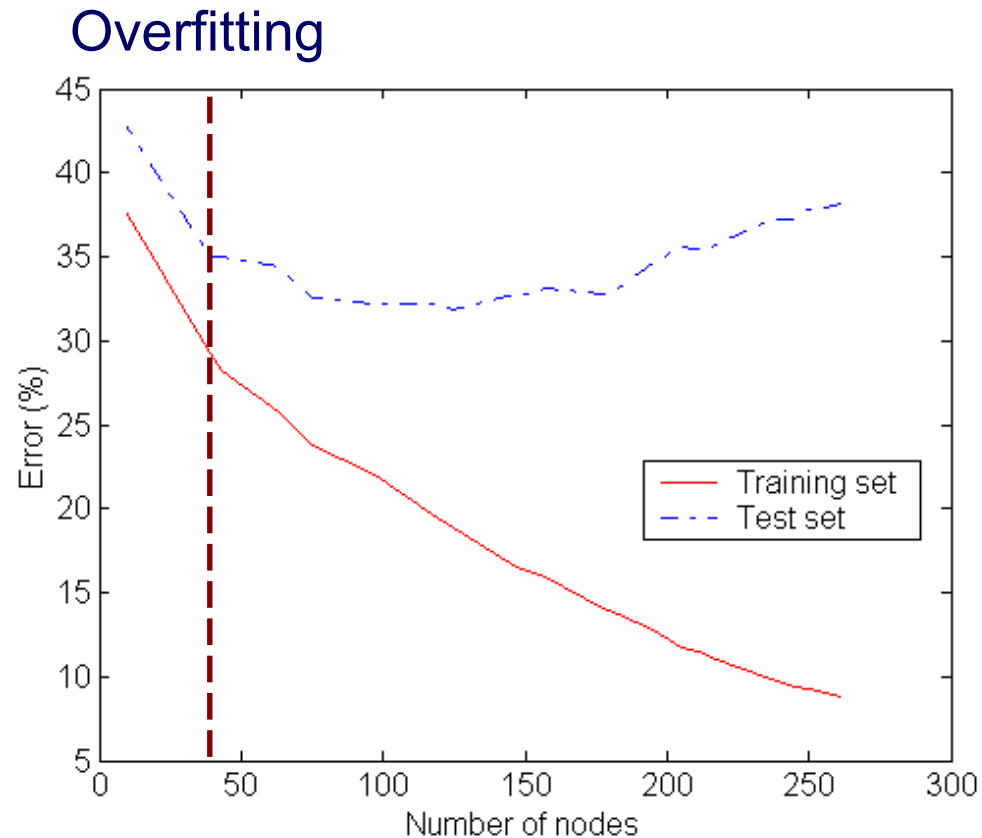
# Cross-Validation in Python

- Note: the gold standard for folds was long set to 10
  - Recent works on very large models have lead to a tendency of lowering that value
    - Or even using one fixed(!) train/test split
  - The default value in scikit-learn is 5

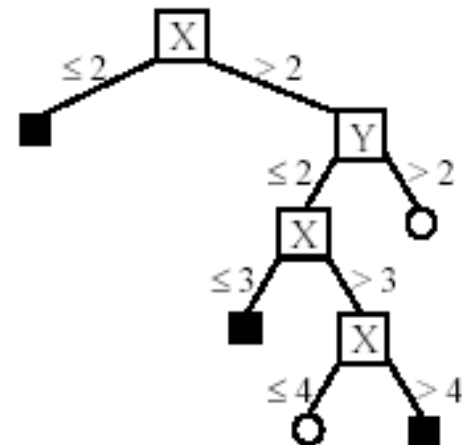
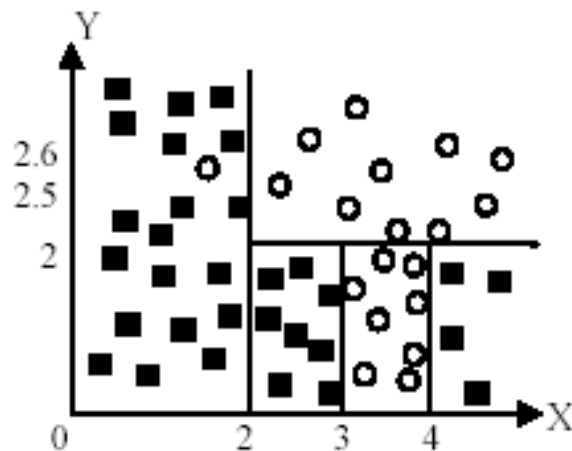
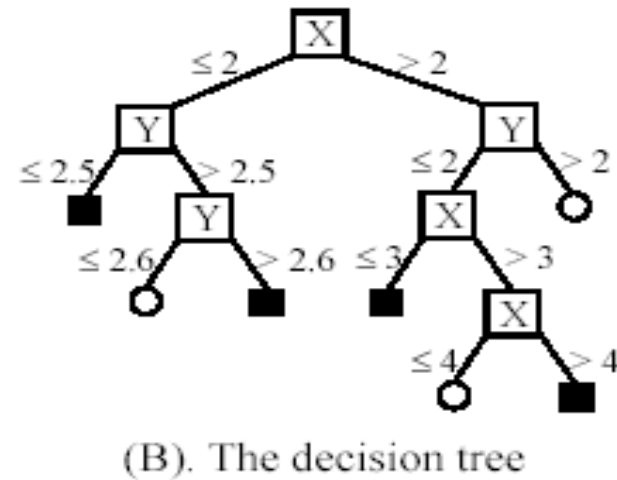
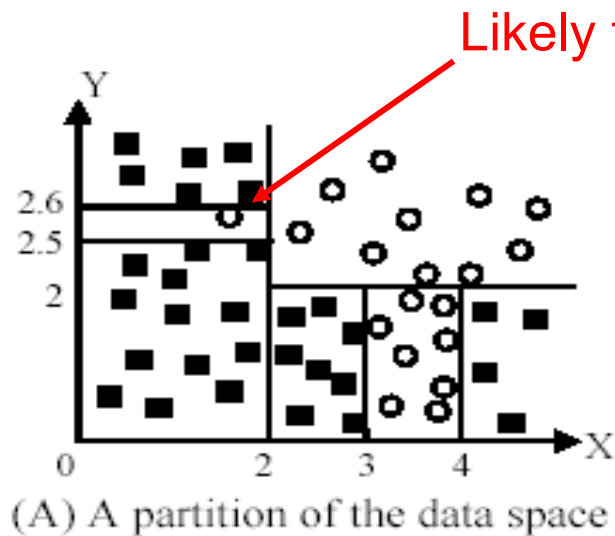
```
clf = KNeighborsClassifier(n_neighbors=3)
scores = cross_val_score(clf, data, target, cv=10)
```

# Back to Overfitting

- Overfitting: Good accuracy on training data, but poor on test data.
- Symptoms: Tree too deep and too many branches
- Typical causes of overfitting
  - too little training data
  - noise
  - poor learning algorithm



# Overfitting and Noise



# How to Address Overfitting?

- **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
  - Less restrictive conditions:
    - Stop if number of instances within a node is less than some user-specified threshold
    - Stop if expanding the current node only slightly improves the impurity measure (user-specified threshold)

# How to Address Overfitting?

- **Post-pruning**
  1. Grow decision tree to its entire size
  2. Trim the nodes of the decision tree in a bottom-up fashion
    - using a validation data set
    - or an estimate of the generalization error
  3. 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

# Training vs. Generalization Errors

- Training error
  - also: resubstitution error, apparent error
  - errors made in training
  - evidence: misclassified training instances
- Generalization error
  - errors made on unseen data
  - evidence: no apparent evidence
- Training error can be computed
- Generalization error must be estimated



# Estimating the Generalization Error

- **Training errors:** error on training ( $\sum e(t)$  )
- **Generalization errors:** error on testing ( $\sum e'(t)$ )
- Methods for estimating generalization errors:
  1. **(Too) Optimistic approach:**  $e'(t) = e(t)$
  2. **Pessimistic approach:**
    - For each leaf node:  $e'(t) = (e(t) + 0.5)$   
(user-defined 0.5 penalty for large trees)
    - Total errors:  $e'(T) = e(T) + N \times 0.5$   
(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 =  $(10 + 30 \times 0.5)/1000 = 2.5\%$
  3. **Reduced Error Pruning (REP):**
    - use validation data set to estimate generalization error

# Example of Post-Pruning

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

Training Error (Before splitting) = 10/30

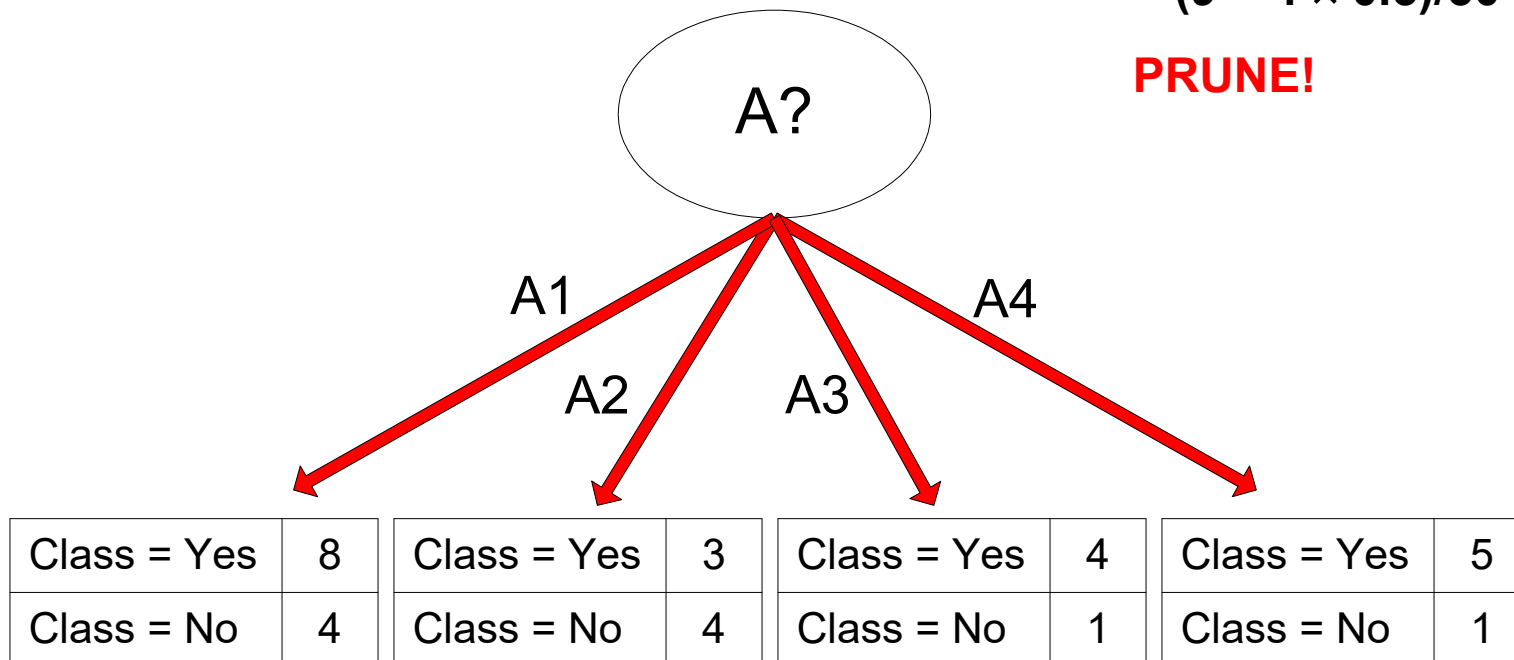
Pessimistic error =  $(10 + 0.5)/30 = 10.5/30$

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

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

**PRUNE!**



# Alternative Classification Methods

- So far, we have seen
  - k-NN
  - Naive Bayes
  - Decision Trees
- There is a whole lot more in RapidMiner & scikit-learn
- Brief intro
  - Artificial Neural Networks

# Example: Credit Rating

- Consider the following example:
  - and try to build a model
  - which is as small as possible (recall: Occam's Razor)

Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

# Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balanced Account are yes  
→ Get Credit is yes
- Not nicely expressible in trees and rule sets
  - as we know them (attribute-value conditions)

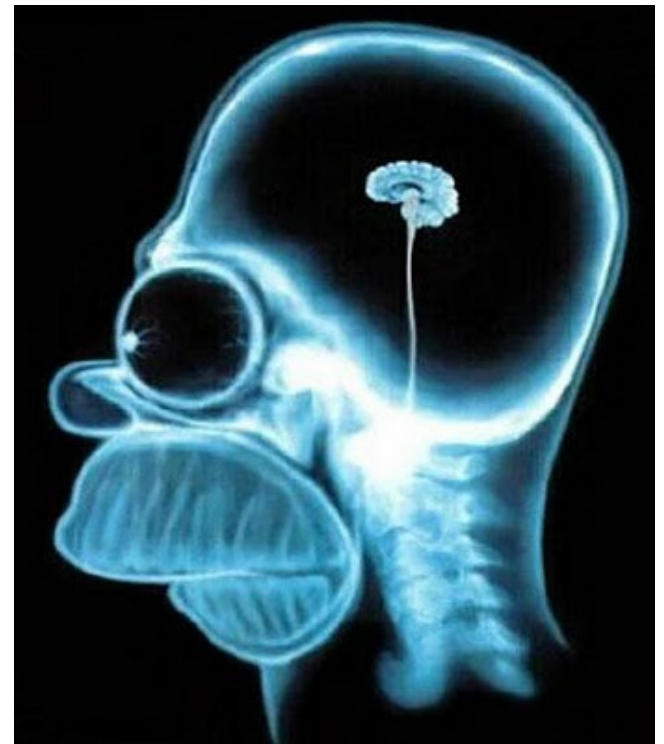
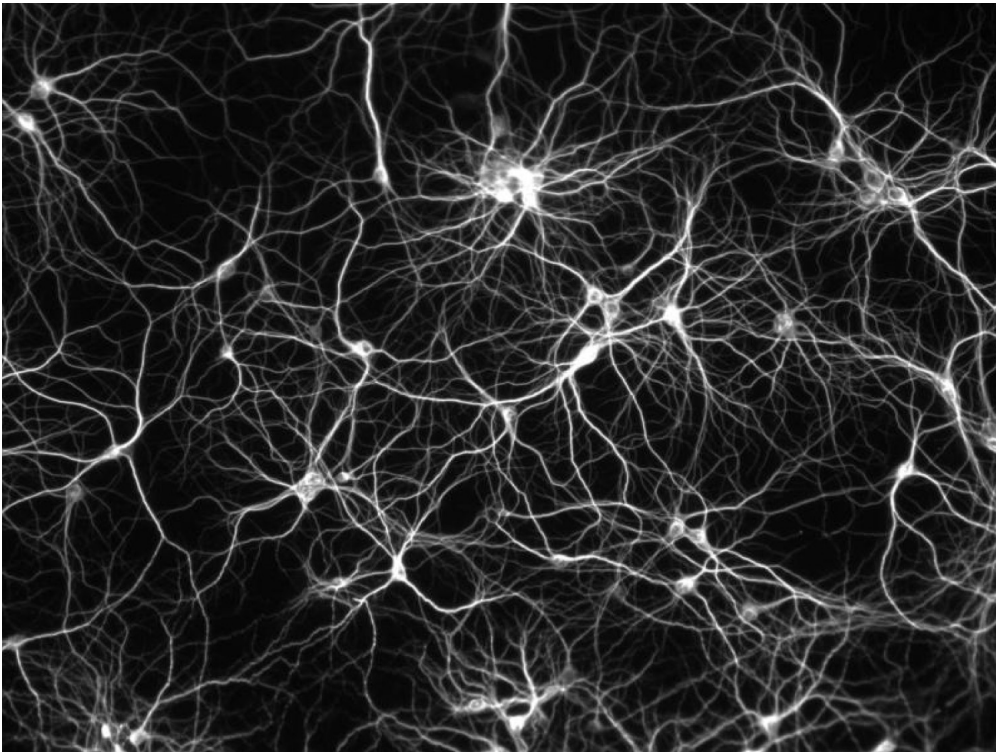
Person	Employed	Owns House	Balanced Account	Get Credit
Peter Smith	yes	yes	no	yes
Julia Miller	no	yes	no	no
Stephen Baker	yes	no	yes	yes
Mary Fisher	no	no	yes	no
Kim Hanson	no	yes	yes	yes
John Page	yes	no	no	no

# Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes  
→ Get Credit is yes
- As rule set:
  - Employed=yes and OwnsHouse=yes => yes
  - Employed=yes and BalanceAccount=yes => yes
  - OwnsHouse=yes and BalanceAccount=yes => yes
  - => no
- General case:
  - at least m out of n attributes need to be yes => yes
  - this requires  $\binom{n}{m}$  rules, i.e.,  $\frac{n!}{m! \cdot (n-m)!}$
  - e.g., “5 out of 10 attributes need to be yes”  
requires more than 15,000 rules!

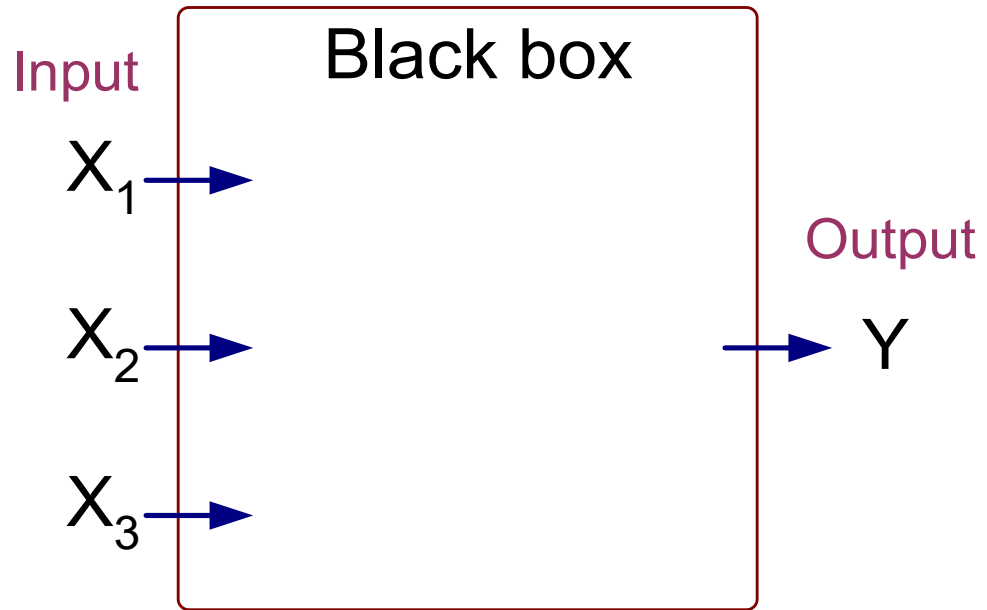
# Artificial Neural Networks

- Inspiration
  - one of the most powerful super computers in the world



# Artificial Neural Networks (ANN)

$X_1$	$X_2$	$X_3$	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



Output  $Y$  is 1 if at least two of the three inputs are equal to 1.

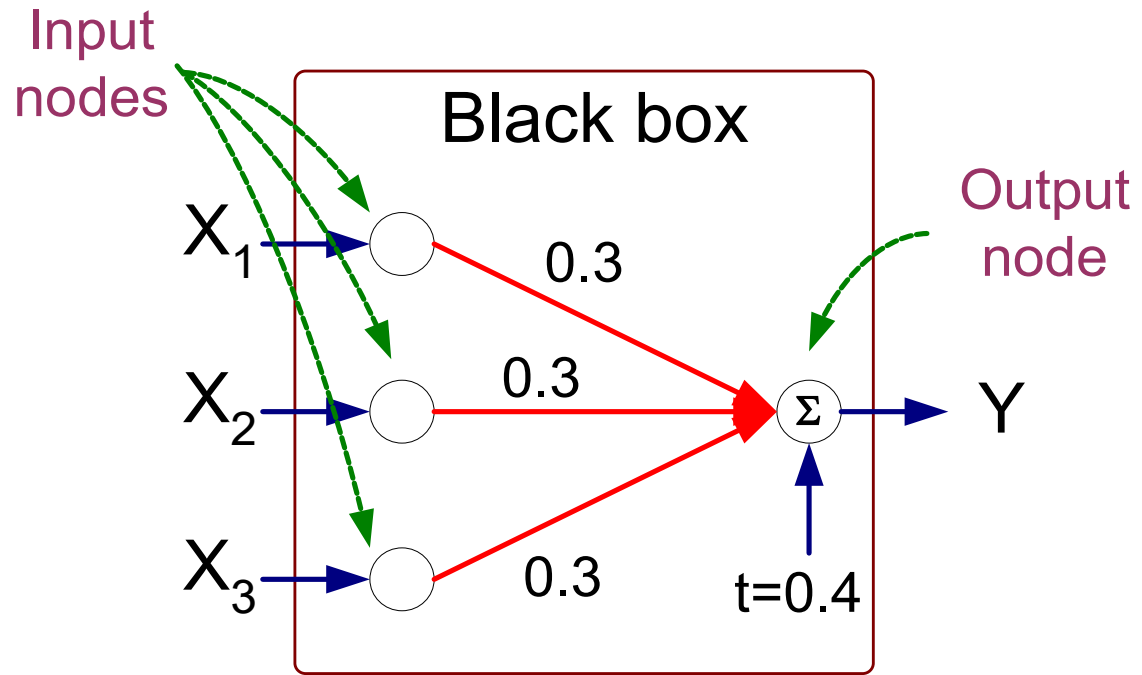


# Example: Credit Rating

- Smallest model:
  - if at least two of Employed, Owns House, and Balance Account are yes  
→ Get Credit is yes
- Given that we represent yes and no by 1 and 0, we want
  - if  $(\text{Employed} + \text{Owns House} + \text{Balance Account}) > 1.5$   
→ Get Credit is yes

# Artificial Neural Networks (ANN)

$X_1$	$X_2$	$X_3$	$Y$
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

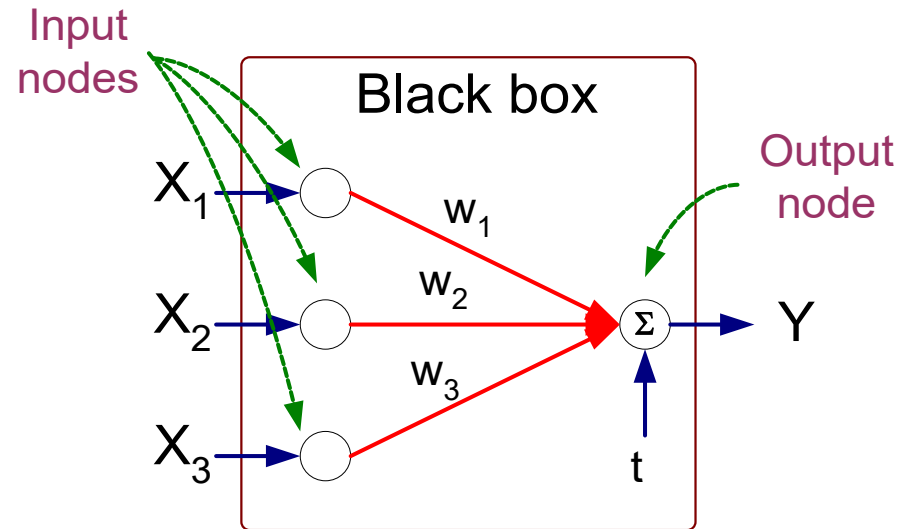


$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

# Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold  $t$

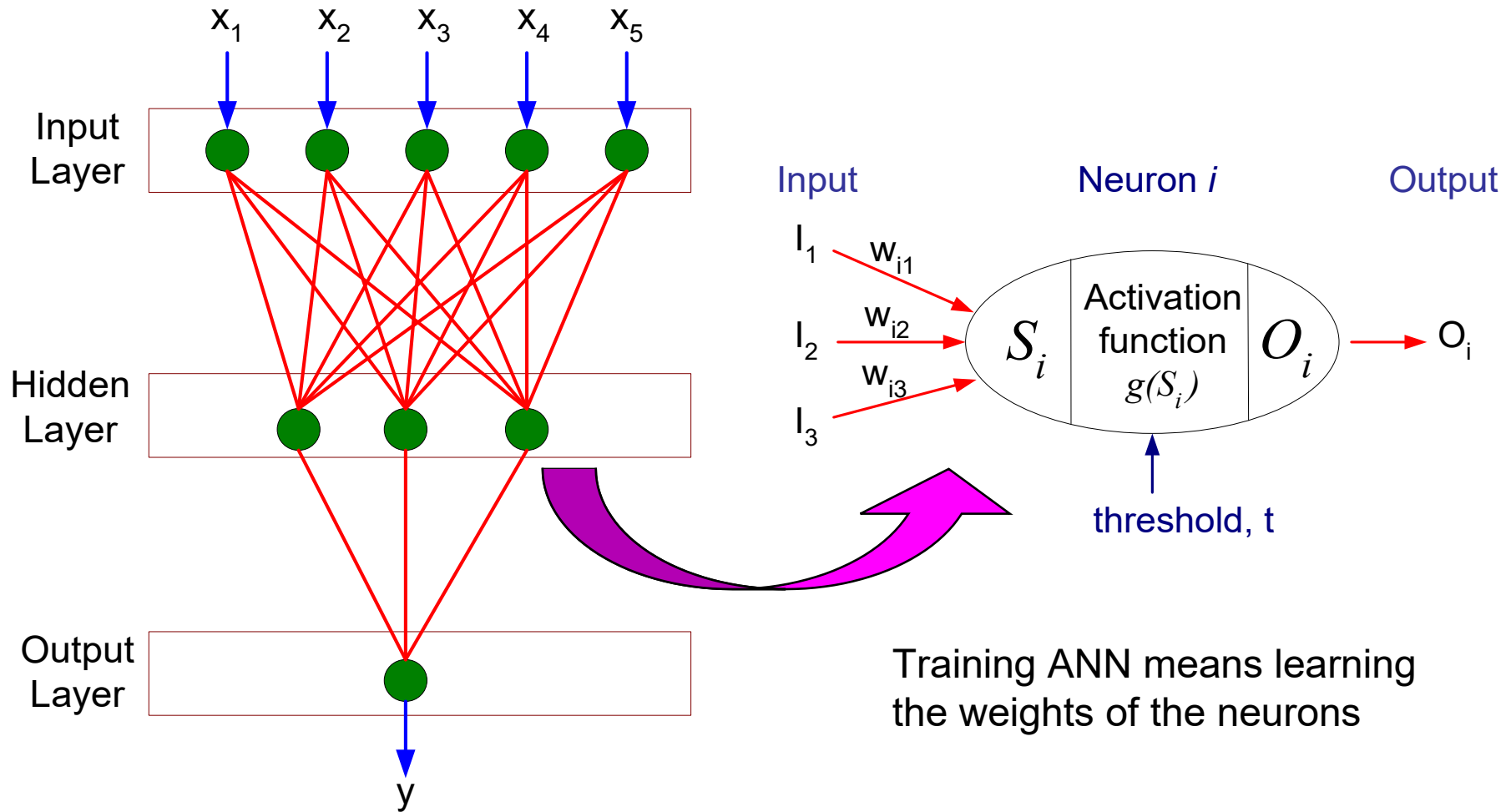


## Perceptron Model

$$Y = I\left(\sum_i w_i X_i - t\right) \quad \text{or}$$

$$Y = \text{sign}\left(\sum_i w_i X_i - t\right)$$

# General Structure of ANN

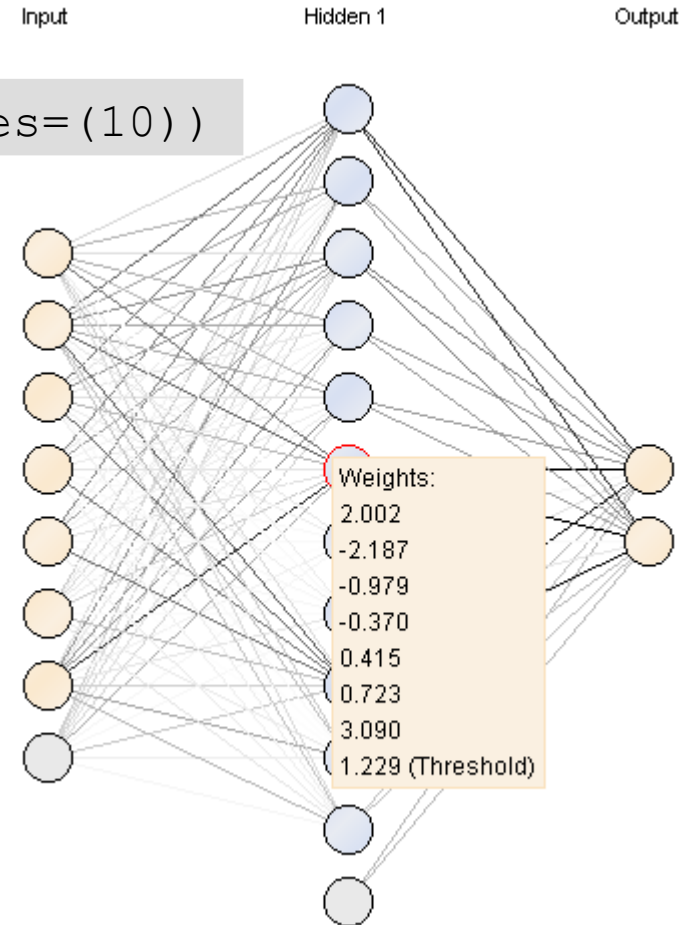


# Algorithm for learning ANN

- Initialize the weights ( $w_0, w_1, \dots, w_k$ ), e.g., all with 1
- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Objective function: 
$$E = \sum_i [Y_i - f(w_i, X_i)]^2$$
  - Find the weights  $w_i$ 's that minimize the above objective function
    - e.g., back propagation algorithm (see books)

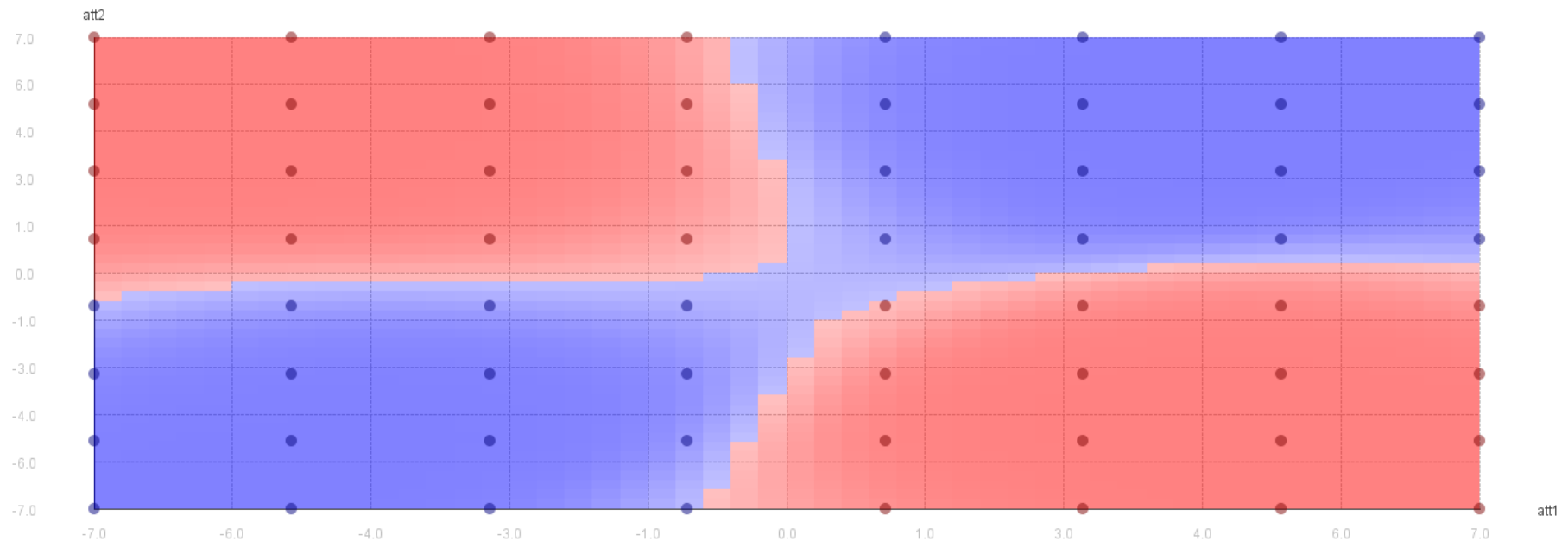
# ANN in Python

```
clf = MLPClassifier(hidden_layer_sizes=(10))
```

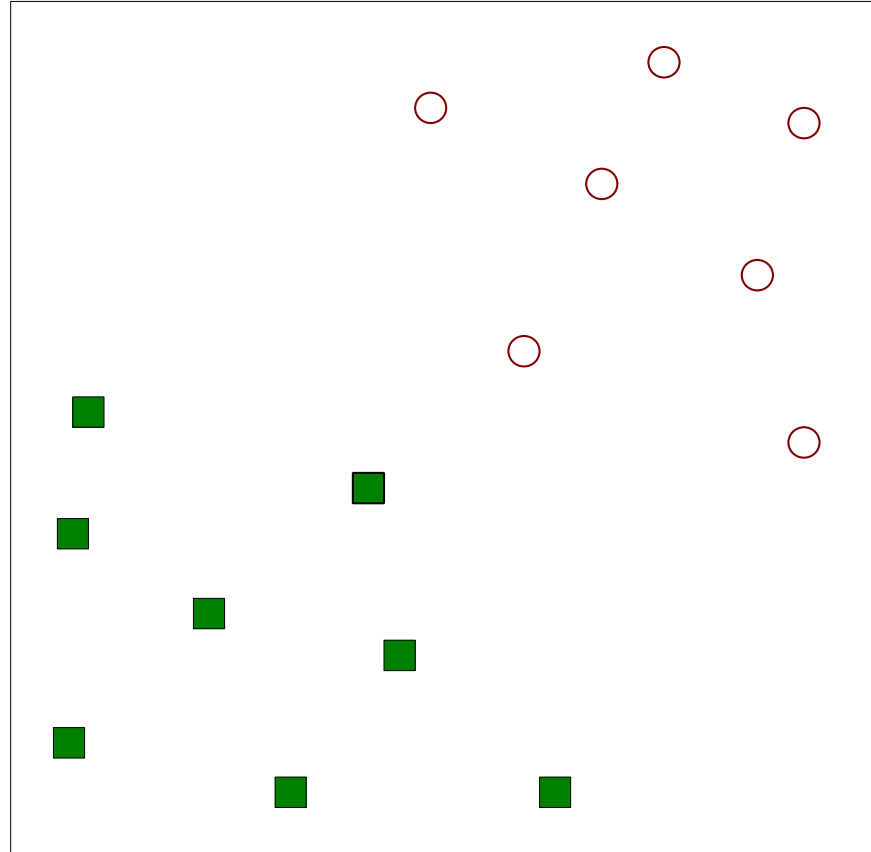


# Decision Boundaries of ANN

- Arbitrarily shaped objects
- Fuzzy boundaries



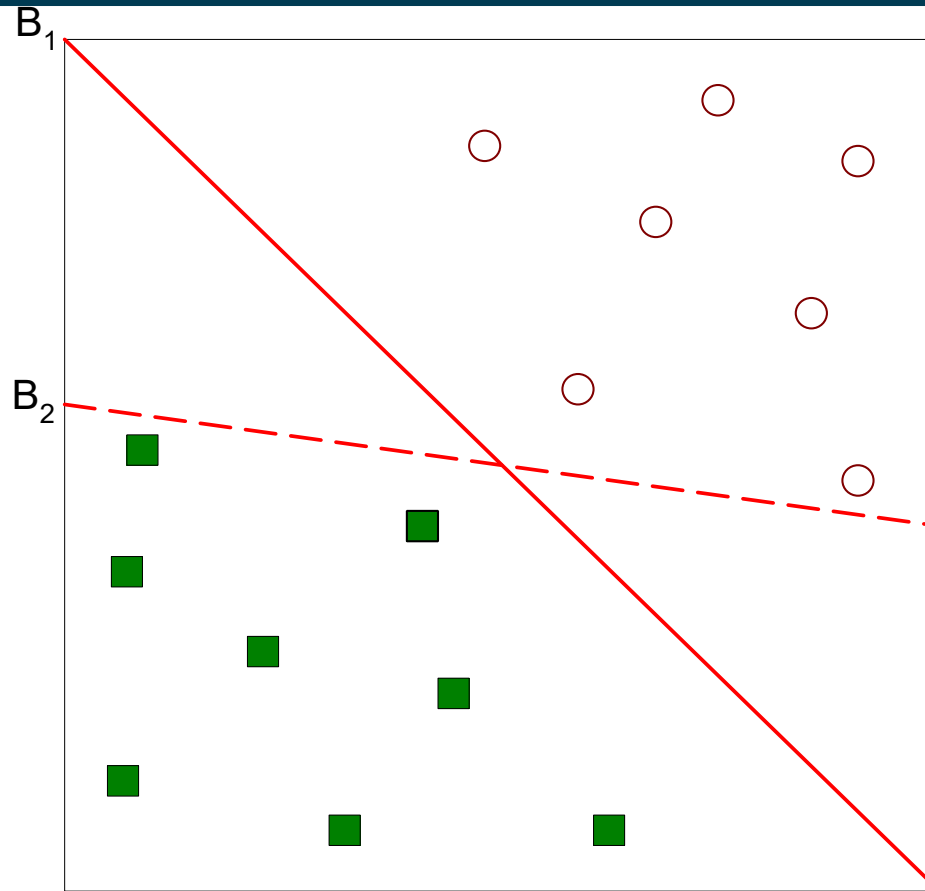
# Support Vector Machines



- Find a linear hyperplane (decision boundary) that will separate the data

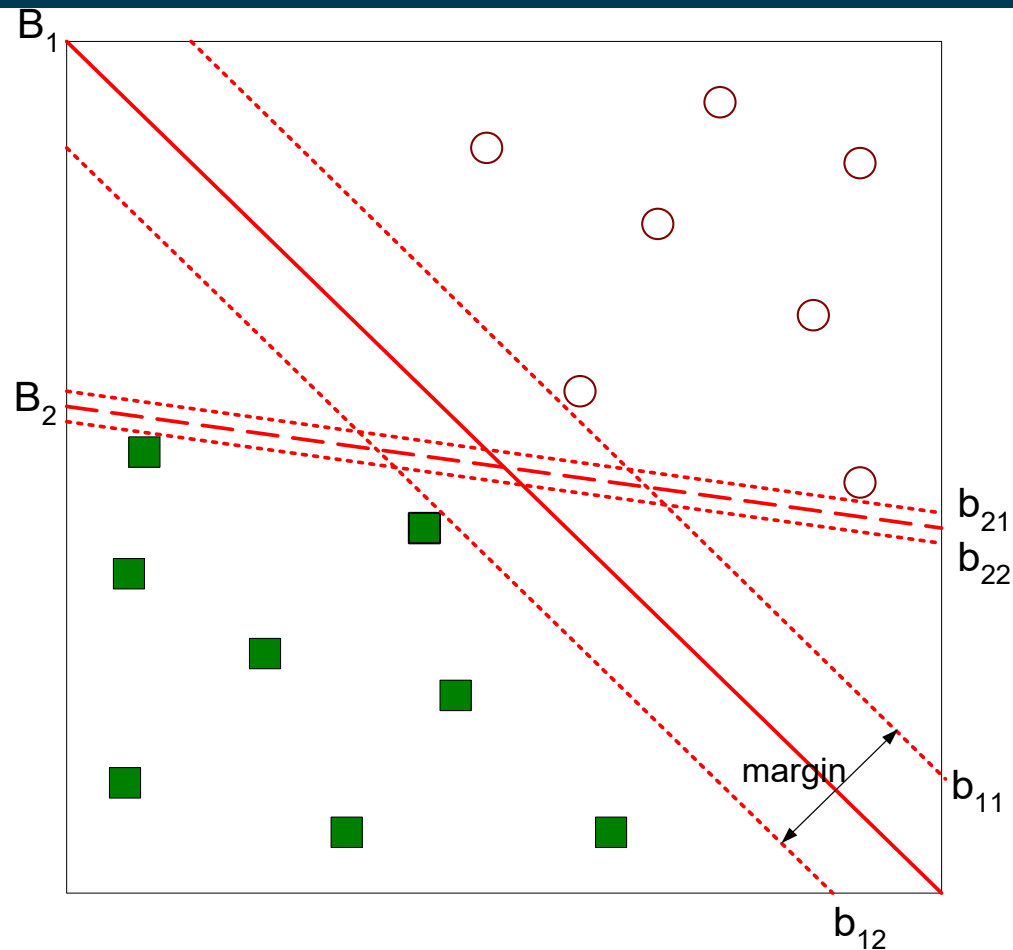


# Support Vector Machines



- Which one is better?  $B_1$  or  $B_2$ ?
- How do you define “better”?

# Support Vector Machines



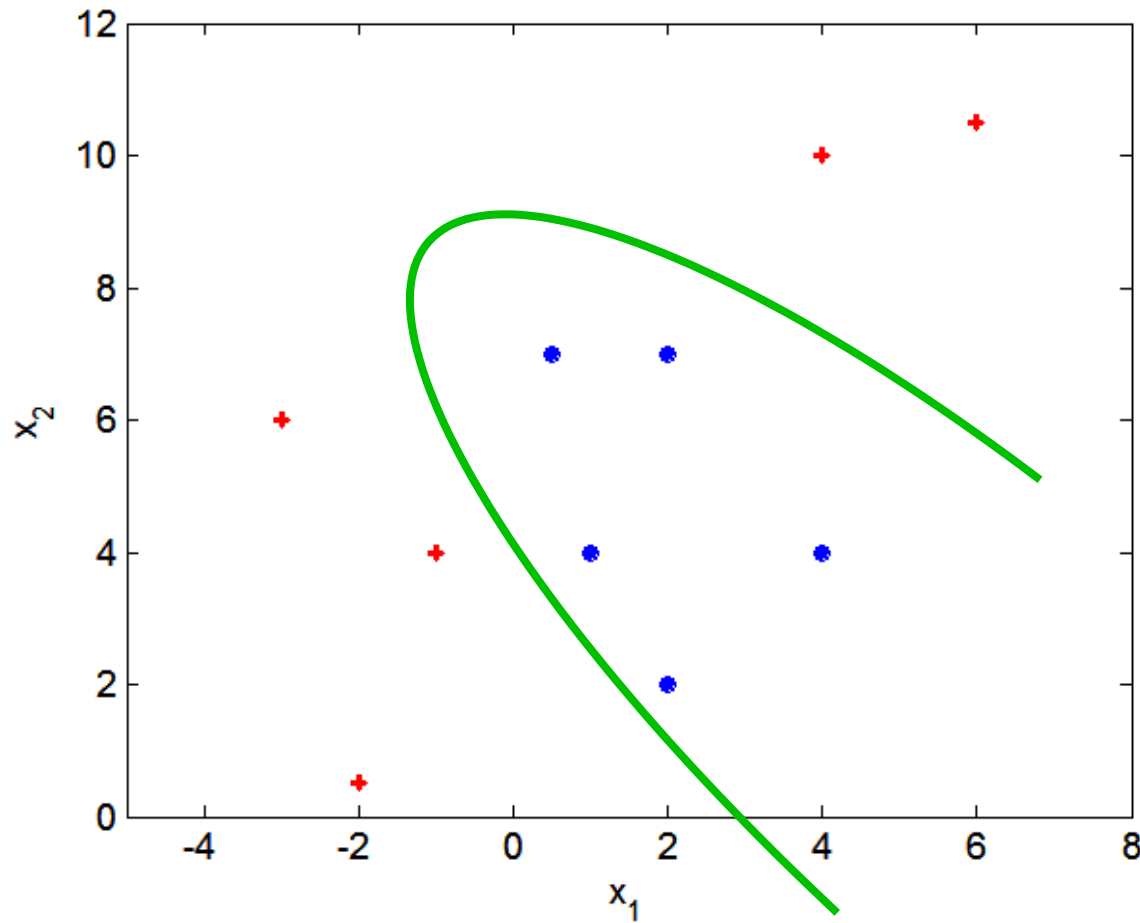
- Find hyperplane **maximizes** the margin  $\Rightarrow$  B1 is better than B2

# Support Vector Machines

- What is computed
  - a separating hyper plane
  - defined by its support vectors (hence the name)
- Challenges
  - Computing an optimal separation is expensive
  - requires good approximations
- Dealing with noisy data
  - introducing “slack variables” in margin computation

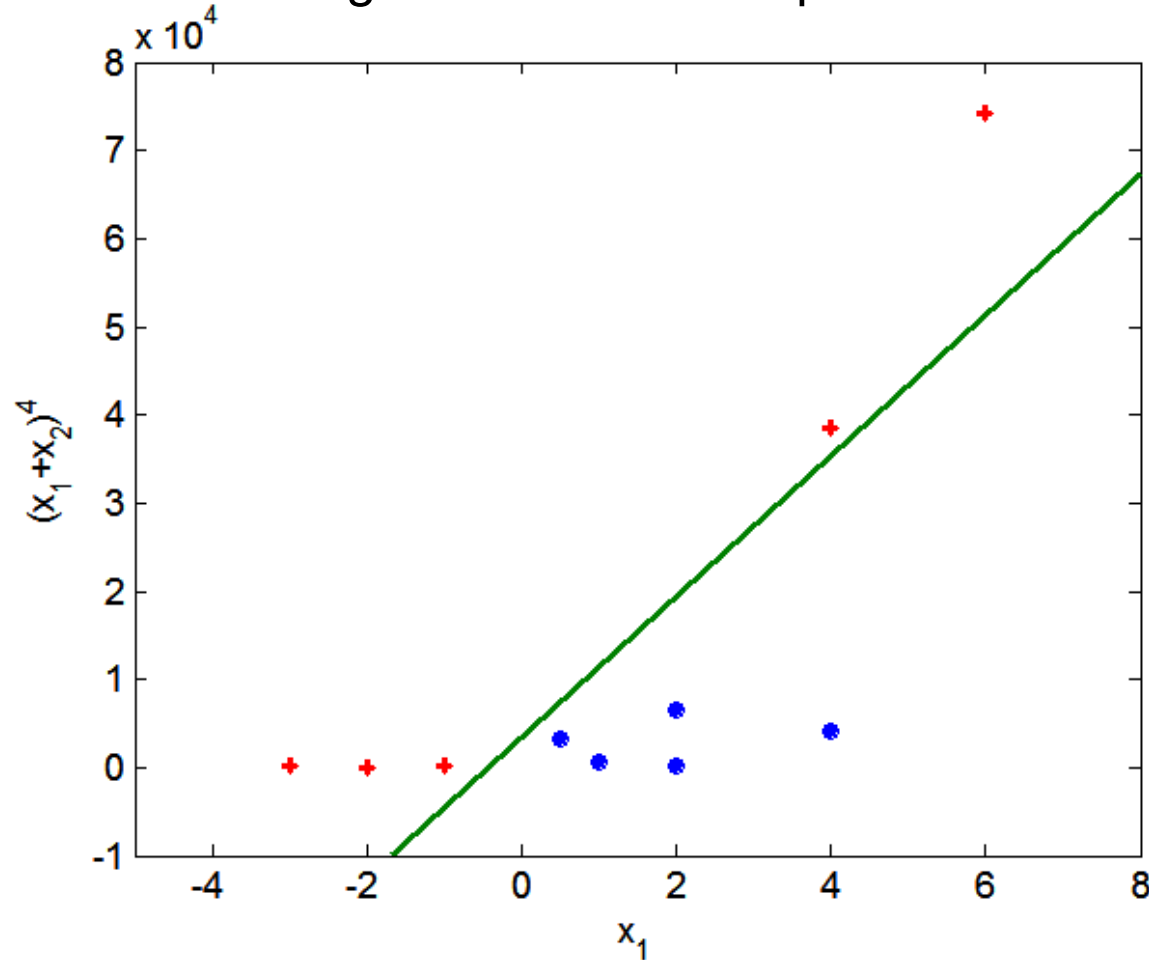
# Nonlinear Support Vector Machines

- What if decision boundary is not linear?



# Nonlinear Support Vector Machines

- Transform data into higher dimensional space



# Nonlinear Support Vector Machines

- Transformation in higher dimensional space
  - Uses so-called Kernel function
  - Different variants: polynomial function, radial basis function, ...
- Finding a hyperplane in higher dimensional space
  - is computationally expensive
  - Kernel trick: expensive parts of the calculation can be performed in lower dimensional space
  - Details: see books...

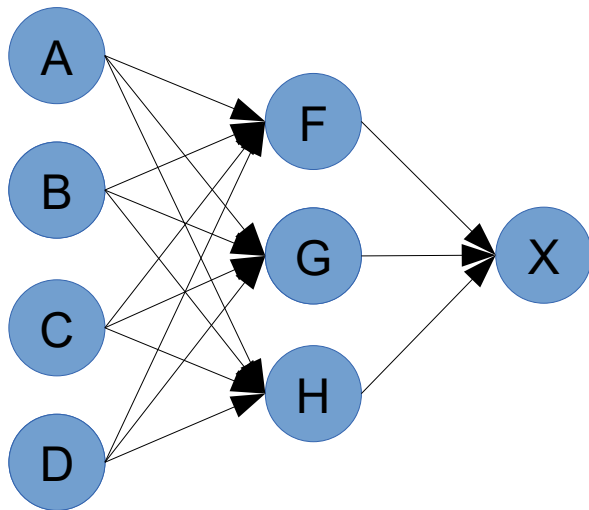
```
clf = SVC(kernel='rbf')
```

# More Exotic Problems

- Consider
  - Four binary features A,B,C,D
  - Goal: Classify true if the number of TRUE values is even (i.e., 0, 2, or 4)
- Very hard for classic machine learning problems
  - Approximate solution can be learned with neural network

# More Exotic Problems

- Consider
  - Four binary features A,B,C,D
  - Goal: Classify true if the number of TRUE values is even (i.e., 0, 2, or 4)



$$F = A + B + C - D < 3$$

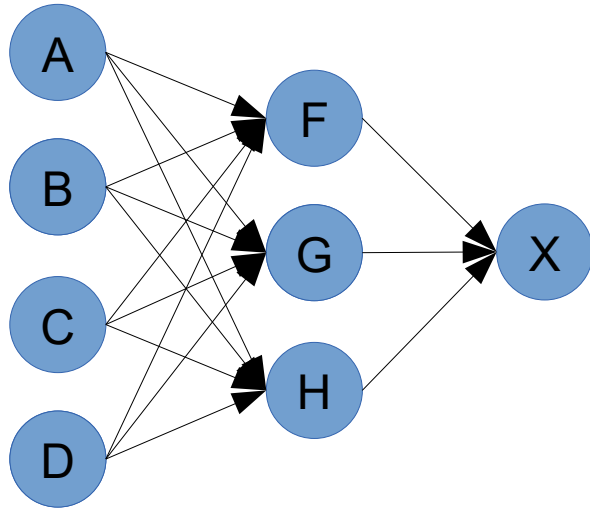
$$G = A + B + C - D < 2$$

$$H = A + B + C - D < 1$$

$$X = F + G - H < 2$$



# More Exotic Problems



$$F = A + B + C - D < 3$$

$$G = A + B + C - D < 2$$

$$H = A + B + C - D < 1$$

$$X = F + G - H < 2$$

A	B	C	D	F	G	H	X	C
0	0	0	0	1	1	1	TRUE	TRUE
1	0	0	0	1	1	0	FALSE	FALSE
0	1	0	0	1	1	0	FALSE	FALSE
0	0	1	0	1	1	0	FALSE	FALSE
0	0	0	1	1	1	1	TRUE	FALSE
1	1	0	0	1	0	0	TRUE	TRUE
0	1	1	0	1	0	0	TRUE	TRUE
0	0	1	1	1	1	1	TRUE	TRUE
1	0	0	1	1	1	1	TRUE	TRUE
1	0	1	0	1	0	0	TRUE	TRUE
0	1	0	1	1	1	1	TRUE	TRUE
1	1	1	0	0	0	0	TRUE	FALSE
1	1	0	1	1	1	0	FALSE	FALSE
1	0	1	1	1	1	0	FALSE	FALSE
0	1	1	1	1	1	0	FALSE	FALSE
1	1	1	1	1	0	0	TRUE	TRUE

# Hyperparameter Selection

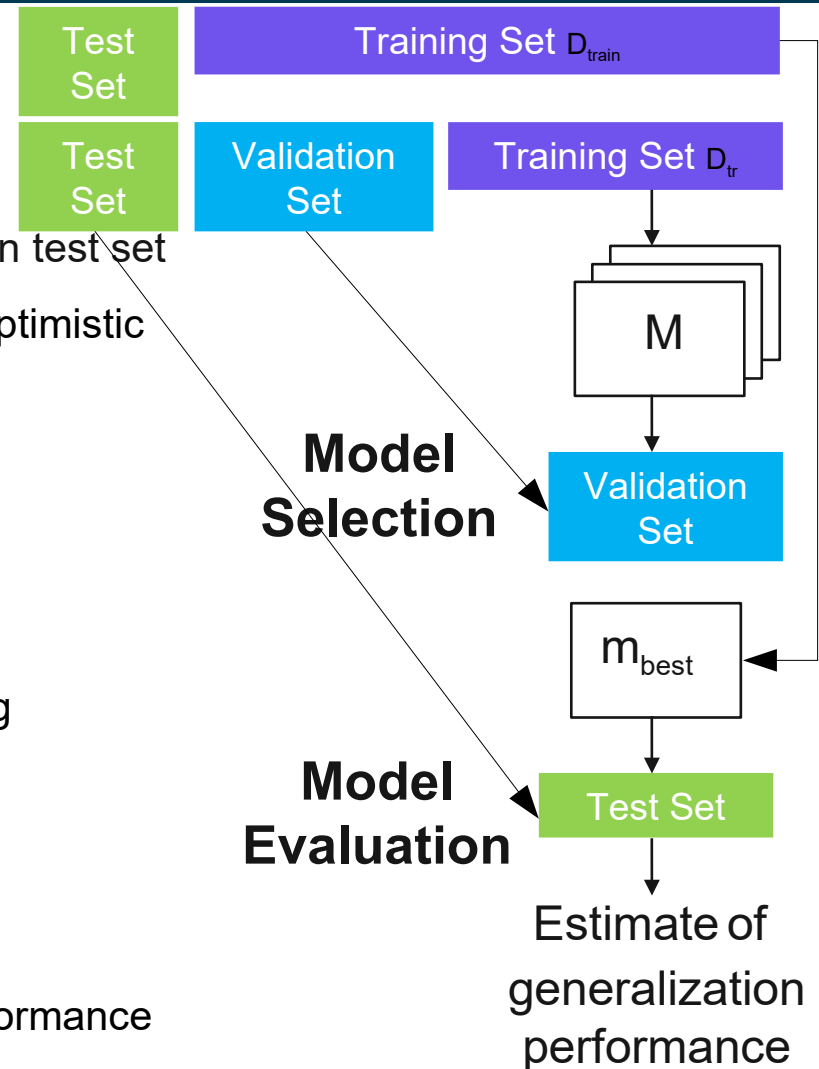
- A **hyperparameter** is a parameter which influences the learning process and whose value is set before the learning begins
  - pruning thresholds for trees and rules
  - gamma and C for SVMs
  - learning rate, hidden layers for ANNs
- By contrast, **parameters** are learned from the training data
  - weights in an ANN, probabilities in Naïve Bayes, splits in a tree
- Many methods work poorly with the default hyperparameters
- How to determine good hyperparameters?
  - manually play around with different hyperparameter settings
  - have your machine automatically test many different settings (hyperparameter optimization)

# Hyperparameter Optimization

- Goal: Find the combination of hyperparameter values that results in learning the model with the lowest generalization error
- How to determine the parameter value combinations to be tested?
  - **Grid Search**: Test all combinations in user-defined ranges
  - **Random Search**: Test combinations of random parameter values
  - **Evolutionary Search**: Keep specific parameter values that worked well
- Often hundreds of combinations are tested
  - reason for cloud computing
- **Model Selection**: From all learned models  $M$ , select the model  $m_{\text{best}}$  that is expected to generalize best to unseen records

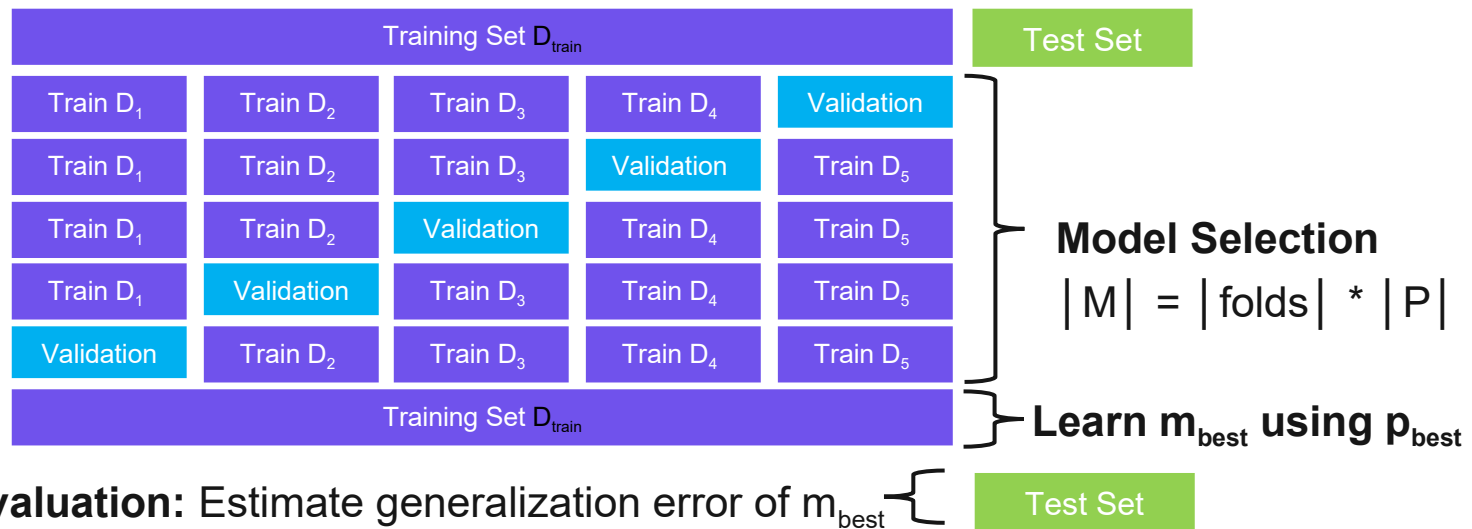
# Model Selection Using a Validation Set

- Keep data used for model selection **strictly separate** from data used for model evaluation, otherwise:
  - selected model  $m_{\text{best}}$  will overfit to patterns in test set
  - estimate of generalization error will be too optimistic
- Method to find the best model:
  - Split training set  $D_{\text{train}}$  into validation set  $D_{\text{val}}$  and training set  $D_{\text{tr}}$
  - Learn models  $M$  on  $D_{\text{tr}}$  using different hyperparameter value combinations  $P$
  - Select best parameter values  $p_{\text{best}}$  by testing each model  $m_i$  on the validation set  $D_{\text{val}}$
  - Learn final model  $m_{\text{best}}$  on complete  $D_{\text{train}}$  using the parameter values  $p_{\text{best}}$
  - Evaluate  $m_{\text{best}}$  on test set in order to get an unbiased estimate of its generalization performance



# Model Selection using Cross-Validation

- But wait, we want to
  1. make sure that all examples are used for validation once
  2. use as much labeled data as possible for training
- Both goals are met by using cross-validation for model selection



- 5 folds, 100 parameter value sets → 501 models learned

# Model Evaluation using Nested Cross-Validation

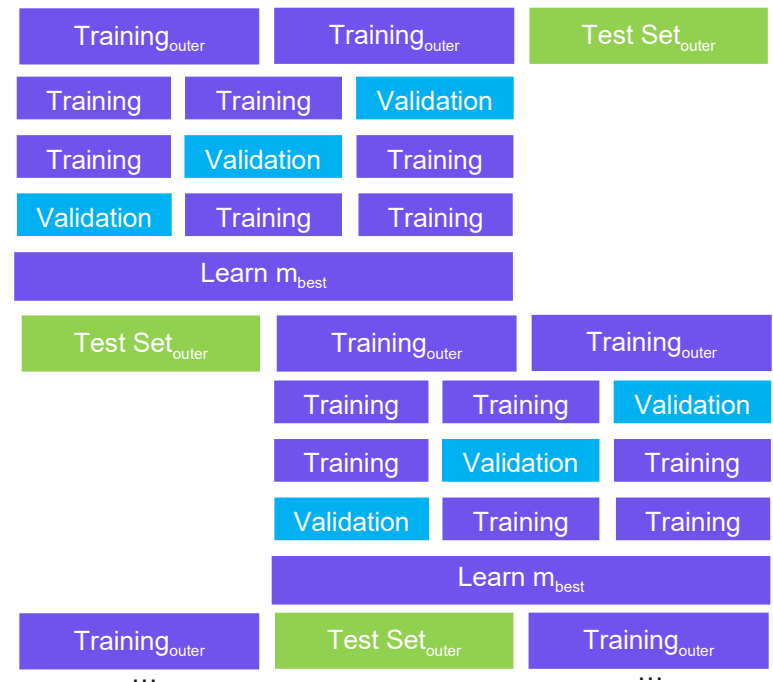
- Nest two cross-validation loops into each other in order to:
  1. find the best hyperparameter setting (model selection)
  2. get a reliable estimate of the generalization error (model evaluation)

## – Outer Cross-Validation

- estimates generalization error of  $m_{\text{best}}$
- training set is passed on to inner cross-validation in each iteration

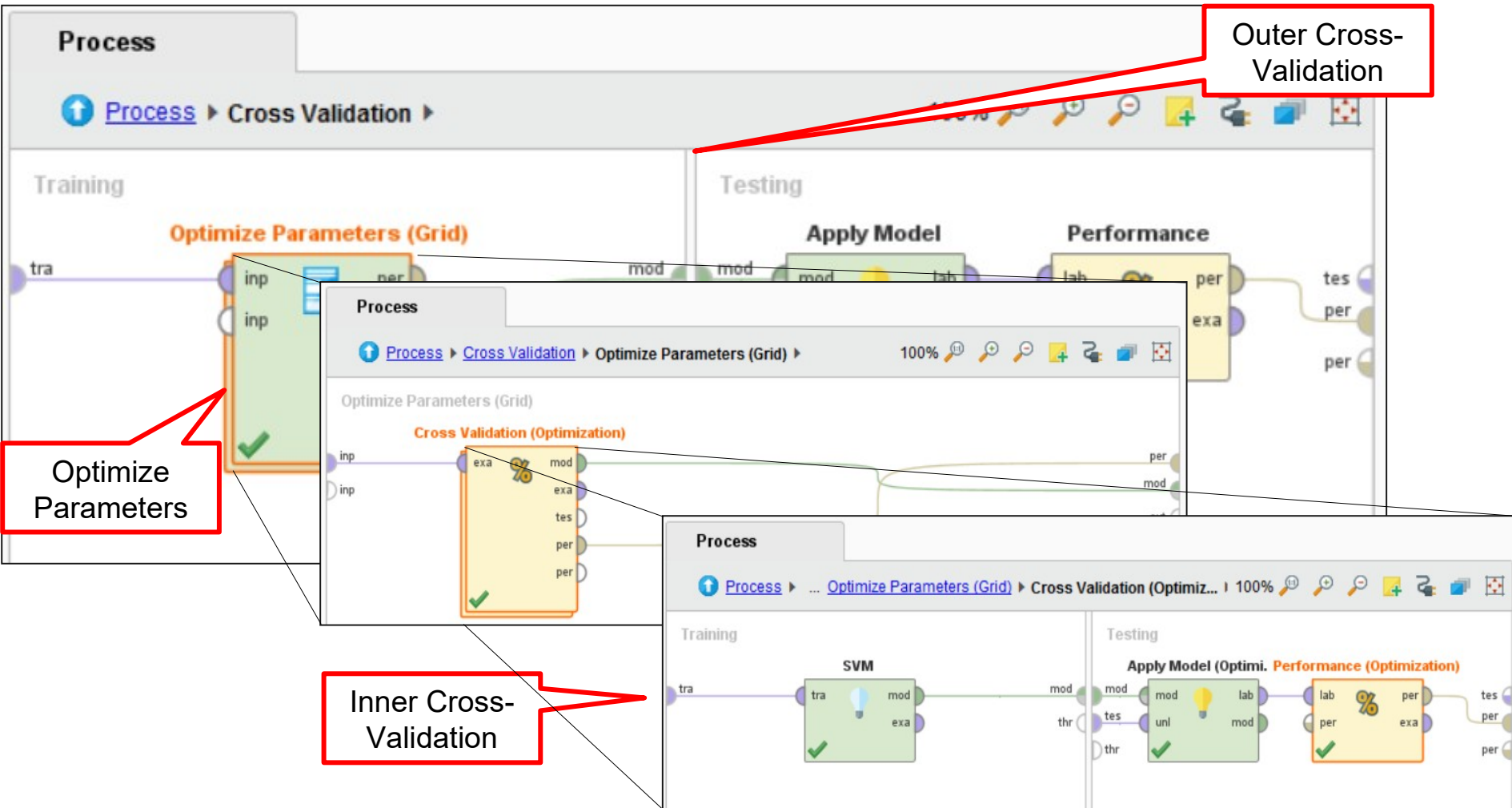
## – Inner Cross-Validation

- searches for best parameter combination
- splits outer training set into inner training and validation set
- learns model  $m_{\text{best}}$  using all outer training data



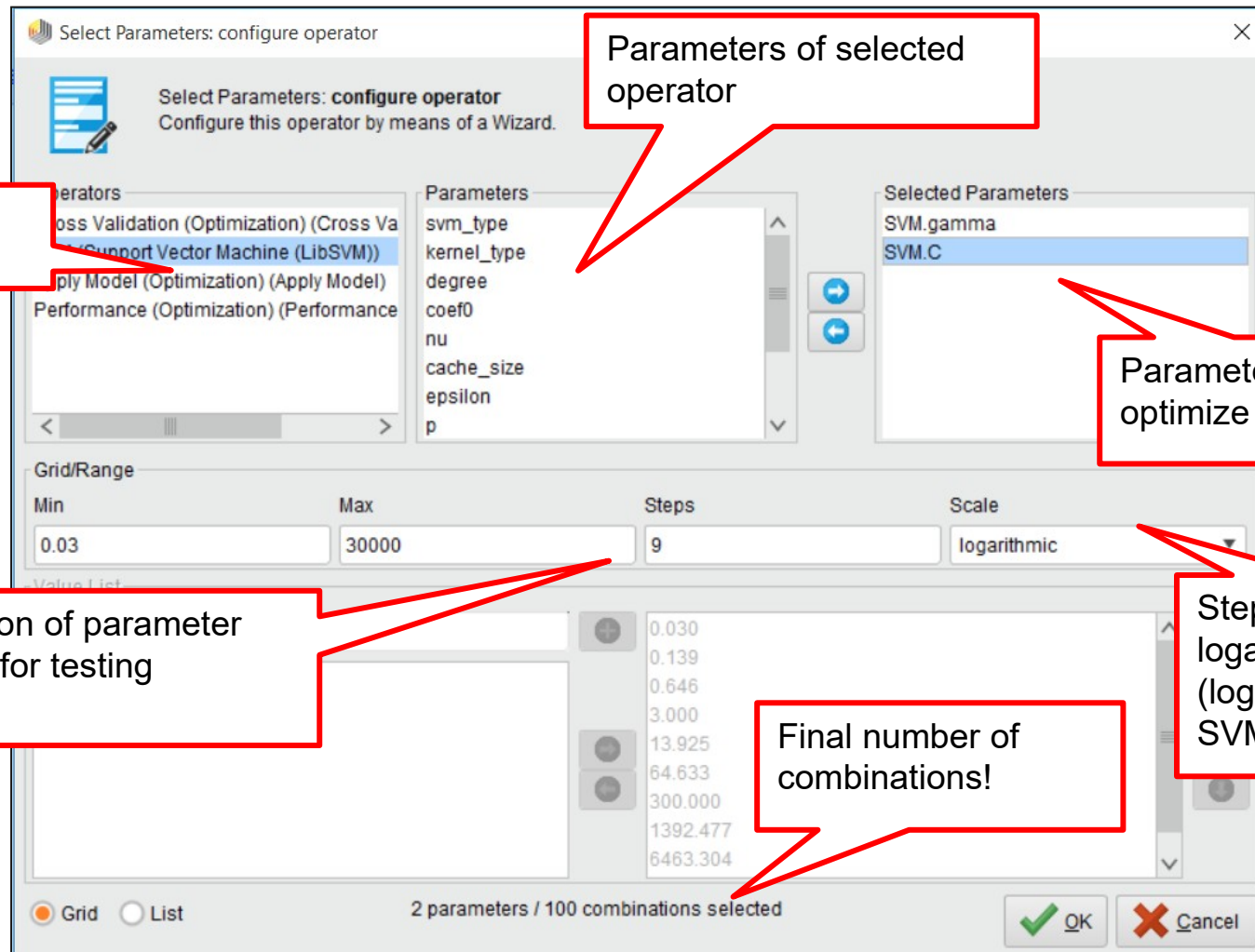
- $5 \text{ folds}_{\text{Outer}} * ((5 \text{ folds}_{\text{Inner}} * 100 \text{ parameter sets}) + 1) \rightarrow 2505 \text{ models learned}$

# Nested Cross-Validation in RapidMiner



<https://rapidminer.com/resource/correct-model-validation/>

# Hyperparameter Optimization in RapidMiner





# Nested Cross-Validation in Python

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

# Specify hyperparameter combinations for search
parameter_grid = {"C": [1, 10, 100, 1000], "gamma": [.001, .01, .1, 1]}

# Create SVM
estimator_svm = SVC(kernel='rbf')

# Create the grid search for model selection
estimator_gs = GridSearchCV(estimator_svm, parameter_grid, scoring='accuracy', cv=5)

# Run nested cross-validation for model evaluation
accuracy_cv = cross_val_score(estimator_gs, dataset, labels, cv=5, scoring='accuracy')
```

# Summary

- Classification approaches
  - There are quite a few: Nearest Neighbors, Naive Bayes, Decision Trees, Rules, SVMs, Neural Networks
- Distinctions
  - Lazy vs. eager
  - Performance (accuracy, training time, testing time, model size)
  - Decision Boundaries (theory and practice)
- Issues
  - Overfitting
  - Parameter tuning

# Questions?

