U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 5d: Supervised Learning - Classification

(adapted from Tan et al, 2020)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence

Agenda

- Classification
- Overfitting
- Model Selection
- Model Evaluation
- Decision Tree based Methods
- Nearest-Neighbor
- Naïve Bayes
- Support Vector Machines
- Neural Networks, Deep Neural Networks
- Conclusions

Classification: Definition

Given a collection of records (training set)

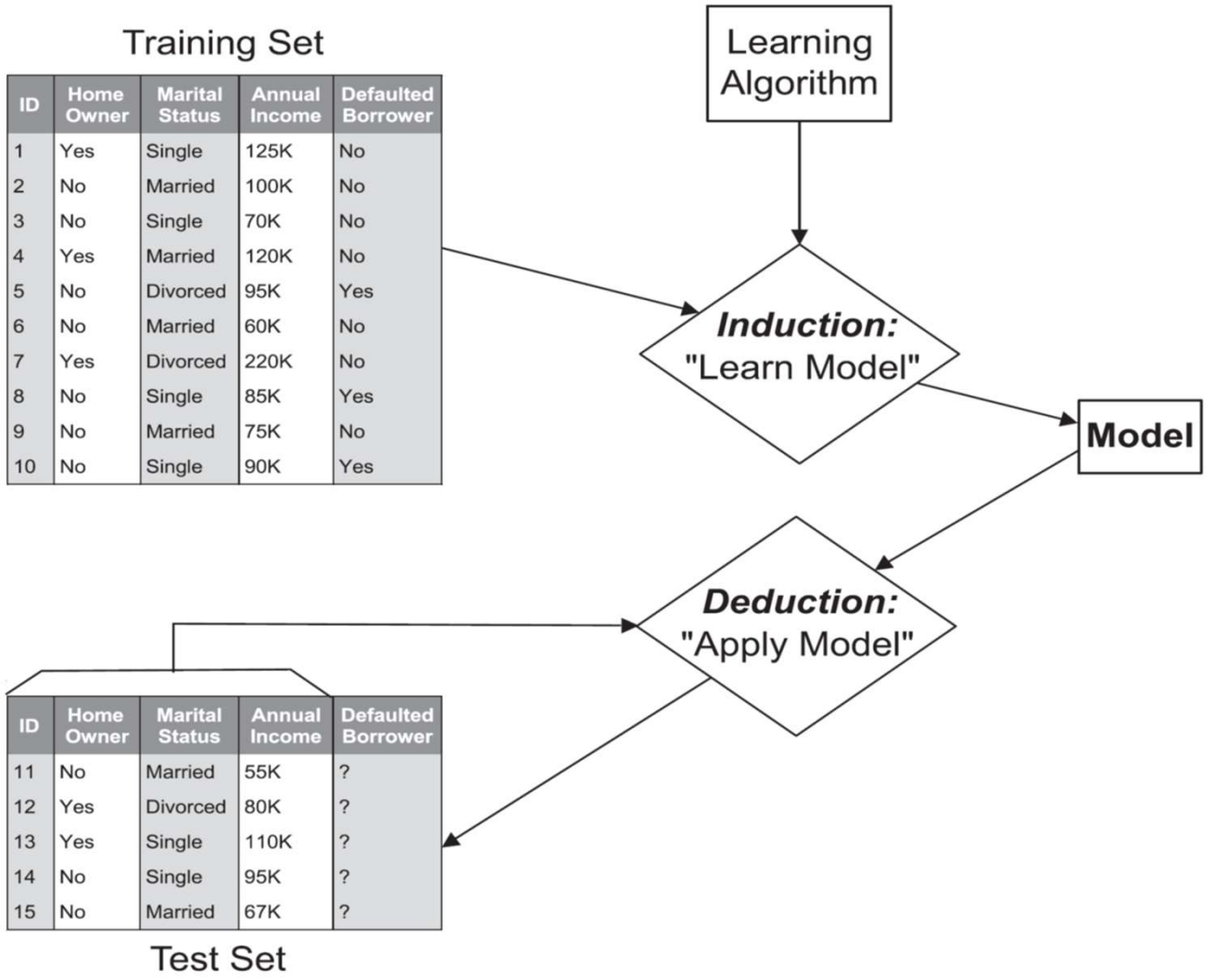
- Each record is by characterized by a tuple ($\boldsymbol{x}$,y), where $\boldsymbol{x}$ is the attribute set and y is the class label
  - $\boldsymbol{x}$: attribute, predictor, independent variable, input
  - $\boldsymbol{y}$: class, response, dependent variable, output

Task: Learn a model that maps each attribute set $\boldsymbol{x}$ into one of the predefined class labels y

Examples of Classification Tasks

Task	Attribute set, $\boldsymbol{x}$	Class label, y
Categorizing email messages	Features extracted from email message header and content	spam or non-spam
Identifying tumor cells	Features extracted from x-rays or MRI scans	malignant or benign cells
Cataloging galaxies	Features extracted from telescope images	Elliptical, spiral, or irregular-shaped galaxies

General Approach for Building Classification Model

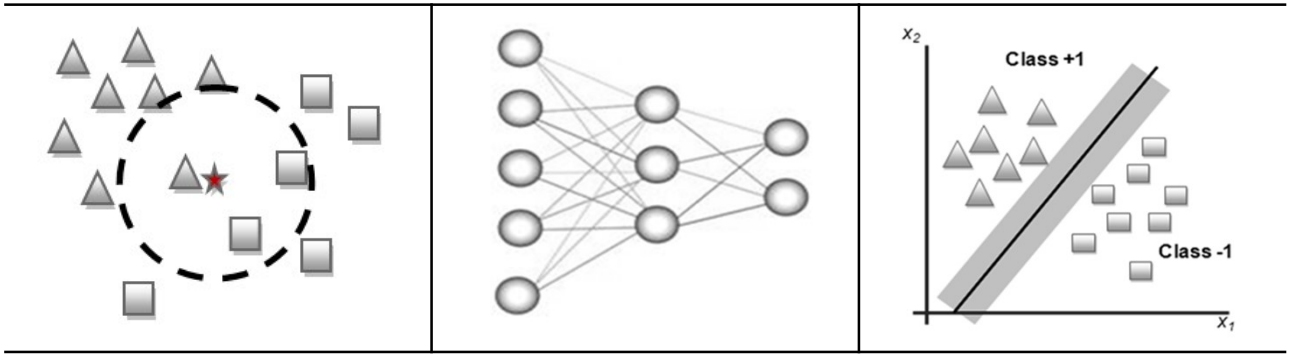


ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

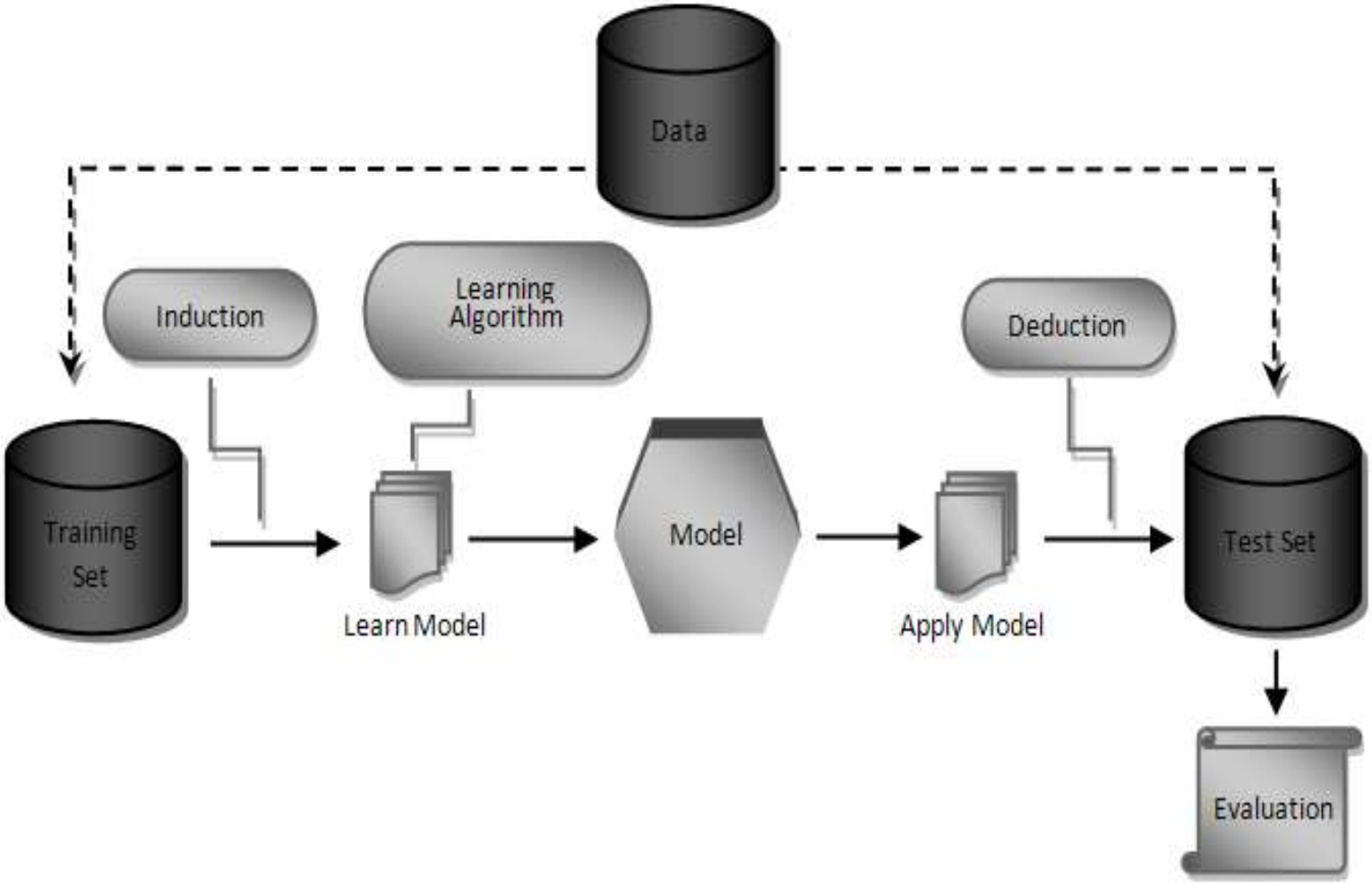
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
11	No	Married	55K	?
12	Yes	Divorced	80K	?
13	Yes	Single	110K	?
14	No	Single	95K	?
15	No	Married	67K	?

Classification Techniques

- Base Classifiers
  - Decision Tree based Methods
  - Rule-based Methods
  - Nearest-Neighbor
  - Naïve Bayes and Bayesian Belief Networks
  - Support Vector Machines
  - Neural Networks, Deep Neural Networks
- Ensemble Classifiers
  - Boosting, Bagging, Random Forests



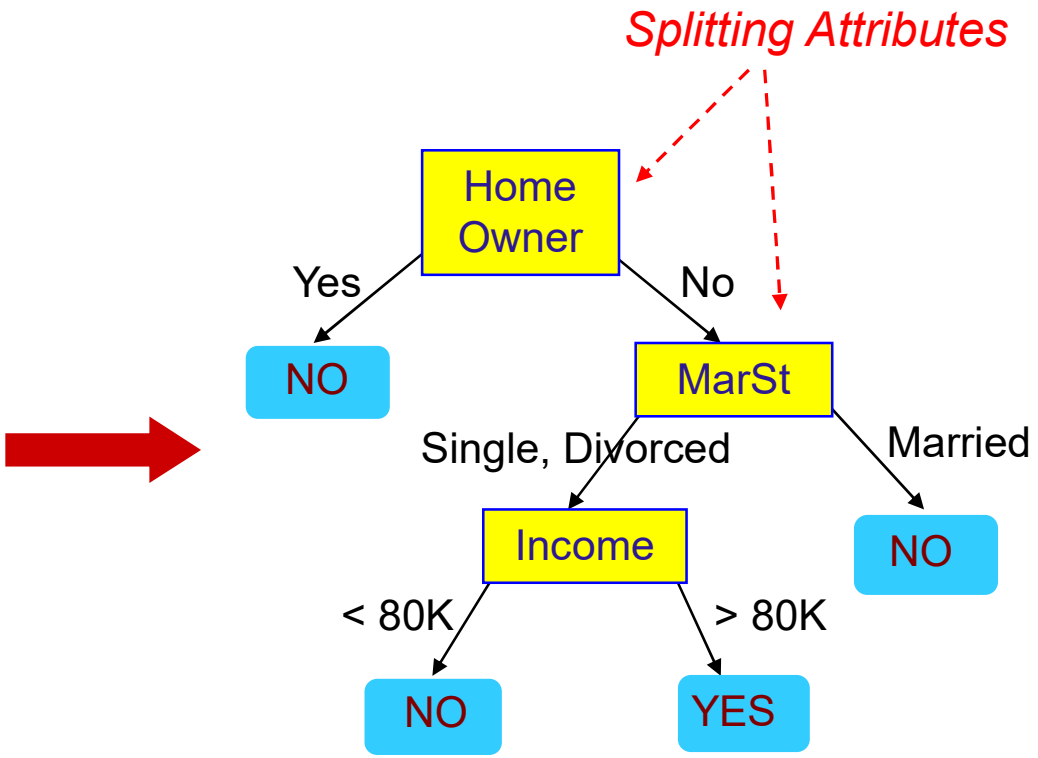
Classification Techniques



Example of a Decision Tree

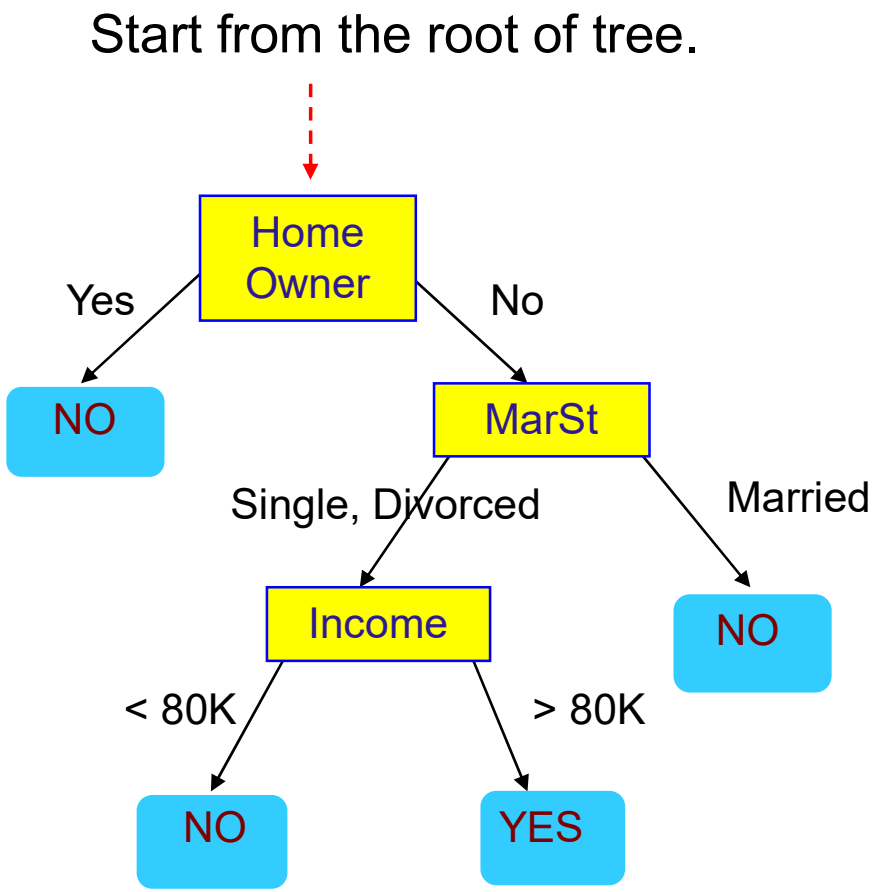
ID	Home Owner (categorical)	Marital Status (categorical)	Annual Income (continuous)	Defaulted Borrower (class)
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

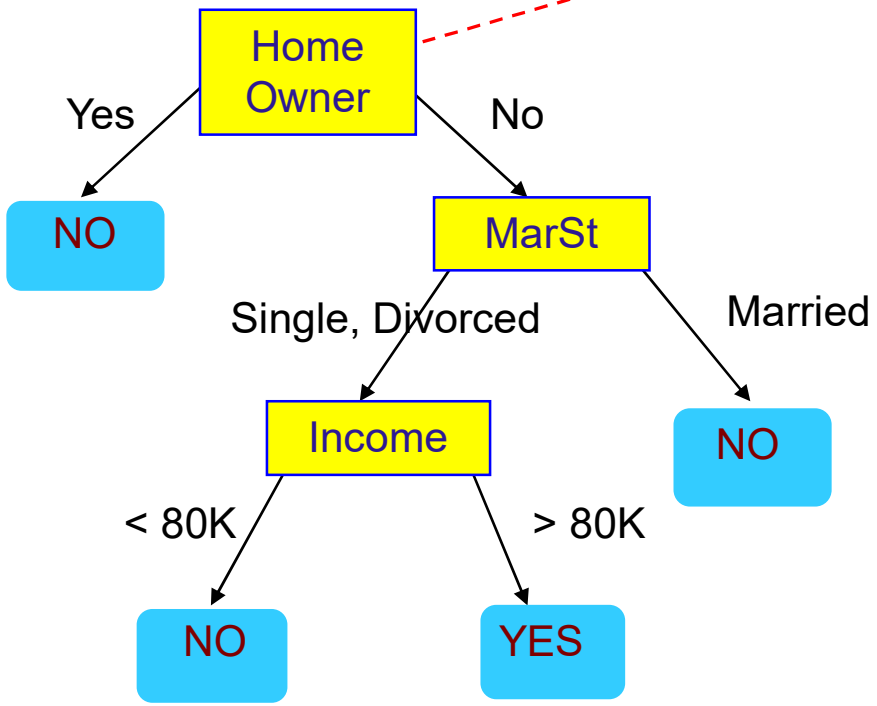
Apply Model to Test Data



Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Apply Model to Test Data



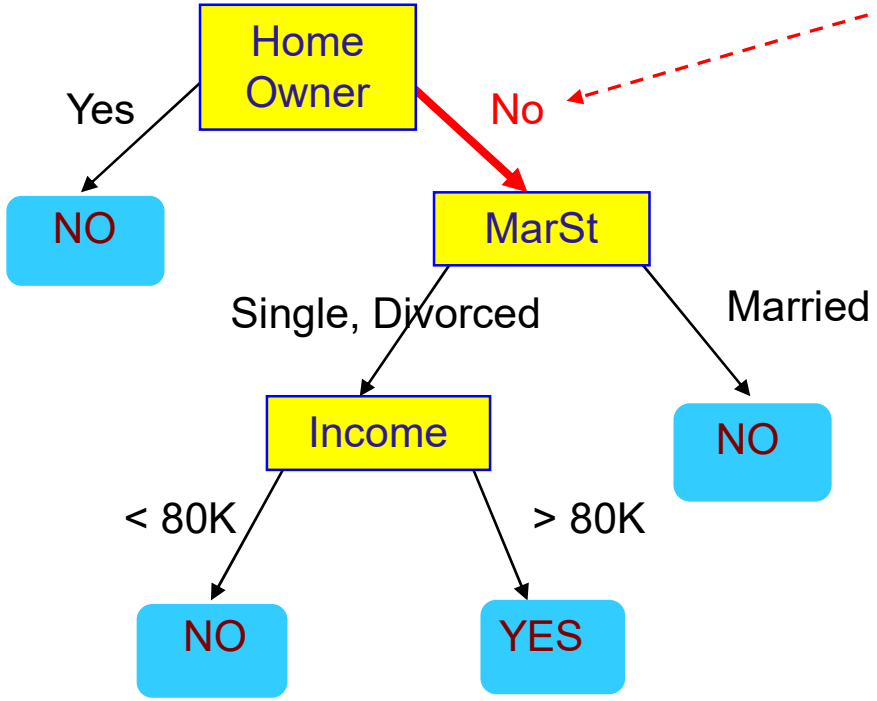
Test Data

Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?

Apply Model to Test Data

Test Data

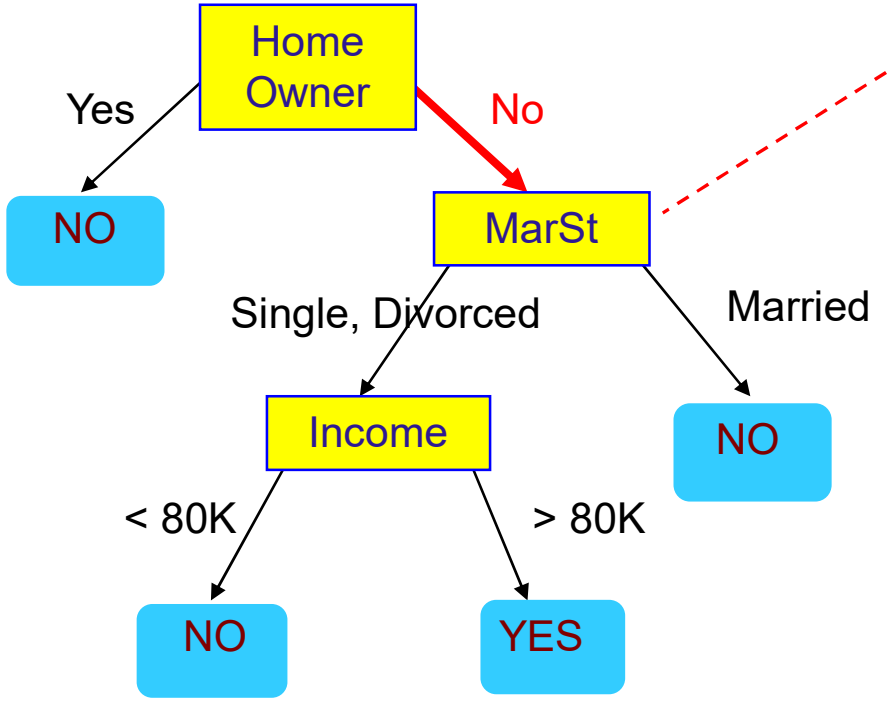
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

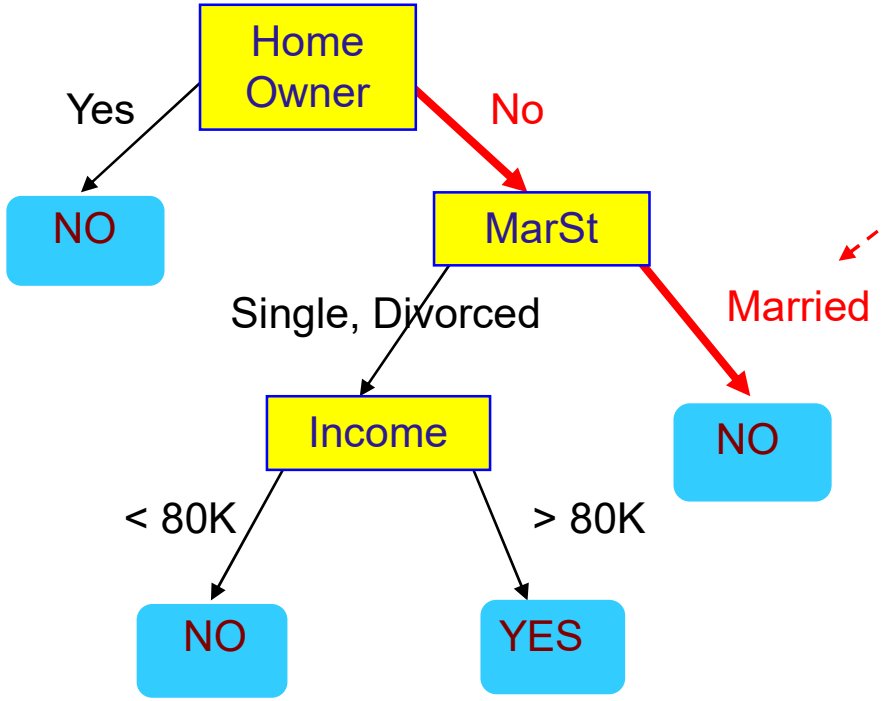
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

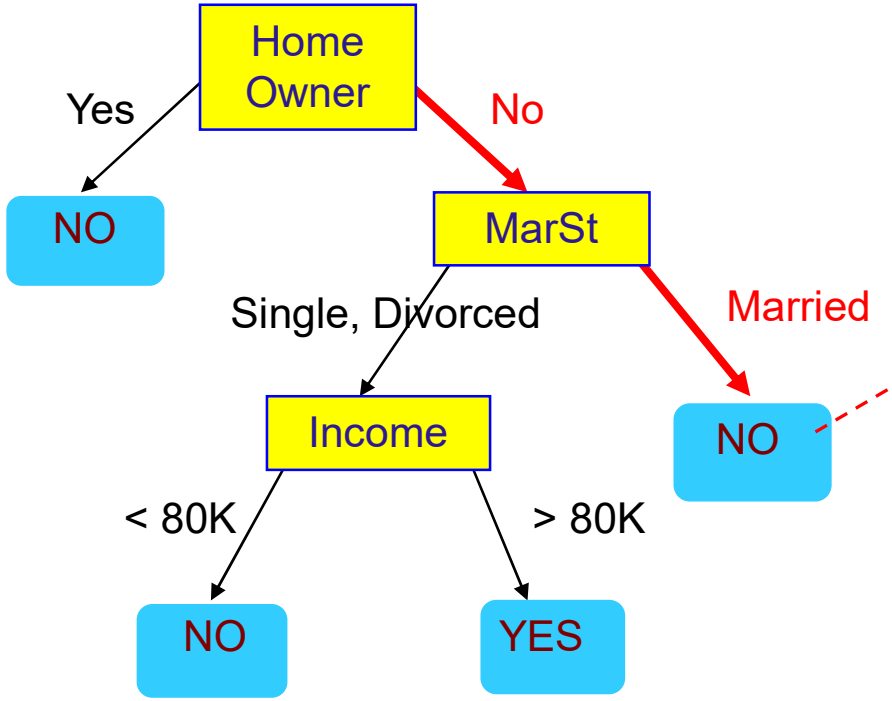
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Apply Model to Test Data

Test Data

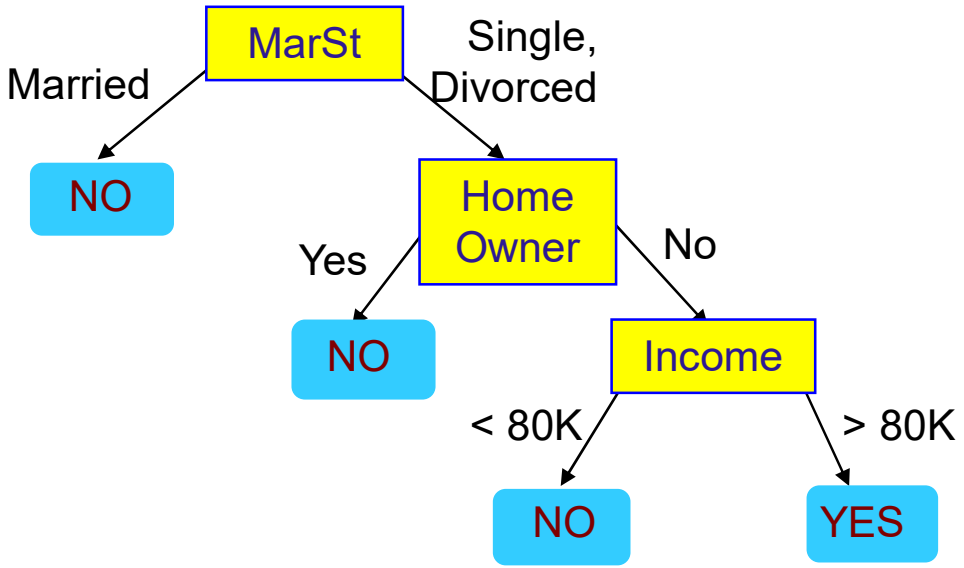
Home Owner	Marital Status	Annual Income	Defaulted Borrower
No	Married	80K	?



Assign Defaulted to "No"

Another Example of Decision Tree

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
		categorical	categorical	continuous	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	No	Single	90K	Yes



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

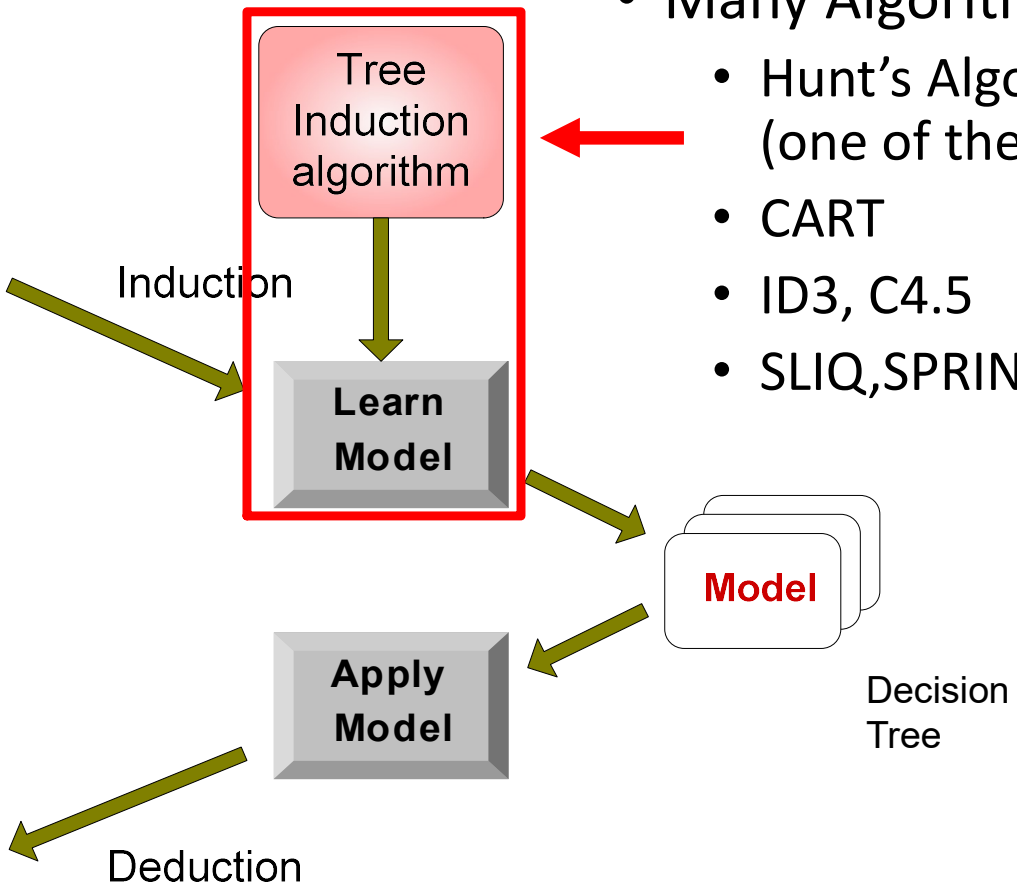
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ,SPRINT

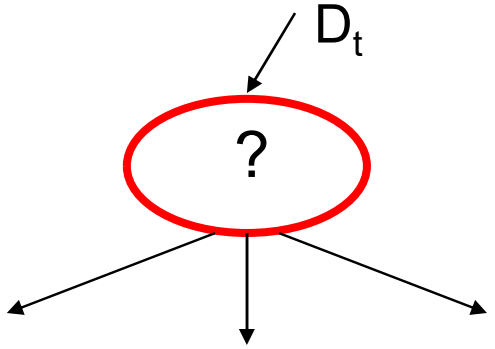
General Structure of Hunt's Algorithm

Let D_t be the set of training records that reach a node t

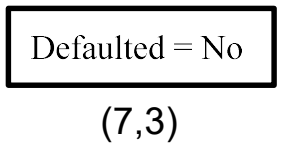
General Procedure:

- If D_t contains records that belong 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.

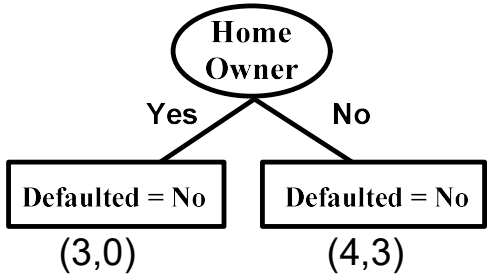
ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



Hunt's Algorithm

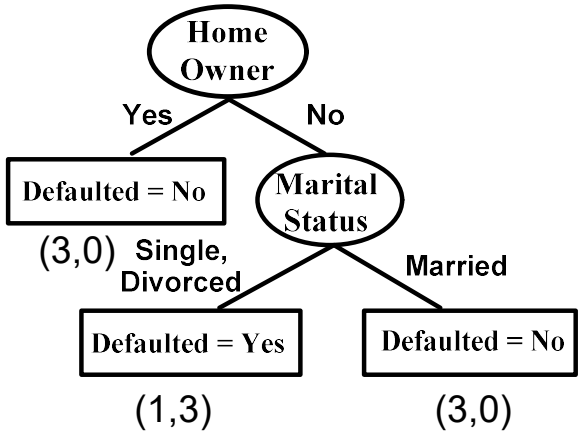


(a)

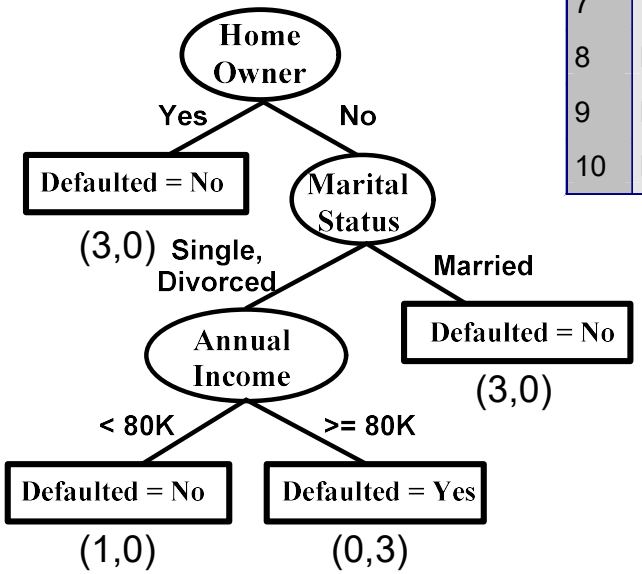


(b)

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
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



(c)



(d)

Design Issues of Decision Tree Induction

How should training records be split?

- Method for expressing test condition
- Depending on attribute types
  - Binary
  - Nominal
  - Ordinal
  - Continuous
- Measure for evaluating the goodness of a test condition

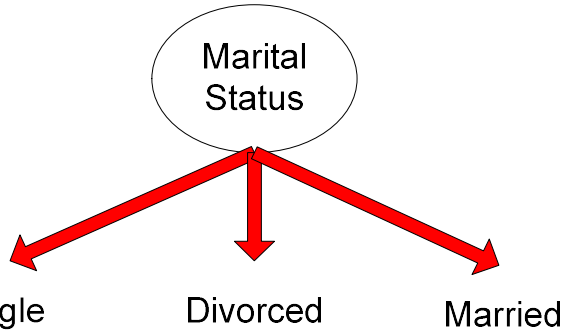
How should the splitting procedure stop?

- Stop splitting if all the records belong to the same class or have identical attribute values
- Early termination

Test Condition for Nominal Attributes

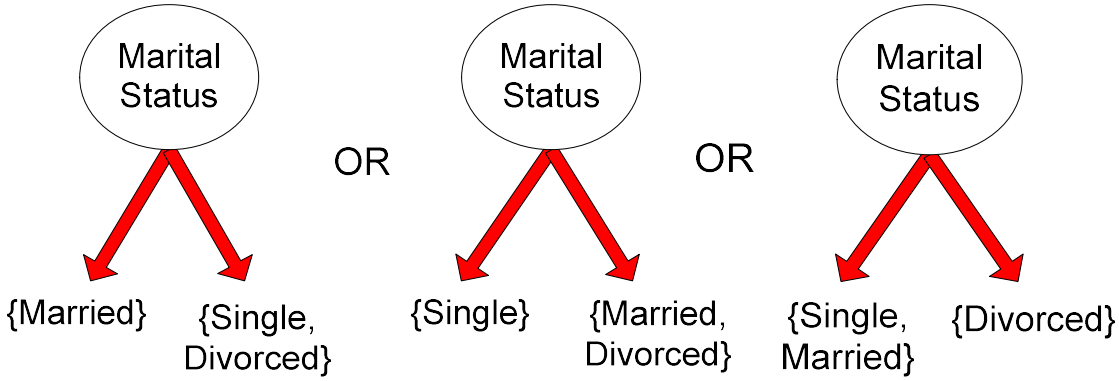
Multi-way split:

- Use as many partitions as distinct values



Binary split:

- Divides values into two subsets



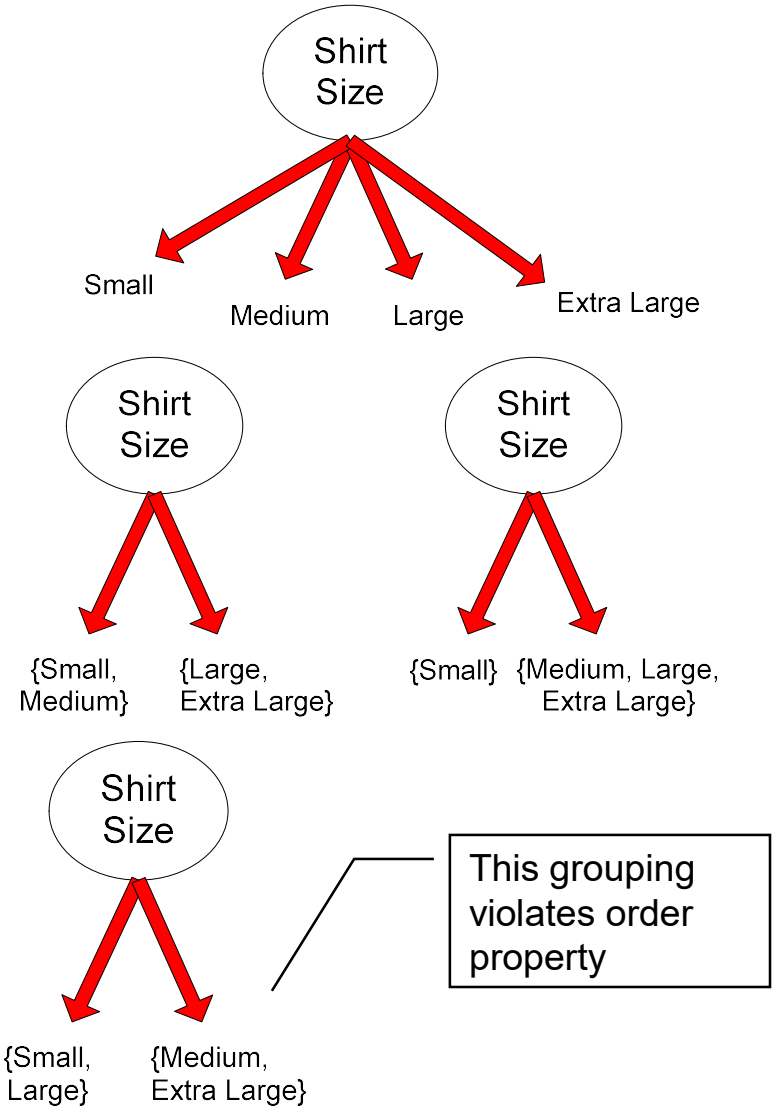
Test Condition for Ordinal Attributes

Multi-way split:

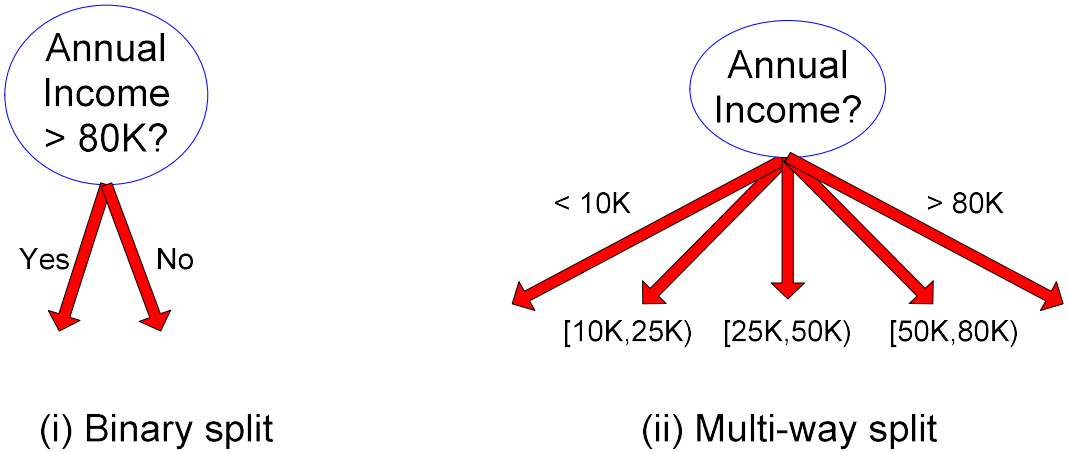
- Use as many partitions as distinct values

Binary split:

- Divides values into two subsets
- Preserve order property among attribute values



Test Condition for Continuous Attributes



(i) Binary split

(ii) Multi-way split

- Splitting based on continuous attributes:
  - Discretization to form an ordinal categorical attribute

    Ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
    - Static – discretize once at the beginning
    - Dynamic – repeat at each node
  - Binary Decision: $(A < v)$ or $(A \geq v)$
    - consider all possible splits and finds the best cut
    - can be more compute intensive

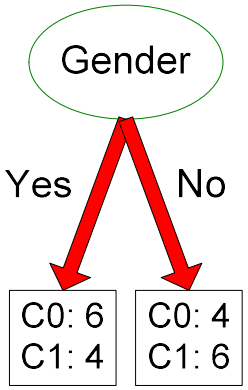
Determining the Best Split

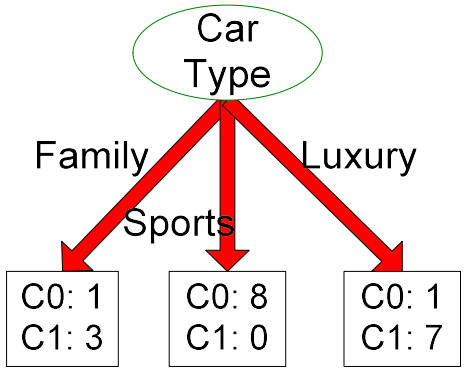
Before Splitting:

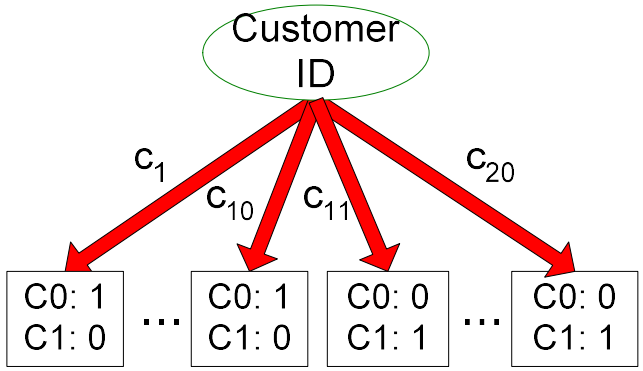
- 10 records of class 0
- 10 records of class 1

Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Which test condition is the best?







Determining the Best Split - Impurity

Greedy approach:

- Nodes with purer class distribution are preferred

Need a measure of node impurity:

C0: 5 C1: 5	C0: 9 C1: 1
High degree of impurity	Low degree of impurity

Measures of Node Impurity:

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node t, and $\boldsymbol{c}$ is the total number of classes

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

$$Classification\ error = 1 - \max[p_i(t)]$$

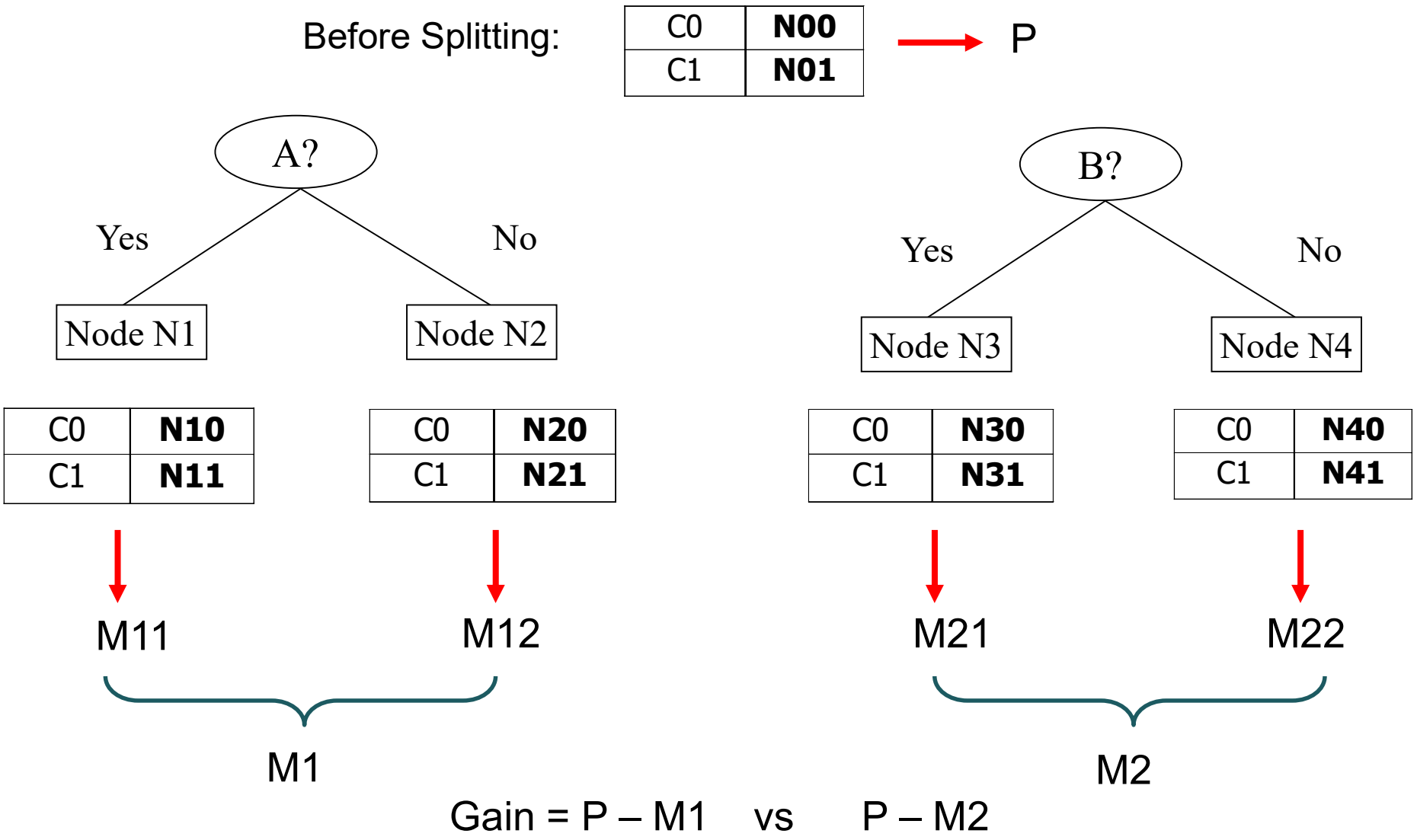
Finding the Best Split

1. Compute impurity measure (P) before splitting
2. Compute impurity measure (M) after splitting
   - Compute impurity measure of each child node
   - M is the weighted impurity of child nodes
3. Choose the attribute test condition that produces the highest gain:

   Gain = P - M

   or equivalently, lowest impurity measure after splitting (M)

Finding the Best Split



Measure of Impurity: GINI

- Gini Index for a given node $\boldsymbol{t}$

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

- Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node $\boldsymbol{t}$, and $\boldsymbol{c}$ is the total number of classes
  - Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
  - Minimum of 0 when all records belong to one class, implying the most beneficial situation for classification
  - Gini index is used in decision tree algorithms such as CART, SLIQ, SPRINT

Measure of Impurity: Entropy

- Entropy at a given node $\boldsymbol{t}$

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

- Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node $\boldsymbol{t}$, and $\boldsymbol{c}$ is the total number of classes
  - Maximum of $\log_2 c$ when records are equally distributed among all classes, implying the least beneficial situation for classification
  - Minimum of 0 when all records belong to one class, implying most beneficial situation for classification
- Entropy computations similar to the GINI index computations

Computing Entropy of a Single Node

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Computing Information Gain After Splitting

Information Gain:

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

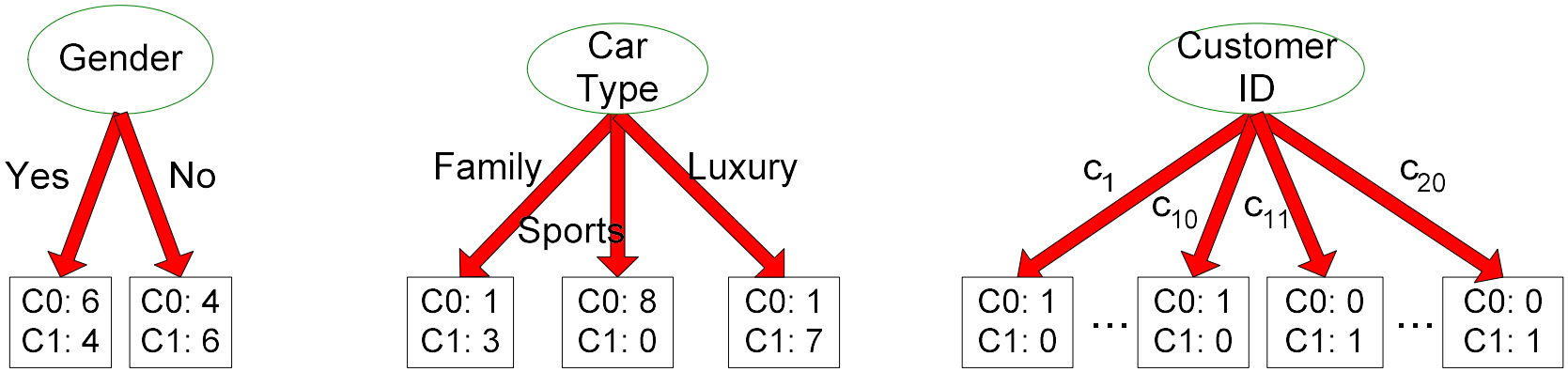
Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

- Choose the split that achieves most reduction (maximizes GAIN)
- Used in the ID3 and C4.5 decision tree algorithms
- Information gain is the mutual information between the class variable and the splitting variable

Problem with large number of partitions

Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure



– Customer ID has highest information gain because entropy for all the children is zero

Gain Ratio

$$Gain\ Ratio = \frac{Gain_{split}}{Split\ Info} \qquad Split\ Info = -\sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

- Adjusts Information Gain by the entropy of the partitioning ($Split\ Info$).
  - Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5 algorithm
- Designed to overcome the disadvantage of Information Gain

Gain Ratio

$$Gain\ Ratio = \frac{Gain_{split}}{Split\ Info} \qquad Split\ Info = \sum_{i=1}^{k} \frac{n_i}{n} log_2 \frac{n_i}{n}$$

Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

SplitINFO = 1.52

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

SplitINFO = 0.72

	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

SplitINFO = 0.97

Measure of Impurity: Classification Error

Classification error at a node t

$$Error(t) = 1 - \max_i [p_i(t)]$$

- Maximum of $1 - 1/c$ when records are equally distributed among all classes, implying the least interesting situation
- Minimum of 0 when all records belong to one class, implying the most interesting situation

Computing Error of a Single Node

$$Error(t) = 1 - \max_i [p_i(t)]$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

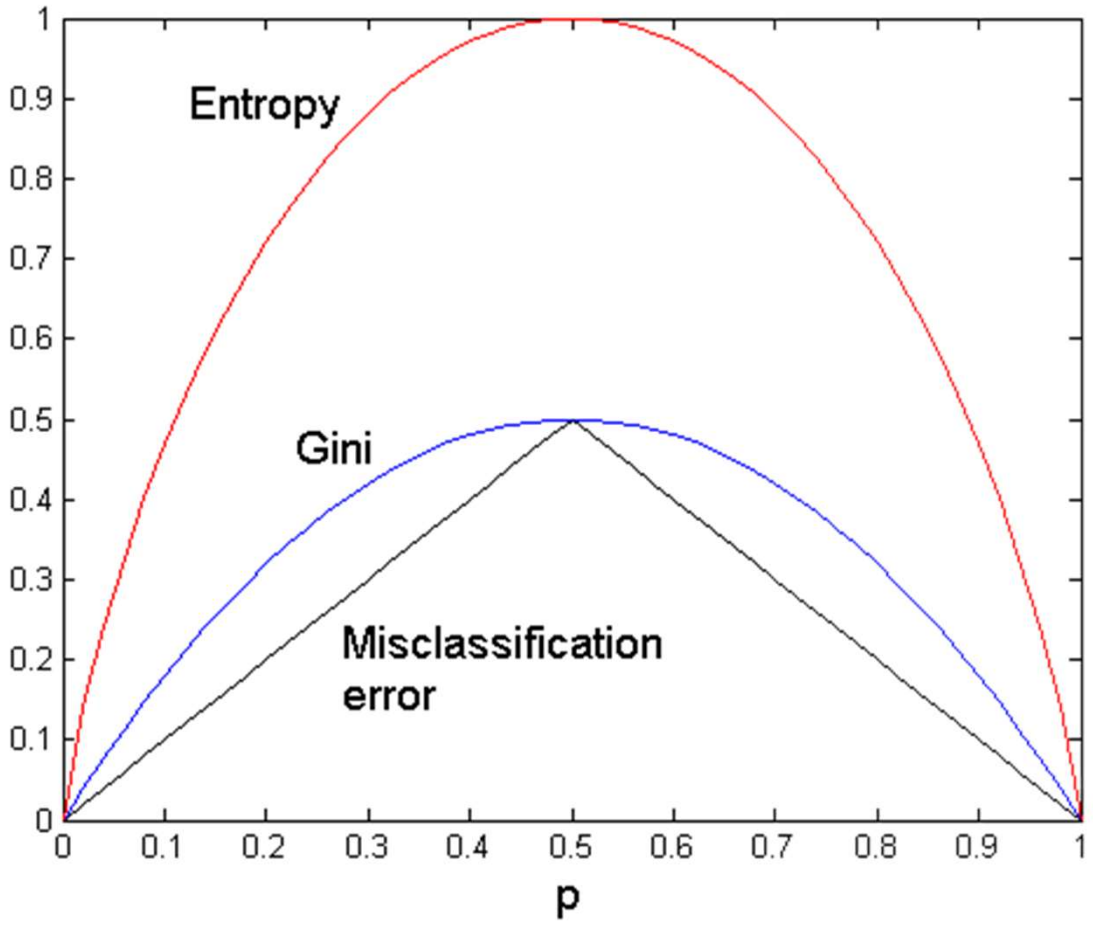
C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Comparison among Impurity Measures

For a 2-class problem:



Decision Tree Based Classification

Advantages:

- Relatively inexpensive to construct
- Extremely fast at classifying unknown records
- Easy to interpret for small-sized trees
- Robust to noise (especially when methods to avoid overfitting are employed)
- Can easily handle redundant attributes
- Can easily handle irrelevant attributes (unless the attributes are interacting)

Disadvantages:

- Due to the greedy nature of splitting criterion, interacting attributes (that can distinguish between classes together but not individually) may be passed over in favor of other attributed that are less discriminating
- Each decision boundary involves only a single attribute

Classification Errors

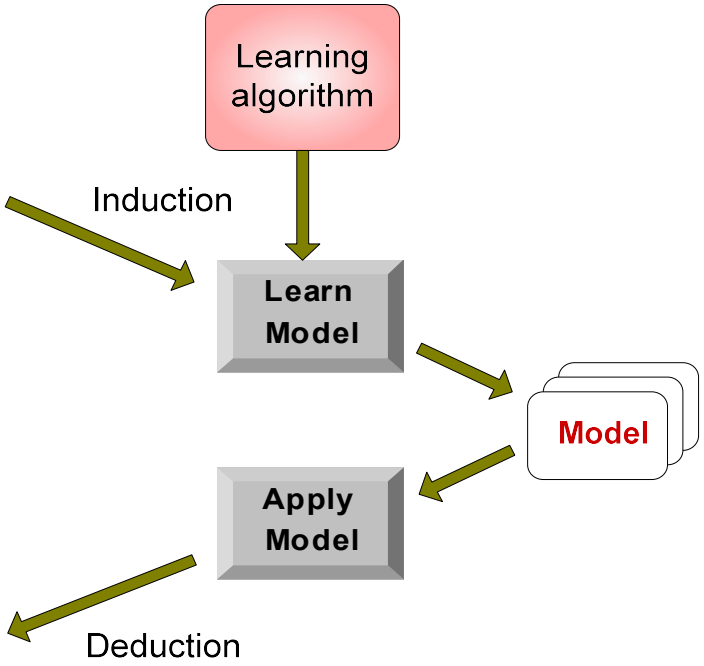
- **Training errors**: Errors committed on the training set
- **Test errors**: Errors committed on the test set
- **Generalization errors**: Expected error of a model over random selection of records from same distribution

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

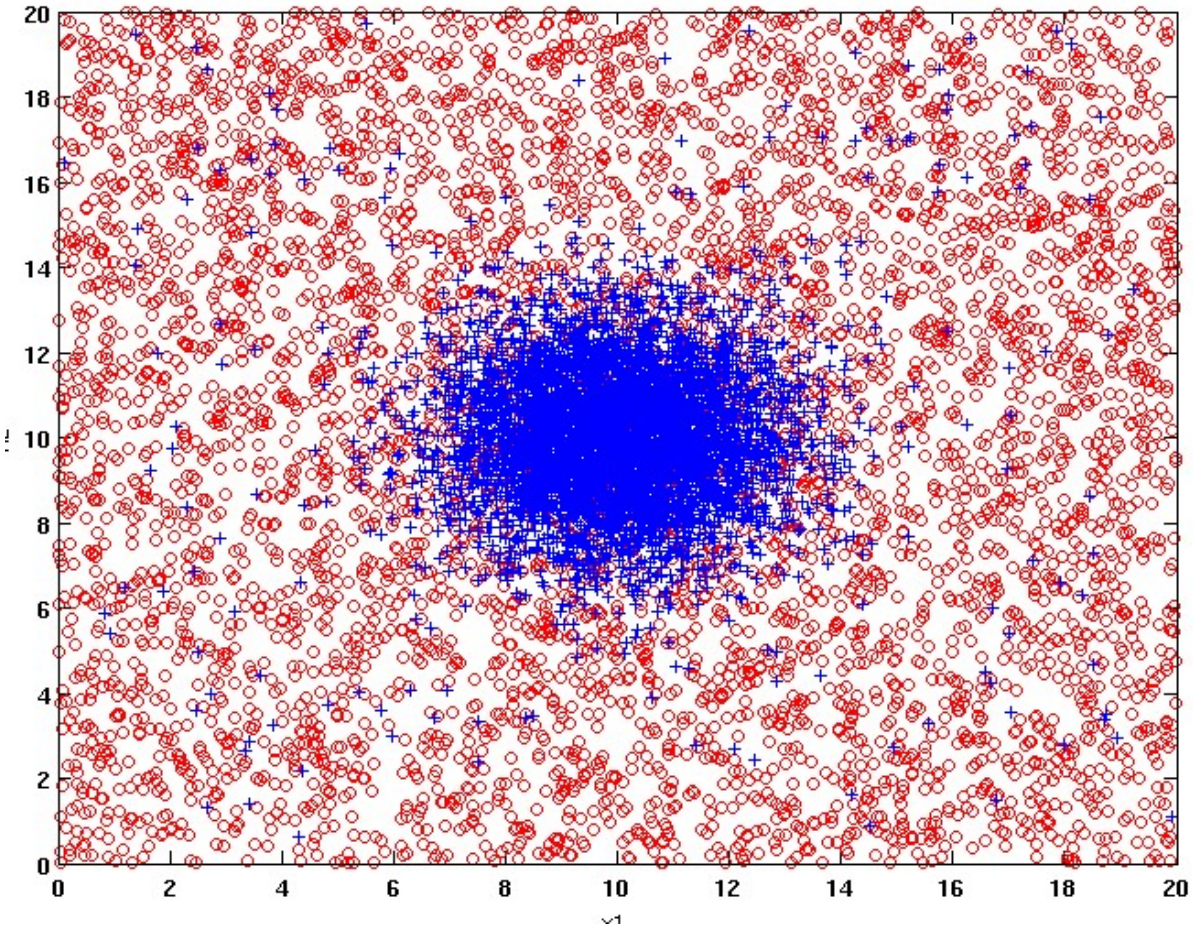
Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Overfitting - Example Data Set



Two class problem:

+ : 5400 instances

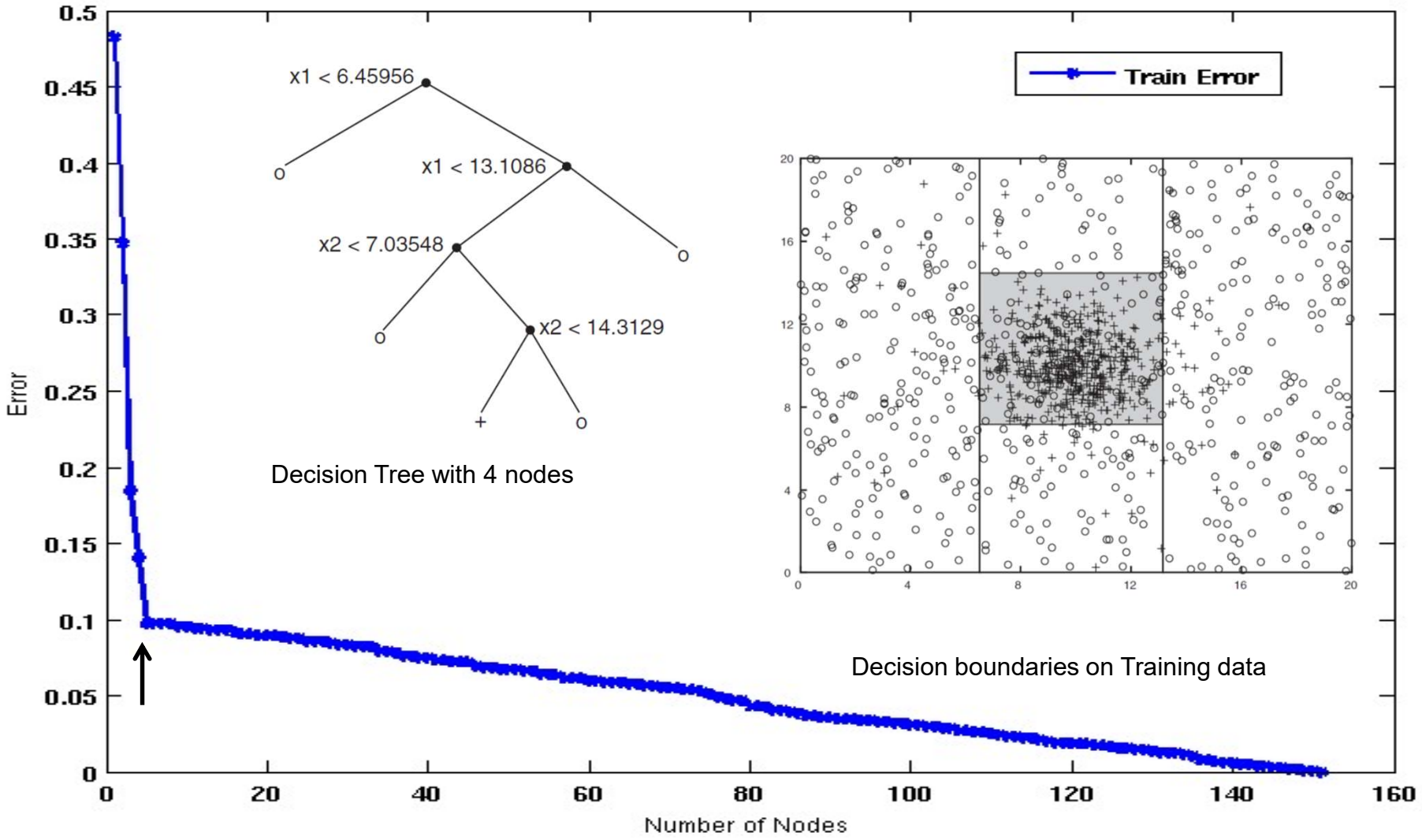
- **5000 instances generated from a Gaussian centered at (10,10)**
- **400 noisy instances added**

o : 5400 instances

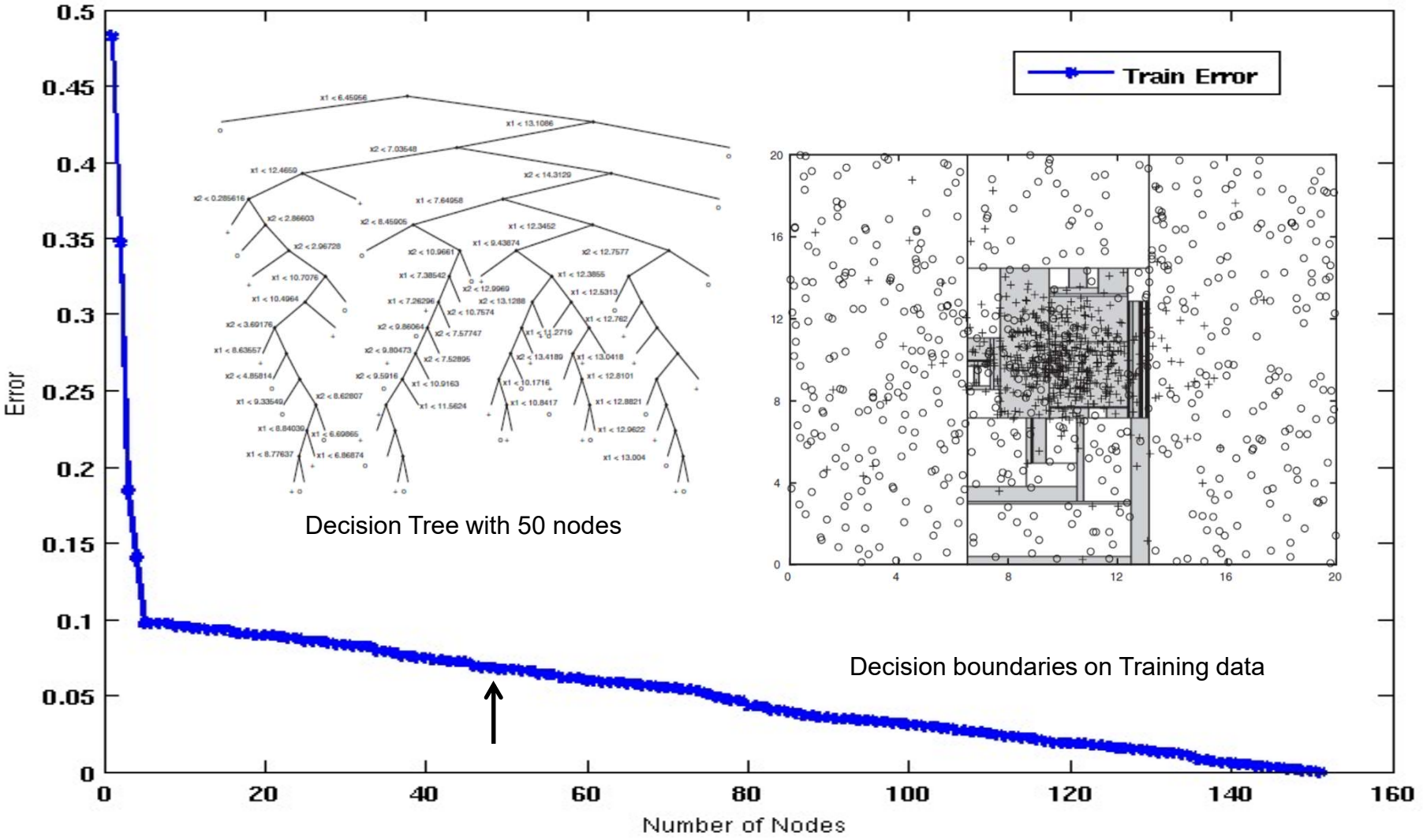
- **Generated from a uniform distribution**

10 % of the data used for training and 90% of the data used for testing

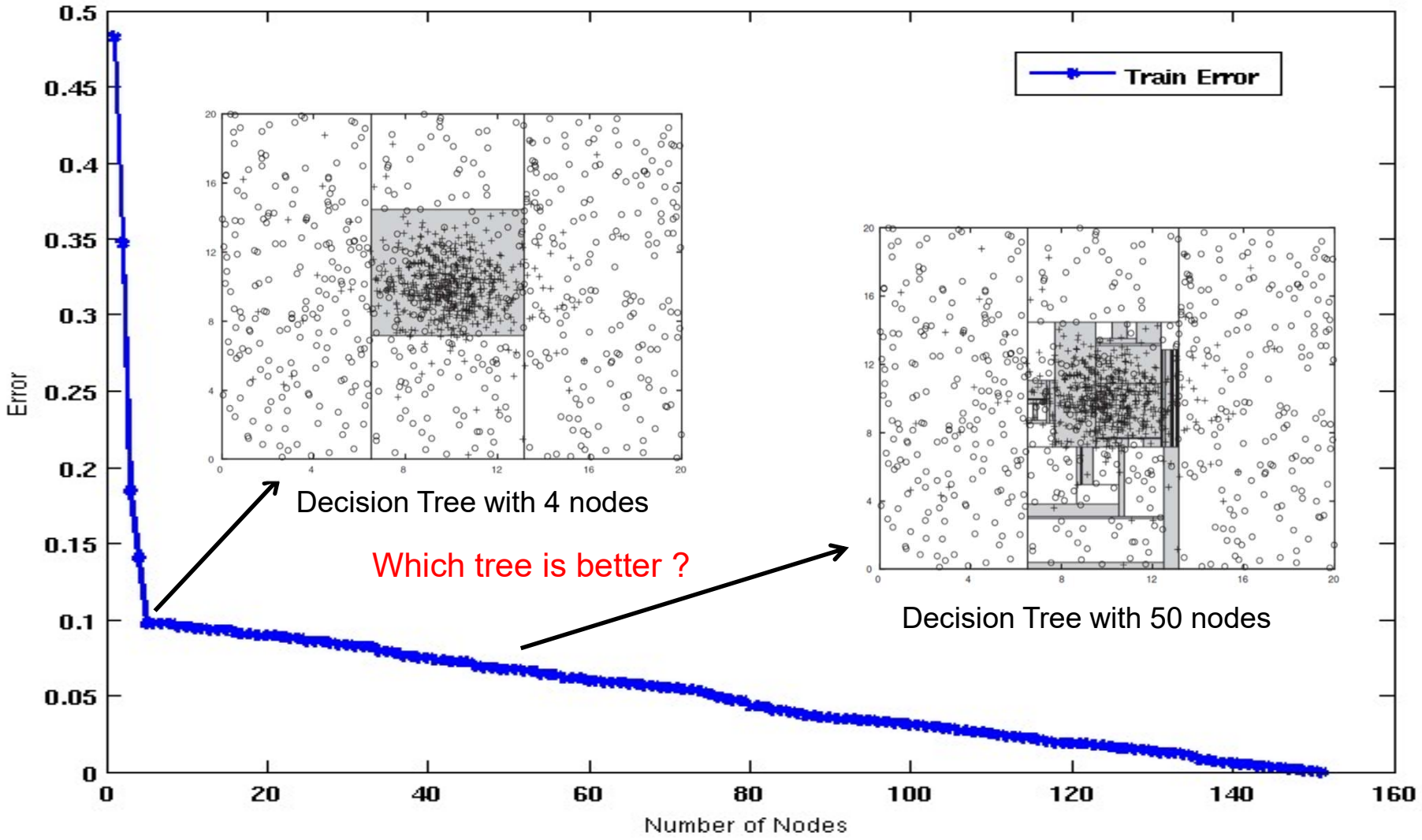
Decision Tree with 4 nodes



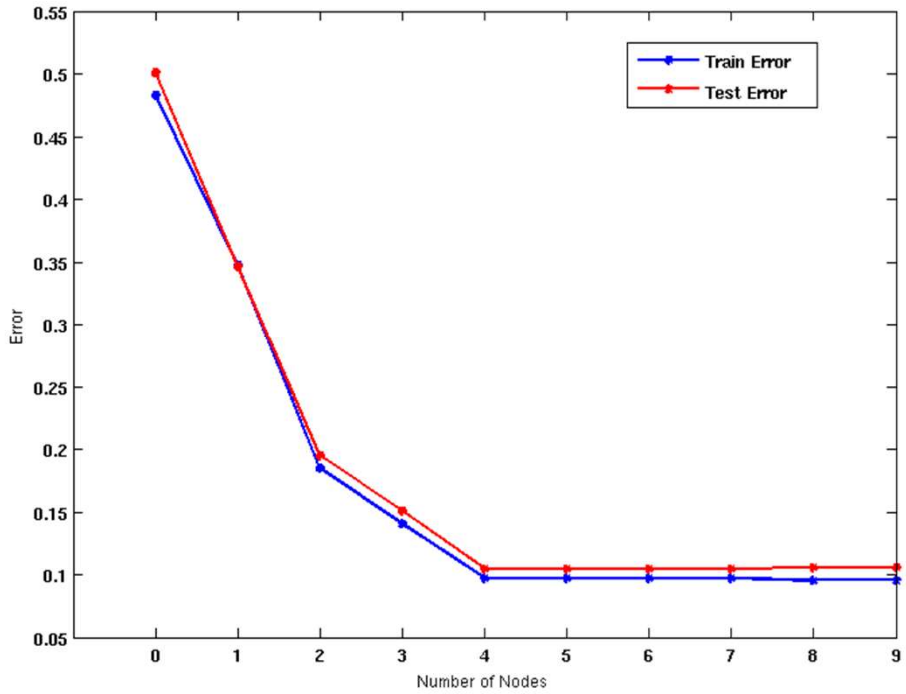
Decision Tree with 50 nodes

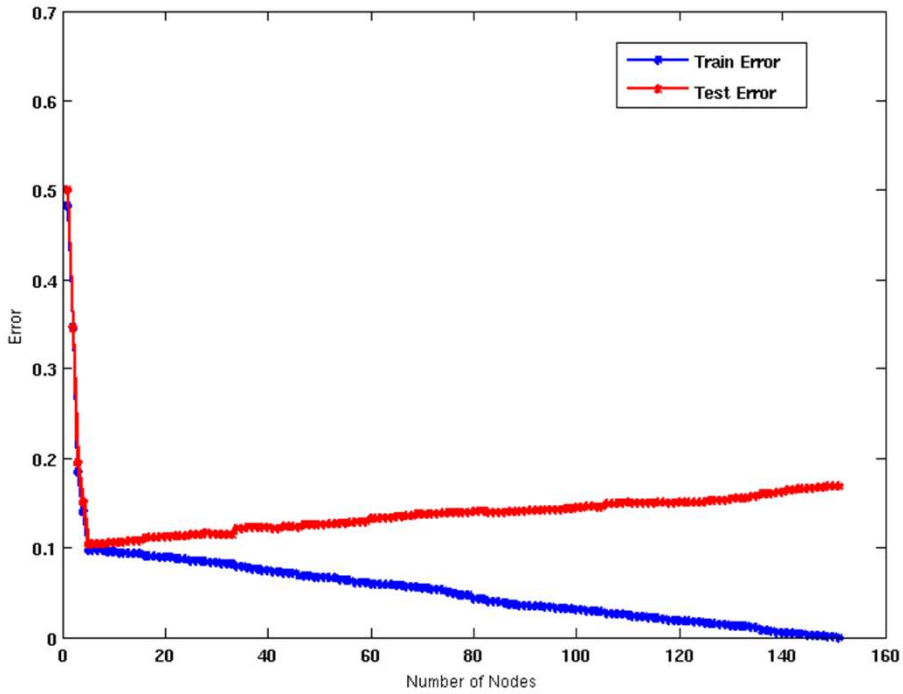


Which tree is better?



Model Underfitting and Overfitting

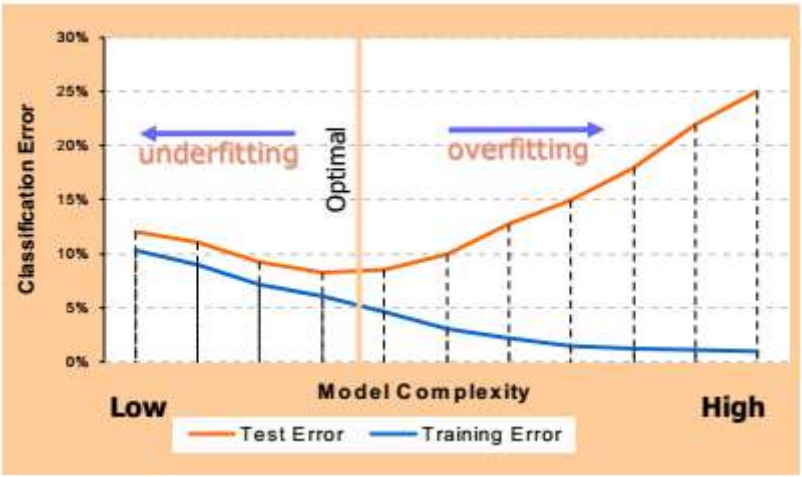




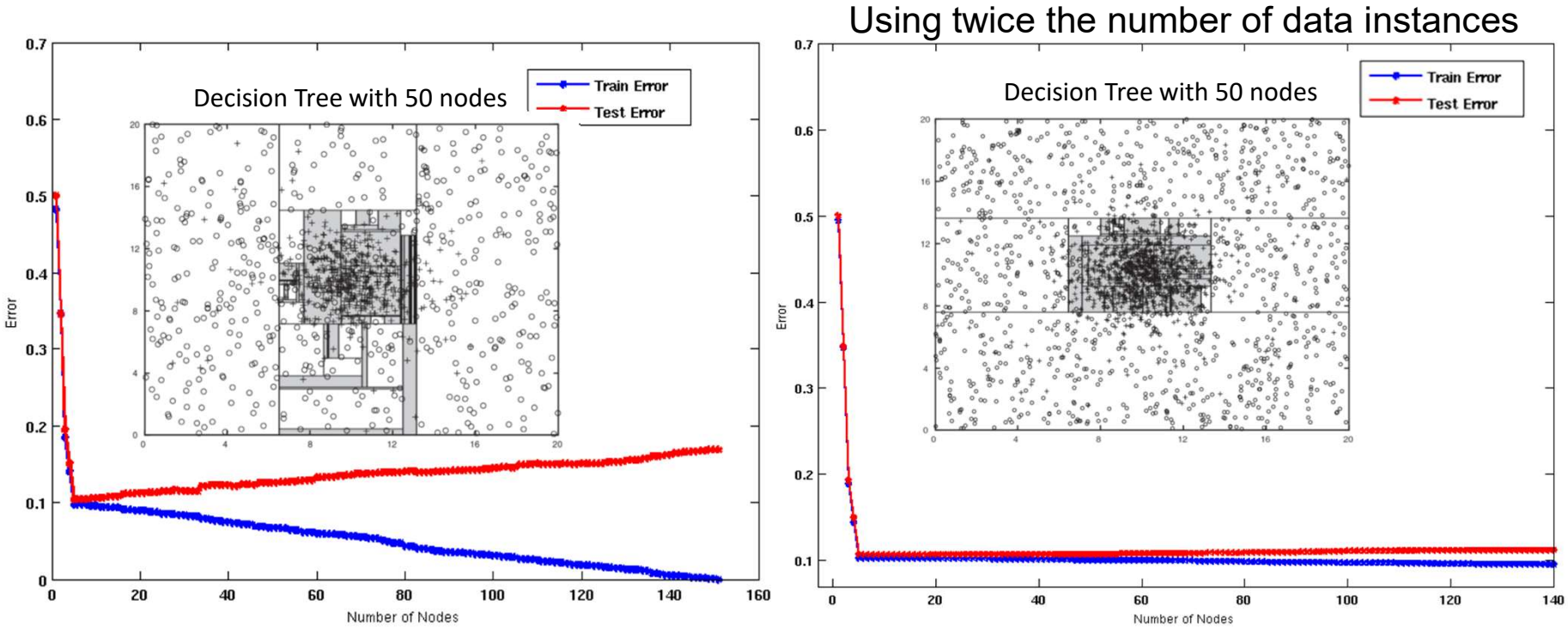
As the model becomes more and more complex, test errors can start increasing even though training error may be decreasing

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

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



Model Overfitting – Impact of Training Data Size



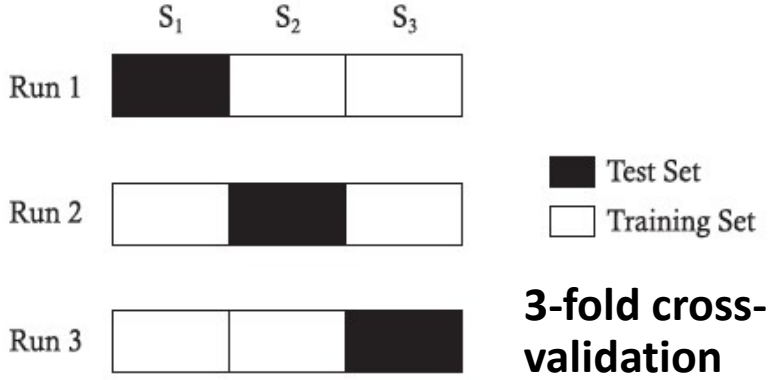
- Increasing the size of training data reduces the difference between training and testing errors at a given size of model
- Overfitting results in decision trees more complex than necessary
- Training error does not provide a good estimate of how well the tree will perform on previously unseen records
- Need ways for estimating generalization errors

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
- Divide training data into two parts:
  - Training set:
    - Used for model building
  - Test set:
    - Used for estimating generalization error
    - Note: validation set and test sets are different in some literature
- Drawback:
  - Less data available for training
- Cross Validation!

Model Evaluation

- Purpose:
  - To estimate performance of classifier on previously unseen data (test set)
- Holdout
  - Reserve k% for training and (100-k)% for testing
  - Random subsampling: repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n



3-fold cross-validation

Variations on Cross-validation

- Repeated cross-validation
  - Perform cross-validation a number of times
  - Gives an estimate of the variance of the generalization error
- Stratified cross-validation
  - Guarantee the same percentage of class labels in training and test
  - Important when classes are imbalanced, and the sample is small
- Use nested cross-validation approach for model selection and evaluation

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix (two classes):

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	**a**	**b**
	Class=No	**c**	**d**

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Metrics for Performance Eval. - Accuracy

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitations:

- Consider a 2-class problem:
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10
- If model predicts everything to be class 0, Accuracy = 9990/10000 = 99.9 % !
  - Accuracy is misleading because model does not detect any class 1 example

Cost Matrix

	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	**Class=Yes**	**Class=No**
	Class=Yes	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of classifying class **j** example as class **i**

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

Cost Matrix	PREDICTED CLASS		
ACTUAL CLASS	C(i|j)	+	-
	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	150	40
	-	60	250

Accuracy = 80%

Cost = 3910

Model M_2	PREDICTED CLASS		
ACTUAL CLASS		+	-
	+	250	45
	-	5	200

Accuracy = 90%

Cost = 4255

Precision, Recall and F Measure

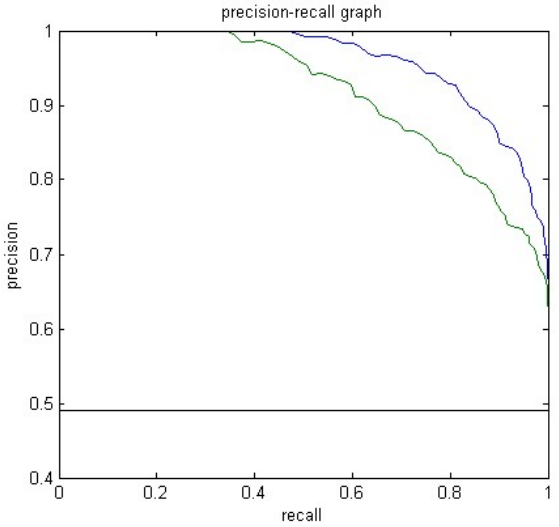
$$\text{Precision (p)} = \frac{a}{a+c} = \frac{TP}{TP+FP}$$

$$\text{Recall (r)} = \frac{a}{a+b} = \frac{TP}{TP+FN}$$

$$\text{F-measure (F)} = \frac{1}{\left(\frac{1/r+1/p}{2}\right)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c} = \frac{2TP}{2TP+FP+FN}$$

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Precision is biased towards **C(Yes|Yes) & C(Yes|No)**
- Recall is biased towards **C(Yes|Yes) & C(No|Yes)**
- F-Measure is biased towards all except **C(No|No)**
  - Relates sensitivity and accuracy through harmonic mean
  - High values of the F measure are obtained only when accuracy and recall have high values



Precision-Recall Plot

Performance Evaluation and Class Imbalance

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

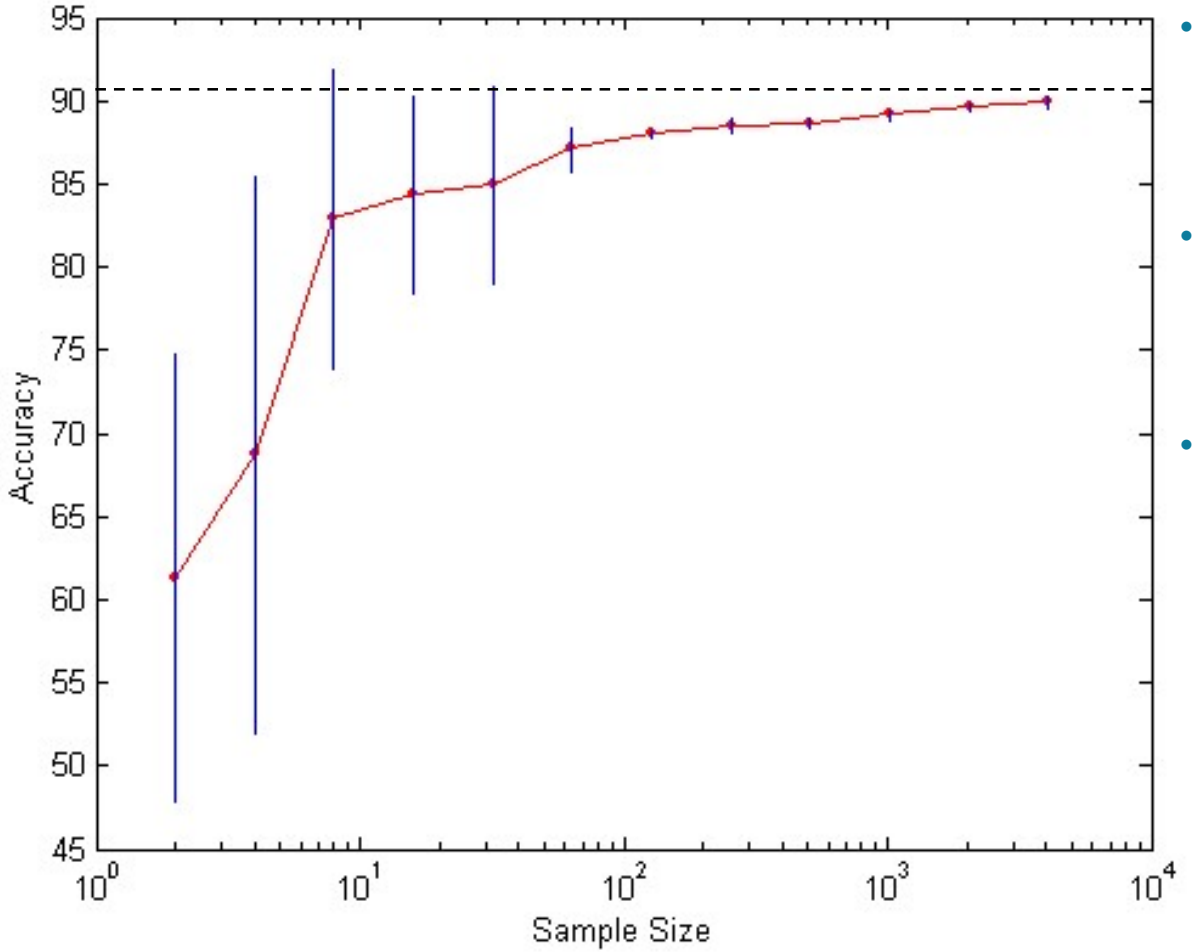
Class Imbalance Problem

- If the class we are interested in is very rare, then the classifier will ignore it
- We can modify the optimization criterion by using a cost sensitive metric
- We can **balance** the class distribution
  - Sample from the larger class so that the size of the two classes is the same
  - Replicate the data of the class of interest so that the classes are balanced: Over-fitting issues

Methods of Estimation

- **Holdout** (large number of examples, >1000 examples)
  - Reserve **2/3** for training and **1/3** for testing
- Random subsampling
  - One sample may be biased -- Repeated holdout
- Cross validation
  - Partition data into **k** disjoint subsets
  - **k**-fold: train on **k-1** partitions, test on the remaining one
  - Leave-one-out: **k=n**
  - Guarantees that each record is used the same number of times for training and testing
- Bootstrap (for small training sets, <500 examples)
  - Sampling with replacement

Learning Curve

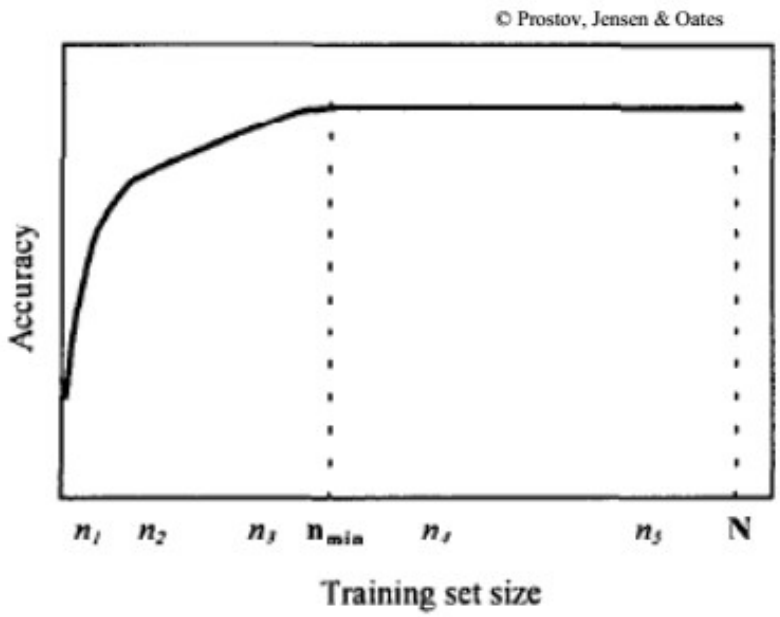


- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve
- Effect of small sample size:
  - Bias in the estimate
  - Variance of estimate

X-axis: number of examples from the training set (ranges from 0 to N: number of available examples)
Y-axis: classifier accuracy induced by a learning algorithm using a training set of size n

Learning Curve

- A learning curve is typically composed of 3 parts
  - Initial part (smaller) that shows very rapid growth
  - Intermediate part (larger) showing slowed growth
  - Final part that shows a plateau ⇒ the addition of new training examples no longer improves performance
- A learning curve converges when it reaches its plateau
- n_{min} represents the size of the training set where the curve converges



ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize trade-off between positive hits and false alarms
- **ROC** curve plots **TPR** (on **y**-axis) against **FPR** (on **x**-axis)

$$TPR = \frac{TP}{TP + FN}$$

Fraction of positive instances predicted correctly

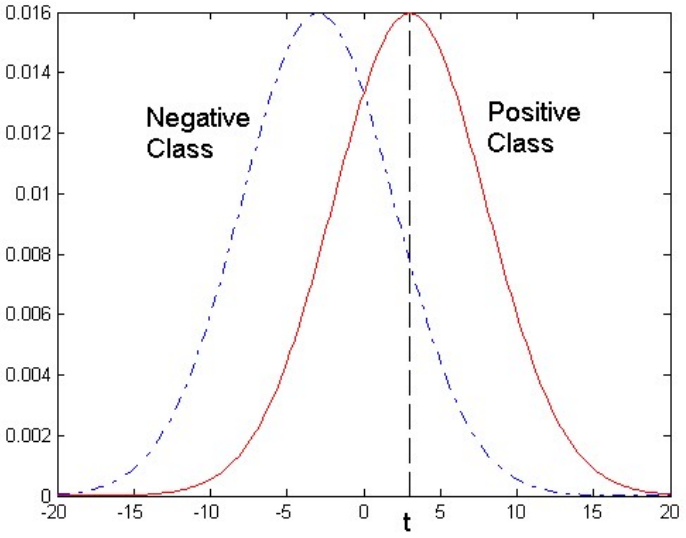
$$FPR = \frac{FP}{FP + TN}$$

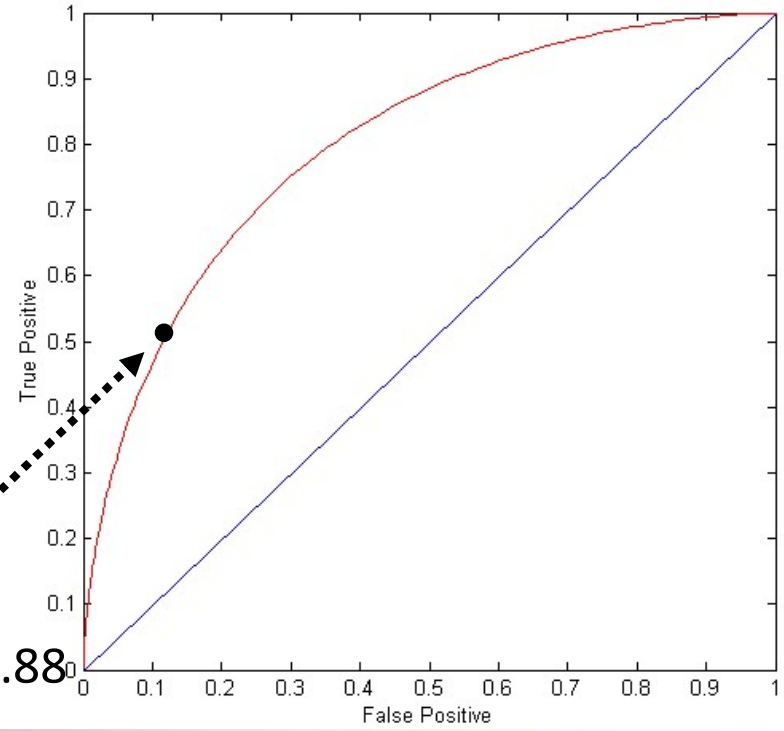
Fraction of negative instances predicted incorrectly

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

ROC Curve

- Classifier Performance represented as a point on the ROC curve
- Changing some parameter of the algorithm, sample distribution or cost matrix changes the point location
  - **1**-dimensional data set containing **2** classes (***positive*** and ***negative***)
  - any points located at **x > t** is classified as ***positive***



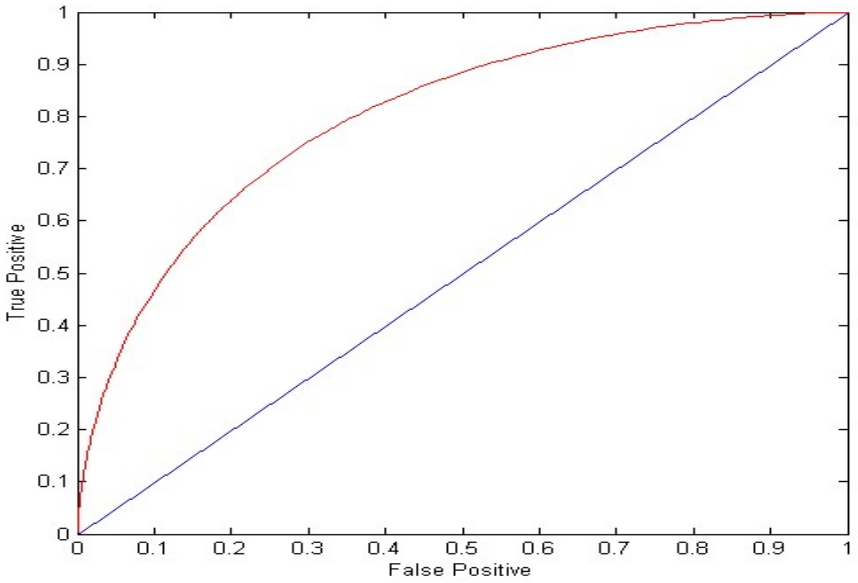


At threshold **t**: TP=0.5, FN=0.5, FP=0.12, FN=0.88

ROC Curve

(TP,FP):

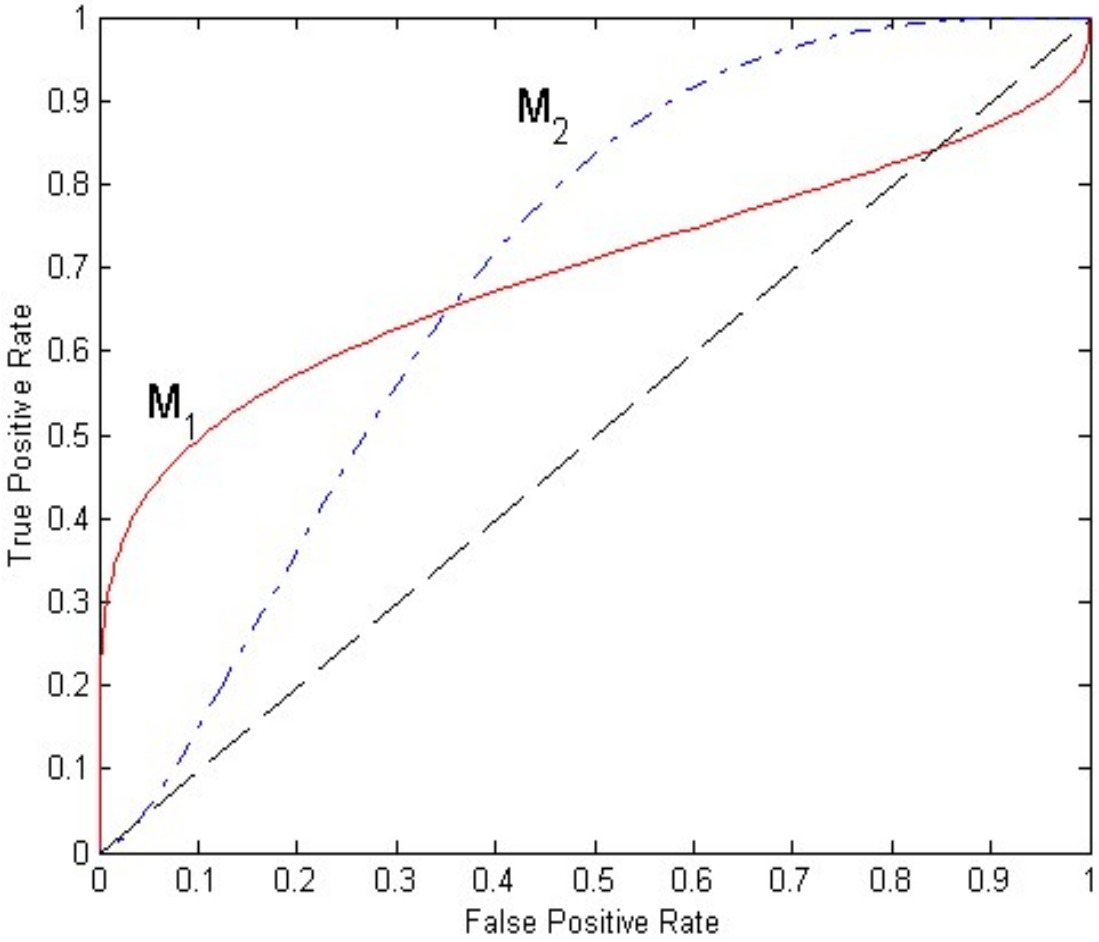
- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (1,0): ideal



- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - Prediction is opposite of the true class

		PREDICTED CLASS	
		Yes	No
ACTUAL CLASS	Yes	a (TP)	b (FN)
	No	c (FP)	d (TN)

Using ROC for Model Comparison



No model consistently outperform the other

- M_1 is better for small FPR
- M_2 is better for large FPR

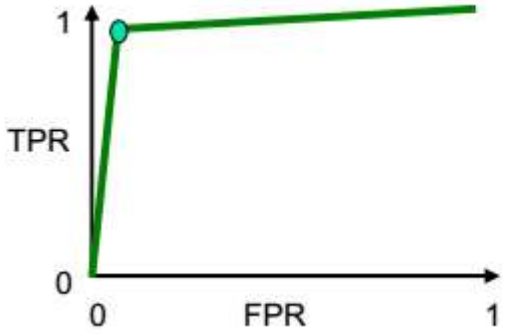
Area Under the ROC curve (AUC)

- Ideal:
  - Area = 1
- Random guess:
  - Area = 0.5

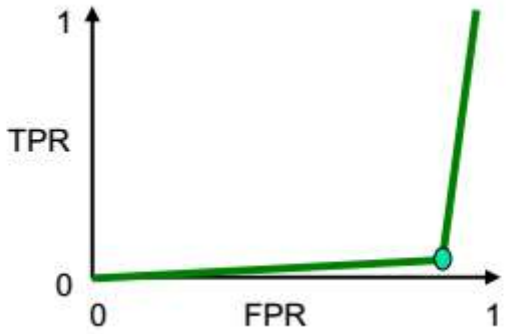
Area Under the Curve (AUC) as a single number for evaluation

ROC for Model Comparison

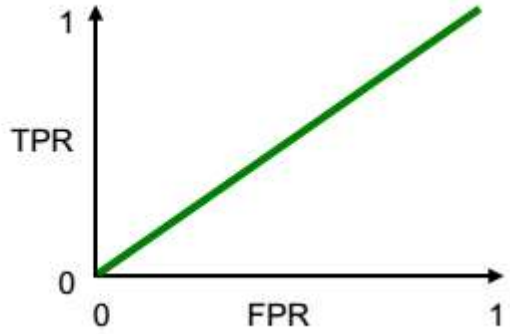
- **ROC Curve - Discrete Classifiers - Good and Bad Classifiers**

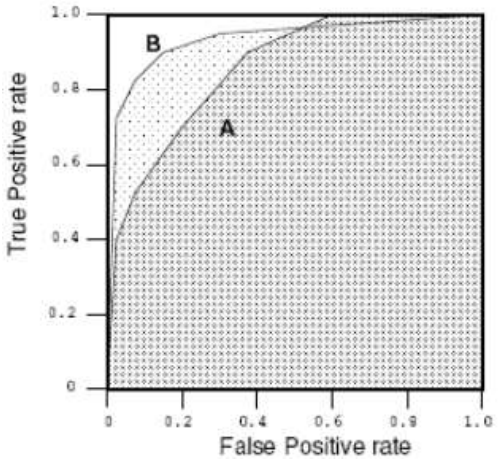


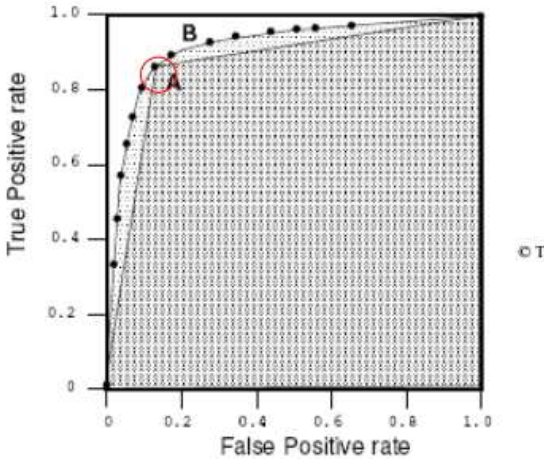
Good classifier
TPR – High/ FPR – Low

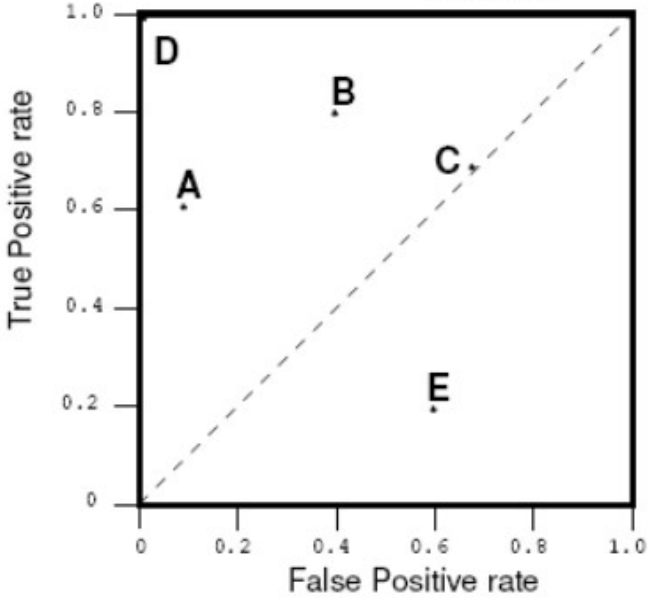


Very Bad classifier
TPR – Low / FPR – High



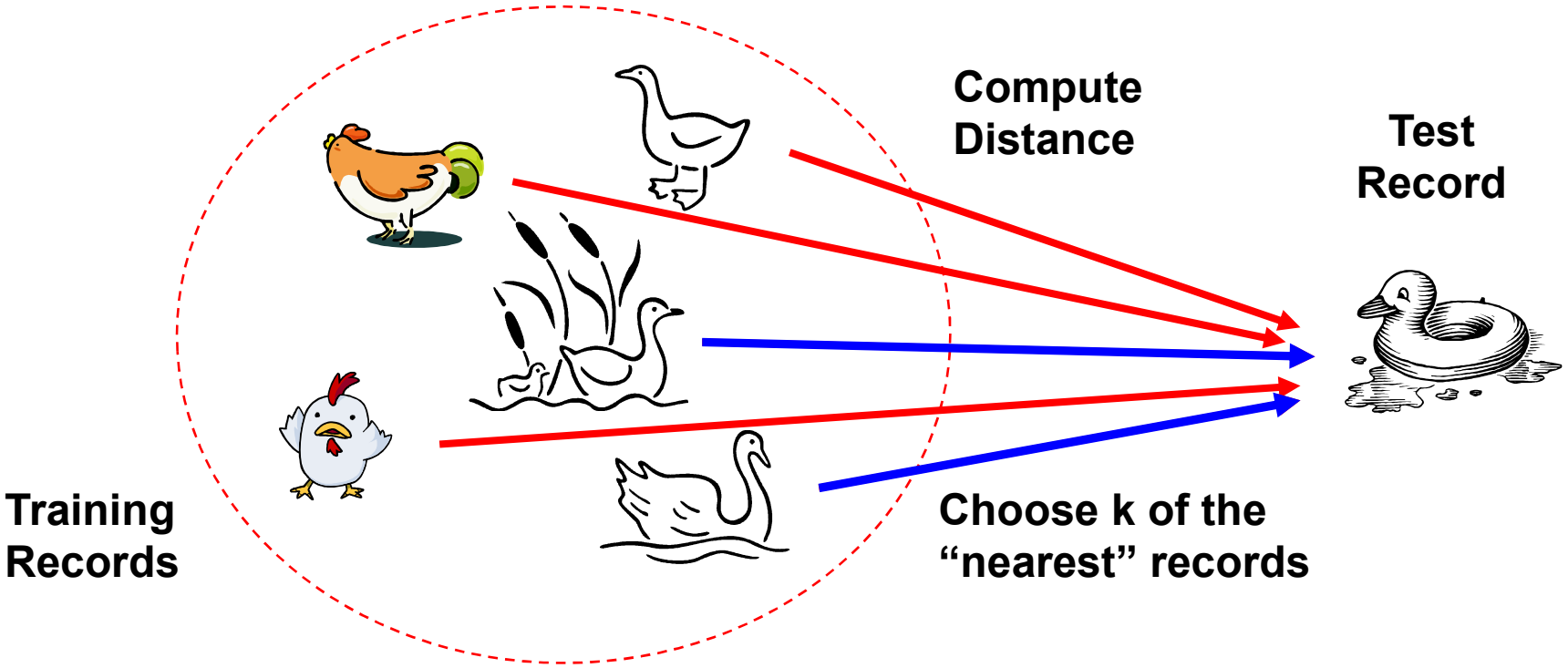




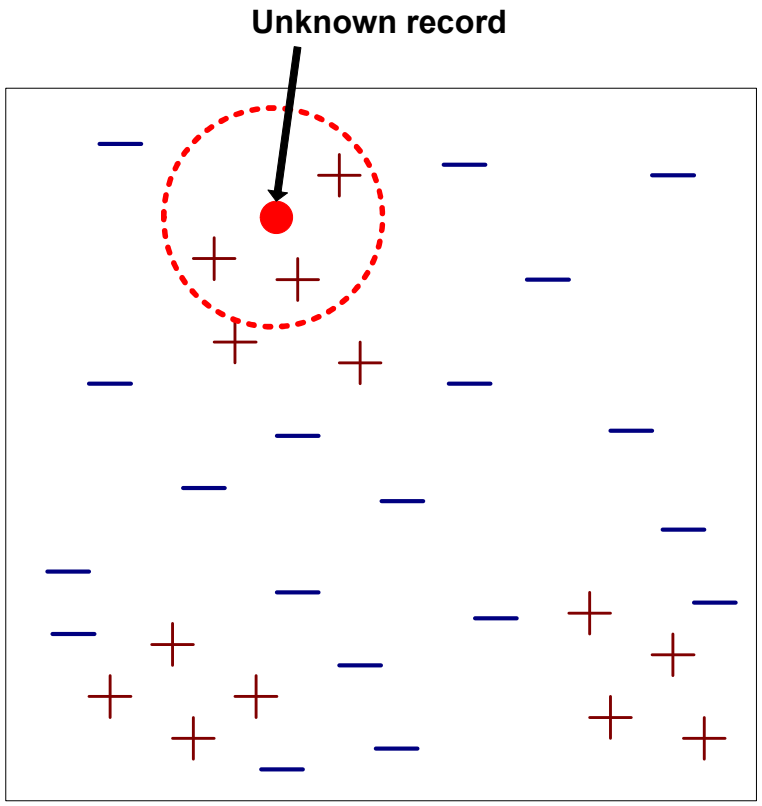


Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it is probably a duck



Nearest-Neighbor Classifiers



Requires the following:

- A set of labeled records
- Proximity metric to compute distance/ similarity between a pair of records
  - e.g., Euclidean distance
- The value of *k*, the number of nearest neighbors to retrieve
- A method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

How to Determine the class label of a Test Sample?

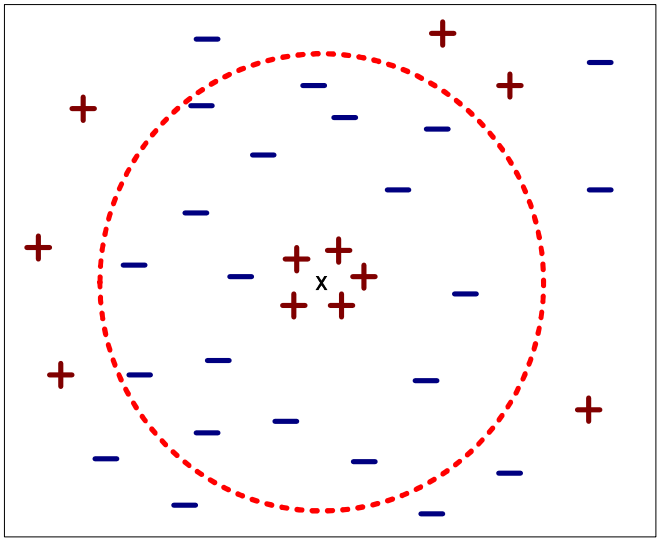
- Take the majority vote of class labels among the k-nearest neighbors
- Weight the vote according to distance: weight factor, $w = 1/d^2$

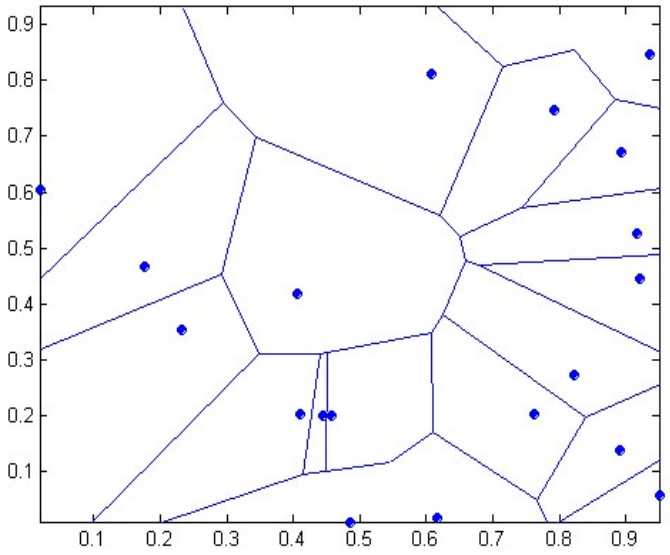
Nearest Neighbor Classification

- **Data preprocessing is often required**
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
    - Examples:
      - Height of a person may vary from 1.5m to 1.9m
      - Weight of a person may vary from 45Kg to 120Kg
      - Income of a person may vary from 10K to 1M Eur
  - Time series are often standardized to have 0 means a standard deviation of 1

Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes
- Nearest neighbor classifiers:
  - Local classifiers
  - Can produce decision boundaries of arbitrary shapes
  - Lazy classifiers

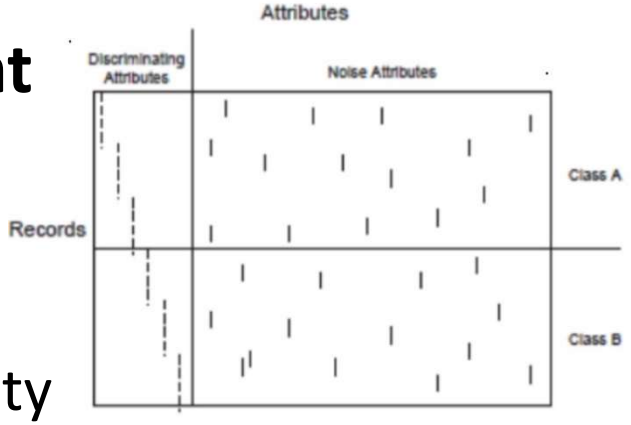




1-nn decision boundary is a Voronoi Diagram

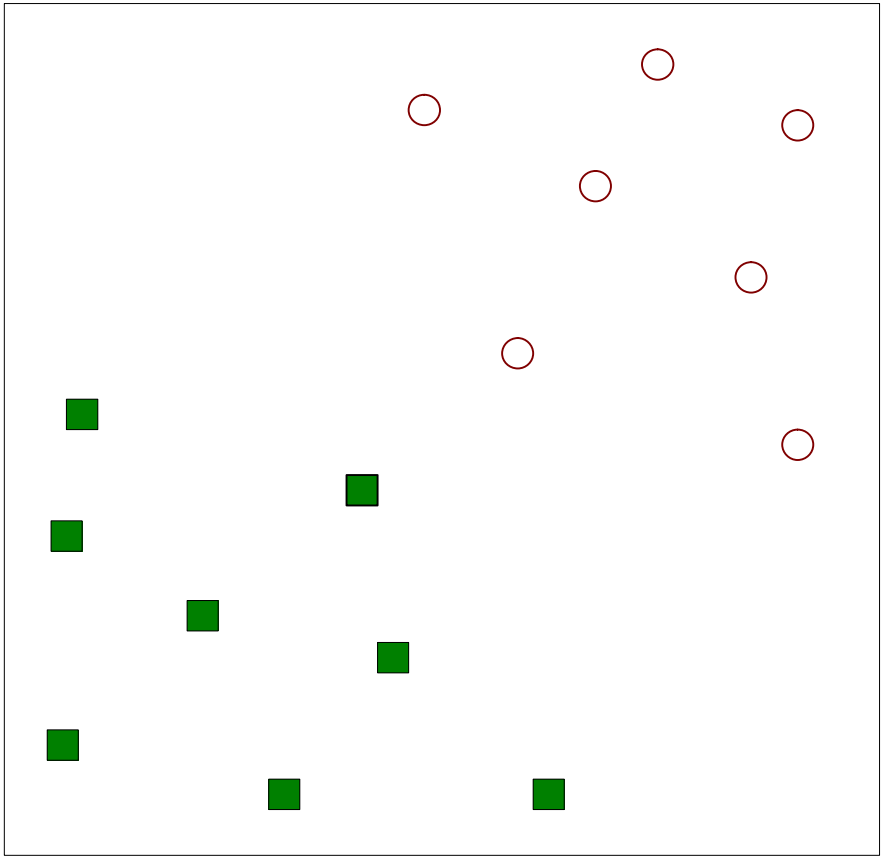
Nearest Neighbor Classification…

- **Handling missing values in training and test sets**
  - Proximity computations normally require the presence of all attributes
  - Some approaches use the subset of attributes present in two instances
    - This may not produce good results since it effectively uses different proximity measures for each pair of instances
    - Thus, proximities are not comparable
- **Handling Irrelevant and Redundant Attributes**
  - Irrelevant attributes add noise to the proximity measure
  - Redundant attributes bias the proximity measure towards certain attributes



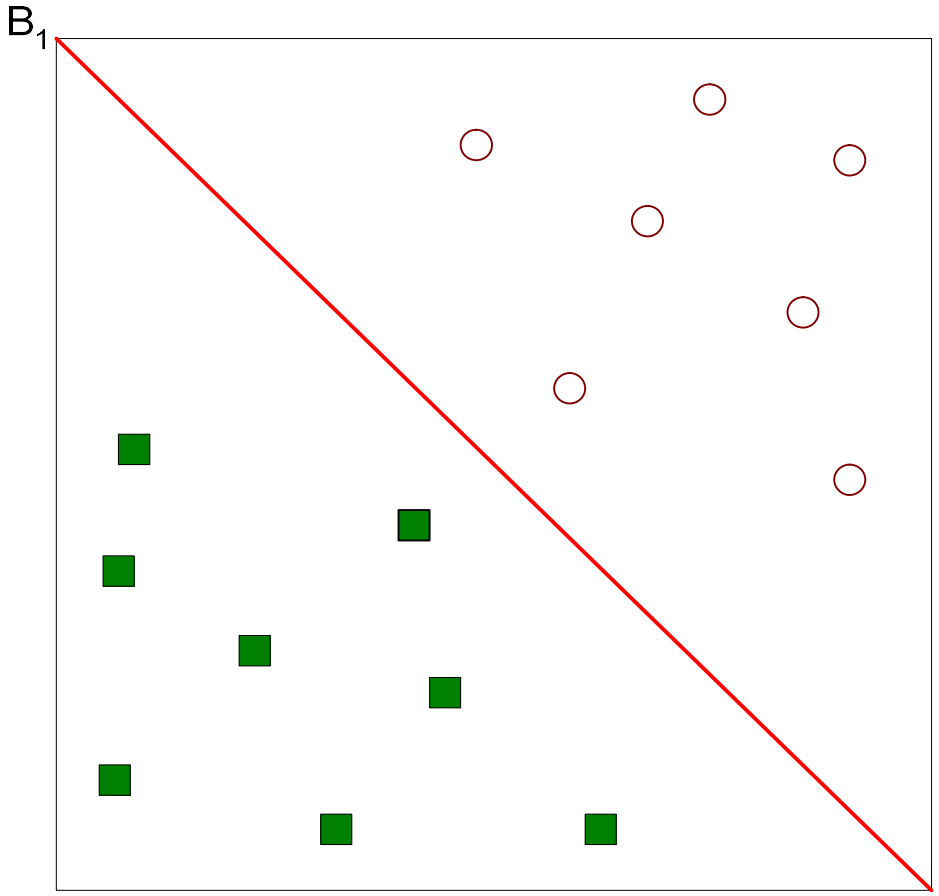
(a) Synthetic data set 1.

Support Vector Machines



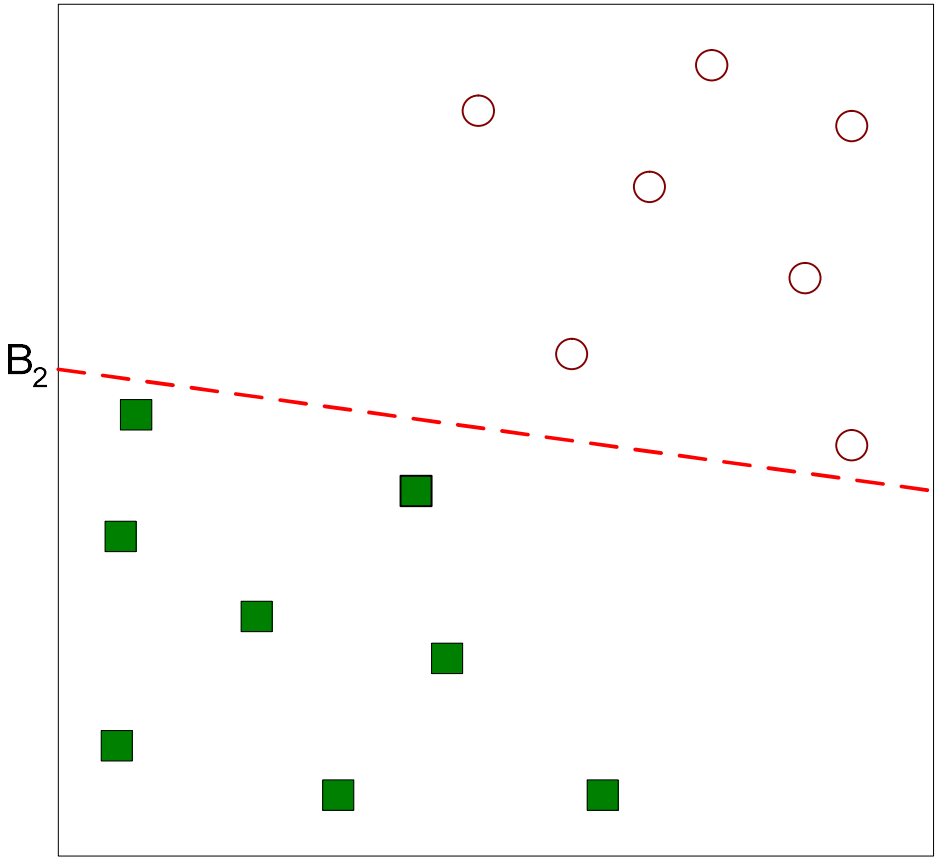
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machines



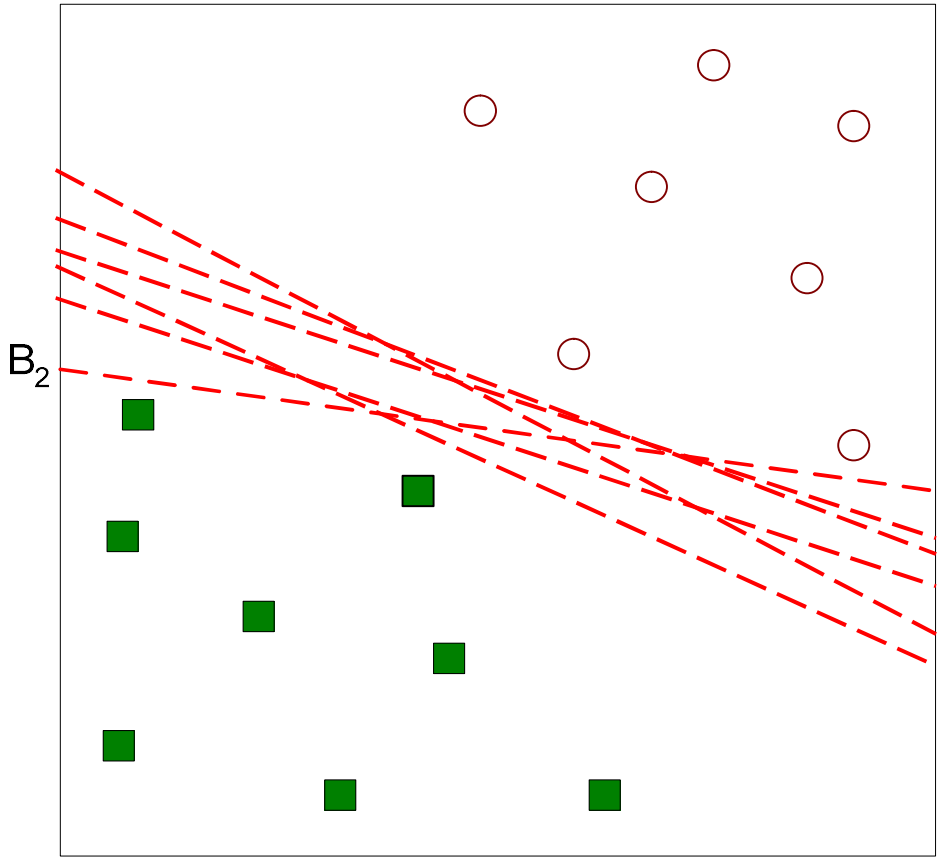
- One Possible Solution

Support Vector Machines



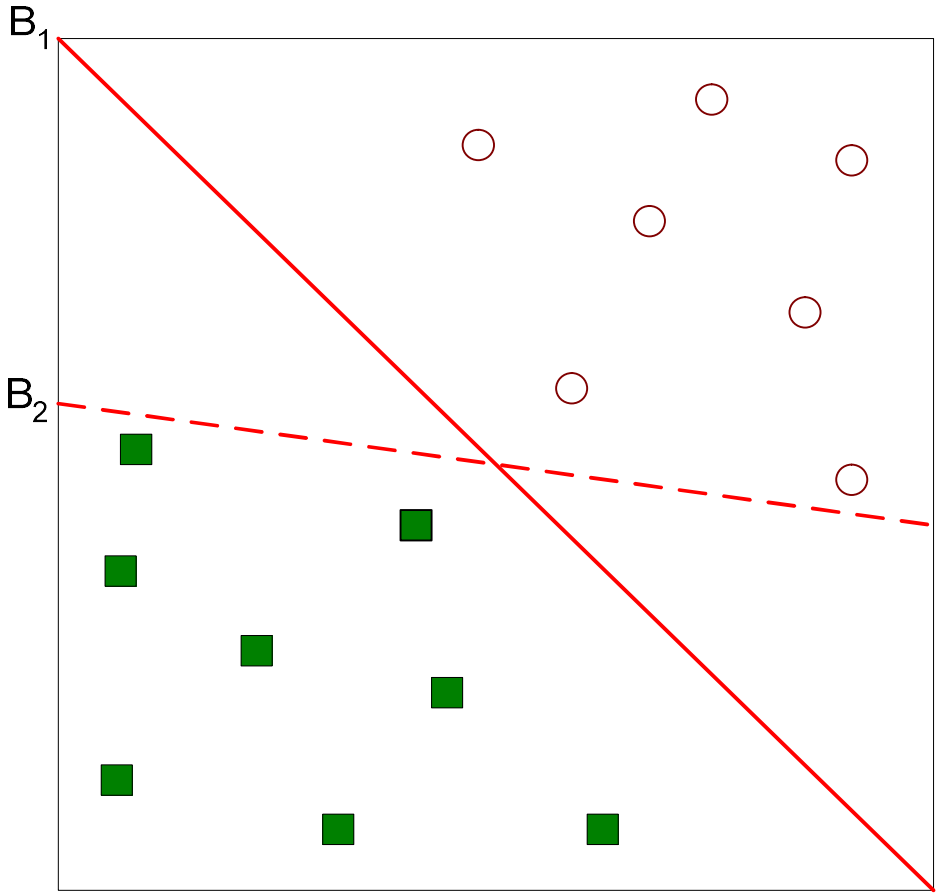
- Another possible solution

Support Vector Machines



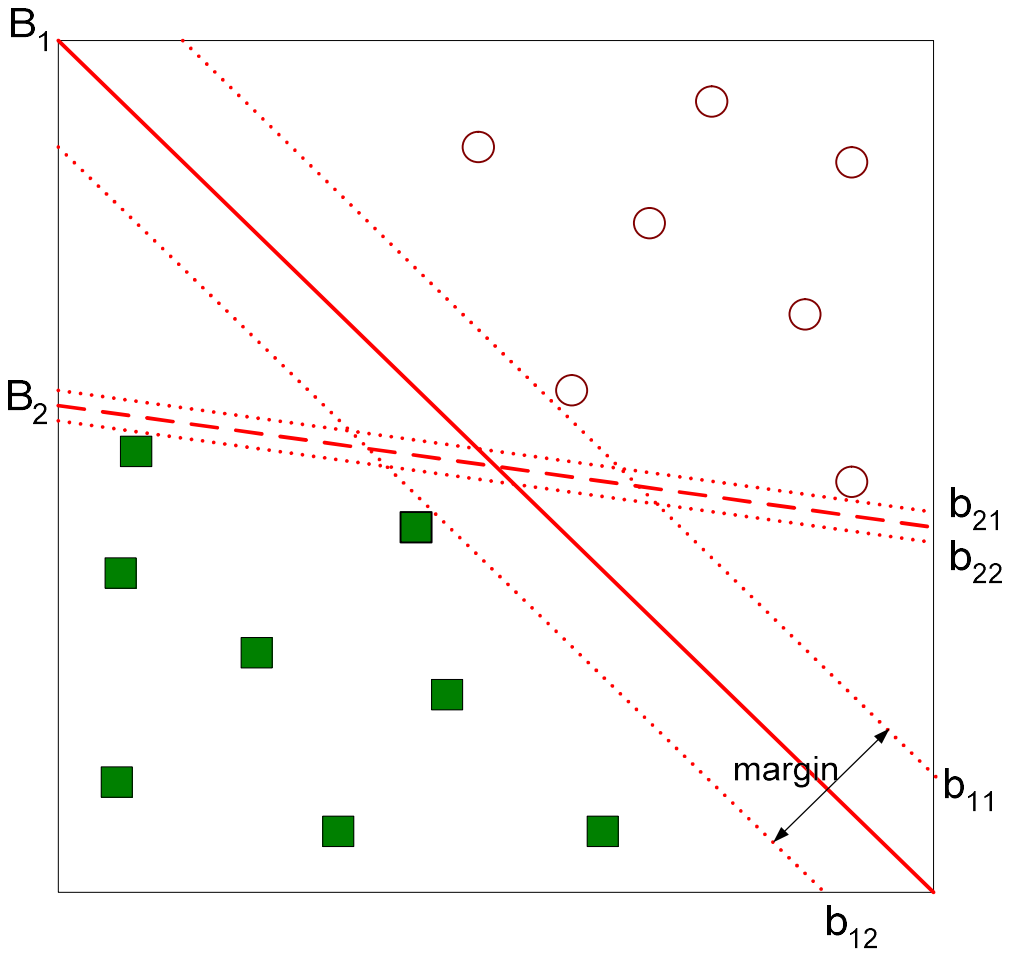
- Other possible solutions

Support Vector Machines



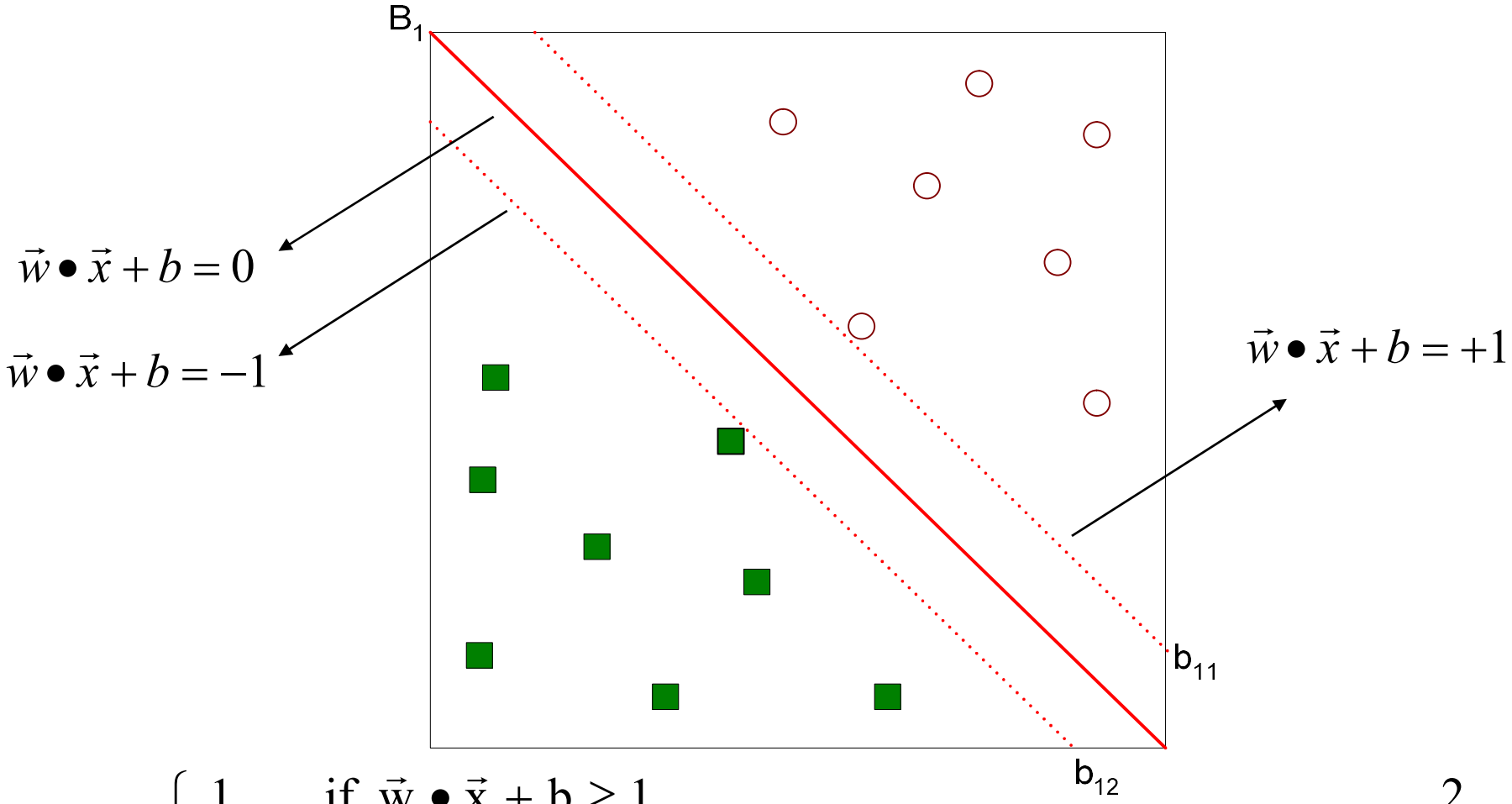
- Which one is better? B1 or B2?
- How do you define better?

Support Vector Machines



- Find hyperplane maximizes the margin => B1 is better than B2

Support Vector Machines



$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$$

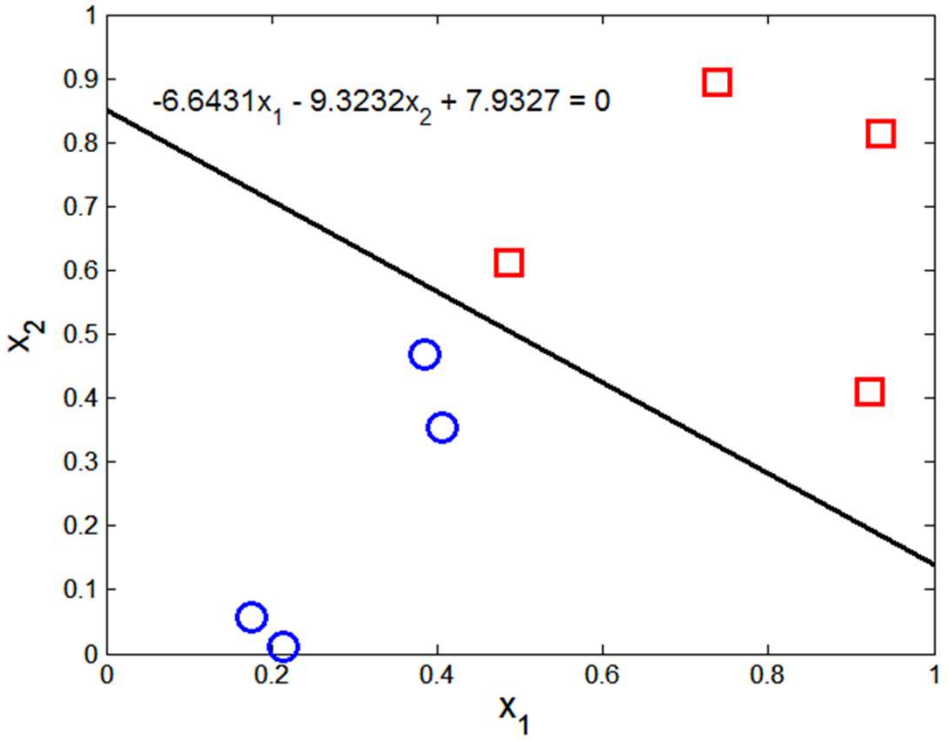
$$\text{Margin} = \frac{2}{\| \vec{w} \|}$$

Learning Linear SVM

- Linear model: $f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$
- Learning the model is equivalent to determining values of $\vec{w}$ and b
  - How to find $\vec{w}$ and b from training data?
- Objective is to maximize: $\text{Margin} = \frac{2}{\|\vec{w}\|}$
- Which is equivalent to minimizing: $L(\vec{w}) = \frac{\|\vec{w}\|^2}{2}$
- Subject to the following constraints: $y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 \end{cases}$

  or $y_i(w \bullet x_i + b) \geq 1, \qquad i = 1,2,\ldots,N$
- This is a constrained optimization problem
  - Solve it using Lagrange multiplier method

Example of Linear SVM



Support vectors

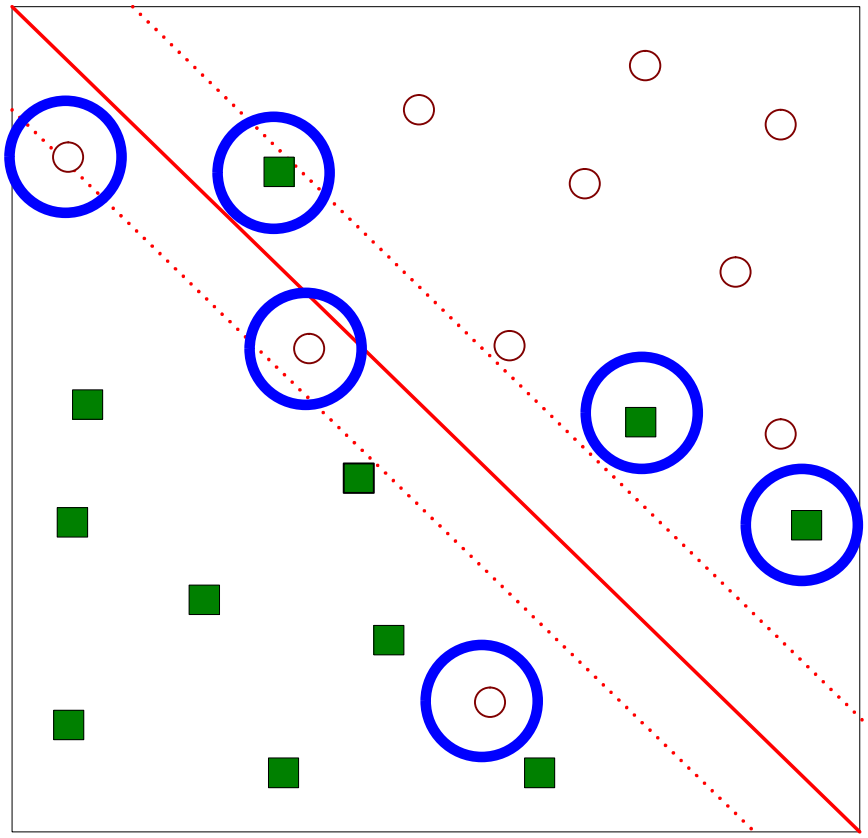
x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

- Decision boundary depends only on support vectors
  - If you have data set with same support vectors, decision boundary will not change
  - How to classify using SVM once **w** and *b* are found? Given a test record, x_i

$$f(\vec{x}_i) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 \end{cases}$$

Support Vector Machines

- What if the problem is not linearly separable?



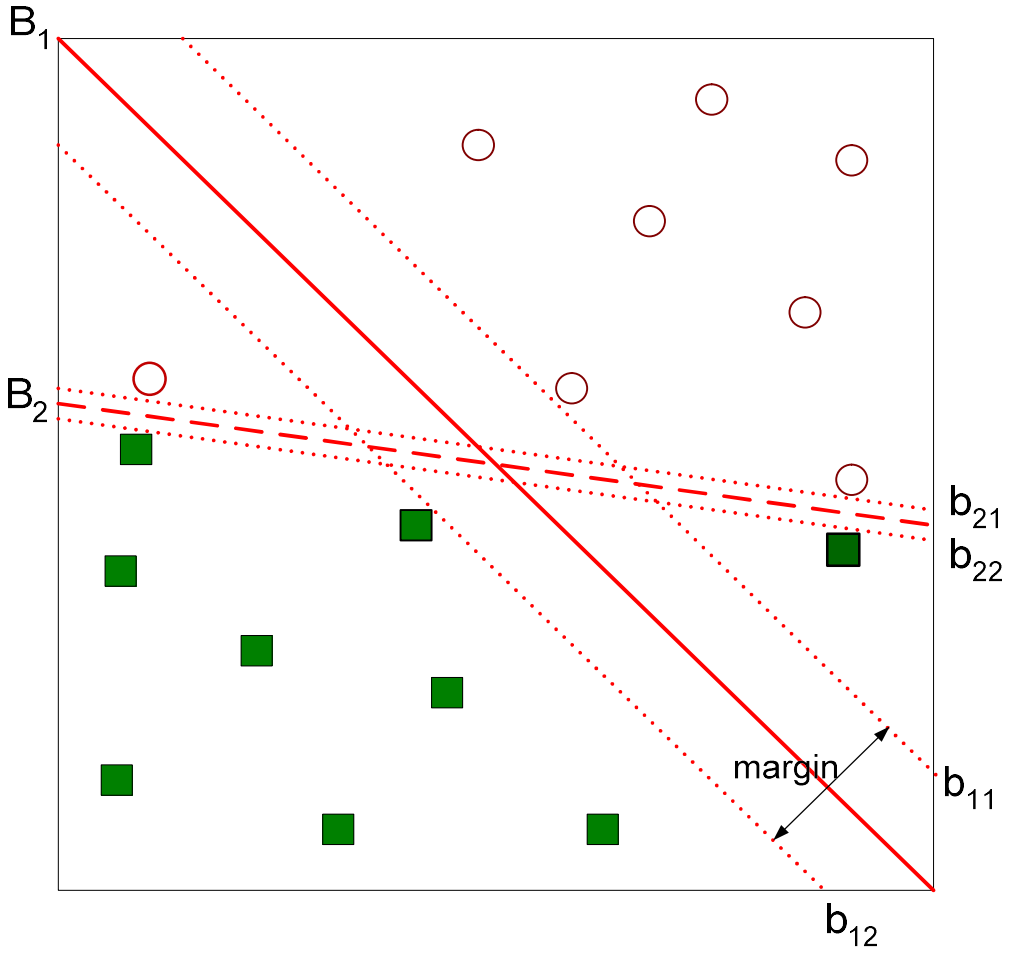
Support Vector Machines

- What if the problem is not linearly separable?
  - Introduce slack variables
    - Need to minimize: $L(w) = \frac{\| \vec{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k \right)$
    - Subject to:

$$y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 - \xi_i \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 + \xi_i \end{cases}$$

    - If k is 1 or 2, this leads to similar objective function as linear SVM but with different constraints

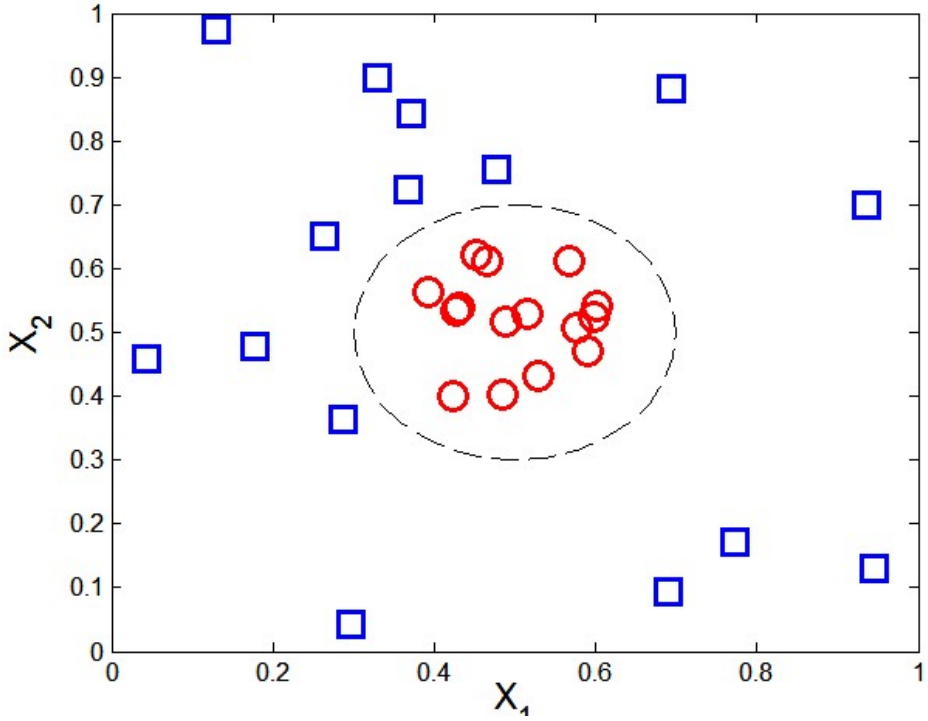
Support Vector Machines



- Find the hyperplane that optimizes both factors

Nonlinear Support Vector Machines

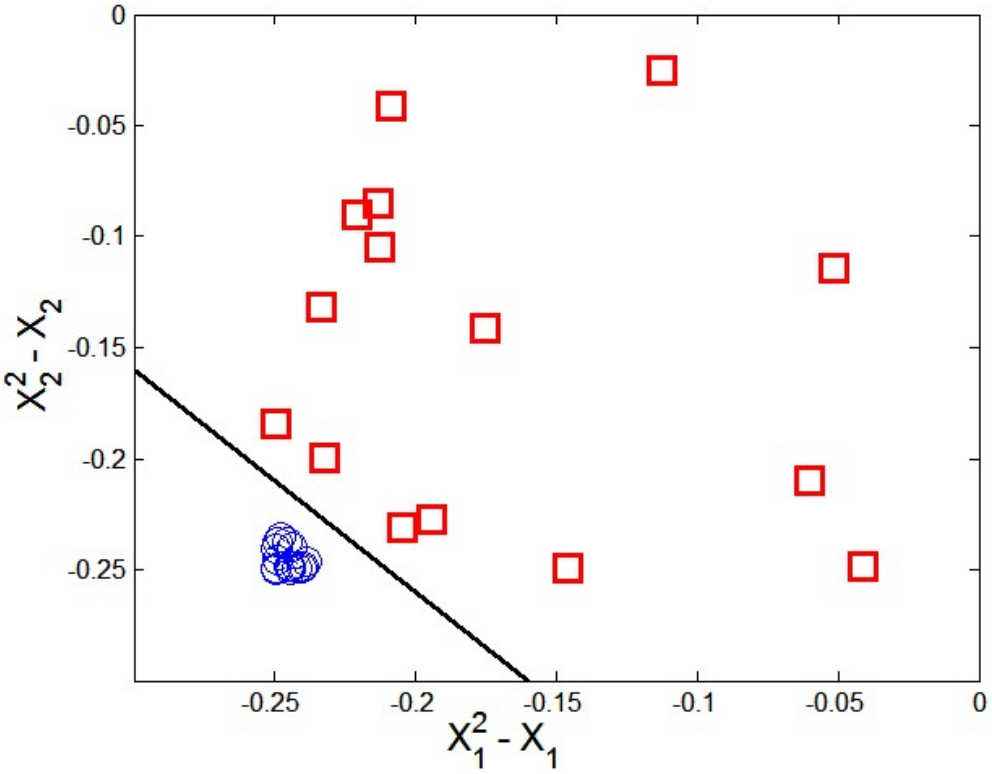
- What if decision boundary is not linear?



$$y(x_1, x_2) = \begin{cases} 1 & \text{if } \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} > 0.2 \\ -1 & \text{otherwise} \end{cases}$$

Nonlinear Support Vector Machines

- Transform data into higher dimensional space



$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46.$$

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1).$$

$$w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

Decision boundary:

$$\vec{w} \bullet \Phi(\vec{x}) + b = 0$$

Learning Nonlinear SVM

- Optimization problem:

$$\min_{w} \frac{\|\mathbf{w}\|^2}{2}$$

$$subject\ to \qquad y_i(\mathbf{w}\cdot\Phi(x_i)+b) \geq 1,\ \forall\{(x_i,y_i)\}$$

- Which leads to the same set of equations (but involve $\Phi(x)$ instead of x)

$$L_D = \sum_{i=1}^{n}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) \qquad \mathbf{w} = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i)$$

$$\lambda_i\{y_i(\sum_j \lambda_j y_j \Phi(\mathbf{x}_j)\cdot\Phi(\mathbf{x}_i)+b)-1\} = 0,$$

$$f(\mathbf{z}) = sign(\mathbf{w}\cdot\Phi(\mathbf{z})+b) = sign(\sum_{i=1}^{n}\lambda_i y_i \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{z})+b).$$

Learning Nonlinear SVM

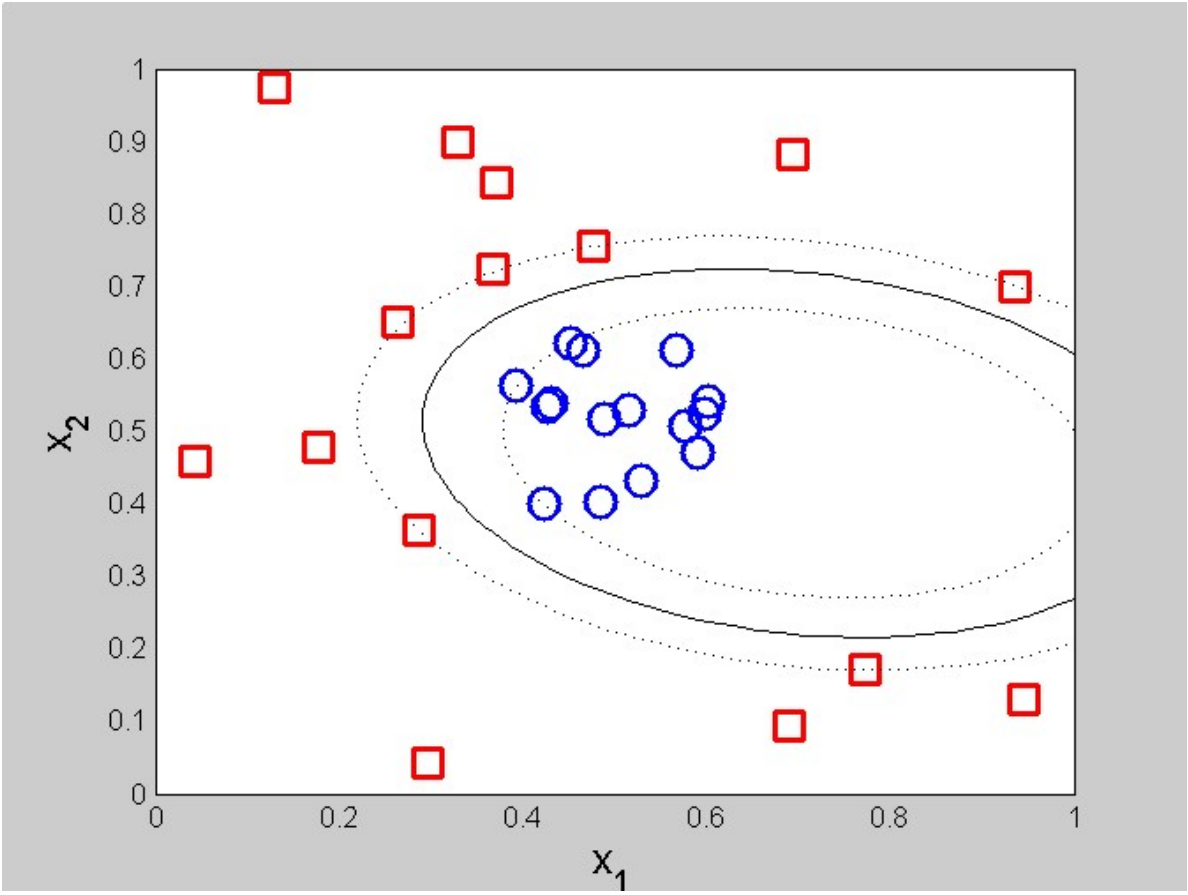
- Issues:
  - What type of mapping function Φ should be used?
  - How to do the computation in high dimensional space?
    - Most computations involve dot product $\Phi(x_i) \bullet \Phi(x_j)$
    - Curse of dimensionality?
- Kernel Trick:
  - $\Phi(x_i) \bullet \Phi(x_j) = K(x_i, x_j)$
  - $K(x_i, x_j)$ is a kernel function (expressed in terms of the coordinates in the original space)
    - Examples:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)}$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta)$$

Example of Nonlinear SVM



SVM with polynomial degree 2 kernel

Learning Nonlinear SVM

- Advantages of using kernel:
  - Don't have to know the mapping function Φ
  - Computing dot product $\Phi(x_i) \bullet \Phi(x_j)$ in the original space avoids curse of dimensionality

- Not all functions can be kernels
  - Must make sure there is a corresponding Φ in some high-dimensional space
  - Mercer's theorem (see textbook)

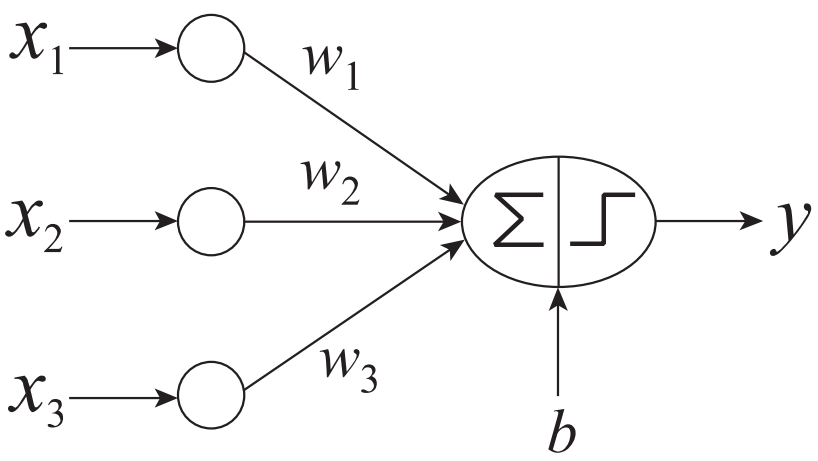
Characteristics of SVM

- Learning problem is formulated as a **convex optimization problem**
  - Efficient algorithms are available to find the global minima
  - Many of the other methods use greedy approaches and find locally optimal solutions
  - High computational complexity for building the model
- Robust to noise
- Overfitting is handled by maximizing the margin of the decision boundary
- SVM can handle irrelevant and redundant better than many other techniques
- User needs to provide the type of kernel function and cost function
- Difficult to handle missing values
- What about categorical variables?

Artificial Neural Networks (ANN)

- **Basic Idea:** A complex non-linear function can be learned as a composition of simple processing units
- ANN is a collection of simple processing units (nodes) that are connected by directed links (edges)
  - Every node receives signals from incoming edges, performs computations, and transmits signals to outgoing edges
  - Analogous to *human brain* where nodes are neurons and signals are electrical impulses
  - Weight of an edge determines the strength of connection between the nodes
- Simplest ANN: **Perceptron** (single neuron)

Basic Architecture of Perceptron



$$y = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x} + b > 0. \\ -1, & \text{otherwise.} \end{cases}$$

$$\tilde{\mathbf{w}} = (\mathbf{w}^T \; b)^T \qquad \tilde{\mathbf{x}} = (\mathbf{x}^T \; 1)^T$$

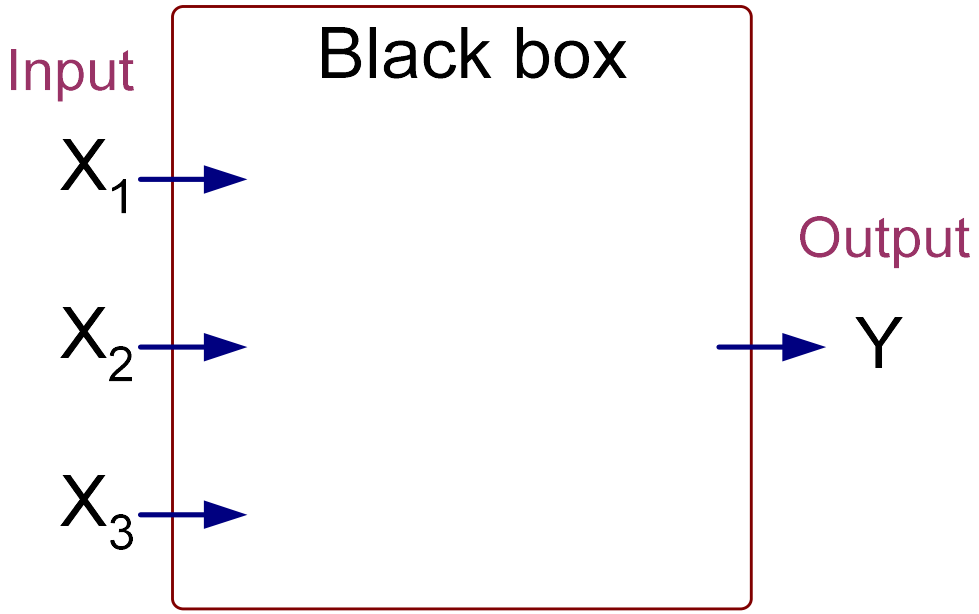
$$\hat{y} = sign(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})$$

Activation Function

- Learns linear decision boundaries
- Related to logistic regression (activation function is sign instead of sigmoid)

Perceptron Example

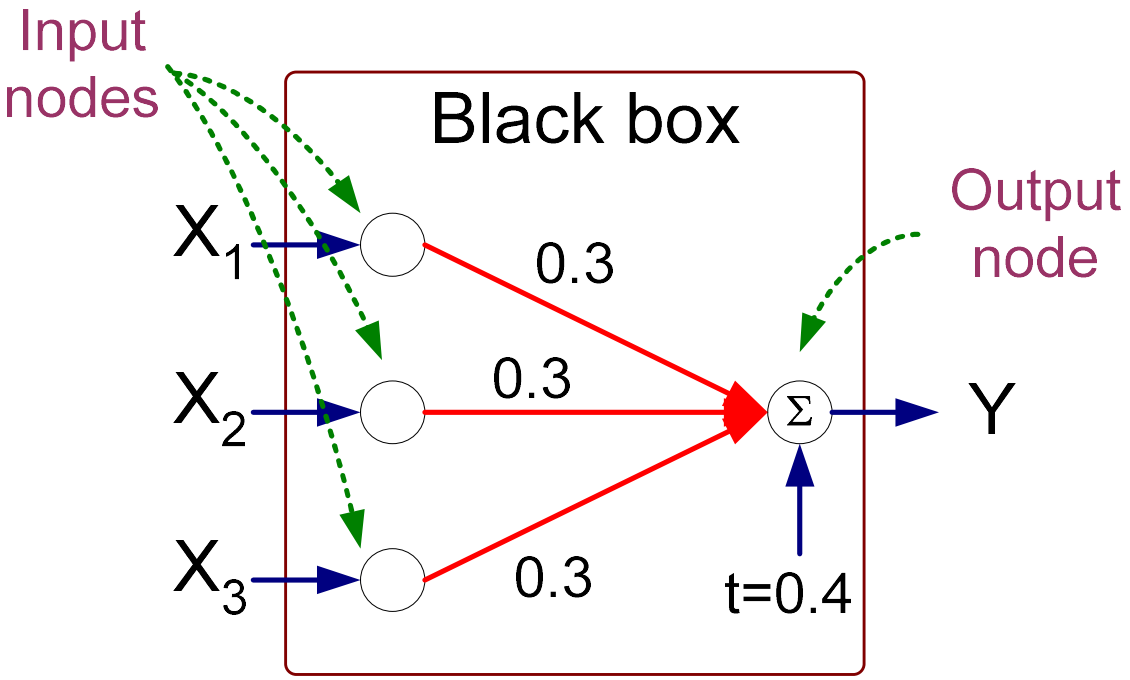
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



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

Perceptron Example

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign\,(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign\,(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute $\hat{y}_i$
    - Update the weights:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$$

- Until stopping condition is met
  - k: iteration number
  - λ: learning rate

Perceptron Learning Rule

- Weight update formula:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda\left(y_i - \hat{y}_i^{(k)}\right)x_{ij}$$

- Intuition:
  - Update weight based on error: e = $(y_i - \hat{y}_i)$
    - If y = $\hat{y}$, e=0: no update needed
    - If y > $\hat{y}$, e=2: weight must be increased (assuming Xij is positive) so that $\hat{y}$ will increase
    - If y < $\hat{y}$, e=-2: weight must be decreased (assuming Xij is positive) so that $\hat{y}$ will decrease

Example of Perceptron Learning

$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

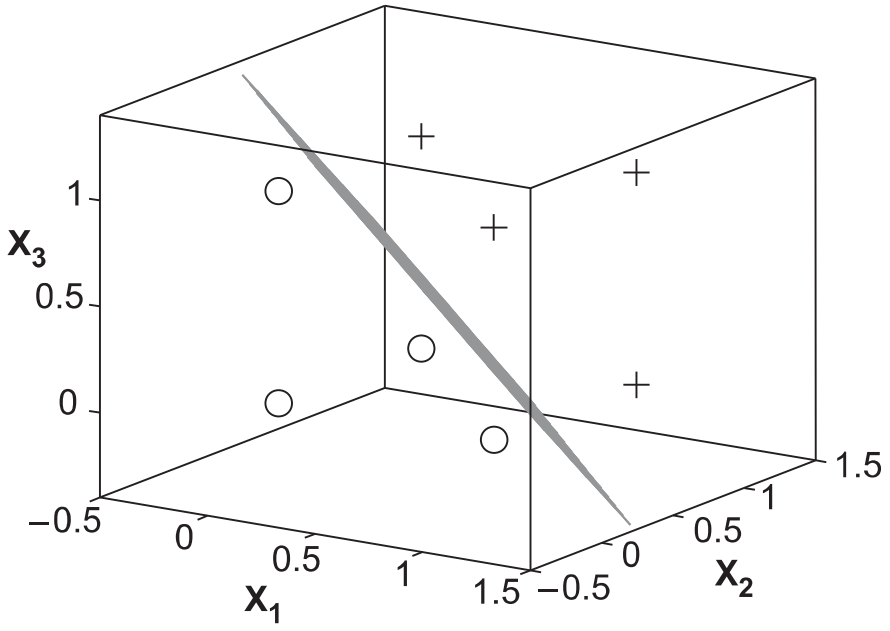
Weight updates over first epoch

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Weight updates over all epochs

Perceptron Learning

- Since y is a linear combination of input variables, decision boundary is linear

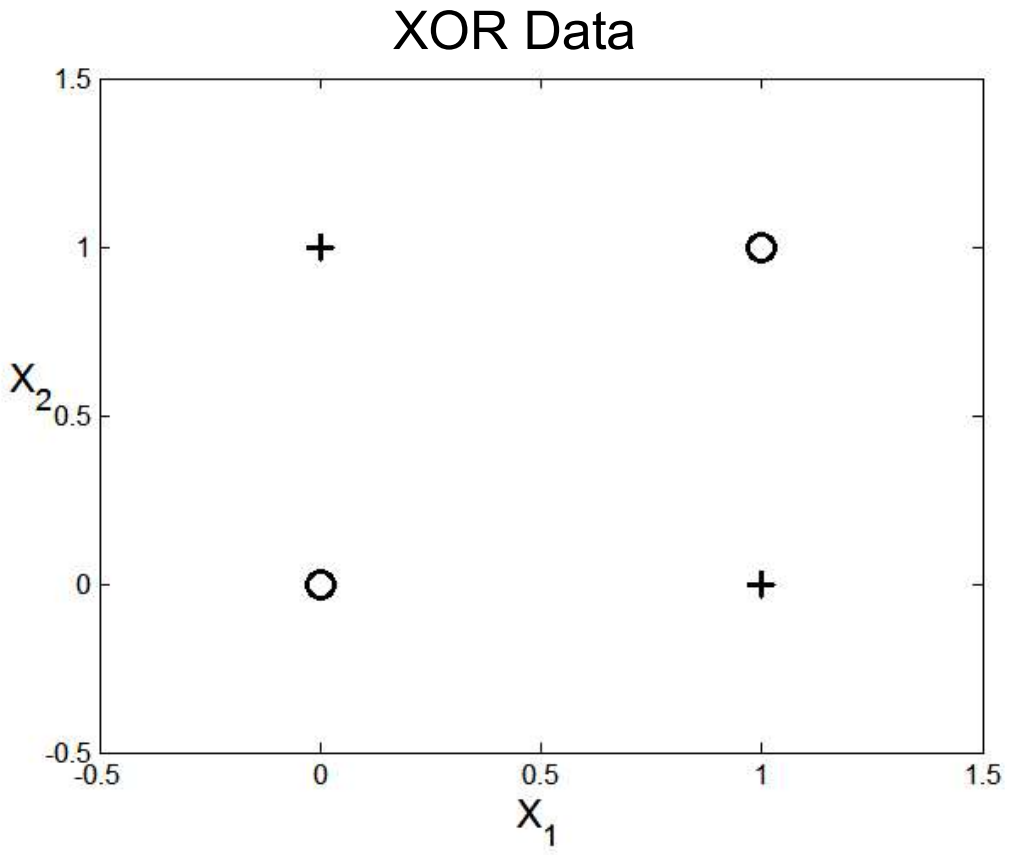


Nonlinearly Separable Data

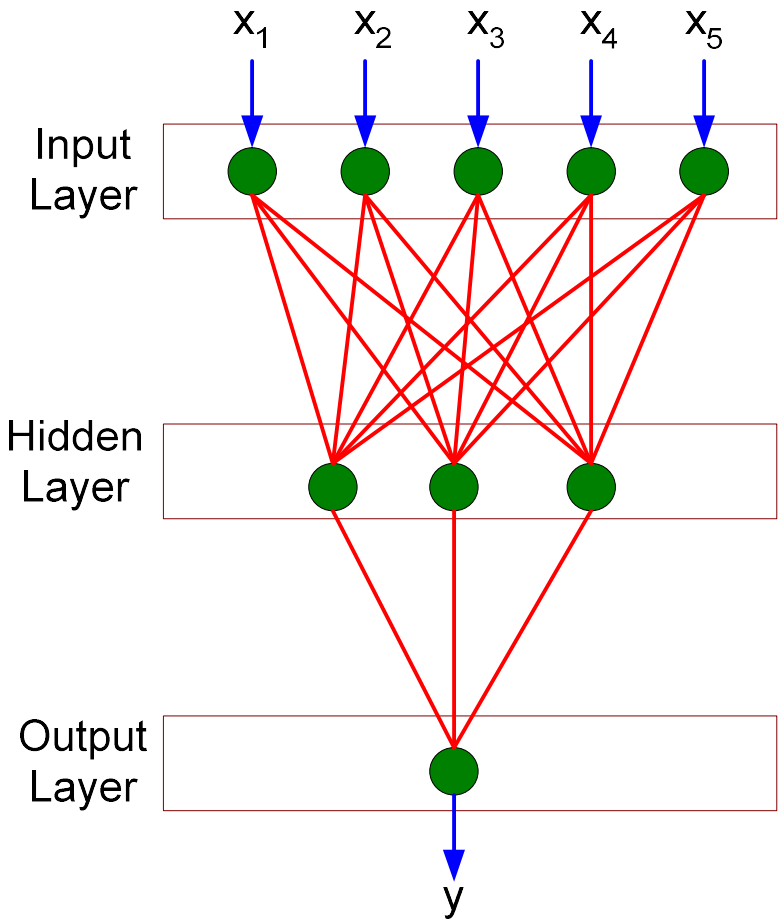
For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

$$y = x_1 \oplus x_2$$

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1



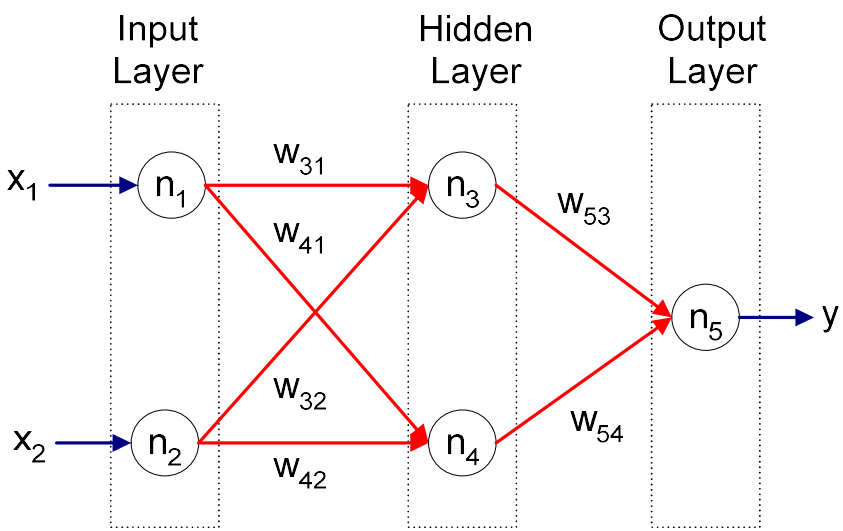
Multi-layer Neural Network

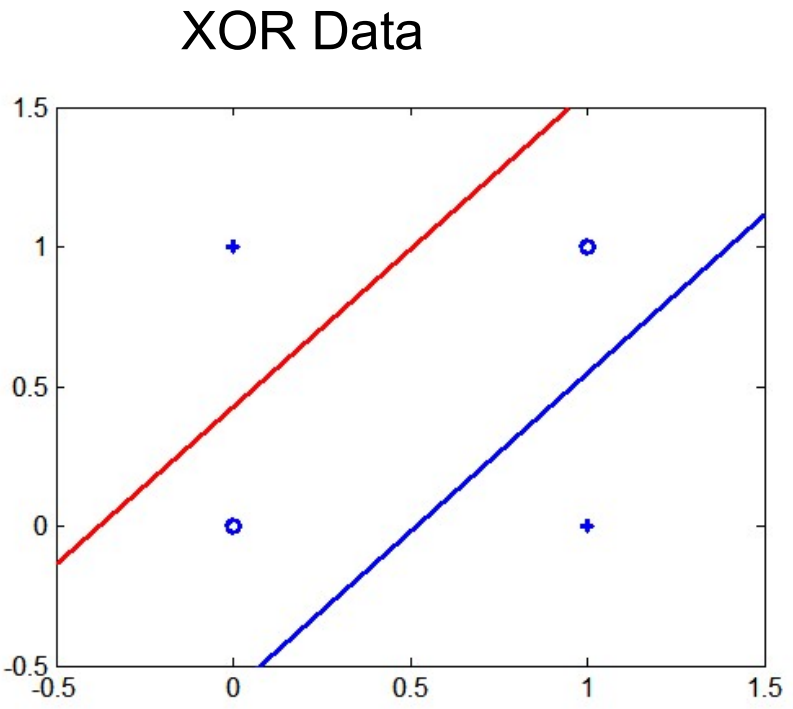


- More than one *hidden layer* of computing nodes
- Every node in a hidden layer operates on activations from preceding layer and transmits activations forward to nodes of next layer
- Also referred to as “feedforward neural networks”

Multi-layer Neural Network

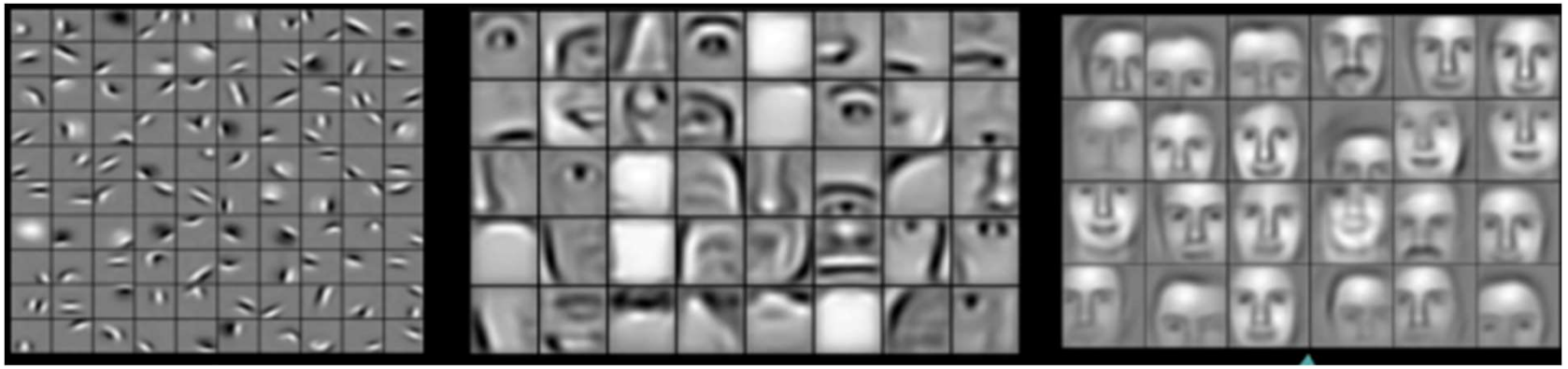
- Multi-layer neural networks with at least one hidden layer can solve any type of classification task involving nonlinear decision surfaces





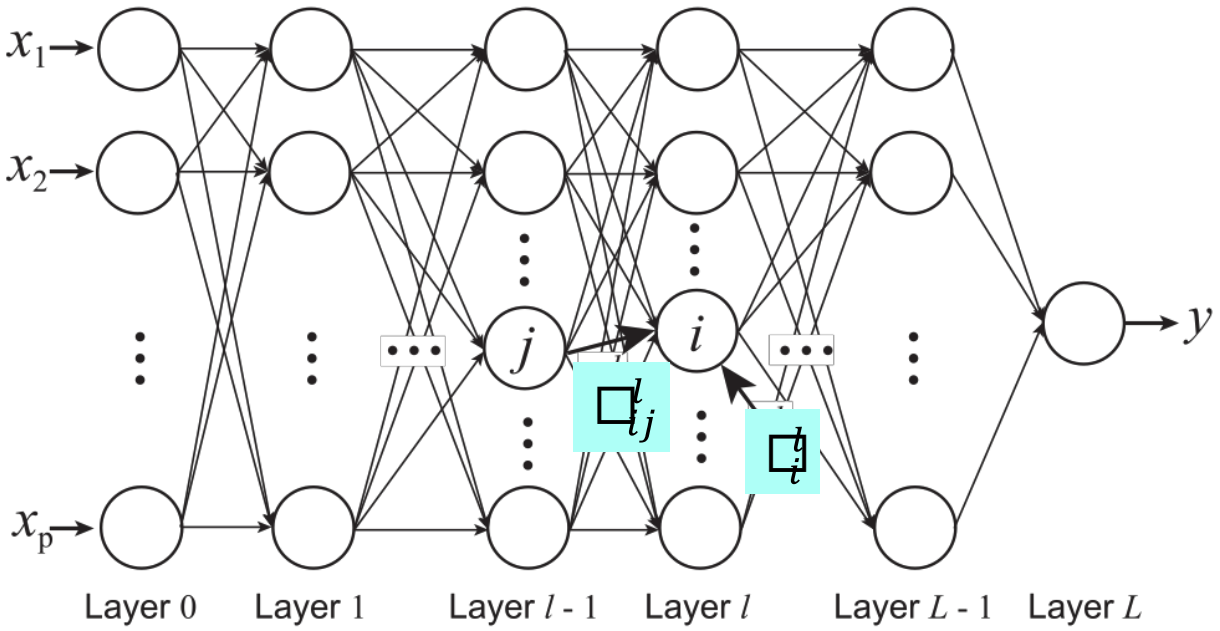
Why Multiple Hidden Layers?

- Activations at hidden layers can be viewed as features extracted as functions of inputs
- Every hidden layer represents a level of abstraction
  - *Complex features are compositions of simpler features*



- Number of layers is known as **depth** of ANN
  - *Deeper networks express complex hierarchy of features*

Multi-Layer Network Architecture



$$a_i^l = f(z_i^l) = f\left(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\right)$$

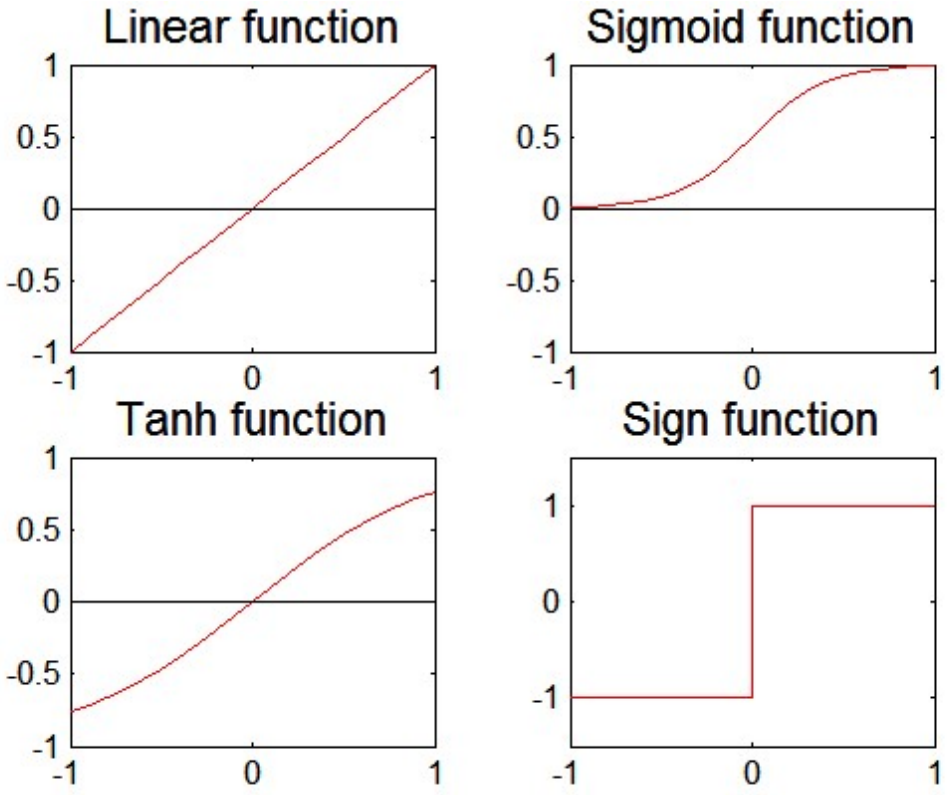
Activation value at node i at layer l

Activation Function

Linear Predictor

Activation Functions

$$a_i^l = f(z_i^l) = f\Big(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\Big)$$



$$a_i^l = \sigma(z_i^l) = \frac{1}{1 + e^{-z_i^l}}.$$

$$\frac{\partial a_i^l}{\partial z_i^l} = \frac{\partial\ \sigma(z_i^l)}{\partial z_i^l} = a_i^l(1 - a_i^l)$$

Learning Multi-layer Neural Network

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term e = y - $\hat{y}$ and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Gradient Descent

- Loss Function to measure errors across all training points

$$E(\mathbf{w}, \mathbf{b}) = \sum_{k=1}^{n} \text{Loss } (y_k,\ \hat{y}_k)$$

Squared Loss:

$$\text{Loss } (y_k,\ \hat{y}_k) = (y_k - \hat{y}_k)^2$$

- Gradient descent: Update parameters in the direction of "maximum descent" in the loss function across all points

$$w_{ij}^l \longleftarrow w_{ij}^l - \lambda \frac{\partial E}{\partial w_{ij}^l},$$

$$b_i^l \longleftarrow b_i^l - \lambda \frac{\partial E}{\partial b_i^l},$$

λ: learning rate

- Stochastic gradient descent (SGD): update the weight for every instance (minibatch SGD: update over min-batches of instances)

Computing Gradients

$$\frac{\partial E}{\partial w_{ij}^l} = \sum_{k=1}^{n} \frac{\partial \text{ Loss } (y_k,\ \hat{y_k})}{\partial w_{ij}^l}.$$

$$\hat{y} = a^L$$

$$a_i^l = f(z_i^l) = f\Big(\sum_j w_{ij}^l a_j^{l-1} + b_i^l\Big)$$

- Using chain rule of differentiation (on a single instance):

$$\frac{\partial \text{ Loss}}{\partial w_{ij}^l} = \frac{\partial \text{ Loss}}{\partial a_i^l} \times \frac{\partial a_i^l}{\partial z_i^l} \times \frac{\partial z_i^l}{\partial w_{ij}^l}.$$

- For sigmoid activation function:

$$\frac{\partial \text{ Loss}}{\partial w_{ij}^l} = \delta_i^l \times a_i^l(1 - a_i^l) \times a_j^{l-1},$$

$$\text{where } \delta_i^l = \frac{\partial \text{ Loss}}{\partial a_i^l}.$$

- How can we compute δ_i^l for every layer?

Backpropagation Algorithm

- At output layer L:

$$\delta^L = \frac{\partial \text{ Loss}}{\partial a^L} = \frac{\partial \ (y - a^L)^2}{\partial a^L} = 2(a^L - y).$$

- At a hidden layer l (using chain rule):

$$\delta_j^l = \sum_i (\delta_i^{l+1} \times a_i^{l+1}(1 - a_i^{l+1}) \times w_{ij}^{l+1}).$$

  - Gradients at layer l can be computed using gradients at layer l + 1
  - Start from layer L and "backpropagate" gradients to all previous layers

- Use gradient descent to update weights at every epoch
- For next epoch, use updated weights to compute loss fn. and its gradient
- Iterate until convergence (loss does not change)

Design Issues in ANN

- Number of nodes in input layer
  - One input node per **binary/continuous** attribute
  - k or $\log_2$ k nodes for each **categorical** attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $\log_2$ k nodes for k-class problem
- Number of hidden layers and nodes per layer
- Initial weights and biases
- Learning rate, max. number of epochs, mini-batch size for mini-batch SGD, …

Characteristics of ANN

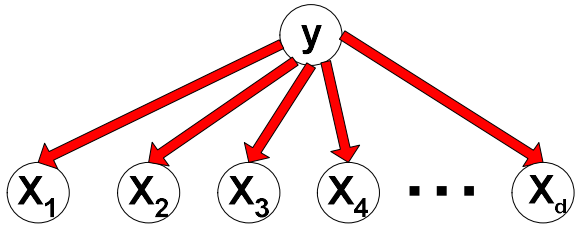
- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large
  - Naturally represents a hierarchy of features at multiple levels of abstractions
- Gradient descent may converge to local minimum
- Model building is compute intensive, but testing is fast
- Can handle redundant and irrelevant attributes because weights are automatically learnt for all attributes
- Sensitive to noise in training data
  - This issue can be addressed by incorporating model complexity in the loss function
- Difficult to handle missing attributes

Deep Learning Trends

- Training **deep** neural networks (more than 5-10 layers) could only be possible in recent times with:
  - Faster computing resources (GPU)
  - Larger labeled training sets
- Algorithmic Improvements in Deep Learning
  - Responsive activation functions (e.g., RELU)
  - Regularization (e.g., Dropout)
  - Supervised pre-training
  - Unsupervised pre-training (auto-encoders)
- Specialized ANN Architectures:
  - Convolutional Neural Networks (for image data)
  - Recurrent Neural Networks (for sequence data)
  - Residual Networks (with skip connections)
- Generative Models: Generative Adversarial Networks

Bayesian Classifiers - Naïve Bayes

- Bayesian Classifiers - Naïve Bayes

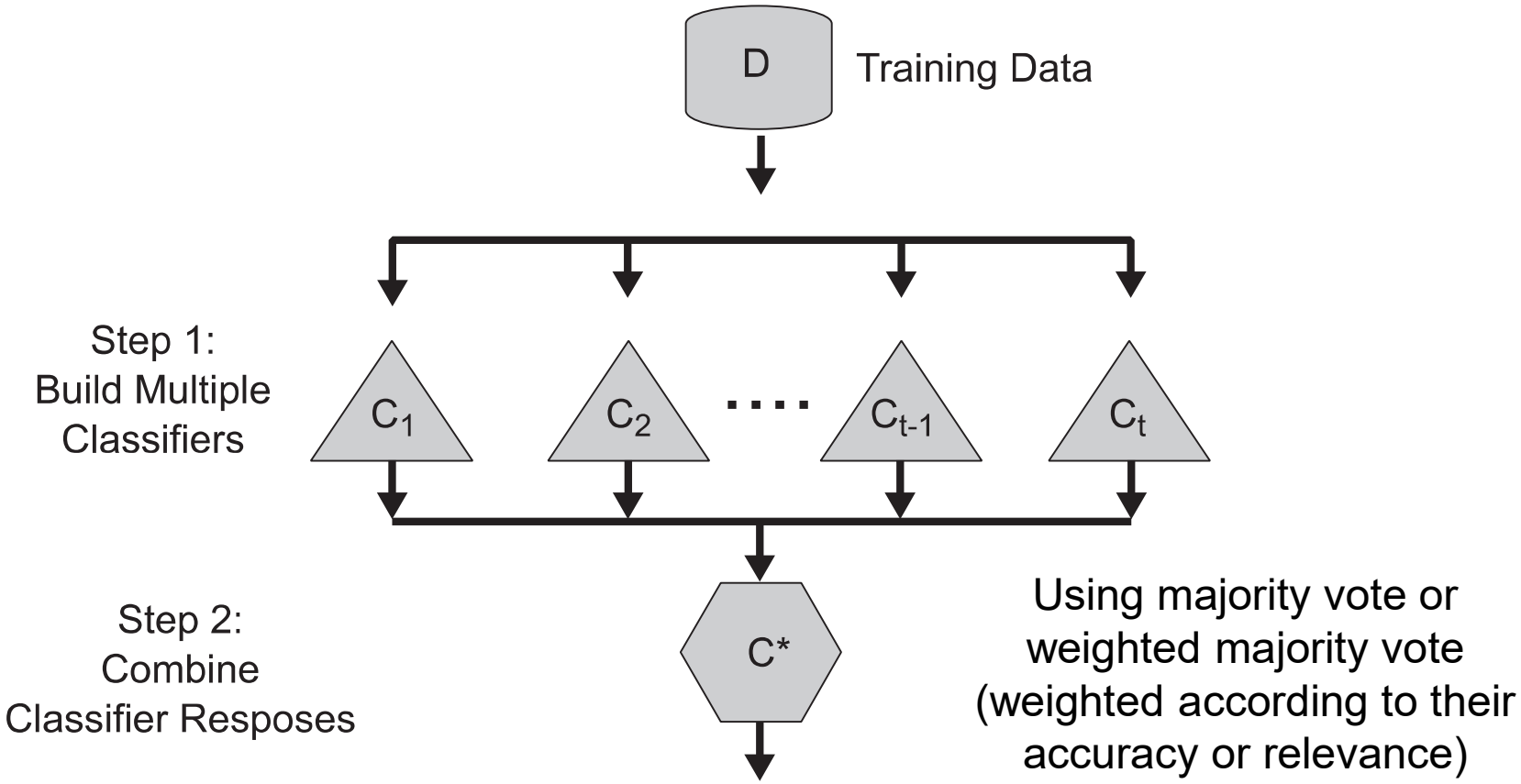


$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)}$$

$$P(Y \mid X_1 X_2 \ldots X_n) = \frac{P(X_1 X_2 \ldots X_d \mid Y)P(Y)}{P(X_1 X_2 \ldots X_d)}$$

  - Characteristics:
    - Robust to isolated noise points
    - Handle missing values by ignoring the instance during probability estimate calculations
    - Robust to irrelevant attributes
    - Redundant and correlated attributes will violate class conditional assumption
      - Use Bayesian Belief Networks (BBN)

Ensemble Learning - Classifiers



Ensemble Classifiers:

- Boosting, Bagging, Random Forests

Summary

- Classification
- Overfitting
- Model Selection
- Model Evaluation
- Decision Tree based Methods
- Nearest-Neighbor
- Naïve Bayes
- Support Vector Machines
- Neural Networks, Deep Neural Networks
- Ensemble Learning
- Conclusions

Bibliography

- Tan, P., Steinbach, M., Karpatne, A. & Kumar, V. (2019). Introduction to Data Mining, 2nd Ed Pearson Addison-Wesley.
- Tan, P., Steinbach, M., Karpatne, A. & Kumar, V. (2020) Introduction to Data Mining Slides, http://www-users.cs.umn.edu/~kumar/dmbook/index.php
- Brígida Mónica Faria, (2018), Slides de Extração de Conhecimento, Instituto Politécnico do Porto, 2018
- Gladys Castillo (2008), Slides de Aprendizagem Computacional (Machine Learning), Universidade de Aveiro, 2008
- RapidMiner: Data Science Platform, (2017), https://rapidminer.com/
- Towards Data Science, (2020), https://towardsdatascience.com/
- DataCamp, https://campus.datacamp.com/courses/data-science-for-everyone/
- Pandas Website, https://pandas.pydata.org/
- Scikit-Learn Website, https://scikit-learn.org/
- MatPlotLib Website, https://matplotlib.org
- Seaborn Website, https://seaborn.pydata.org/

U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Artificial Intelligence/ Inteligência Artificial

Lecture 5d: Supervised Learning - Classification

(adapted from Tan et al, 2020)

Luís Paulo Reis

lpreis@fe.up.pt

Director of LIACC – Artificial Intelligence and Computer Science Lab.
Associate Professor at DEI/FEUP – Informatics Engineering Department,
Faculty of Engineering of the University of Porto, Portugal
President of APPIA – Portuguese Association for Artificial Intelligence