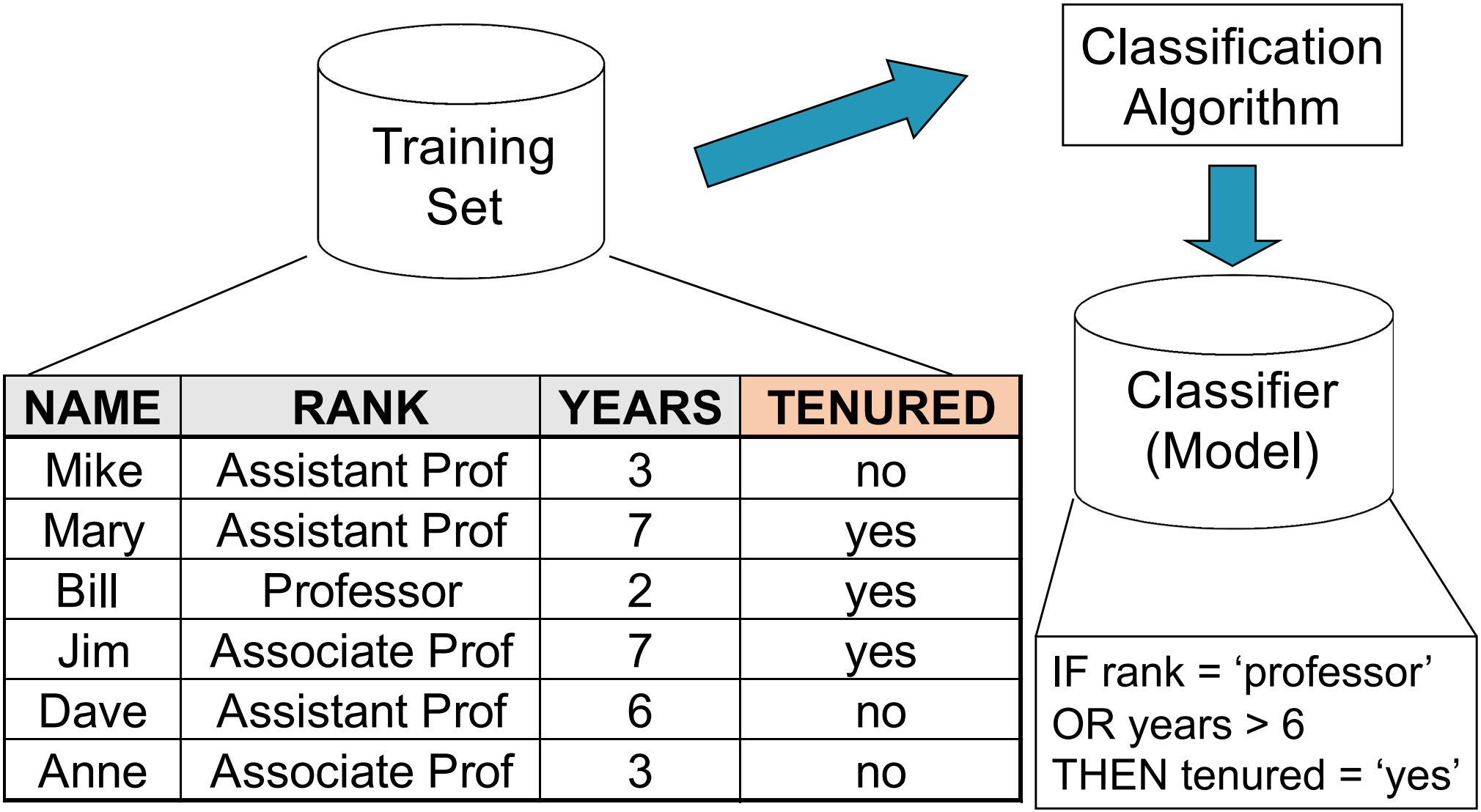
Classification

Man-Kwan Shan
Dept. of Computer Science
National Cheng-Chi Univ.

Classification

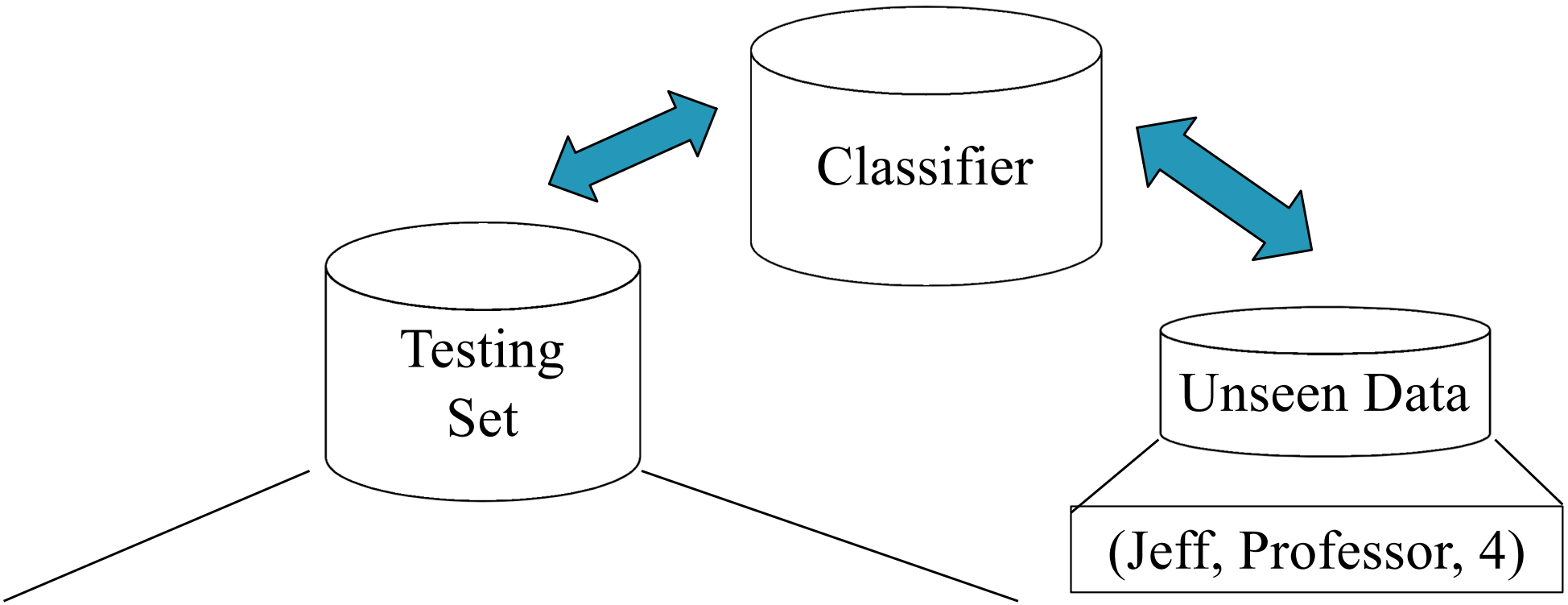
- Classification
  - Classify objects into categories.
  - Given an unseen object, determine its class.
  - Approaches
    - Knowledge driven
      - Classification rules (knowledge) are specified by human
      - Expert System
    - Data driven
      - Classification rules are learned from training data
      - Machine Learning

Relational Data Classification : Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Relational Data Classification: Model Usage



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Tenured?

yes

3C Shopping Mall

age	income	student	credit_rating	buys_computer
≤30	high	no	fair	no
≤30	high	no	excellent	no
30…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
≤30	medium	no	fair	no
≤30	low	yes	fair	yes
>40	medium	yes	fair	yes
≤30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Decision Tree for "*buys_computer*"

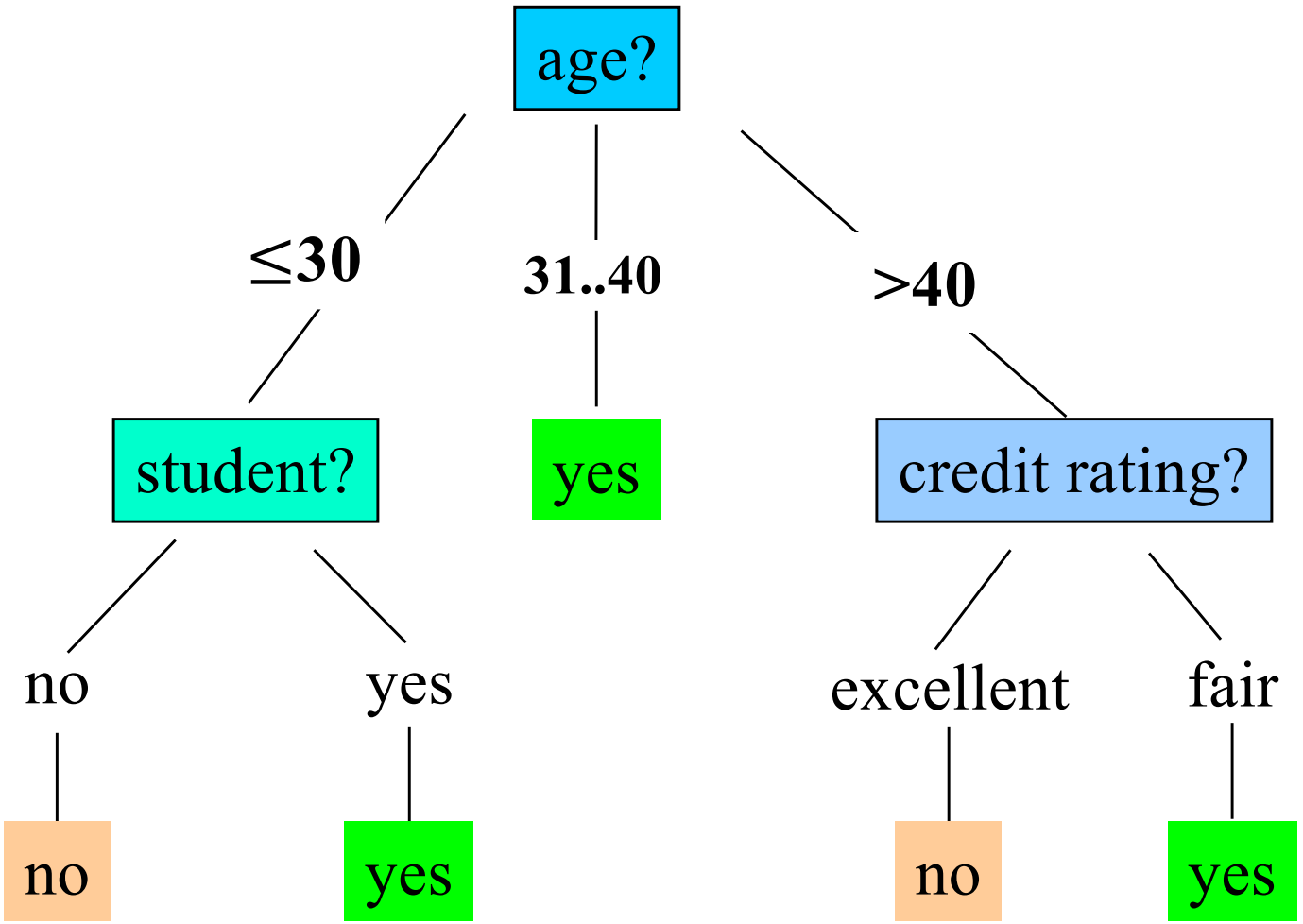
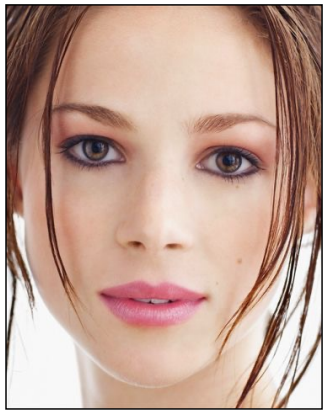


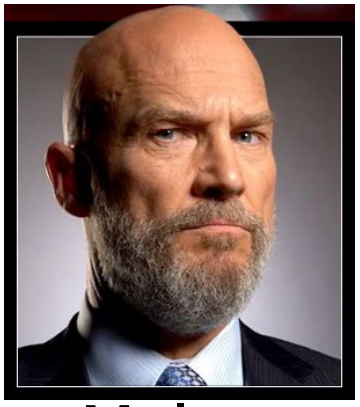
Image Gender Classification

Male or Female ?

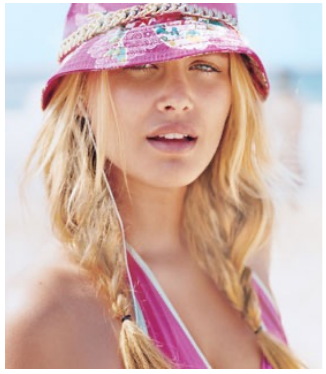


Female

Male

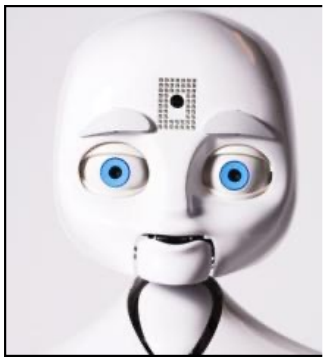


Male



Female

Female



Male

Image Feature Extraction

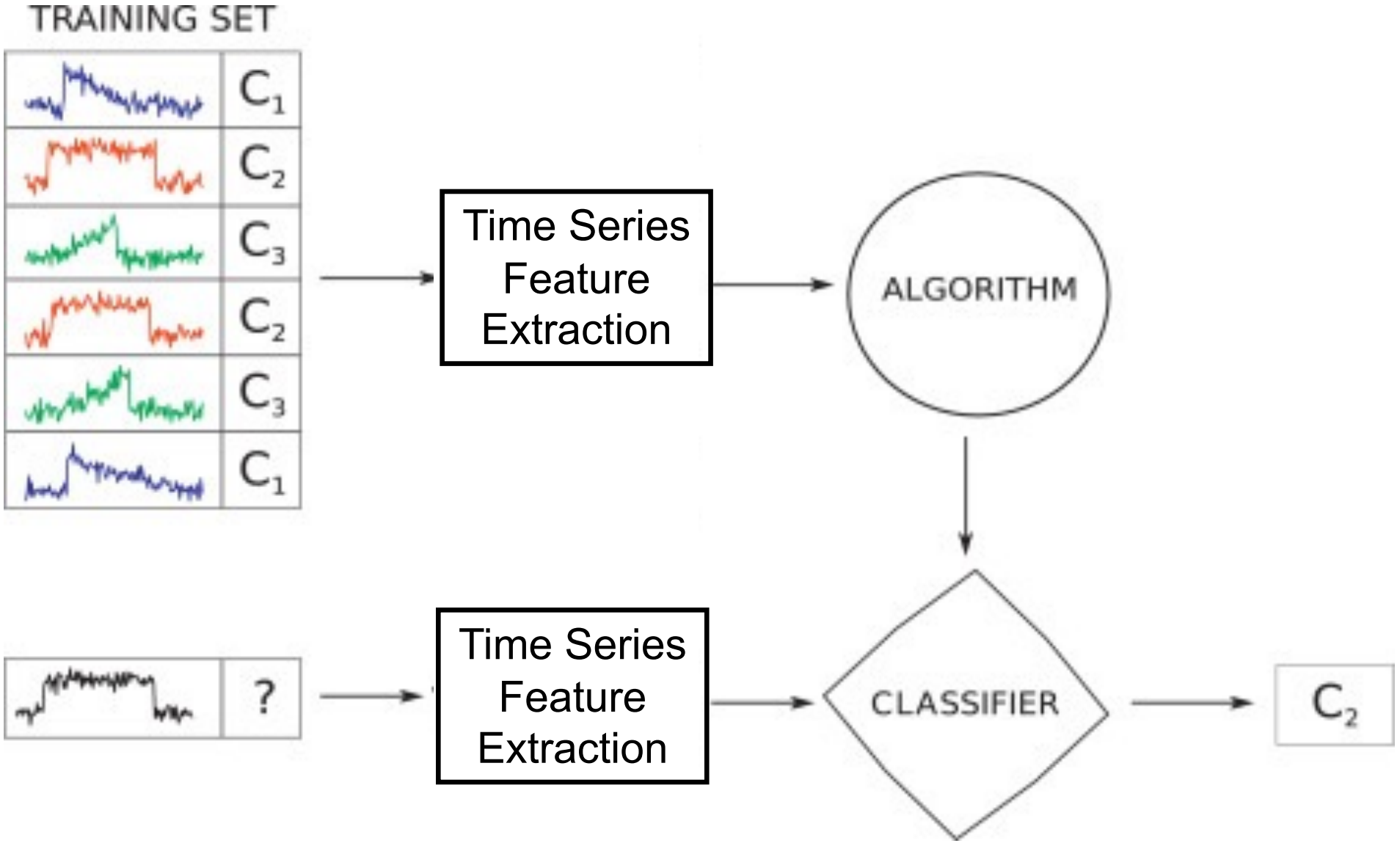
Classification Algorithm

Image Feature Extraction

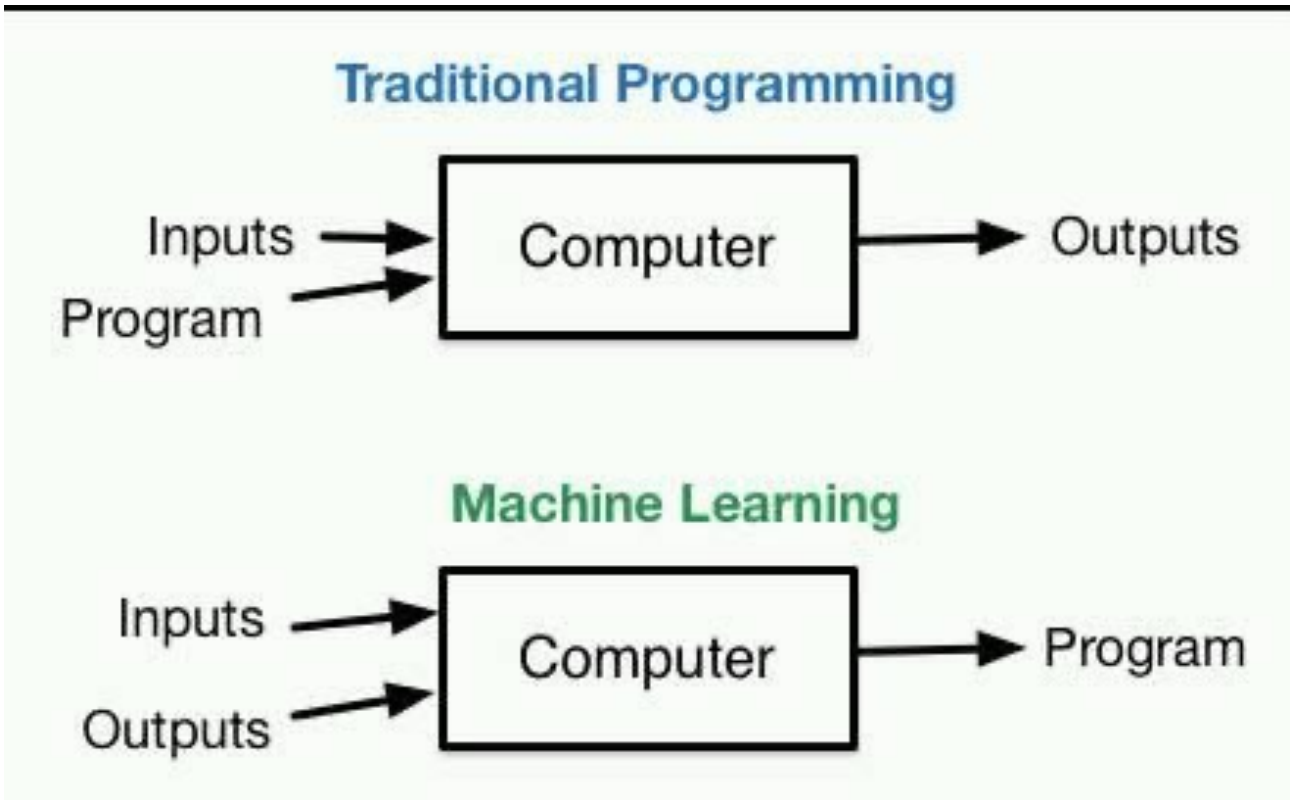
Classifier (Model)

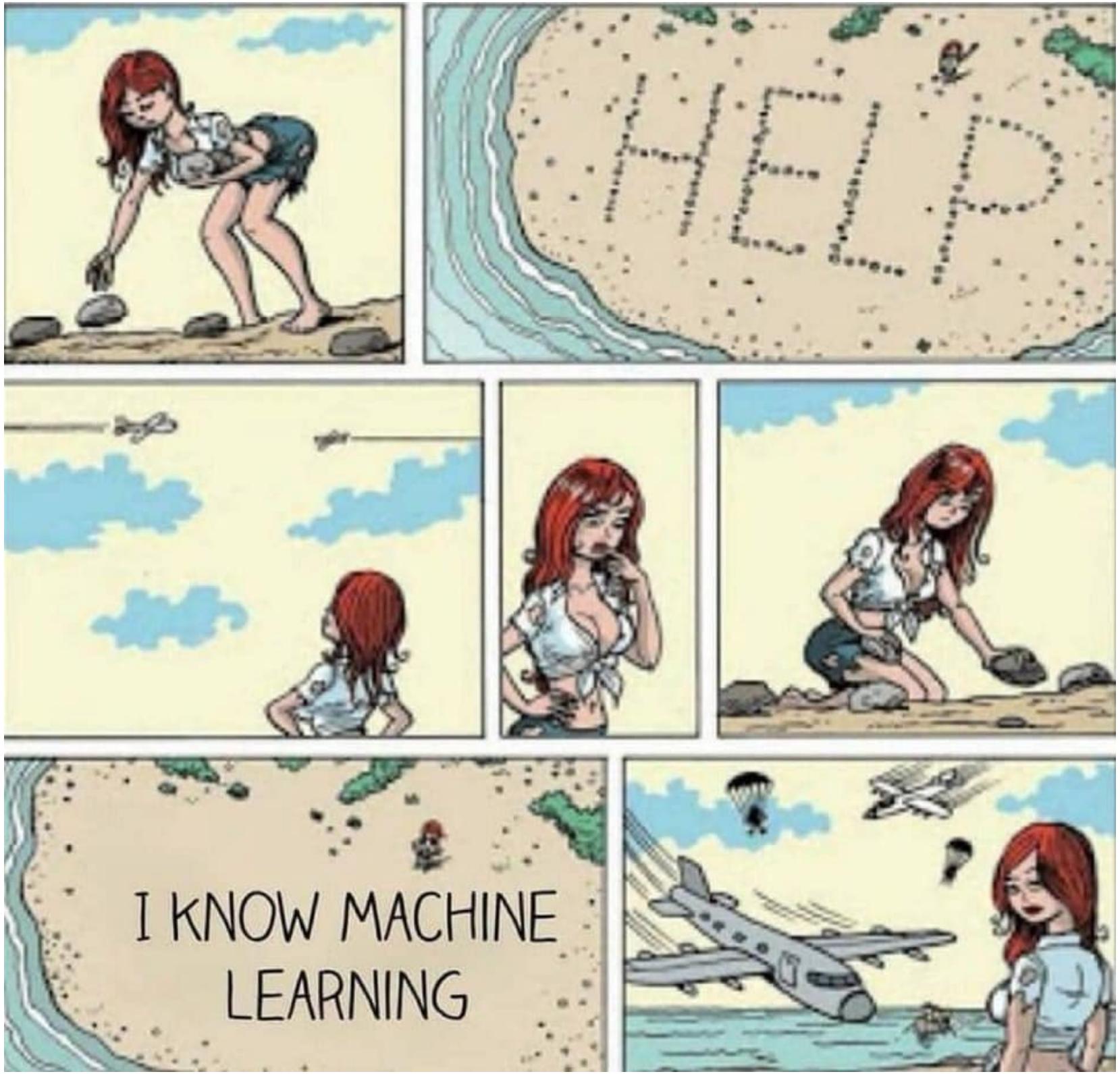
Female

Time Series Classification



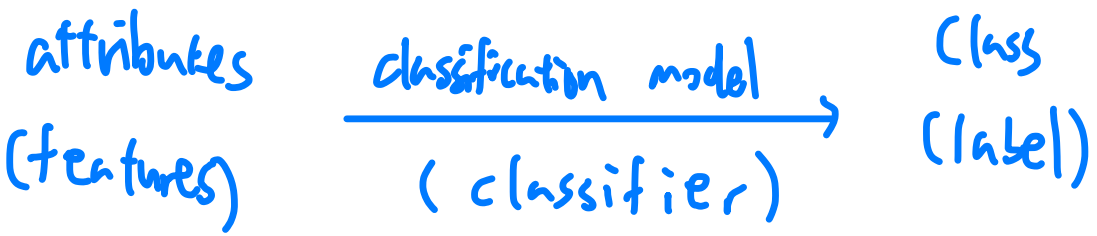
Machine Learning




HELP
I KNOW MACHINE LEARNING

Classification

- Given a collection of records (training set)
  - Each record contains a set of attributes (features), one of the attributes is the class (label).
- Find a classification model for class attribute as a **function** of the values of other attributes.
- Goal
  - previously unseen records should be assigned a class as accurately as possible.
  - A test set is used to determine the accuracy of the model.
  - Usually, the given data set (known label) is divided into
    - training set (used to build the model) &
    - test set (used to validate the model).

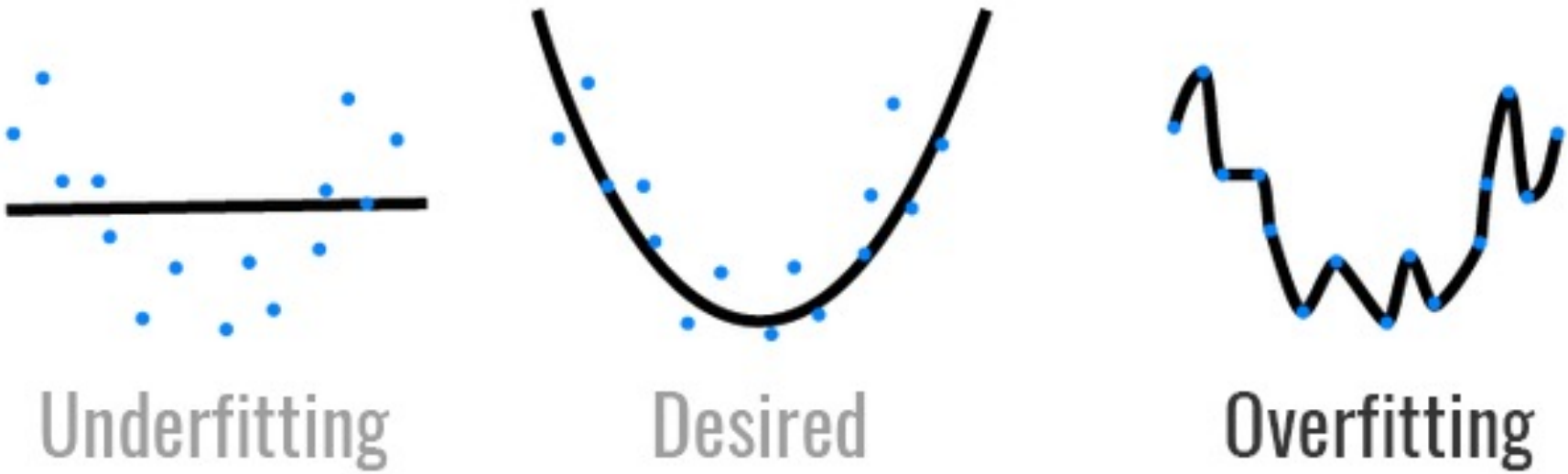


Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class attribute (label attribute)
  - The set of tuples used for model construction: training set
  - The model is represented as classification rules, decision trees, or mathematical models.
  - Estimate accuracy of the model
    - The known label of test set is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
- Model usage: for classifying unseen objects

Overfitting

Overfitting



Supervised vs. Unsupervised Learning

- Classification: Supervised Learning
  - The training data are accompanied by labels indicating the class of the observations
  - Learning classification model from the training data
- Clustering : Unsupervised Learning
  - The class labels of training data is unknown
  - Grouping training data based on similarity, proximity, density

Approaches of Classification Model

- Decision Tree (ID3, C4.5)
- Rule-based Classifier
- Associative Classifier
- Bayesian Network (based on Bayesian Theory)
- Nearest–Neighbor Classifier
- Support Vector Machine
- Ensemble (Ada boost, Random Forest, XGBoost)
- Deep Learning approach
- …

Classification by Decision Tree Induction

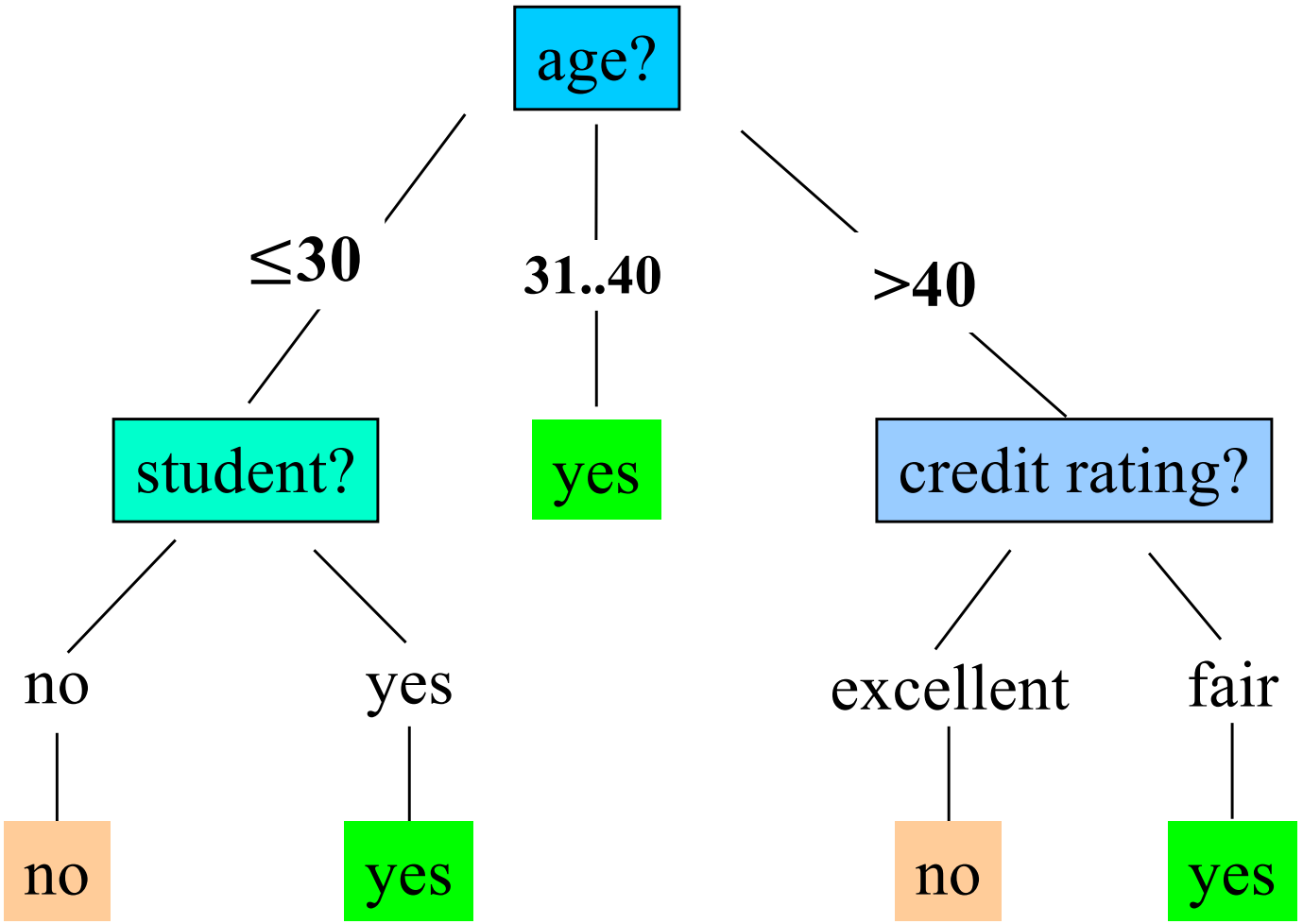
Classification by Decision Tree Induction

- Decision tree
  - A flow-chart-like tree structure
  - Internal node denotes a test on an attribute
  - Branch represents an outcome of the test
  - Leaf nodes represent class labels or class distribution
- Decision tree generation consists of two phases
  - Tree construction
    - At start, all the training examples are at the root
    - Partition examples recursively based on selected attributes
  - Tree pruning
    - Identify and remove branches that reflect noise or outliers
- Use of decision tree: Classifying an unknown example
  - Test the attribute values of the example against the decision tree

Training Dataset

age	income	student	credit_rating	buys_computer
≤30	high	no	fair	no
≤30	high	no	excellent	no
30…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
≤30	medium	no	fair	no
≤30	low	yes	fair	yes
>40	medium	yes	fair	yes
≤30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

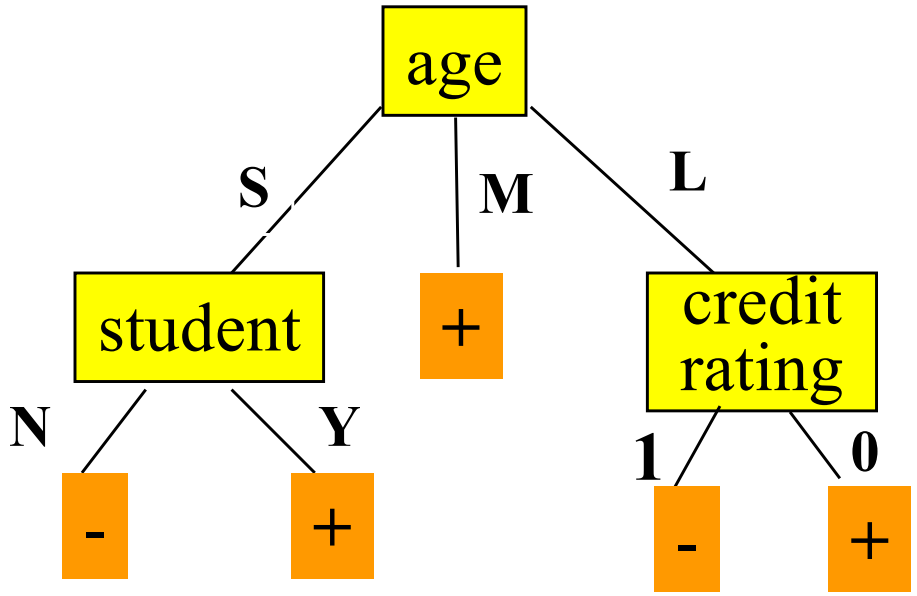
Output: A Decision Tree for "*buys_computer*"



Decision Tree

Training Set

age	income	student	credit rating	buys computer
S	L	N	0	-
S	L	N	1	-
M	L	N	0	+
L	M	N	0	+
L	S	Y	0	+
L	S	Y	1	-
M	S	Y	1	+
S	M	N	0	-
S	S	Y	0	+
L	M	Y	0	+
S	M	Y	1	+
M	M	N	1	+
M	L	Y	0	+
L	M	N	1	-



age	income	student	credit rating	buys computer
L	L	Y	0	?

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on tested attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Attribute Selection Measure

- Information gain (ID3/C4.5)
  - All attributes are assumed to be categorical
  - Can be modified for continuous-valued attributes
- Gini index (IBM IntelligentMiner)
  - All attributes are assumed continuous-valued
  - Assume there exist several possible split values for each attribute
  - May need other tools, such as clustering, to get the possible split values
  - Can be modified for categorical attributes

Some Observations

income	student	buys_computer
high	no	no
high	no	no
high	no	no
medium	no	no
low	yes	yes
low	yes	yes
low	yes	yes
medium	no	no
low	yes	yes
medium	yes	yes
medium	yes	yes
medium	no	no
high	yes	yes
medium	no	no

buy student	Y	N
Y	7/7	0/7
N	0/7	7/7

Some Observations (cont.)

income	student	buys_computer
high	no	yes
high	no	yes
high	no	yes
medium	no	yes
low	yes	no
low	yes	no
low	yes	no
medium	no	yes
low	yes	no
medium	yes	no
medium	yes	no
medium	no	yes
high	yes	no
medium	no	yes

buy student	Y	N
Y	0/7	7/7
N	7/7	0/7

Some Observations (cont.)

income	student	buys_computer
high	no	no
high	no	no
high	yes	no
medium	yes	no
low	yes	yes
low	yes	yes
low	yes	yes
medium	no	no
low	yes	yes
medium	yes	yes
medium	yes	yes
medium	yes	no
high	yes	yes
medium	no	no

buy student	Y	N
Y	7/10	3/10
N	0/4	4/4

Some Observations (cont.)

income	student	buys_computer
high	no	no
high	no	no
high	yes	no
medium	yes	no
low	yes	no
low	yes	yes
low	yes	yes
medium	no	no
low	yes	yes
medium	yes	yes
medium	yes	no
medium	yes	no
high	yes	yes
medium	no	no

buy student	Y	N
Y	5/10	5/10
N	0/4	4/4

Some Observations (cont.)

income	student	buys_computer
high	no	yes
high	no	yes
high	yes	no
medium	yes	no
low	yes	no
low	yes	yes
low	yes	yes
medium	no	no
low	yes	yes
medium	yes	yes
medium	yes	no
medium	yes	no
high	yes	yes
medium	no	no

buy student	Y	N
Y	5/10	5/10
N	2/4	2/4

Entropy

- Proposed by Claude E. Shannon of Bell Labs.
- Entropy of message *M* with alphabet {m_1, m_2, …,m_n}

$$E(M) = -\sum_i p(m_i)\log_2 p(m_i)$$ (bits)

where $p(m_i)$ is the probability that m_i in *M* will occur

- e.g. Given *p*(male)=1/2, *p*(female)=1/2

$$E(M) = -(\frac{1}{2} * (\log \frac{1}{2}) + \frac{1}{2} * (\log \frac{1}{2})) = \frac{1}{2} + \frac{1}{2} = 1$$

But if p(male)=2/3, p(female)=1/3 ?

$$E(M) = -(-\tfrac{2}{3} * (\log \tfrac{2}{3}) + -\tfrac{1}{3} * (\log \tfrac{1}{3}))$$
$$= -(0.39 + 0.53) = 0.92$$

Entropy (cont.)

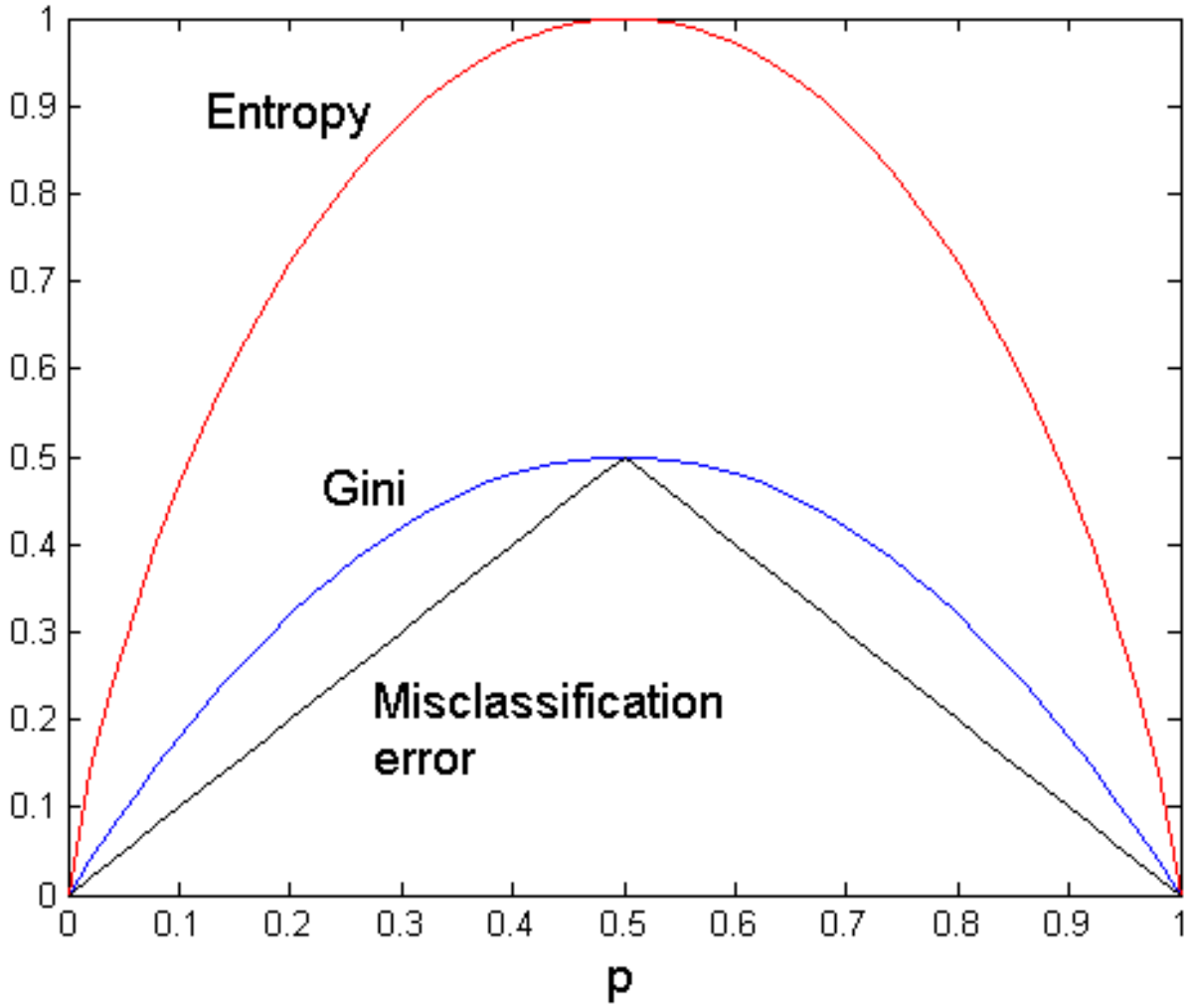
- Entropy in science
  - A measure of the disorder of a system
  - The more entropy, the more energy uncertainty
- In general, the entropy is greater when the probability distribution is flat & smaller when it is more peaked

When $p = 0.5$, both outcomes are equally likely. That is to say, the uncertainty level peaks.

Comparison among Splitting Criteria

Gini Impurity = $1 - \sum (p(i)^2) \equiv 1 - \sum (p(i)(1-p(i)))$

For a 2-class problem:



information gain ↖ ↗ gain ratio

Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Assume there are two classes, *P* and *N*
  - Let the set of examples *S* contain *p* elements of class *P* and *n* elements of class *N*
  - The amount of information, needed to decide if an arbitrary example in *S* belongs to *P* or *N* is defined as

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

S { p elements of class P
  n " N

2. Limitation of Information Gain

Attributes with many unique values (e.g., unique IDs, timestamps) tend to have high Information Gain because they perfectly split the data, even if they don't provide meaningful patterns. This leads to overfitting.

---

3. Gain Ratio Formula

To address this bias, **Gain Ratio** normalizes Information Gain by dividing it by a value called **Split Information** (SI):

$$\text{Gain Ratio (GR)} = \frac{\text{Information Gain}}{\text{Split Information}}$$

Where:

$$\text{Split Information (SI)} = -\sum_{i=1}^{k} \frac{|S_i|}{|S|} \cdot \log_2 \left(\frac{|S_i|}{|S|} \right)$$

- $|S|$: Total number of instances in the dataset.
- $|S_i|$: Number of instances in subset S_i after splitting by the attribute.
- k: Number of subsets created by the split.

Information Gain in Decision Tree Induction

- Assume that using attribute A a set *S* will be partitioned into sets $\{S_1, S_2, \ldots, S_v\}$
  - If S_i contains p_i examples of *P* and n_i examples of *N*, the entropy, or the expected information (impurity) needed to classify objects in all subtrees S_i is

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

- The encoding information that would be gained by branching on *A* （difference of impurity after splitting *A*)

$$Gain(A) = I(p, n) - E(A)$$

Training Dataset

age	income	student	credit_rating	buys_computer
≤30	high	no	fair	no
≤30	high	no	excellent	no
30…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
≤30	medium	no	fair	no
≤30	low	yes	fair	yes
>40	medium	yes	fair	yes
≤30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute Selection by Information Gain Computation

= 0.94

$I(p,n) = I(9,5) = -\left(\frac{9}{14}\log\frac{9}{14} + \frac{5}{14}\log\frac{5}{14}\right)$

- Class *P*: buys_computer = "yes"
- Class *N*: buys_computer = "no"
- $I(P, N) = I(9, 5) = 0.940$
- Compute the entropy for *age*:

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.971 * \frac{10}{14}$$

≈ .694

$$Gain(age) = I(p,n) - E(age)$$

= 0.940-0.694 = 0.246

	age	p_i	n_i	$I(p_i, n_i)$
5	<=30	2	3	0.971
4	31…40	4	0	0.000
5	>40	3	2	0.971
14		9	5	

Similarly

Gain(*income*) = 0.940-0.911 = 0.029

Gain(*student*) = 0.940-0.789 = 0.151

~~*Gain*(*student*) = 0.940-0.892 = 0.048~~

Gain (credit rating) = .94 - .857 = 0.083

∵ Gain (age) > other ∴ choose "age" as the 1st splitting criterion

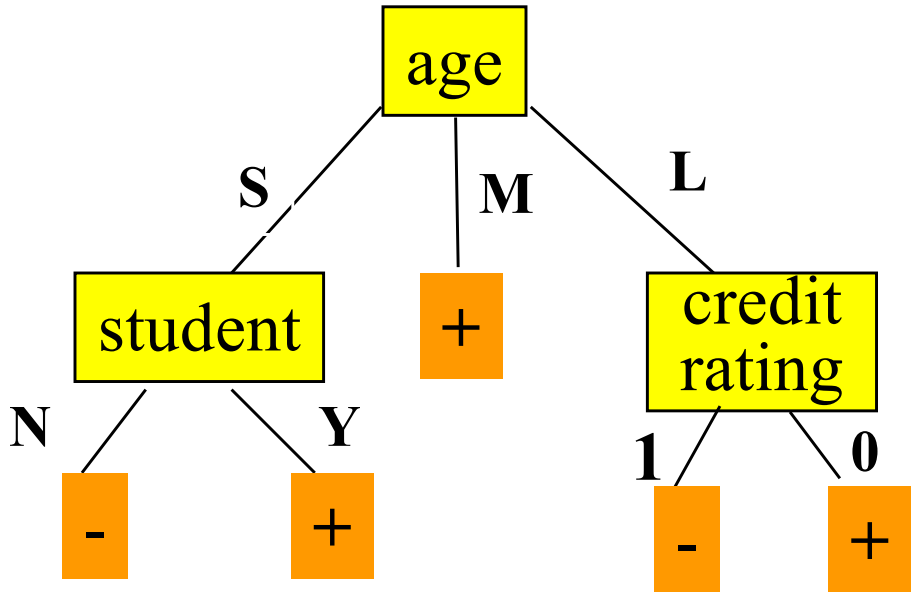
17 Equations That Changed the World
by Ian Stewart

1.	**Pythagoras's Theorem**	$a^2 + b^2 = c^2$	Pythagoras,530 BC
2.	**Logarithms**	$\log xy = \log x + \log y$	John Napier, 1610
3.	**Calculus**	$\frac{\mathrm{d}f}{\mathrm{d}t} = \lim_{h \to 0} = \frac{f(t+h) - f(t)}{h}$	Newton, 1668
4.	**Law of Gravity**	$F = G\frac{m_1 m_2}{r^2}$	Newton, 1687
5.	**The Square Root of Minus One**	$i^2 = -1$	Euler, 1750
6.	**Euler's Formula for Polyhedra**	$V - E + F = 2$	Euler, 1751
7.	**Normal Distribution**	$\Phi(x) = \frac{1}{\sqrt{2\pi\rho}} e^{\frac{(x-\mu)^2}{2\rho^2}}$	C.F. Gauss, 1810
8.	**Wave Equation**	$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$	J. d'Almbert, 1746
9.	**Fourier Transform**	$f(\omega) = \int_{\infty}^{\infty} f(x) e^{-2\pi i x \omega} \mathrm{d}\, x$	J. Fourier, 1822
10.	**Navier-Stokes Equation**	$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \nabla \cdot \mathbf{T} + \mathbf{f}$	C. Navier, G. Stokes, 1845
11.	**Maxwell's Equations**	$\nabla \cdot \mathbf{E} = 0$ $\quad \nabla \cdot \mathbf{H} = 0$ $\nabla \times \mathbf{E} = -\frac{1}{c}\frac{\partial \mathbf{H}}{\partial t}$ $\quad \nabla \times \mathbf{H} = \frac{1}{c}\frac{\partial E}{\partial t}$	J.C. Maxwell, 1865
12.	**Second Law of Thermodynamics**	$\mathrm{d}S \geq 0$	L. Boltzmann, 1874
13.	**Relativity**	$E = mc^2$	Einstein, 1905
14.	**Schrodinger's Equation**	$i\hbar\frac{\partial}{\partial t}\Psi = H\Psi$	E. Schrodinger, 1927
15.	**Information Theory**	$H = -\sum p(x) \log p(x)$	C. Shannon, 1949
16.	**Chaos Theory**	$x_{t+1} = k x_t (1 - x_t)$	Robert May, 1975
17.	**Black-Scholes Equation**	$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} - rV = 0$	F. Black, M. Scholes, 1990

Decision Tree

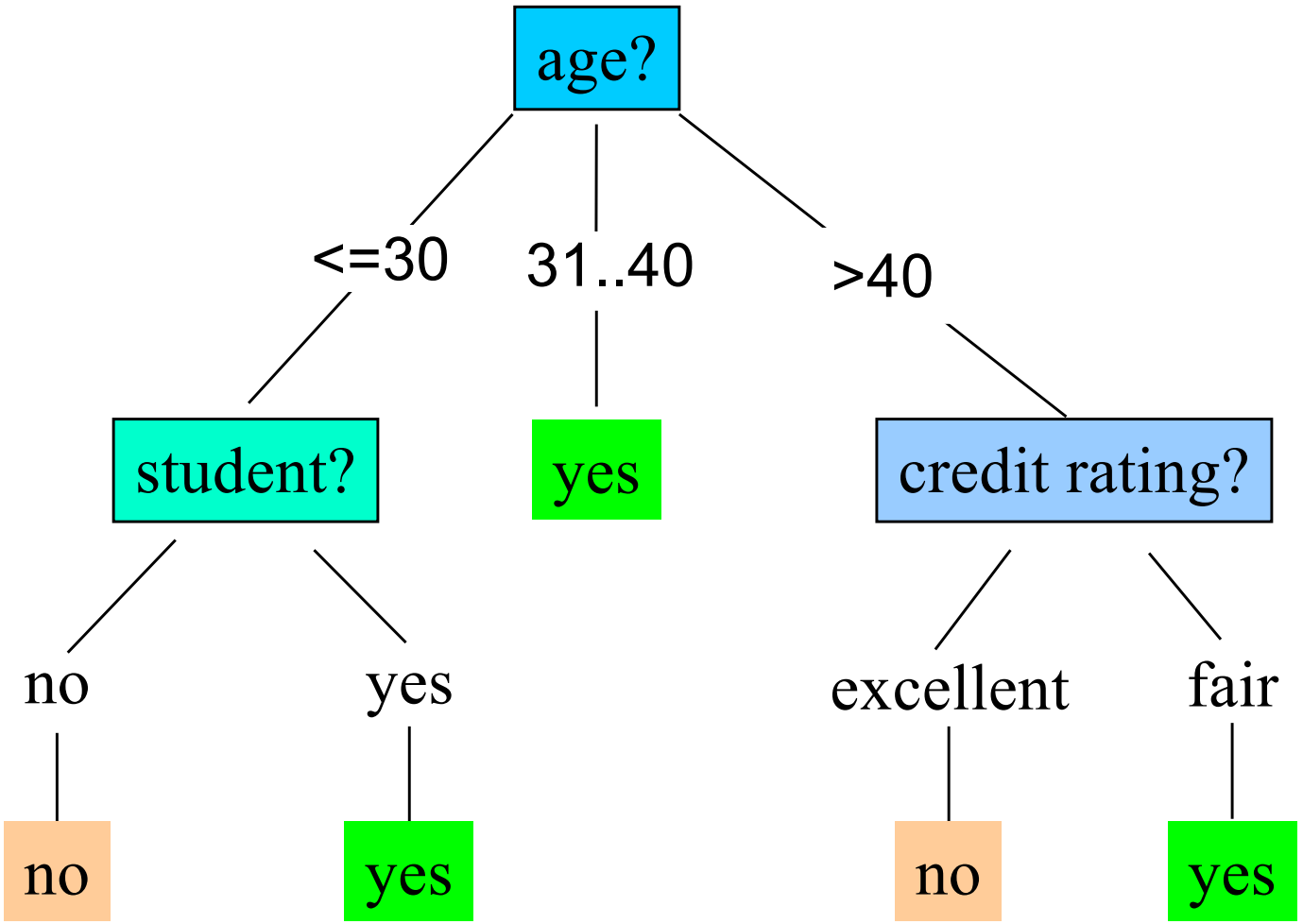
Training Set

age	income	student	credit rating	buys computer
S	L	N	0	-
S	L	N	1	-
M	L	N	0	+
L	M	N	0	+
L	S	Y	0	+
L	S	Y	1	-
M	S	Y	1	+
S	M	N	0	-
S	S	Y	0	+
L	M	Y	0	+
S	M	Y	1	+
M	M	N	1	+
M	L	Y	0	+
L	M	N	1	-



age	income	student	credit rating	buys computer
L	L	Y	0	?

Output: A Decision Tree for "*buys_computer*"



Extracting Classification Rules from Trees

- Represent the knowledge in the form of IF-THEN rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = "≤30" AND *student* = "*no*" **THEN** *buys_computer* = "*no*"

IF *age* = "≤ 30" AND *student* = "*yes*" **THEN** *buys_computer* = "*yes*"

IF *age* = "31…40" **THEN** *buys_computer* = "*yes*"

IF *age* = ">40" **AND** *credit_rating* = "*excellent*"

THEN *buys_computer* = "*yes*"

IF *age* = ">40" AND *credit_rating* = "*fair*"

THEN *buys_computer* = "*no*"

Rule-Based Classifier

Rule-Based Classifier

- Classify records by using a collection of "if…then…" rules
- Rule: (*Condition*) $\rightarrow$ *y*
  - where
    - *Condition* is a conjunctions of attributes
    - *y* is the class label
  - *LHS*: rule antecedent or condition
  - *RHS*: rule consequent
  - Examples of classification rules:
    - (Blood Type=Warm) $\wedge$ (Lay Eggs=Yes) $\rightarrow$ Birds
    - (Taxable Income < 50K) $\wedge$ (Refund=Yes) $\rightarrow$ Evade=No

Rule-based Classifier (Example)

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
human	warm	yes	no	no	mammals
python	cold	no	no	no	reptiles
salmon	cold	no	no	yes	fishes
whale	warm	yes	no	yes	mammals
frog	cold	no	no	sometimes	amphibians
komodo	cold	no	no	no	reptiles
bat	warm	yes	yes	no	mammals
pigeon	warm	no	yes	no	birds
cat	warm	yes	no	no	mammals
leopard shark	cold	yes	no	yes	fishes
turtle	cold	no	no	sometimes	reptiles
penguin	warm	no	no	sometimes	birds
porcupine	warm	yes	no	no	mammals
eel	cold	no	no	yes	fishes
salamander	cold	no	no	sometimes	amphibians
gila monster	cold	no	no	no	reptiles
platypus	warm	no	no	no	mammals
owl	warm	no	yes	no	birds
dolphin	warm	yes	no	yes	mammals
eagle	warm	no	yes	no	birds

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Application of Rule-Based Classifier

- A rule *r* covers an instance **x** if the attributes of the instance satisfy the condition of the rule

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
hawk	warm	no	yes	no	?
grizzly bear	warm	yes	no	no	?

The rule R1 covers a hawk => Bird

The rule R3 covers the grizzly bear => Mammal

Rule Coverage and Accuracy

- Coverage of a rule
  - Fraction of records that satisfy the antecedent of a rule
- Accuracy of a rule:
  - Fraction of records that satisfy both the antecedent and consequent of a rule

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

(Status=Single) → No

Coverage = 40%, Accuracy = 50%

How does Rule-based Classifier Work?

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
lemur	warm	yes	no	no	?
turtle	cold	no	no	sometimes	?
dogfish shark	cold	yes	no	yes	?

A lemur triggers rule R3, so it is classified as a mammal

A turtle triggers both R4 and R5

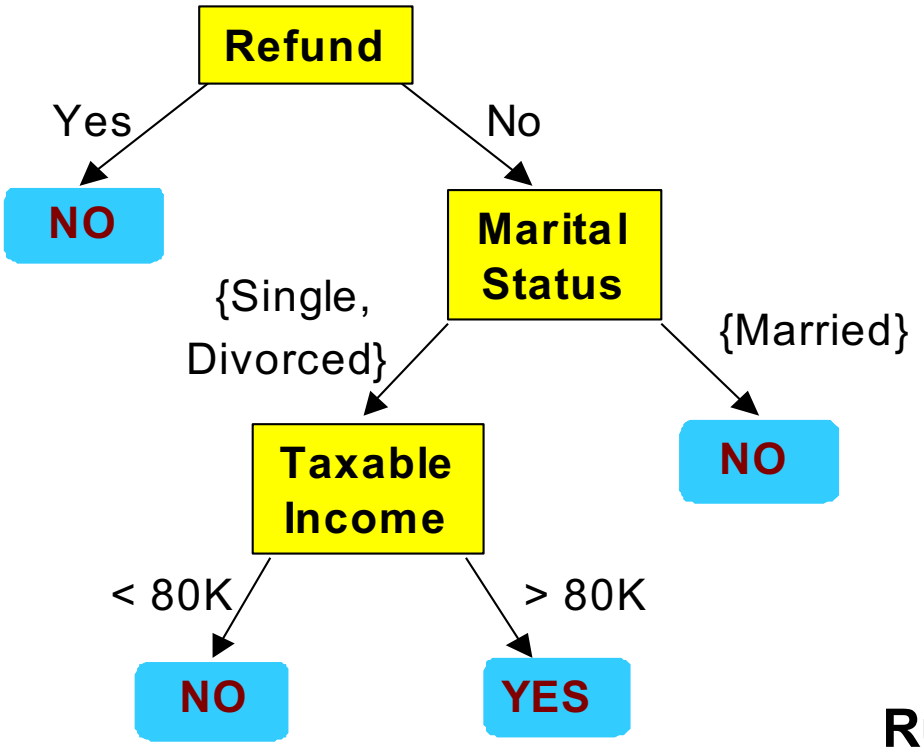
A dogfish shark triggers none of the rules

Characteristics of Rule-Based Classifier

- Mutually exclusive rules
  - Classifier contains mutually exclusive rules if the rules are independent of each other
  - Every record is covered by at most one rule

- Exhaustive rules
  - Classifier has exhaustive coverage if it accounts for every possible combination of attribute values
  - Each record is covered by at least one rule

From Decision Trees To Rules



Classification Rules

(Refund=Yes) ==> No

(Refund=No, Marital Status={Single,Divorced}, Taxable Income<80K) ==> No

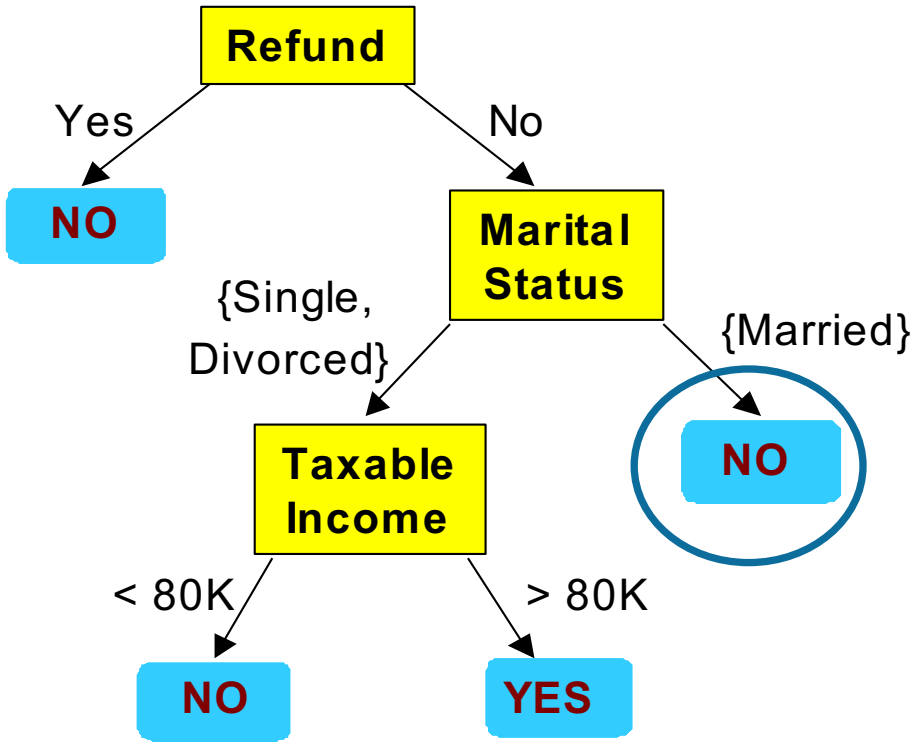
(Refund=No, Marital Status={Single,Divorced}, Taxable Income>80K) ==> Yes

(Refund=No, Marital Status={Married}) ==> No

Rules are mutually exclusive and exhaustive

Rule set contains as much information as the tree

Rules Can Be Simplified



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

Initial Rule: **(Refund=No) $\wedge$ (Status=Married) $\rightarrow$ No**

Simplified Rule: **(Status=Married) $\rightarrow$ No**

Effect of Rule Simplification

- Rules are no longer mutually exclusive
  - A record may trigger more than one rule
  - Solution?
    - Ordered rule set
    - Unordered rule set – use voting schemes

- Rules are no longer exhaustive
  - A record may not trigger any rules
  - Solution?
    - Use a default class

Ordered Rule Set

- Rules are rank ordered according to their priority
  - An ordered rule set is known as a decision list
- When a test record is presented to the classifier
  - It is assigned to the class label of the highest ranked rule it has triggered
  - If none of the rules fired, it is assigned to the default class

R1: (Give Birth = no) $\wedge$ (Can Fly = yes) $\rightarrow$ Birds

R2: (Give Birth = no) $\wedge$ (Live in Water = yes) $\rightarrow$ Fishes

R3: (Give Birth = yes) $\wedge$ (Blood Type = warm) $\rightarrow$ Mammals

R4: (Give Birth = no) $\wedge$ (Can Fly = no) $\rightarrow$ Reptiles

R5: (Live in Water = sometimes) $\rightarrow$ Amphibians

Name	Blood Type	Give Birth	Can Fly	Live in Water	Class
turtle	cold	no	no	sometimes	?

Building Classification Rules

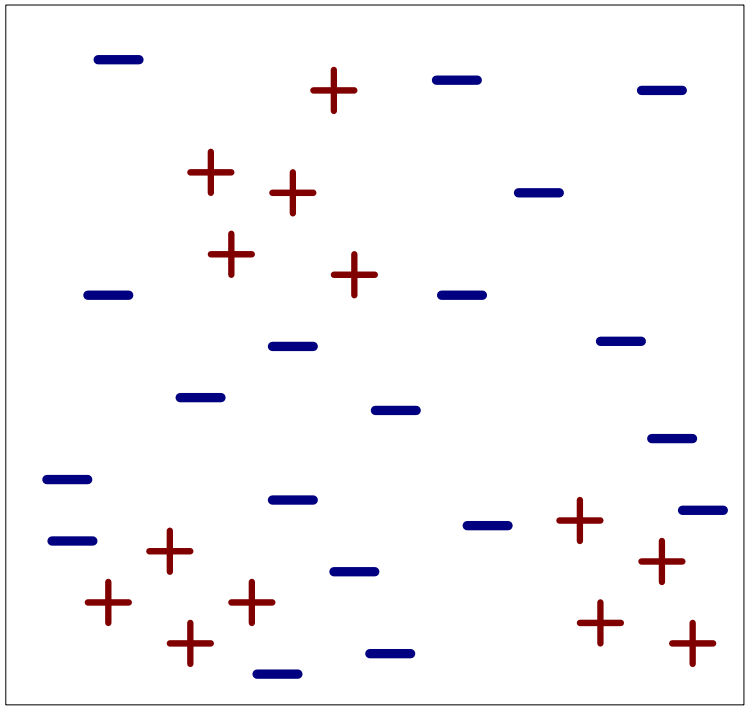
- Direct Method:
  - Extract rules directly from data
  - e.g.: Associated Classification, RIPPER, CN2, Holte's 1R

- Indirect Method:
  - Extract rules from other classification models (e.g. decision trees, neural networks, etc).
  - e.g: ID3 rules, C4.5 rules, CART

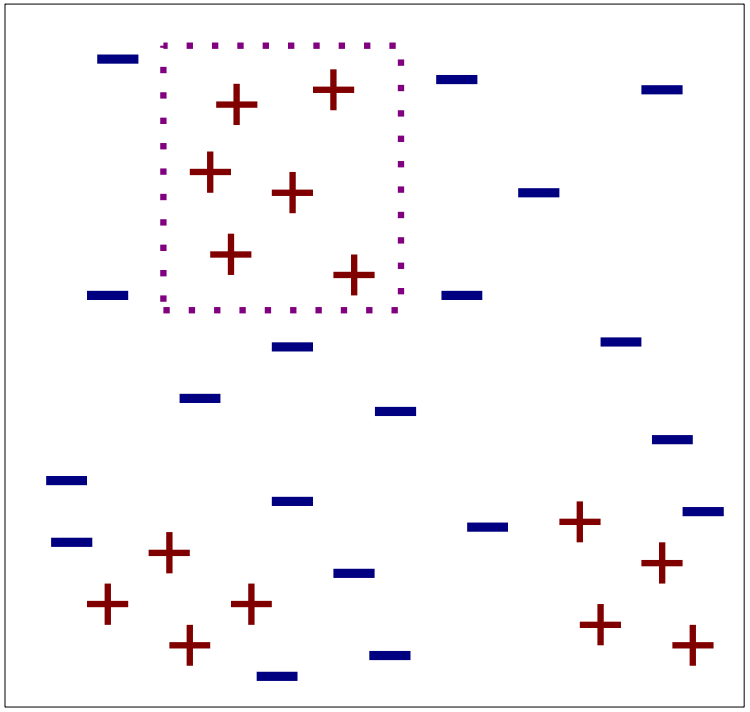
Direct Method: Sequential Covering

1. Start from an empty rule
2. Grow a rule using the Learn-One-Rule function
3. Remove training records covered by the rule
4. Repeat Step (2) and (3) until stopping criterion is met

Example of Sequential Covering

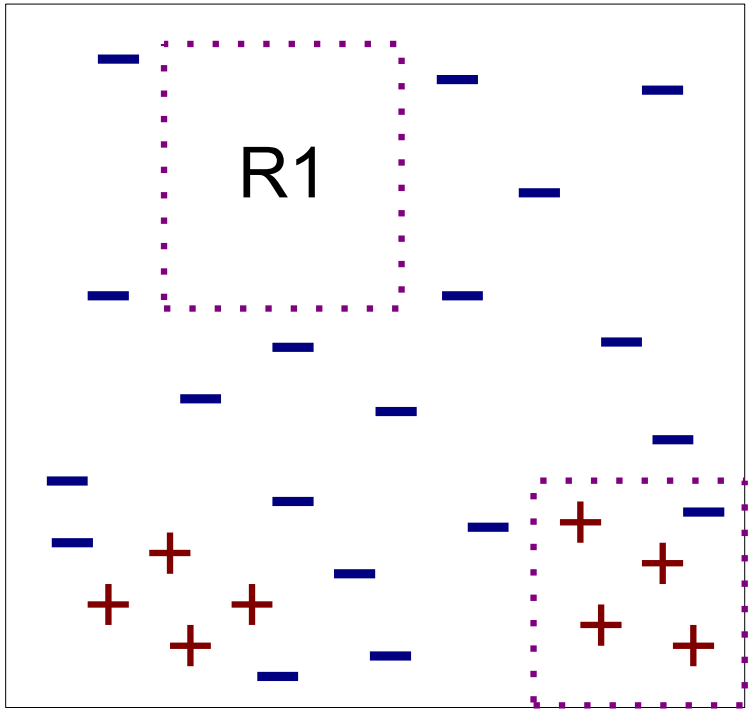


(i) Original Data

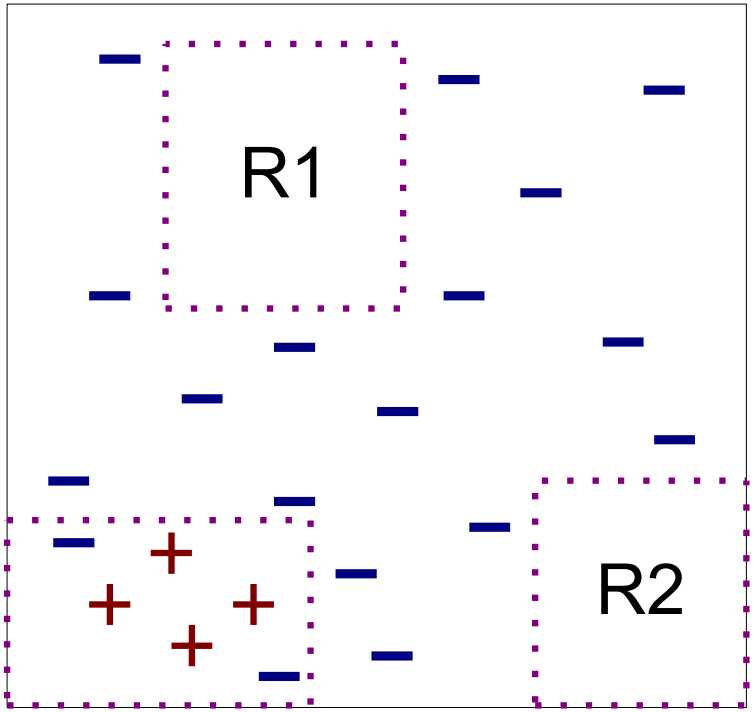


(ii) Step 1

Example of Sequential Covering…



(iii) Step 2



(iv) Step 3

Advantages of Rule-Based Classifiers

- As highly expressive as decision trees
- Easy to interpret
- Easy to generate
- Can classify new instances rapidly
- Performance comparable to decision trees

Associated Classifier

Classification Based on Association

Associated Classification CBA

Steps:

1. Discretizing continuous attributes, (if any)
2. Generating all Class Association Rules (CARs)
3. Building classifier based on the generated Class Association Rules

Association Rule Mining : Apriori, FP-Growth

(the consequent must always be a class label)

Algorithm of CBA classifier

incorrect indentation

```
1   R = sort(R);
2   for each rule r ∈ R in sequence do
3       temp = ∅;
4       for each case d ∈ D do
5           if d satisfies the conditions of r then
6               store d.id in temp and mark r if it correctly
                    classifies d;
7       if r is marked then
8           insert r at the end of C;
9           delete all the cases with the ids in temp from D;
10          selecting a default class for the current C;
11          compute the total number of errors of C;
12      end
13  end
14  Find the first rule p in C with the lowest total number
        of errors and drop all the rules after p in C;
15  Add the default class associated with p to end of C,
        and return C (our classifier).
```

Algorithm of CBA classifier

```
# Sort the rules R
R = sort(R)

# Initialize the classifier
for each rule r in R do:
    temp = ø

    # Check each case in the dataset D
    for each case d in D do:
        if d satisfies the conditions of r then:
            # Store the case ID in temp and mark the rule if it correctly classifies the case
            store d.id in temp
            mark r if it correctly classifies d

    # If the rule is marked, include it in the classifier (correctly classifies at least 1 record)
    if r is marked then:
        # Add the rule to the classifier
        insert r at the end of C
        # Remove the cases covered by this rule
        delete all the cases with the ids in temp from D
        # Update the default class for the current classifier
        select a default class for the current C
        # Compute the total number of errors for the current classifier
        compute the total number of errors of C
    end
end

# Pruning step: find the rule with the minimum error
find the first rule p in C with the lowest total number of errors
drop all the rules after p in C

# Add the default class associated with p to the end of the classifier
add the default class associated with p to the end of C

# Return the final classifier
return C
```

CBA

- Satisfy two main conditions:
  1. Each training case is covered by the rule with highest precedence among the rules that can cover the case
  2. Every rule in C correctly classifies at least one remaining training case.

Example

min_sup	25%
min_conf	50%

ID	Itemset	Class
d1	a, c, g	1
d2	a, b, e	2
d3	b, c, g	1
d4	a, g, h	1
d5	c, f, h	2
d6	c, d, g	2
d7	b, f, g	2
d8	c, f, h	1

Rule	Sup(%)	Conf
a, g→ 1	**25**	**100**
a→ 1	**25**	**66.67**
h→ 1	**25**	**66.67**
b→ 2	**25**	**66.67**
f→ 2	**25**	**66.67**
c, g→ 1	**25**	**66.67**
c→ 1	**37.5**	**60**
g→ 1	**37.5**	**60**
c→ 2	**25**	**40**
g→ 2	**25**	**40**

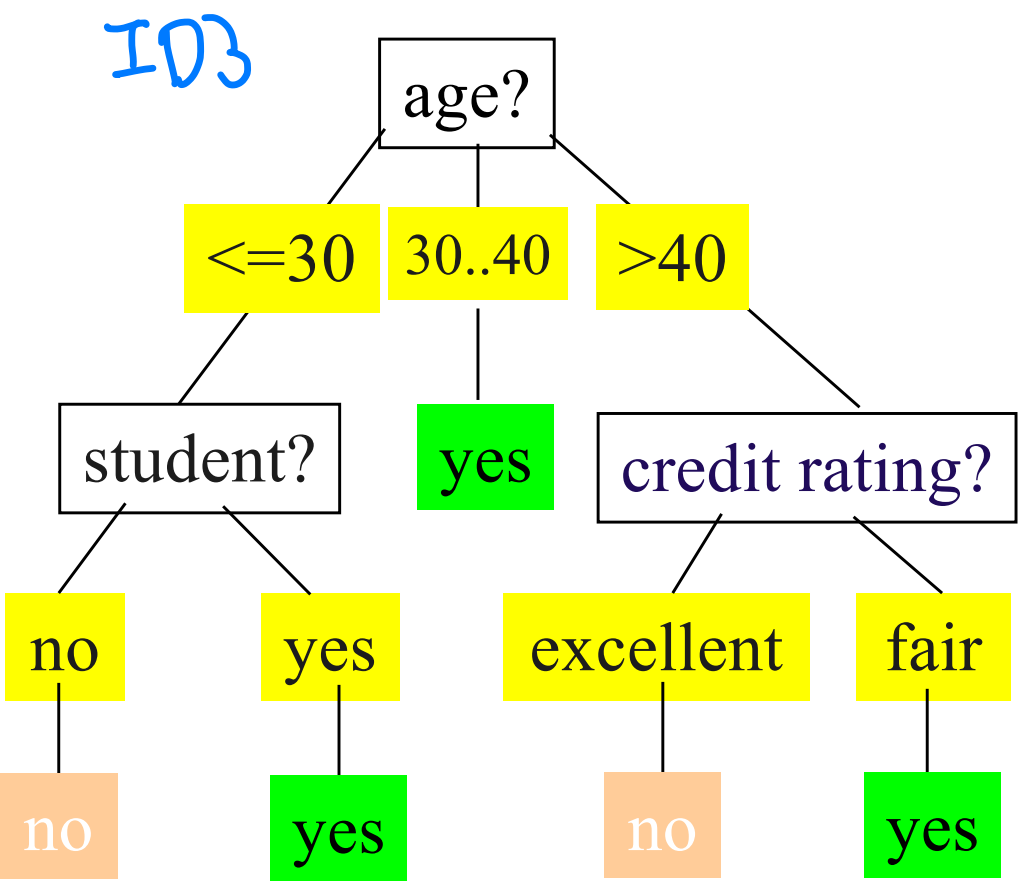
the consequent must be a class label in CARs

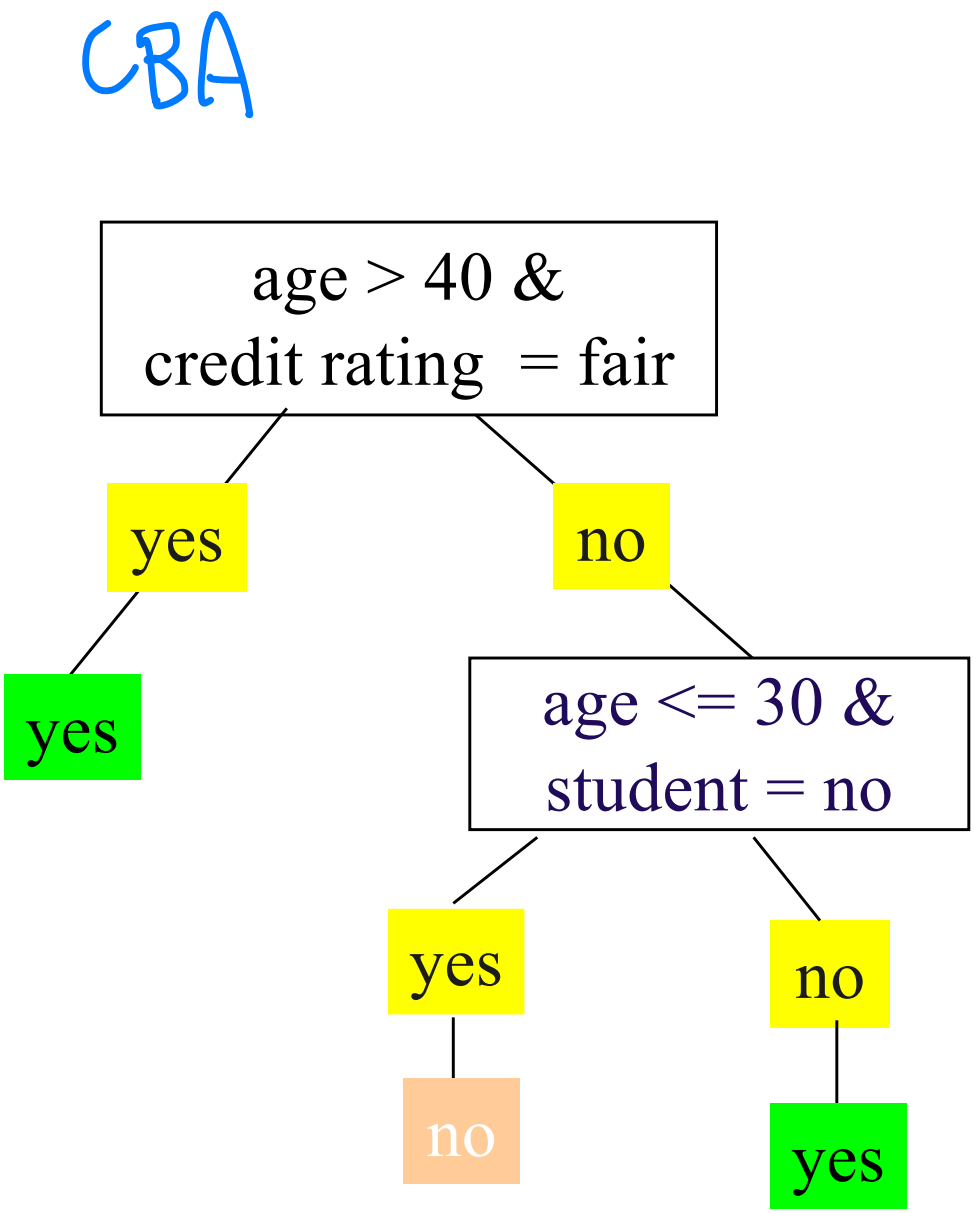
Classifier

a, g→1	default_class: 2 Error: 2
h→ 1	default_class: 2 Error: 2
b→ 2	default_class: 2 Error: 2

a, g, 1
default_class : 2

Comparison between ID3 & CBA





Bayesian Classification

Bayesian classification

- The classification problem may be formalized using a-posteriori probabilities:
- P(C|X) = prob. that the sample tuple

  $X=<x_1,\ldots,x_k>$ is of class C.
- E.g. P(class=N | outlook=sunny,windy=true,…)
- Idea: assign to sample X the class label C such that P(C|X) is maximal

Estimating A-Posteriori Probabilities

- Bayes theorem:

$$P(C|X) = P(X|C)\cdot P(C) / P(X)$$

- P(X) is constant for all classes
- P(C) = relative freq of class C samples
- C such that P(C|X) is maximum =

  C such that P(X|C)·P(C) is maximum
- Problem: computing P(X|C)

Naïve Bayesian Classification

- Naïve assumption: attribute independence

$$P(x_1,\ldots,x_k|C) = P(x_1|C)\cdot\ldots\cdot P(x_k|C)$$

- If i-th attribute is categorical:
  $P(x_i|C)$ is estimated as the relative freq of samples having value x_i as i-th attribute in class C
- If i-th attribute is continuous:
  $P(x_i|C)$ is estimated thru a Gaussian density function
- Computationally easy in both cases

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

P(p) = 9/14
P(n) = 5/14

$P(c)$

$P(X_i|c)$

Outlook	
P(sunny|p) = 2/9	**P(sunny|n) = 3/5**
P(overcast|p) = 4/9	**P(overcast|n) = 0**
P(rain|p) = 3/9	**P(rain|n) = 2/5**
Temperature	
P(hot|p) = 2/9	**P(hot|n) = 2/5**
P(mild|p) = 4/9	**P(mild|n) = 2/5**
P(cool|p) = 3/9	**P(cool|n) = 1/5**
Humidity	
P(high|p) = 3/9	**P(high|n) = 4/5**
P(normal|p) = 6/9	**P(normal|n) = 2/5**
Windy	
P(true|p) = 3/9	**P(true|n) = 3/5**
P(false|p) = 6/9	**P(false|n) = 2/5**

Play-tennis example: classifying X

- An unseen sample X = <rain, hot, high, false>
- P(*p*|X), P(*n*|X), which is maximum?
  - P(*p*|X)=P(X|*p*)·P(*p*)/P(X)
  - P(*n*|X)=P(X|*n*)·P(*n*)/P(X)
- P(X|*p*)·P(*p*)=P(rain, hot, high, false|*p*)·P(*p*)
  ≈ P(rain|*p*)·P(hot|*p*)·P(high|*p*)·P(false|*p*)·P(*p*)
  =3/9·2/9·3/9·6/9·9/14=0.010582
- P(X|*n*)·P(*n*)=P(rain, hot, high, false|*n*)·P(*n*)
  ≈ P(rain|*n*)·P(hot|*n*)·P(high|*n*)·P(false|*n*)·P(*n*)
  =2/5·2/5·4/5·2/5·5/14=0.018286

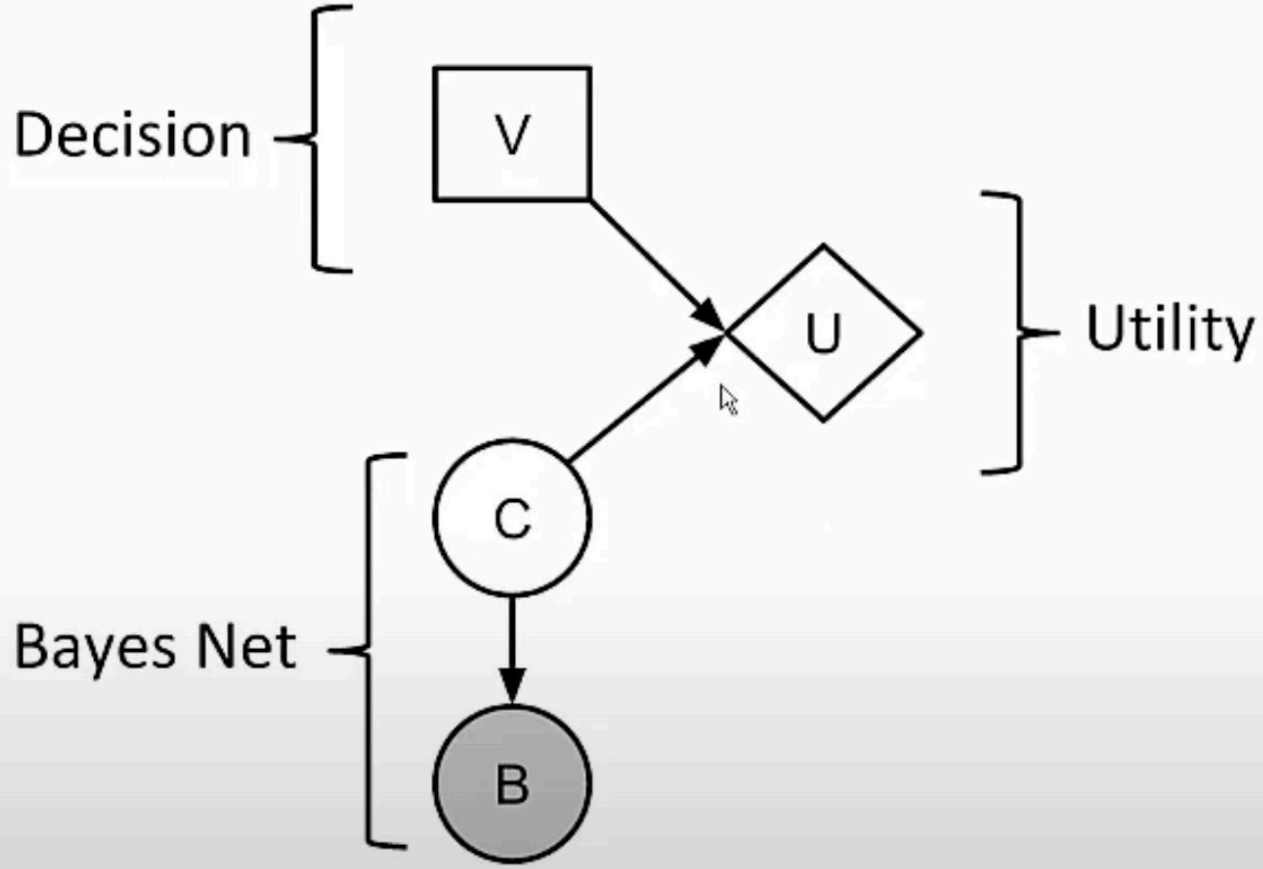
Sample X is classified in class *n* (don't play)

Outlook	
P(sunny|p) = 2/9	**P(sunny|n) = 3/5**
P(overcast|p) = 4/9	**P(overcast|n) = 0**
P(rain|p) = 3/9	**P(rain|n) = 2/5**
Temperature	
P(hot|p) = 2/9	**P(hot|n) = 2/5**
P(mild|p) = 4/9	**P(mild|n) = 2/5**
P(cool|p) = 3/9	**P(cool|n) = 1/5**
Humidity	
P(high|p) = 3/9	**P(high|n) = 4/5**
P(normal|p) = 6/9	**P(normal|n) = 2/5**
Windy	
P(true|p) = 3/9	**P(true|n) = 3/5**
P(false|p) = 6/9	**P(false|n) = 2/5**

The independence hypothesis…

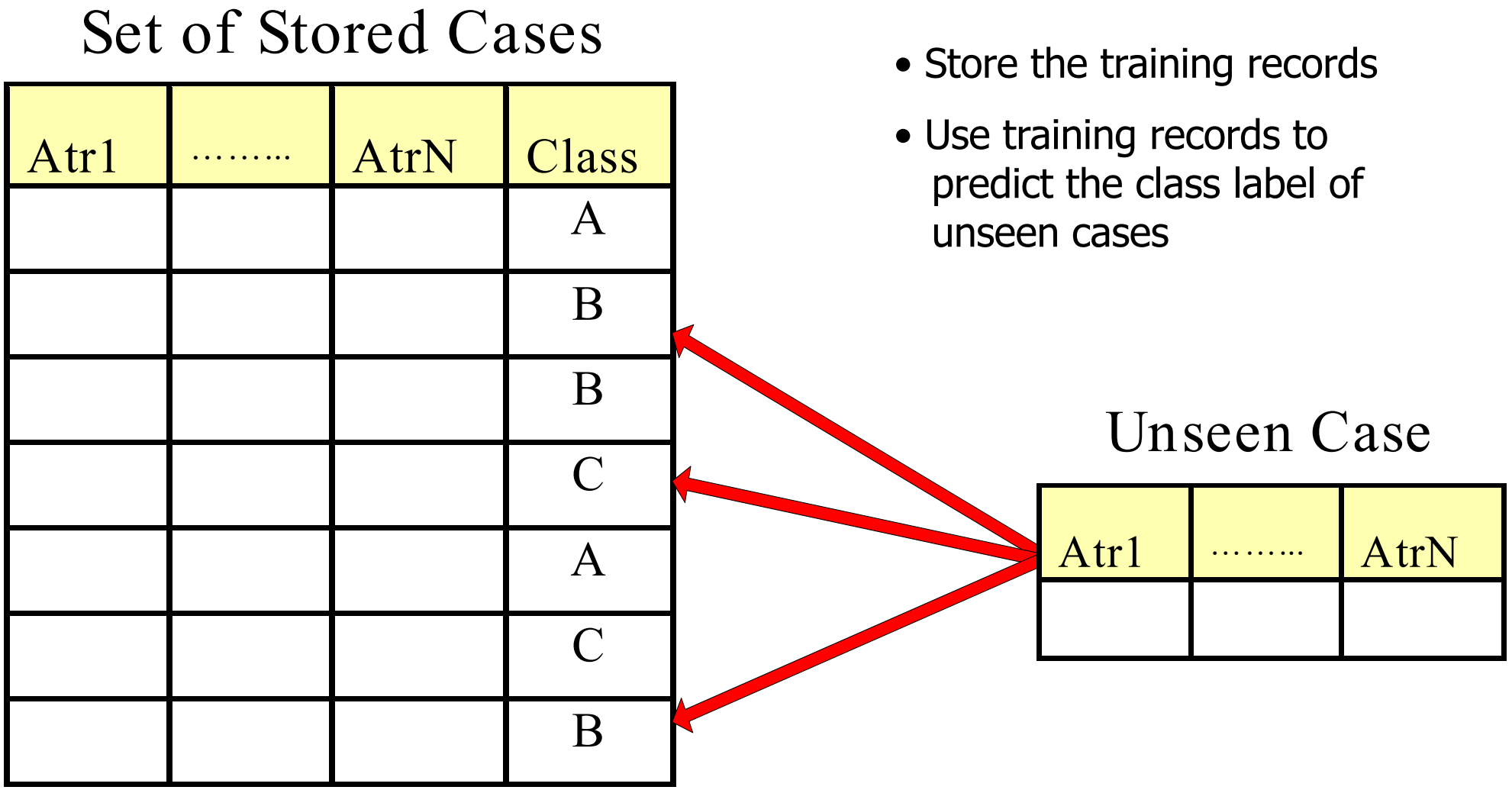
- … makes computation possible
- … yields optimal classifiers when satisfied
- … but is seldom satisfied in practice, as attributes (variables) are often correlated.
- Attempts to overcome this limitation:
  - Bayesian networks, that combine Bayesian reasoning with causal relationships between attributes

Decision Networks



Instance-based Learning

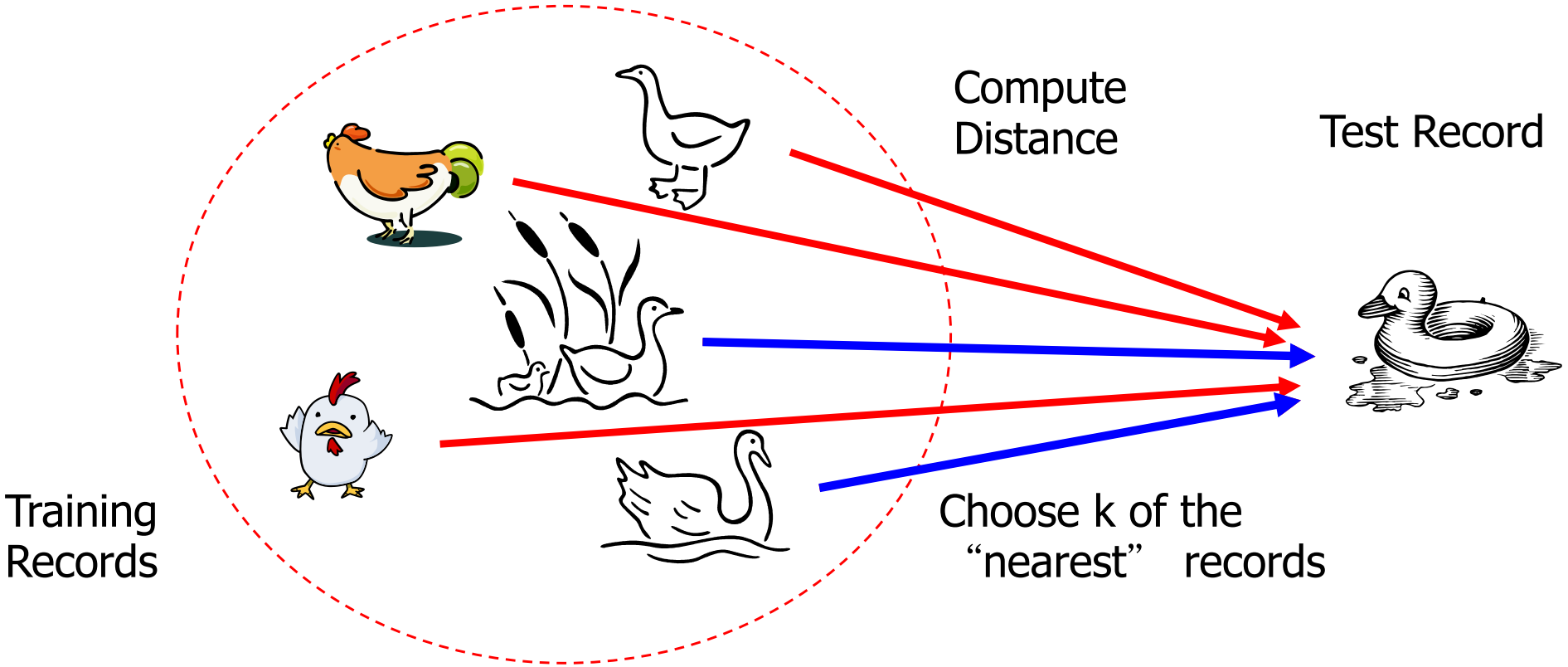
Instance-Based Classifiers



Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, looks like a duck

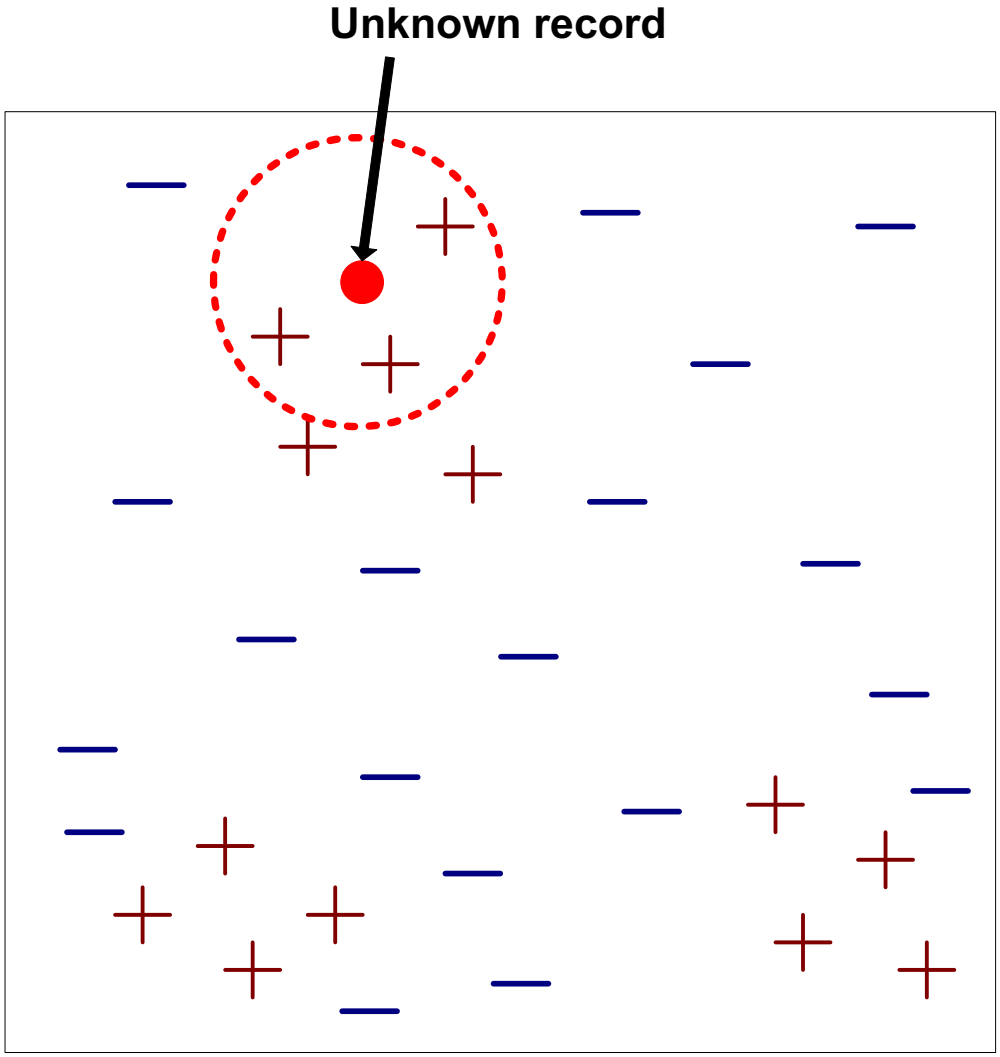
then it's probably a duck



Nearest neighbor Classification (cont.)

- lazy learners
  - It does not build models explicitly
  - Unlike eager learners such as decision tree induction and rule-based systems
  - Classifying unknown records are relatively expensive

Nearest-Neighbor Classifiers



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve
- To classify an unknown record:
  - Compute distance to other training records
  - Identify *k* nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Nearest Neighbor Classification: Distance

- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

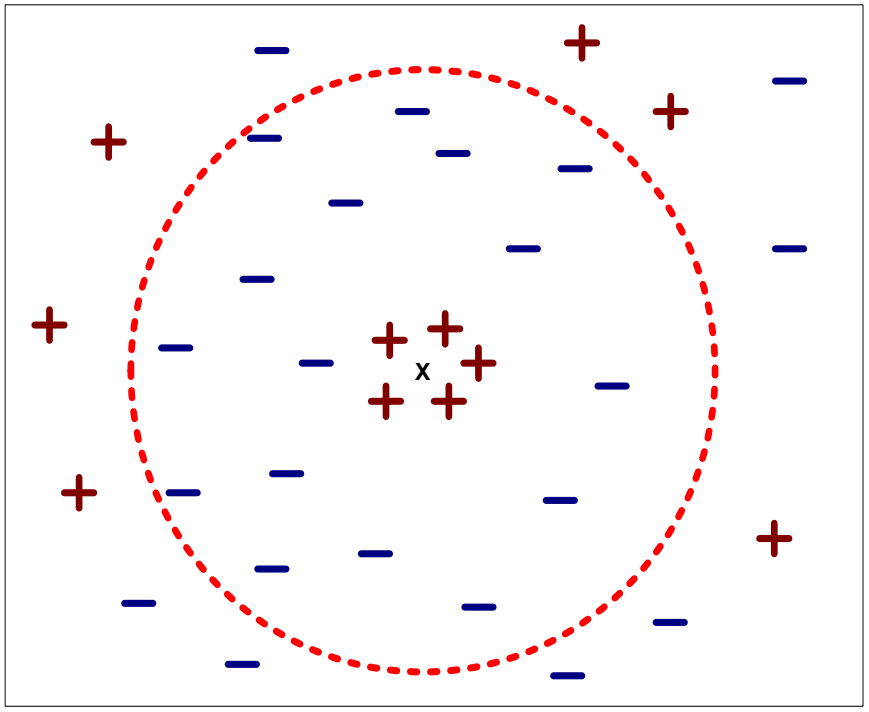
- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

Nearest Neighbor Classification: Distance

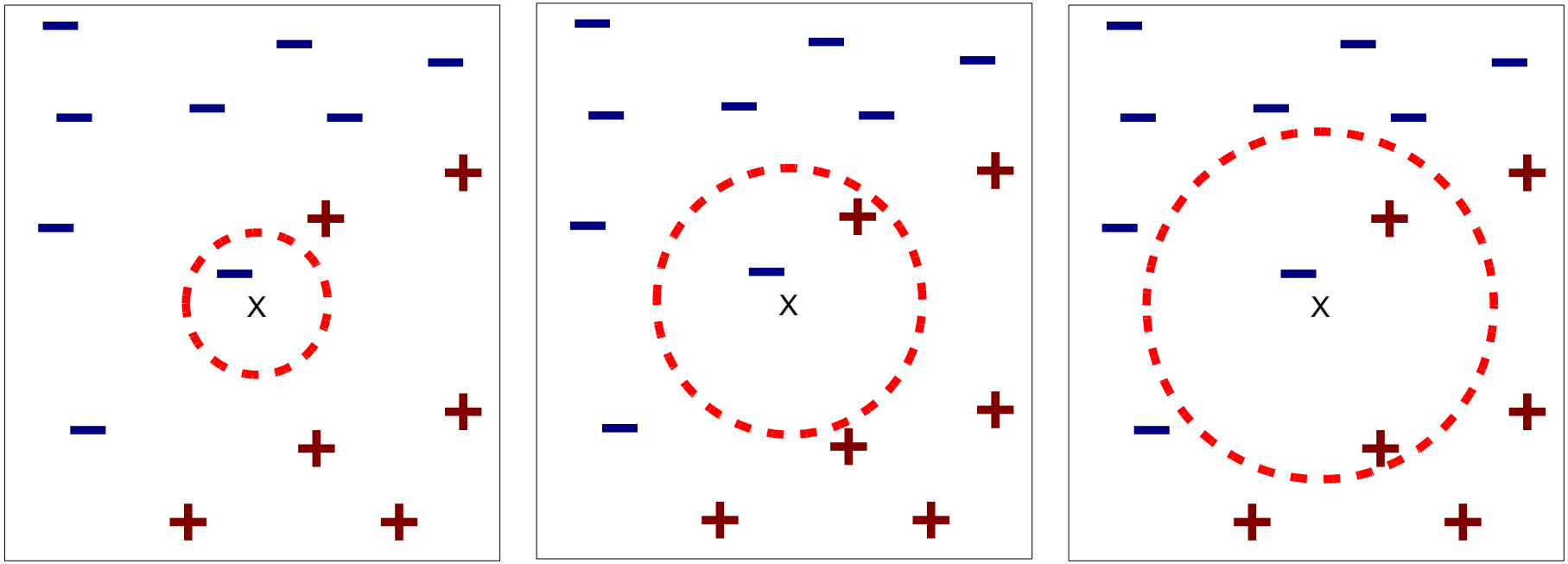
- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

Nearest Neighbor Classification: Choosing the Value k

- 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 Classification: Choosing the Value k (cont.)

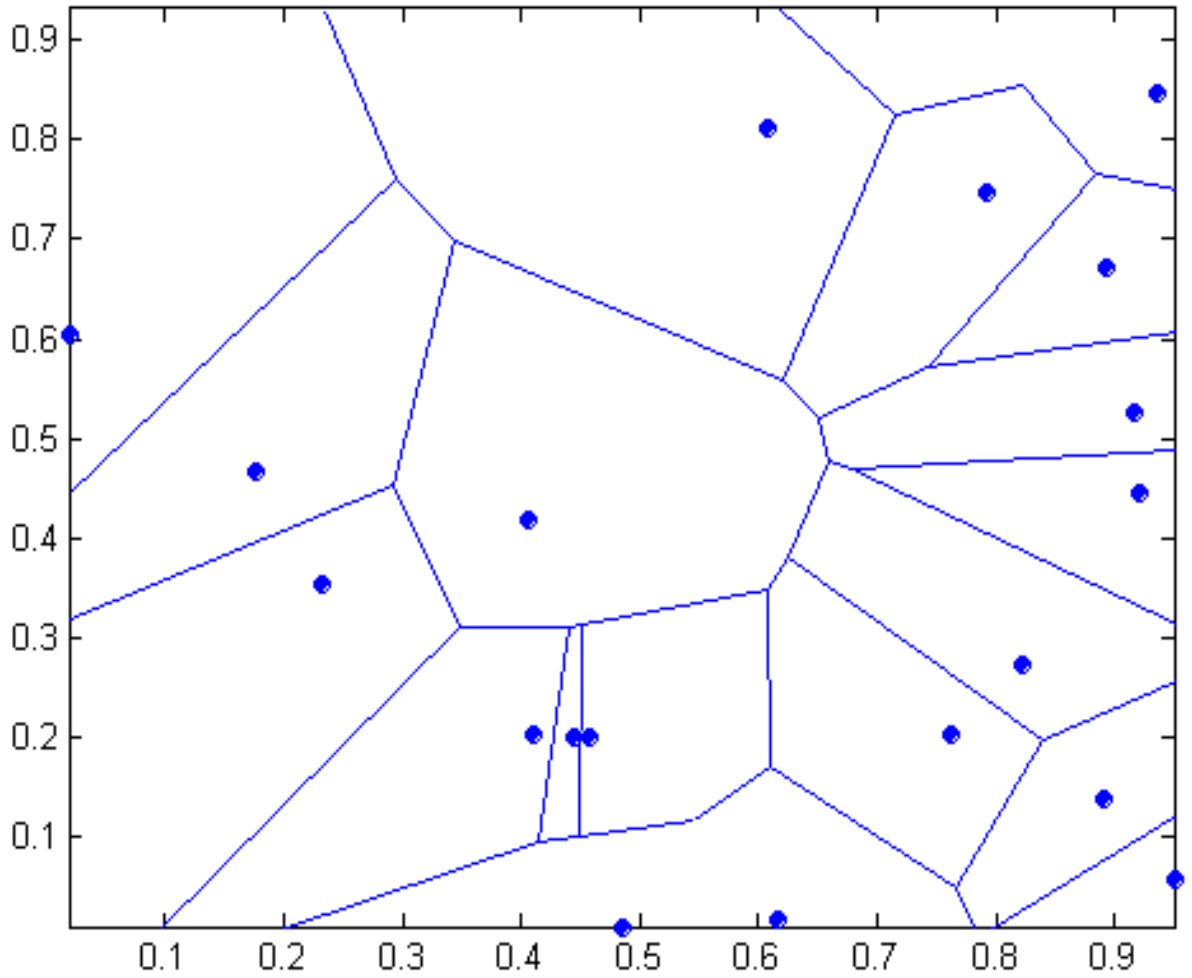


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

1 nearest-neighbor

Voronoi Diagram



Characteristics of Nearest Neighbor Classifier

- Instance-based learning: model free, lazy learner, uses the training examples to predict for a test instance
- Classify a test instance can be quite expensive to compute proximity

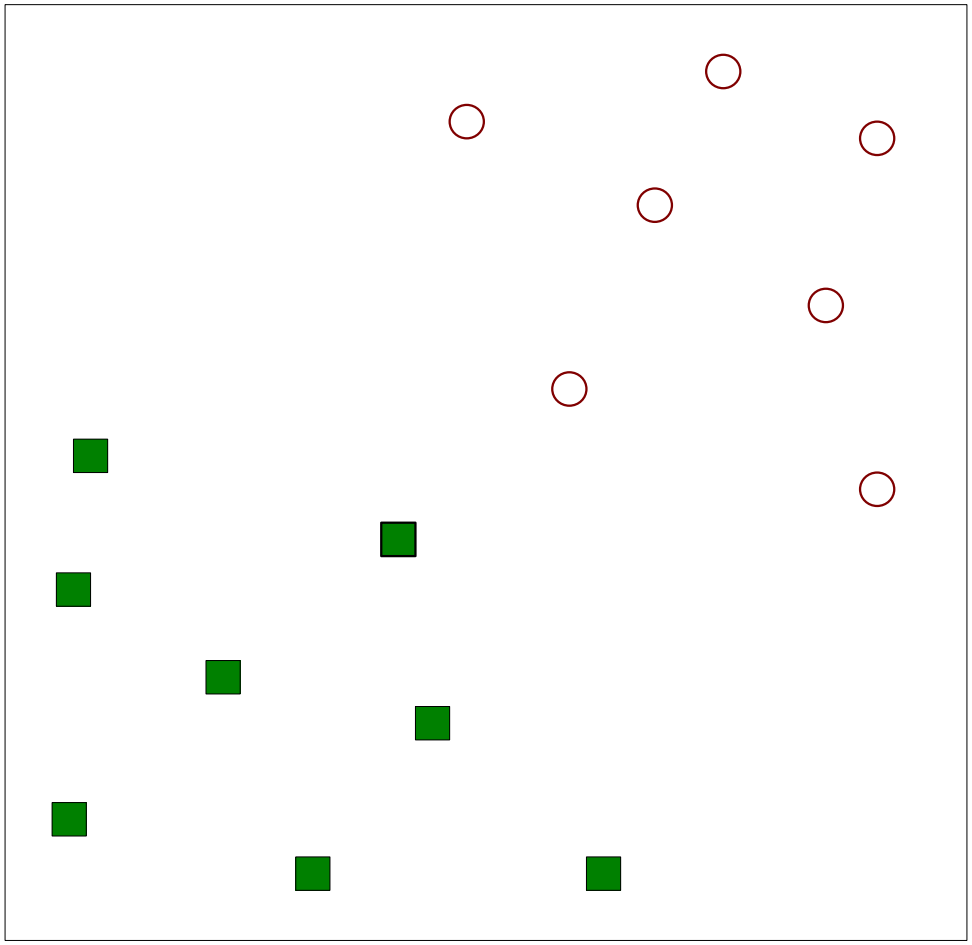
  (Multidimensional Indexing for K-NN)
- Make predictions based on local information
- Decision boundaries of arbitrary shape and high variability

Characteristics of Nearest Neighbor Classifier (cont.)

- Difficult to handle missing values of attributes
- Can handle presence of interacting attributes
- Irrelevant attributes can distort proximity measures
- Curse of Dimensionality : in high dimensional spaces, all pts tend to be equidistant
- Can produce poor predictions unless proximity measure and data preprocessing are taken

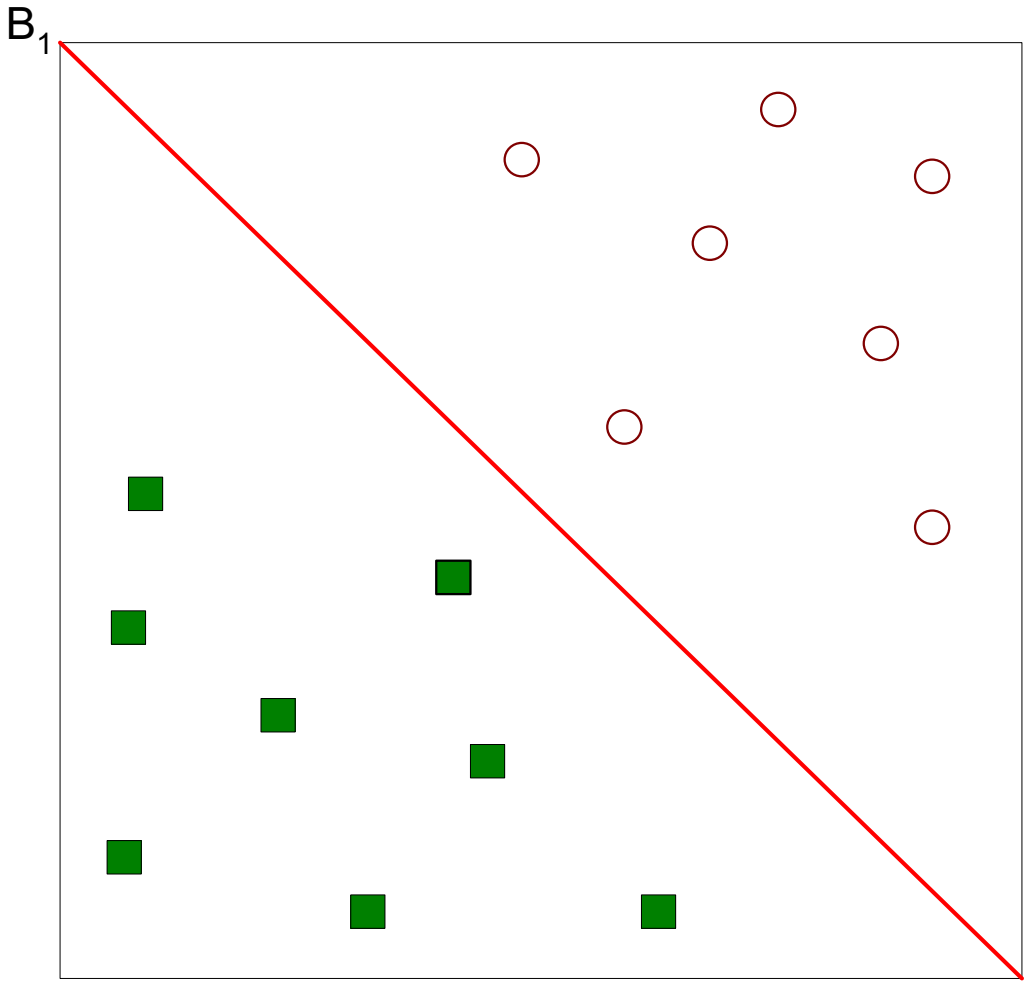
Support Vector Machine

Support Vector Machine



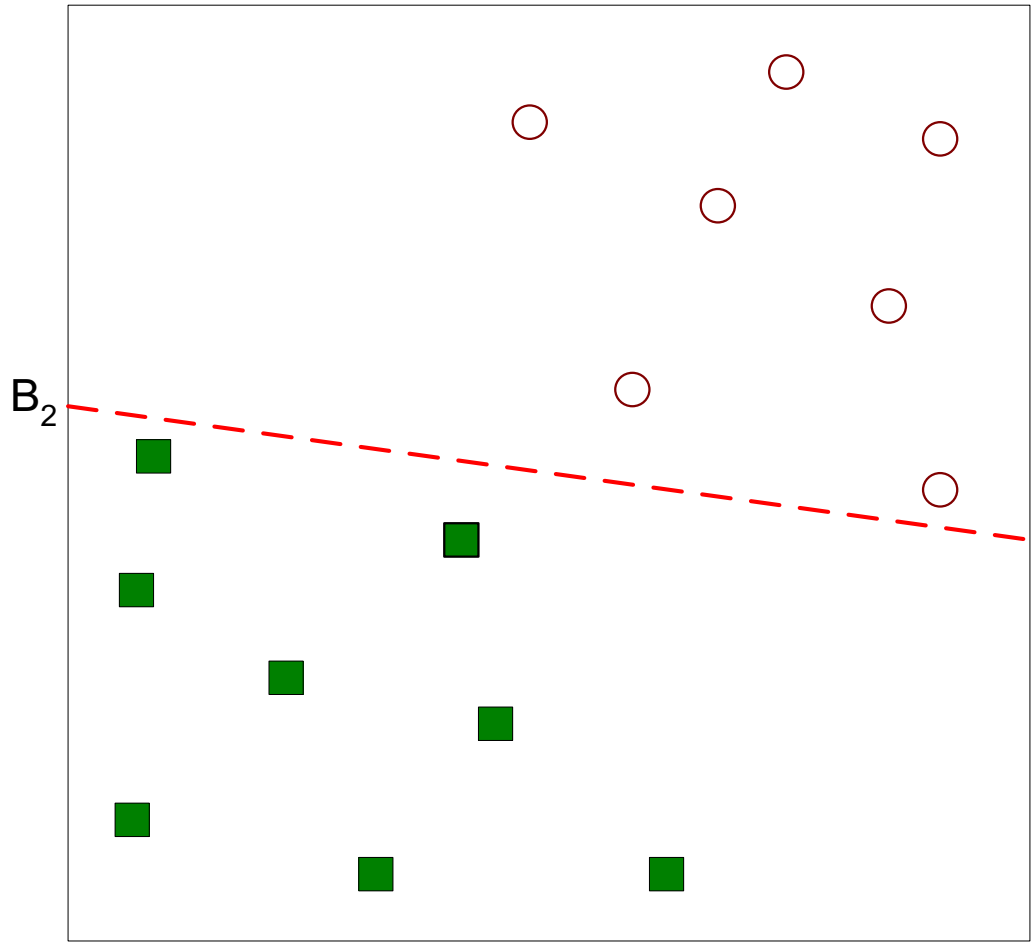
- Find a linear hyperplane (decision boundary) that will separate the data

Support Vector Machine (cont.)



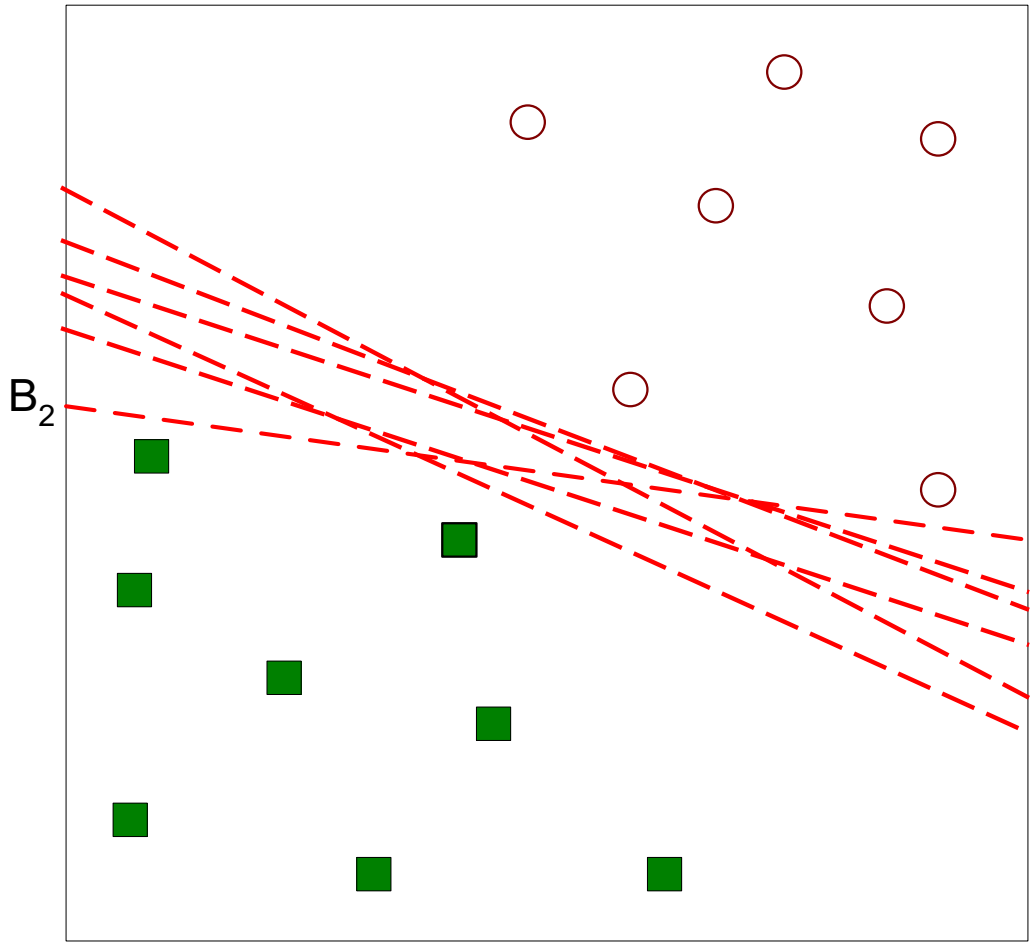
- One Possible Solution

Support Vector Machine (cont.)



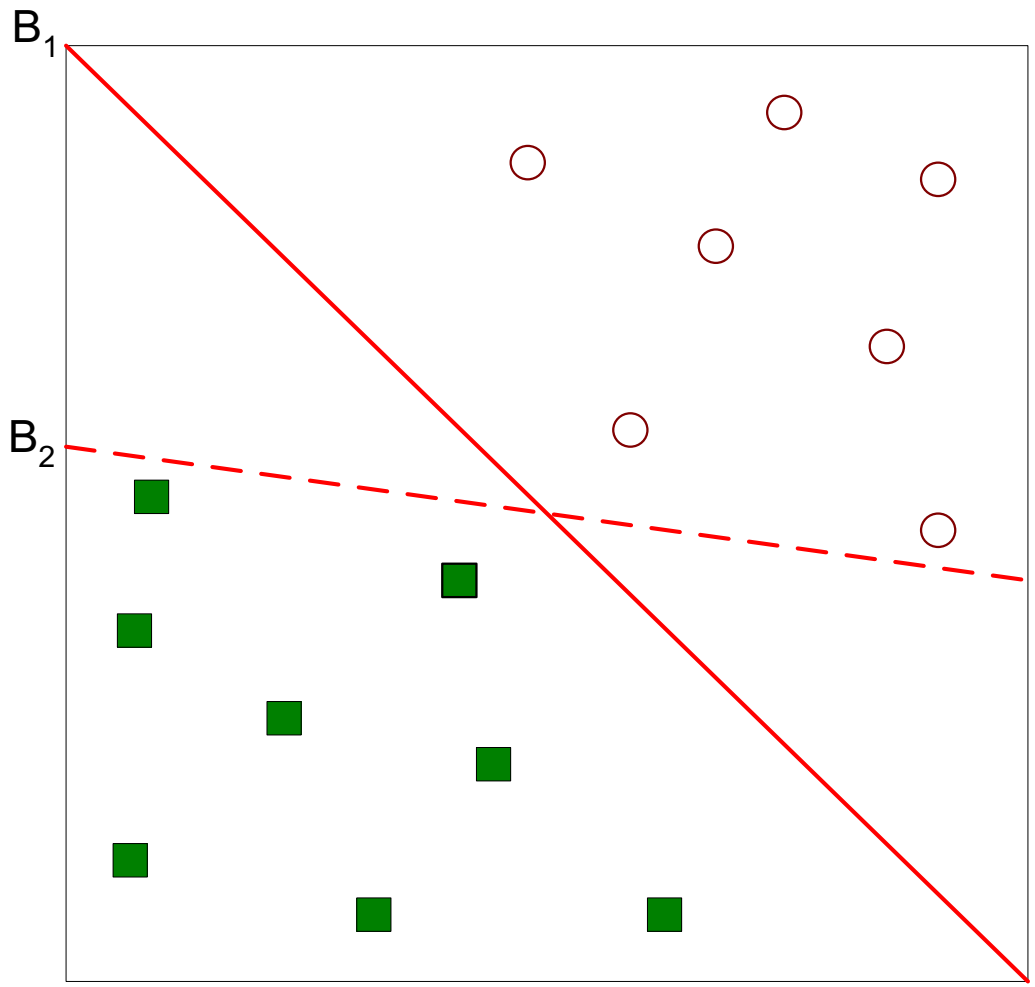
- Another possible solution

Support Vector Machine (cont.)



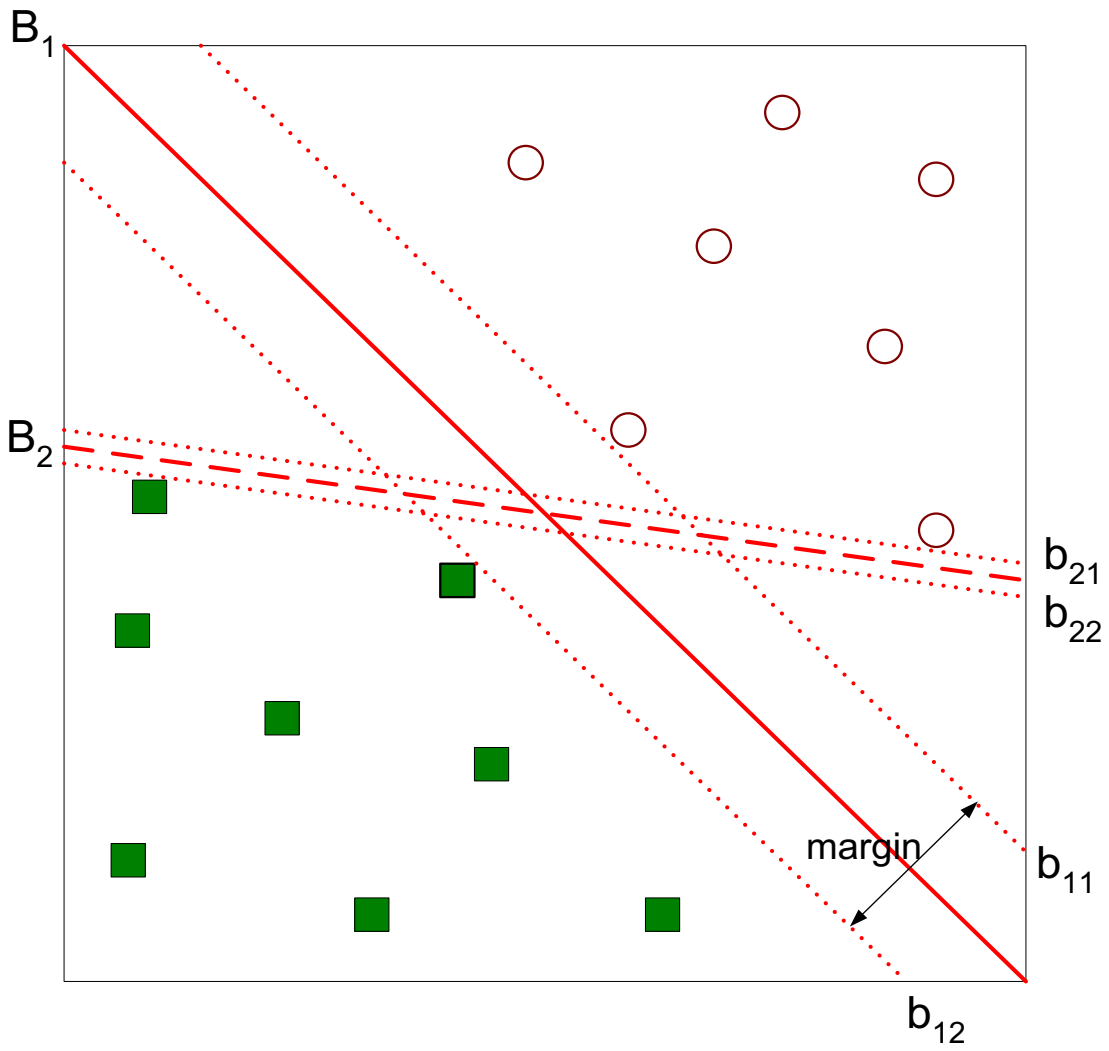
- Other possible solutions

Support Vector Machine (cont.)



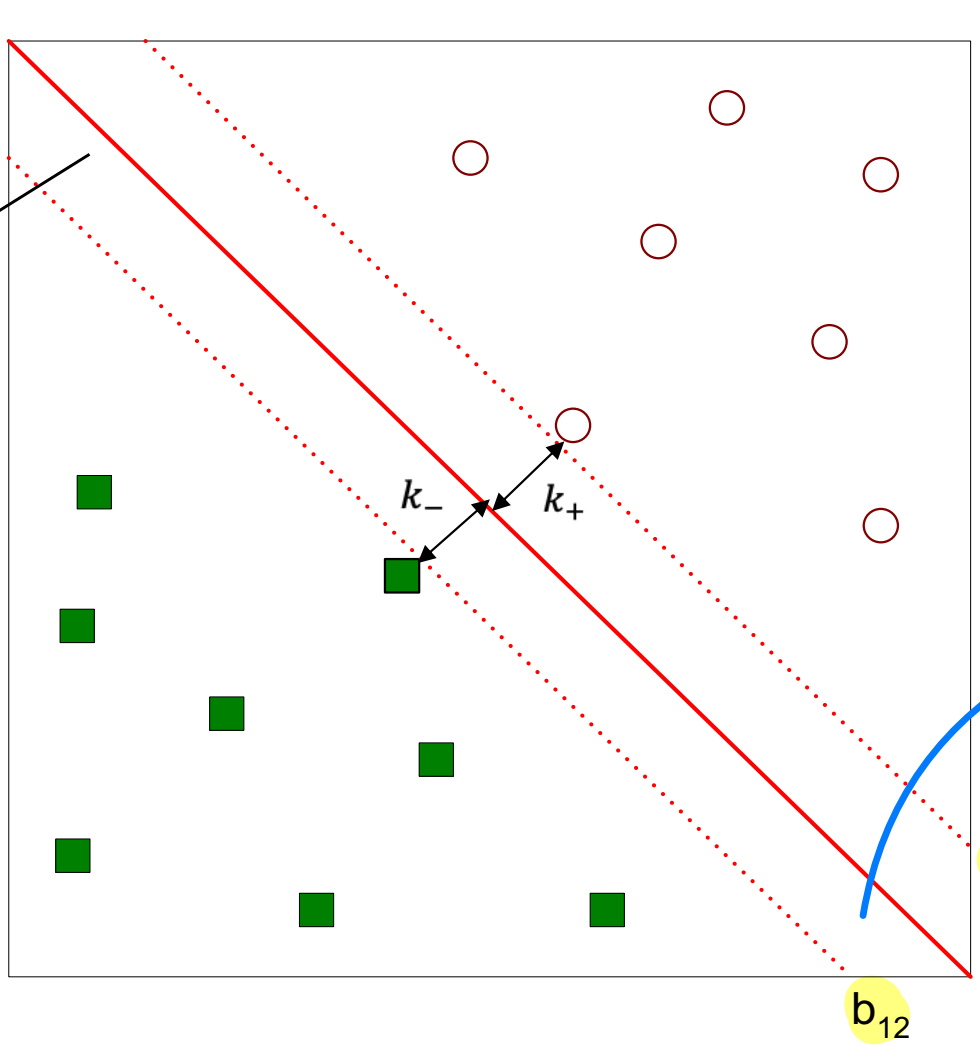
- Which one is better? B1 or B2?
- How do you define better?

Support Vector Machine (cont.)



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

Support Vector Machine (cont.)



$$D(x) = \frac{w^T x + b}{\|w\|}$$

$$\frac{w^T x_i + b}{\|w\|} \geq k_+ \quad if\ y_i = 1$$

$$\frac{w^T x_i + b}{\|w\|} \leq -k_- \quad if\ y_i = -1$$

$$y_i(w^T x_i + b) \geq M\|w\|$$

$$margin = 2M = k_+ - k_-$$

1. Hyperplane Equation:

$$\vec{w} \cdot \vec{x} + b = 0$$

- $\vec{w}$: Weight vector, normal to the hyperplane, determines the orientation of the hyperplane.
- $\vec{x}$: Feature vector (data point).
- b: Bias term, adjusts the hyperplane position.
- $\vec{w} \cdot \vec{x}$: Dot product, computes the projection of $\vec{x}$ onto $\vec{w}$.

This equation represents the decision boundary (hyperplane) that separates the data into two classes.

---

2. Decision Function:

$$D(x) = \frac{\vec{w}^T\vec{x} + b}{\|\vec{w}\|}$$

- $\vec{w}^T$: Transpose of $\vec{w}$, aligns with $\vec{x}$ for computation.
- $\|\vec{w}\|$: Magnitude (norm) of $\vec{w}$, ensures that the distance is normalized.
- $D(x)$: Signed distance of a point $\vec{x}$ from the hyperplane.

The sign of $D(x)$ determines which side of the hyperplane the point lies on:

- $D(x) > 0$: Point lies on one side (e.g., class +1).
- $D(x) < 0$: Point lies on the other side (e.g., class -1).

3. Constraints for Classification:

$$\frac{\vec{w}^T\vec{x}i + b}{\|\vec{w}\|} \geq k+ \quad \text{if } y_i = +1$$

$$\frac{\vec{w}^T\vec{x}i + b}{\|\vec{w}\|} \leq k- \quad \text{if } y_i = -1$$

- y_i: Class label ($+1$ or -1).
- k_+: Positive margin threshold for class $+1$.
- k_-: Negative margin threshold for class -1.

These constraints ensure that the data points are correctly classified and lie on the correct side of the margins.

---

4. General Constraint:

$$y_i(\vec{w}^T\vec{x}_i + b) \geq M\|\vec{w}\|$$

- M: Margin multiplier.
- Ensures that the points are classified with a margin proportional to M.

This combined constraint incorporates the margin distance and classifies points based on their proximity to the margin boundaries.

5. Margin Width:

$$\text{Margin} = 2M = k_+ - k_-$$

- The margin is the perpendicular distance between the two parallel hyperplanes that bound the data of different classes.
- k_+ and k_- are the signed distances of the positive and negative class boundaries, respectively.

The goal of SVM is to **maximize the margin (2M)** while ensuring that all points are correctly classified and lie on the correct side of their respective margins.

Key SVM Objective:

The optimal hyperplane is the one that maximizes the margin, which is equivalent to minimizing $\|\vec{w}\|$ (since the margin is inversely proportional to $\|\vec{w}\|$).

The optimization problem for SVM can be expressed as:

$$\text{Minimize}: \ \frac{1}{2}\|\vec{w}\|^2 \quad \text{Subject to}: \ y_i(\vec{w}^T\vec{x}_i + b) \geq 1$$

Support Vector Machine: Problem Formulation

maximize the margin

$$\max_{w,b} 2M$$

$$subject\ to \quad y_i(w^T x_i + b) \geq M\|w\|$$

which is equivalent to

$$\min_{w,b} \frac{\|w\|^2}{2}$$

$$subject\ to \quad y_i(w^T x_i + b) \geq 1$$

(by choosing ||*w*|| =1/*M*)

1. Maximizing the Margin

Original Objective:

$$\max_{w,b} 2M$$

- $2M$: The width of the margin (distance between the two parallel hyperplanes).
- The goal of SVM is to maximize this margin to achieve better separation between the classes.

Subject to Constraints:

$$y_i \left(\vec{w}^T \vec{x}_i + b \right) \geq M \|\vec{w}\|$$

- y_i: Class label ($+1$ or -1).
- $\vec{w}^T \vec{x}_i + b$: Decision function for data point $\vec{x}_i$.
- $\|\vec{w}\|$: Norm of the weight vector, representing the scale of $\vec{w}$.

This constraint ensures that all data points are classified correctly with at least a margin $M\|\vec{w}\|$. This formulation directly maximizes the margin while ensuring correct classification.

2. Reformulation to Simplify the Objective

Choosing a Scaling:

- To simplify the math, the norm of $\vec{w}$ is scaled such that:

$$\|\vec{w}\| = \frac{1}{M}$$

  - This allows the margin M to be inversely proportional to $\|\vec{w}\|$.

Equivalent Formulation:

$$\min_{w,b} \frac{\|\vec{w}\|^2}{2}$$

- The objective changes to **minimizing** $\frac{\|\vec{w}\|^2}{2}$, which indirectly maximizes the margin because $\|\vec{w}\|$ is inversely related to M.

Updated Constraints:

$$y_i \left(\vec{w}^T \vec{x}_i + b\right) \geq 1$$

- The margin constraint $y_i(\vec{w}^T \vec{x}_i + b) \geq M\|\vec{w}\|$ simplifies to this form when $\|\vec{w}\| = \frac{1}{M}$.
- This ensures that all data points are classified correctly with a normalized margin $M = 1$.

3. Final SVM Optimization Problem

Objective:

$$\min_{w,b} \frac{\|\vec{w}\|^2}{2}$$

- Minimize the squared norm of the weight vector to maximize the margin.

Subject to:

$$y_i \left(\vec{w}^T \vec{x}_i + b\right) \geq 1$$

- Ensure correct classification for all training data points.

Why This Formulation?

1. **Simplifies Optimization**:
   - By fixing $\|\vec{w}\| = \frac{1}{M}$, the margin M is implicitly maximized by minimizing $\|\vec{w}\|^2$.
2. **Quadratic Programming**:
   - The problem becomes a **convex quadratic optimization** problem, which can be solved efficiently using standard optimization techniques.
3. **Interpretation**:
   - The margin is maximized by minimizing the weight vector's norm, ensuring the hyperplane is optimally positioned between the two classes.

Support Vector Machine: Problem Formulation (cont.)

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

$$\text{subject to } y_i (w \cdot x_i + b) \geq 1,$$

$$i = 1, 2, 3, \ldots N$$

- Constrained optimization problem
  - Convex optimization problem
  - Objective function is quadratic, constraints are linear in w and *b*.
  - Can be solved using Lagrange multiplier

Support Vector Machine: Lagrangian Primal Problem

$$\min_{w,b} L_P = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(w^T x_i + b) - 1)$$

penalize the violation of margin constraints

margin constraints

the objective to minimize

λ_i: Lagrange multipliers (non-negative)

$$\begin{cases} \frac{\partial L_p}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{n} \lambda_i y_i x_i \\ \frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \lambda_i y_i = 0 \end{cases}$$

2. Lagrangian Formulation

The constraints $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$ must hold for the solution to be valid. In the Lagrangian, each constraint is turned into a **penalty term**:

$$L_P(\vec{w}, b, \lambda) = \frac{1}{2}\|\vec{w}\|^2 - \sum_{i=1}^{n} \lambda_i \left[y_i(\vec{w}^T\vec{x}_i + b) - 1\right]$$

Penalty Term:

$$\lambda_i \left[y_i(\vec{w}^T\vec{x}_i + b) - 1\right]$$

- If the constraint $y_i(\vec{w}^T\vec{x}_i + b) - 1 \geq 0$ is satisfied, this term contributes nothing (or is non-positive, as $\lambda_i \geq 0$).
- If the constraint is violated ($y_i(\vec{w}^T\vec{x}_i + b) - 1 < 0$), the penalty term adds a positive value to the objective, penalizing the violation.

The **Lagrange multipliers (λ_i)** act as weights for these penalties:

- $\lambda_i > 0$: Indicates that the corresponding constraint is active or violated (i.e., the point lies on or inside the margin).
- $\lambda_i = 0$: Indicates that the constraint is satisfied (the point lies outside the margin and does not influence the optimization).

3. Why Are λ_i the Coefficients?

Balancing the Objective and Constraints:

The Lagrange multipliers λ_i control the trade-off between minimizing the objective $\frac{1}{2}\|\vec{w}\|^2$ and satisfying the constraints $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$. Here's how:

1. If a constraint is far from being violated, $\lambda_i = 0$, meaning the corresponding penalty term does not contribute to the objective.
2. If a constraint is exactly on the boundary or violated (e.g., a support vector lies on the margin), $\lambda_i > 0$, meaning the penalty term actively influences the optimization.

Penalizing Violations Proportionally:

The penalty term $\lambda_i \left[y_i(\vec{w}^T\vec{x}_i + b) - 1\right]$ ensures that violations of the constraints are penalized proportionally:

- A larger λ_i corresponds to a higher weight on the importance of satisfying the constraint for point i.

Gradient Relationships:

The partial derivatives of the Lagrangian with respect to $\vec{w}$ and b involve λ_i. The resulting optimality conditions reveal that:

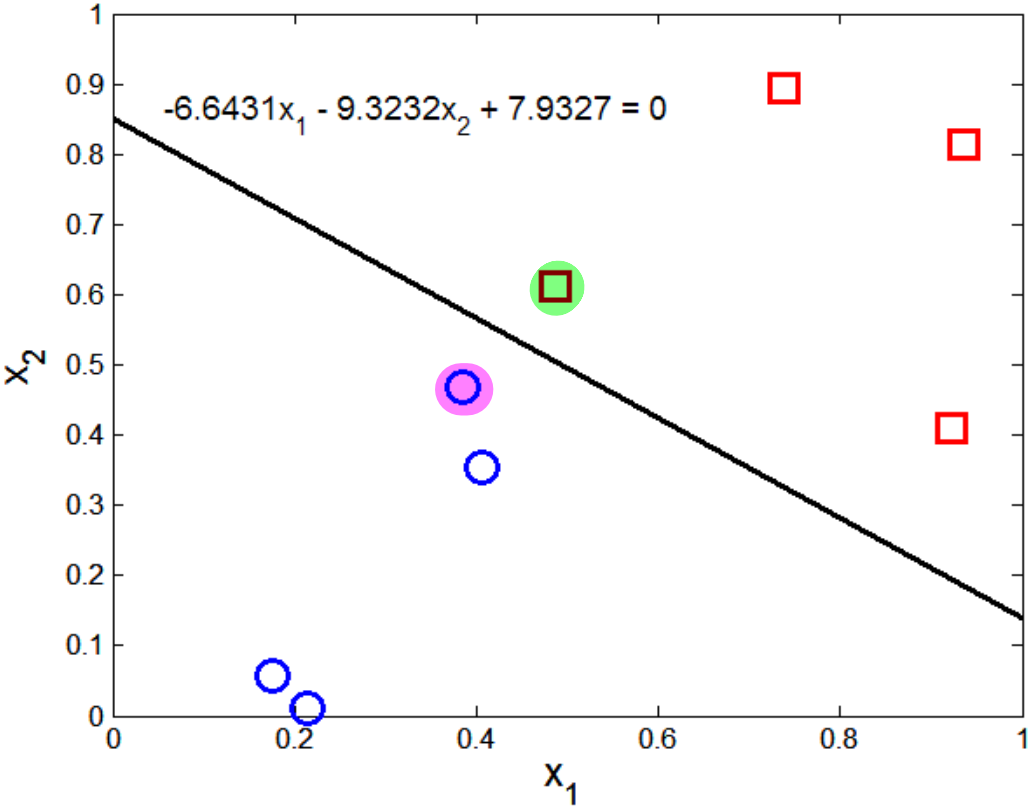
- $\vec{w} = \sum_{i=1}^{n} \lambda_i y_i \vec{x}_i$
  - λ_i determine how much each point contributes to $\vec{w}$.
- Only points with $\lambda_i > 0$ (the **support vectors**) contribute to the final decision boundary.

Support Vector Machine: Complementary Slackness Condition

$$\lambda_i[y_i(w^T x_i + b) - 1] = 0$$

support vectors

- Derived from KKT conditions (Karush-Kuhn-Tucker)



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

The term **"complementary slackness"** comes from the mathematical optimization framework, particularly in the **Karush-Kuhn-Tucker (KKT) conditions**, which are used to solve constrained optimization problems. It describes a relationship between the Lagrange multipliers and the constraints, ensuring that either:

- A constraint is **active** (i.e., binding and influences the solution), or
- The corresponding Lagrange multiplier is **zero** (i.e., the constraint has no effect on the optimization).

This mutual exclusivity — one being "active" while the other is "slack" — gives rise to the term **"complementary slackness."**

Complementarity:

- The condition ties the Lagrange multiplier λ_i and the constraint $y_i(\vec{w}^T\vec{x}_i + b) - 1$:

$$\lambda_i \cdot \Big[y_i(\vec{w}^T\vec{x}_i + b) - 1\Big] = 0$$

- These two terms are complementary because **if one is non-zero, the other must be zero**:
  - If $\lambda_i > 0$, then $y_i(\vec{w}^T\vec{x}_i + b) - 1 = 0$ (the constraint is active).
  - If $y_i(\vec{w}^T\vec{x}_i + b) - 1 > 0$, then $\lambda_i = 0$ (the constraint is inactive).

Slackness:

- **Slackness** refers to the degree to which a constraint is "loose" or inactive:
  - If a constraint $y_i(\vec{w}^T\vec{x}_i + b) - 1 > 0$, it is not tight, and its associated $\lambda_i = 0$.
  - Conversely, if the constraint is tight ($y_i(\vec{w}^T\vec{x}_i + b) - 1 = 0$), the multiplier $\lambda_i > 0$.

The term **"complementary slackness"** encapsulates this relationship: one term being active (non-zero) means the other is slack (zero).

Support Vector Machine: Dual Optimization Problem

$$\max_{\lambda_i} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

$$subject\ to \sum_{i=1}^{n} \lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

depends on the dot product, allowing the kernel trick being used.

Support Vector Machine: Primal & Dual Problem

Example

Primal

$$
\begin{aligned}
Max\ & 40x_1 + 30x_2 && (profits)\\
s.t.\ & x_1 + x_2 \le 120 && (land)\\
& 4x_1 + 2x_2 \le 320 && (labor)\\
& x_1,\ x_2 \ge 0
\end{aligned}
$$

Dual

$$
\begin{aligned}
& (land)\ \ (labor)\\
Min\ & 120y_1 + 320y_2\\
s.t.\ & y_1 + 4y_2 \ge 40 && (x_1)\\
& y_1 + 2y_2 \ge 30 && (x_2)\\
& y_1,\ y_2 \ge 0
\end{aligned}
$$

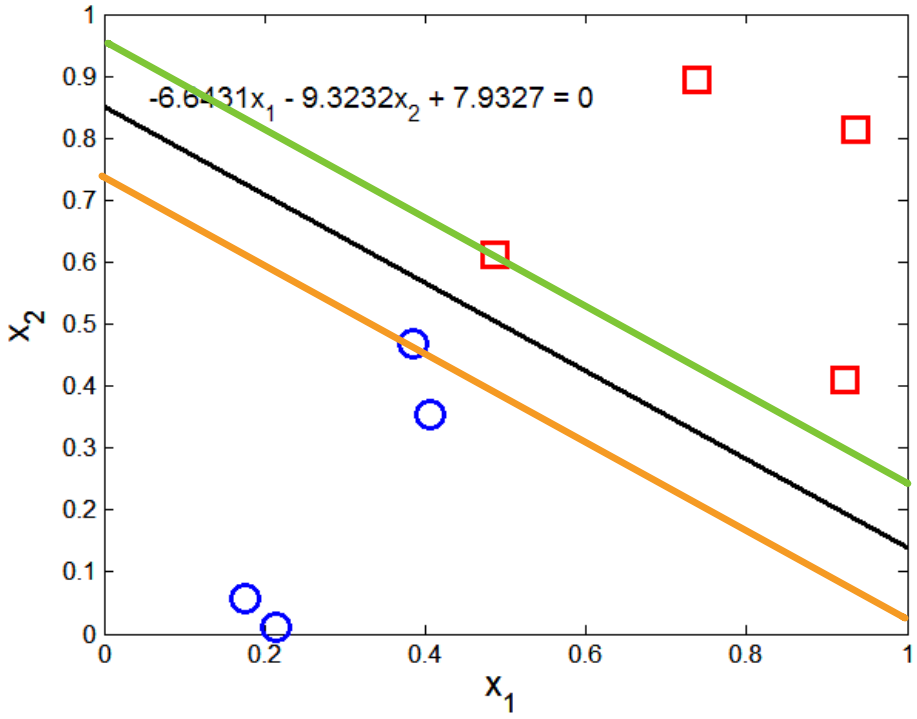
Support Vector Machine: Solving Dual Optimization Problem

$$\frac{\partial L_p}{\partial w} = 0 \Longrightarrow w = \sum_{i=1}^{n} \lambda_i y_i x_i$$

$$b = \frac{1}{n_S} \sum_{i \in S} \frac{1 - y_i w^T x_i}{y_i} \quad (\because \ \lambda_i [y_i (w^T x_i + b) - 1] = 0)$$

$$f(x) = \left(\sum_{i=1}^{n} \lambda_i y_i x_i^T x \right) + b = 0$$

Support Vector Machine: Examples



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

$$w_1 = \sum_i \lambda_i y_i x_{i1} = 65.5261 \times 1 \times 0.3858 + 65.5261 \times -1 \times 0.4871 = -6.64$$

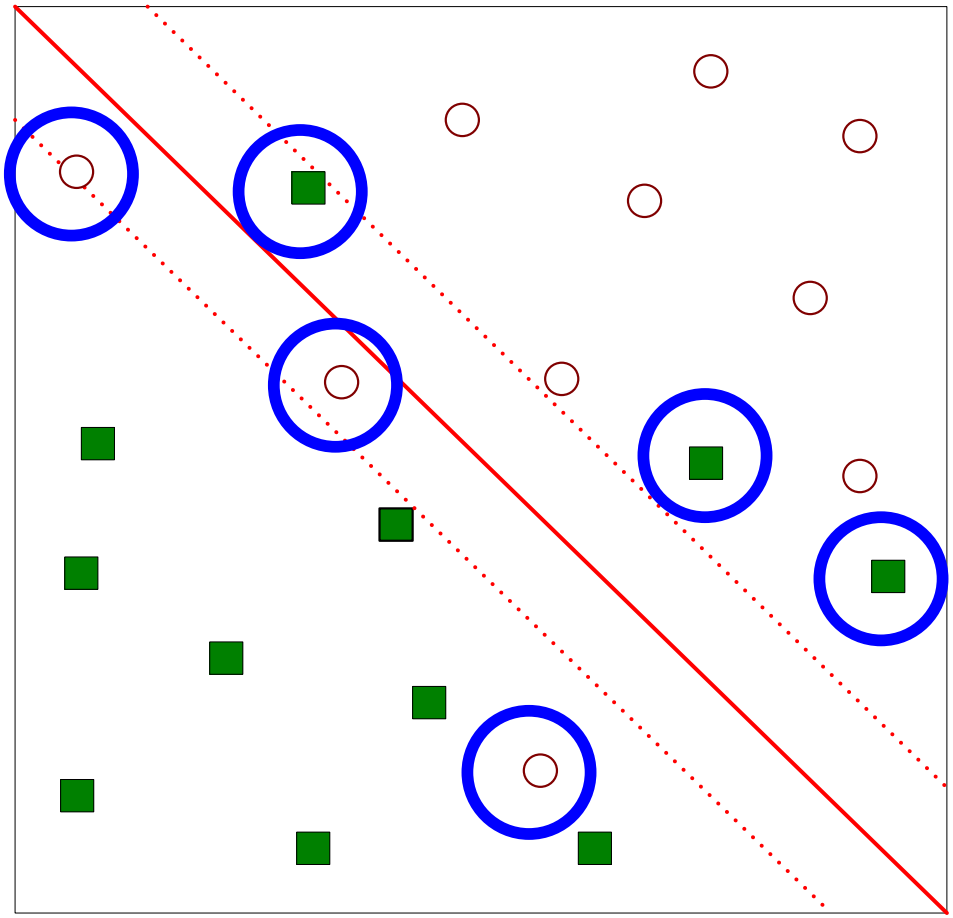
$$w_2 = \sum_i \lambda_i y_i x_{i2} = 65.5261 \times 1 \times 0.4687 + 65.5261 \times -1 \times 0.611 = -9.32$$

$$b^{(1)} = 1 - w \cdot x_1 = 1 - (-6.64)(0.3858) - (-9.32)(0.4687) = 7.9300$$

$$b^{(2)} = -1 - w \cdot x_2 = -1 - (-6.64)(0.4871) - (-9.32)(0.611) = 7.9289$$

Soft-margin Support Vector Machine

- What if the problem is not linearly separable?



Soft-margin Support Vector Machine (cont.)

- Introduce slack variables (regularization)
  - To minimize:

$$L(w) = \frac{\| \vec{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k \right)$$

  - Subject to:

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

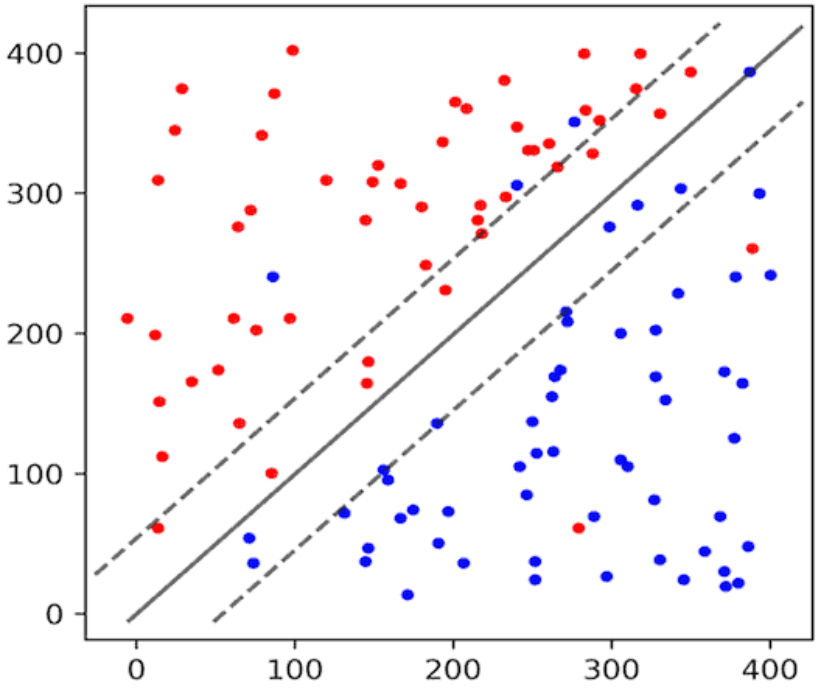
- *C*: hyper-parameter that makes a trade-off between maximizing the margin & minimizing the training error
- A large value of C pays more emphasis on minimizing training errors than maximizing the margin
- *k*: user-specified parameters representing penalty of misclassifying training instance

The soft-margin SVM minimizes the following objective function:

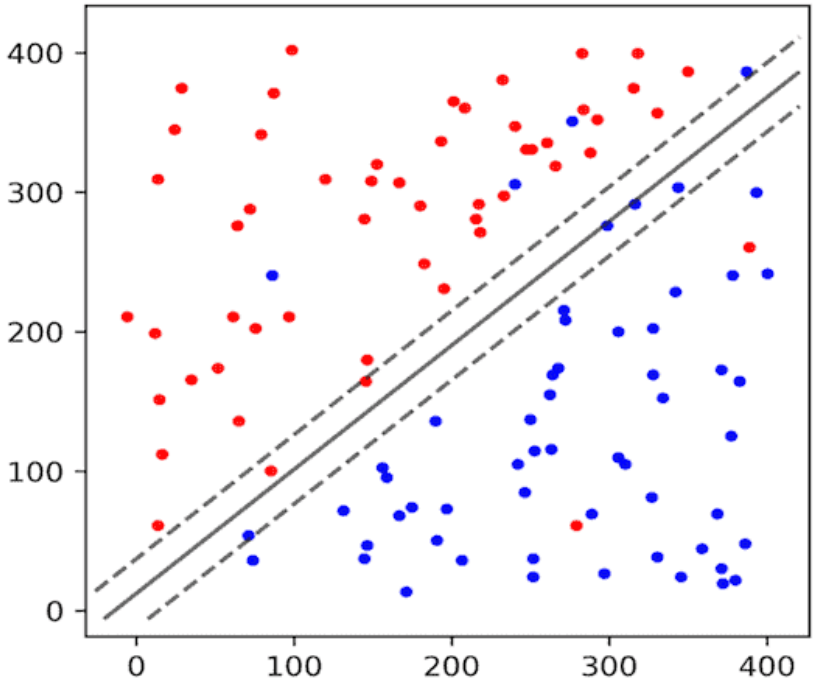
$$L(\vec{w}) = \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{N}\xi_i^k$$

Explanation of Terms:

1. $\frac{1}{2}\|\vec{w}\|^2$:
   - Maximizes the margin width.
   - Smaller $\|\vec{w}\|$ corresponds to a larger margin.
2. $\sum_{i=1}^{N}\xi_i^k$:
   - Penalizes the total slack ξ_i, which measures the margin violation for each point.
   - $\xi_i = 0$: Point lies on the correct side of the margin.
   - $0 < \xi_i \leq 1$: Point lies inside the margin but is correctly classified.
   - $\xi_i > 1$: Point is misclassified.
3. C (Regularization Hyperparameter):
   - Controls the trade-off between margin maximization and misclassification.
   - Higher C: Prioritizes minimizing classification error over maximizing the margin (less tolerant to violations).
   - Lower C: Prioritizes maximizing the margin, allowing more misclassification (more tolerant to violations).



C = 1

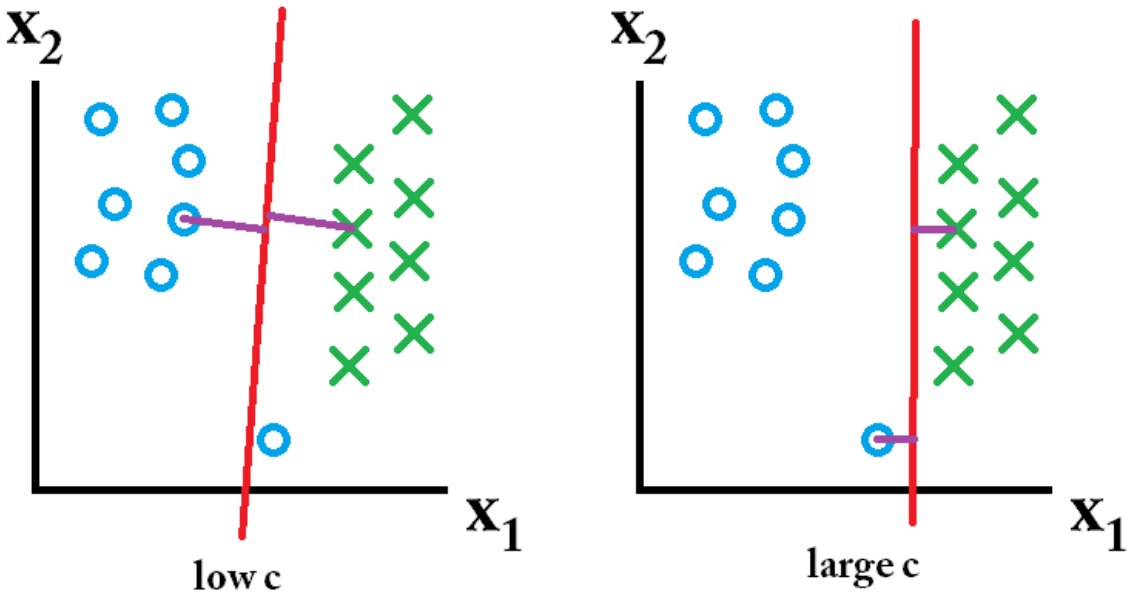


C = 100

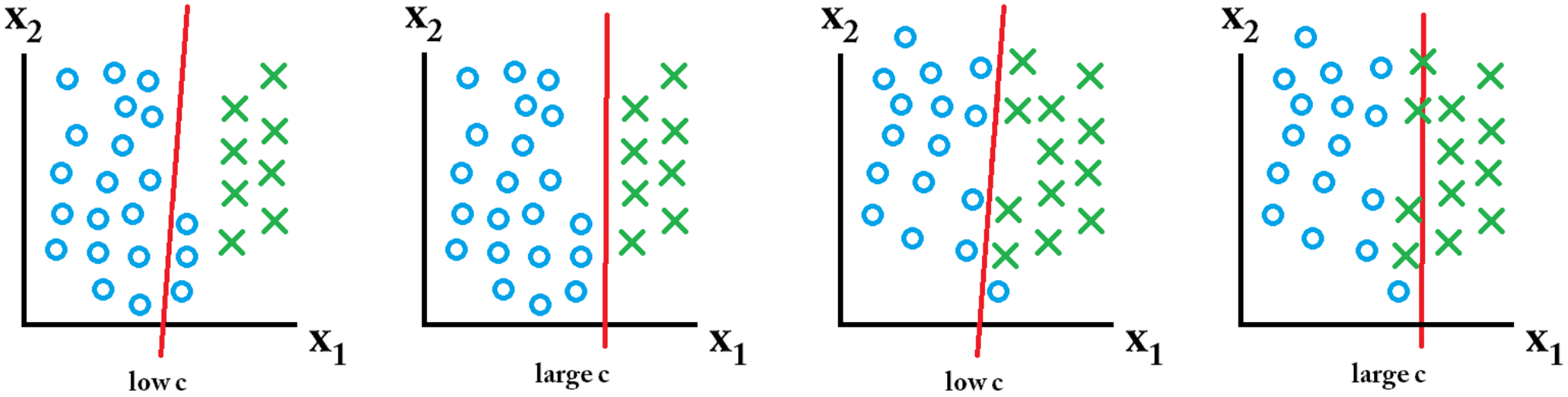
(cf https://learnopencv.com/svm-using-scikit-learn-in-python/)

- For large values of C, → look out for overfitting !
  the optimization will choose a smaller-margin hyperplane
  if that hyperplane does a better job of
  getting all the training points classified correctly.

- Conversely, a very small value of C
  will cause the optimizer to look for a larger-margin hyperplane
  even if that hyperplane misclassifies more points.



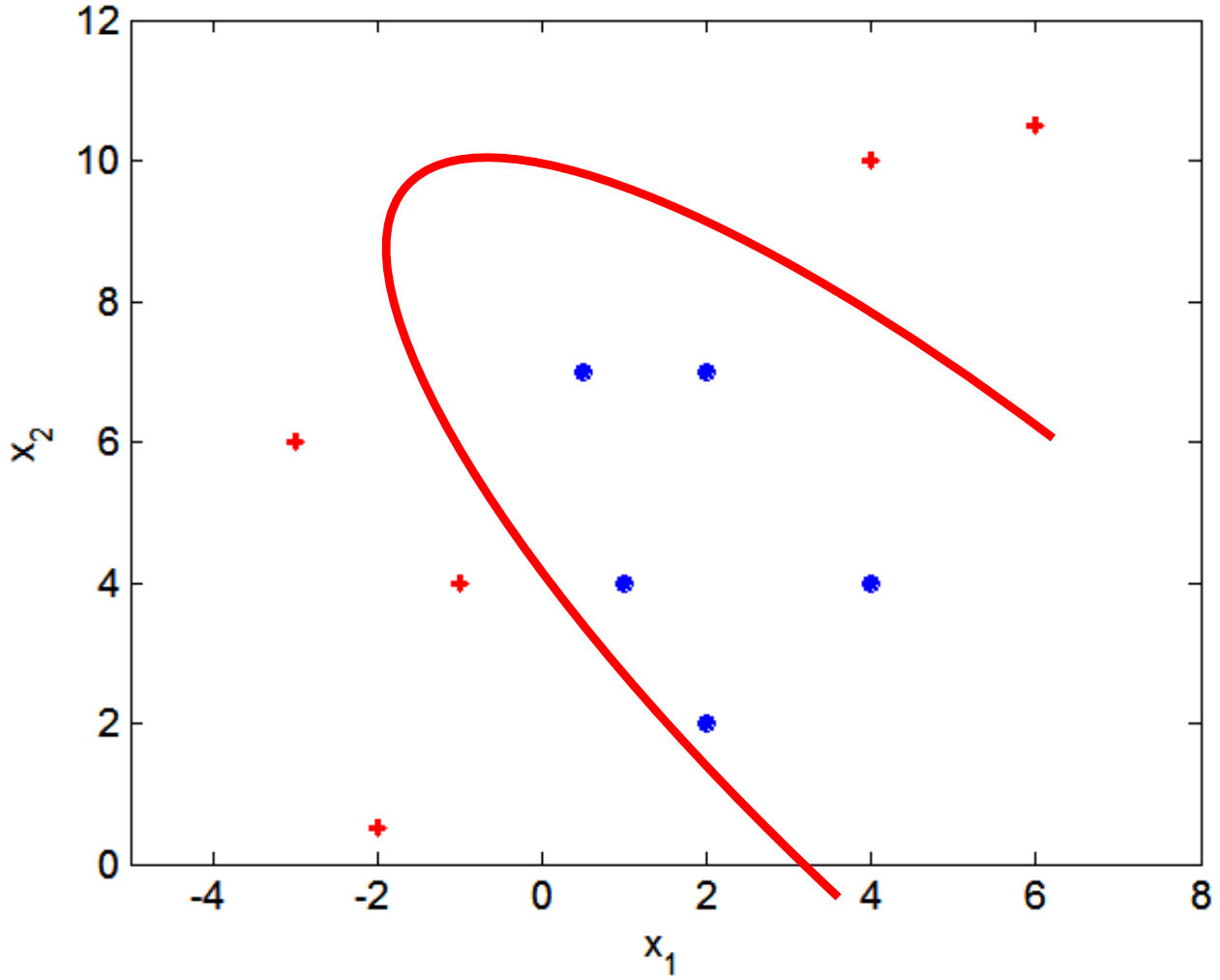
training



test

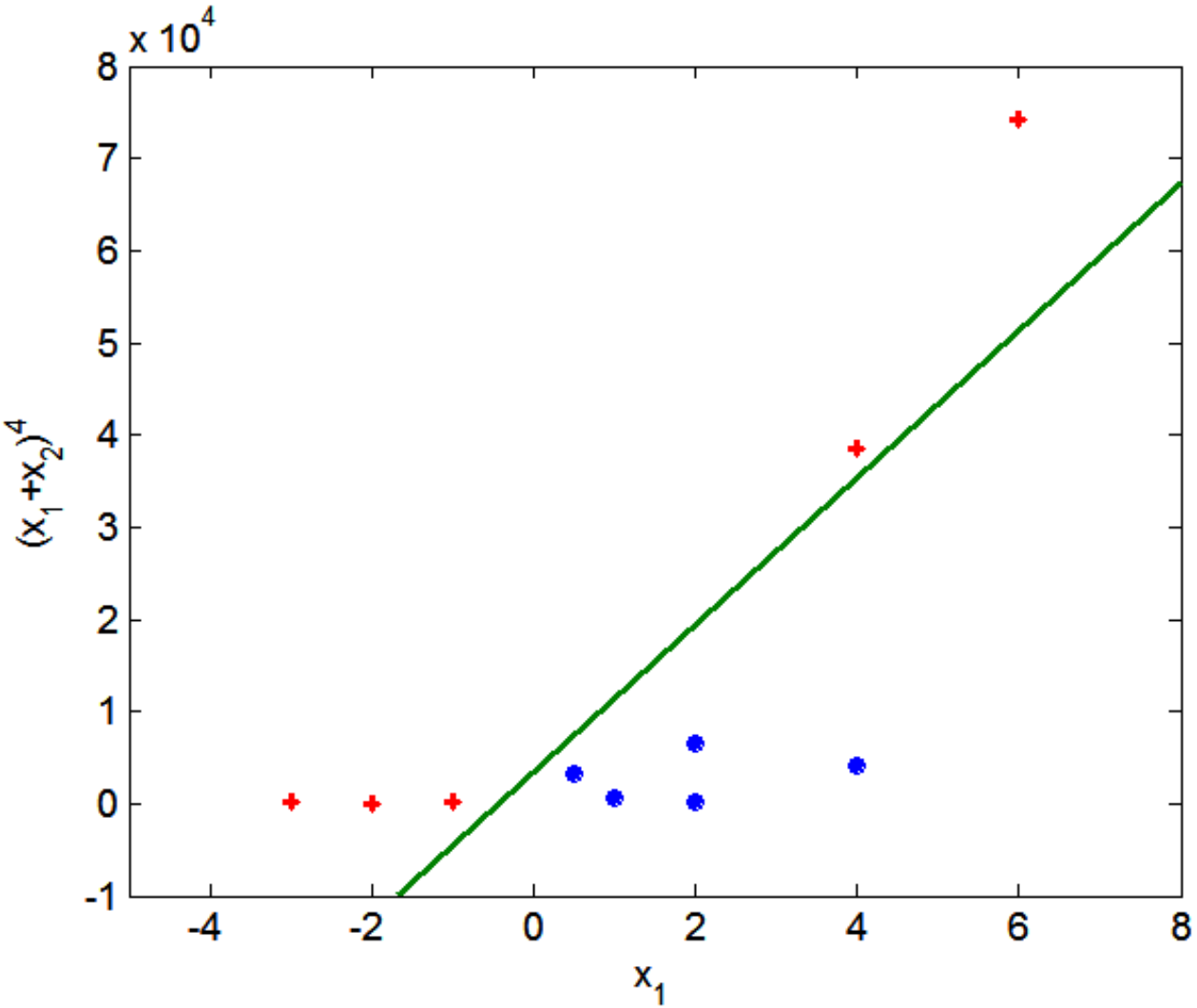
Nonlinear Support Vector Machine

- What if decision boundary is not linear?



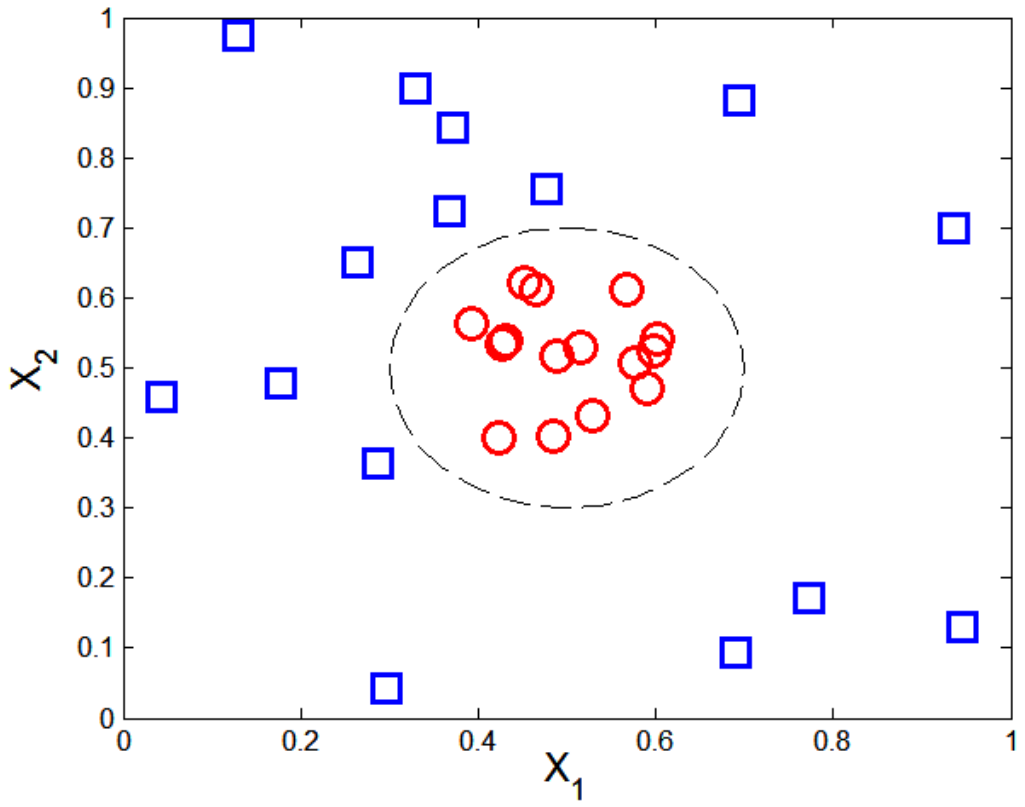
Nonlinear Support Vector Machine (cont.)

- Transform data into higher dimensional space



Nonlinear Support Vector Machine (cont.)

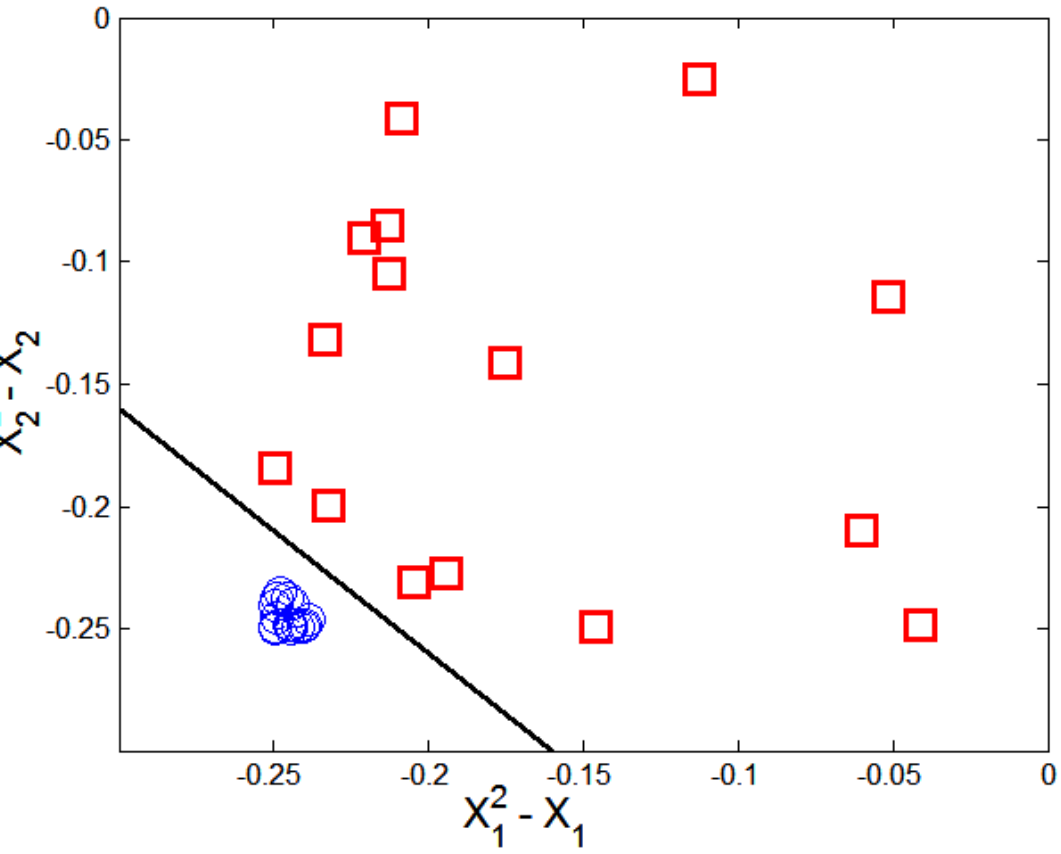
- What if decision boundary is not linear?



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

Nonlinear Support Vector Machine

- Transform data into higher dimensional space



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

Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

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

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

↳ a mapping function

Learning Nonlinear SVM

- Optimization problem:

$$\min_{w} \frac{\|\mathbf{w}\|^2}{2}$$
$$subject\ to \qquad y_i(\boldsymbol{w} \cdot \Phi(\boldsymbol{x}_i) + b) \geq 1,\ \forall\{(\boldsymbol{x}_i, y_i)\}$$

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

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

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

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

Learning Nonlinear SVM (cont.)

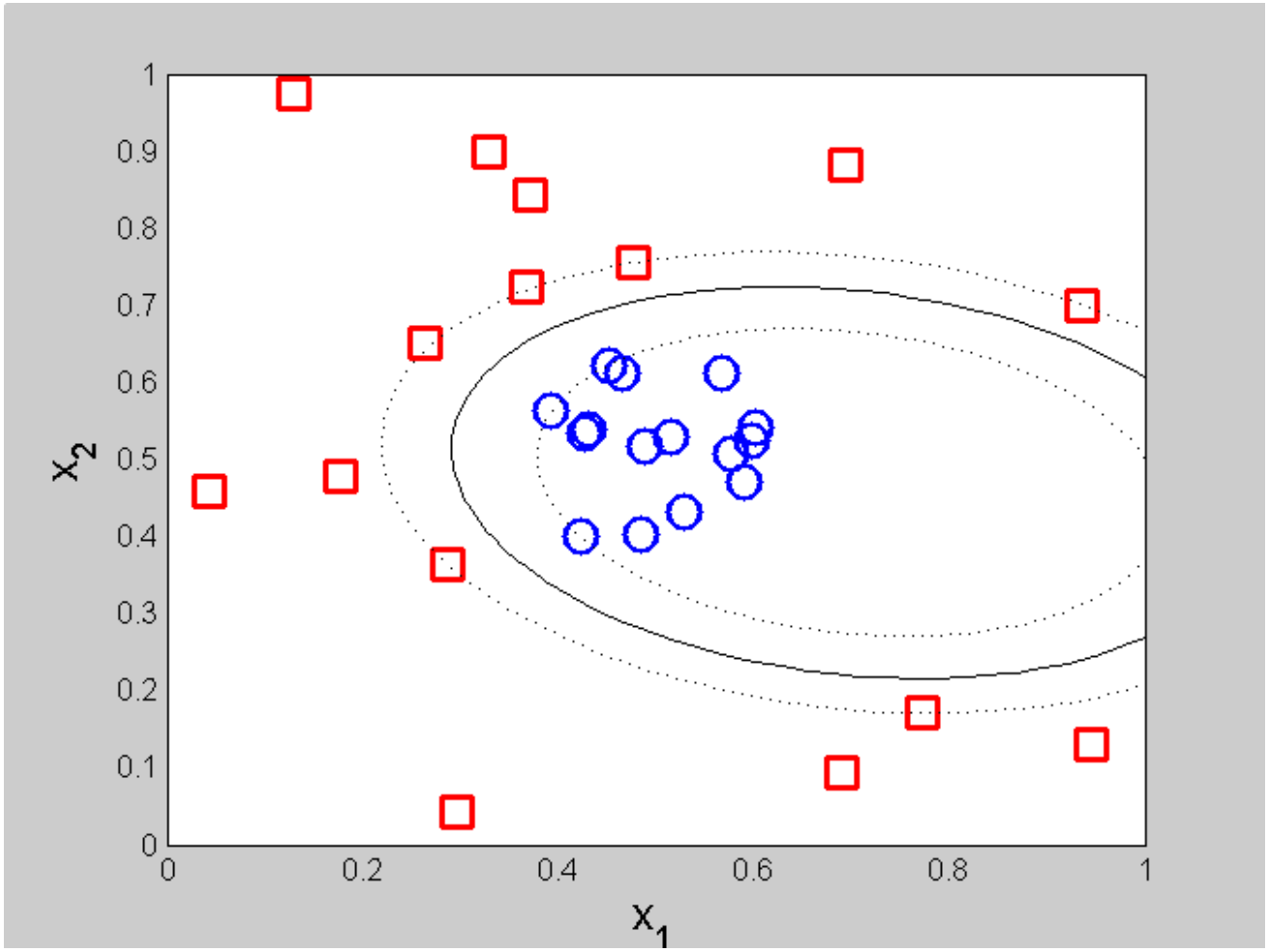
- Issues:
  - What type of mapping function Φ should be used? (It's not clear what type of mapping function Φ should be used to ensure a linear decision boundary)
  - How to do the computation in high dimensional space?
    - Most computations involve dot product $\Phi(x_i) \bullet \Phi(x_j)$
    - Curse of dimensionality?

Learning Nonlinear SVM: Kernel Trick

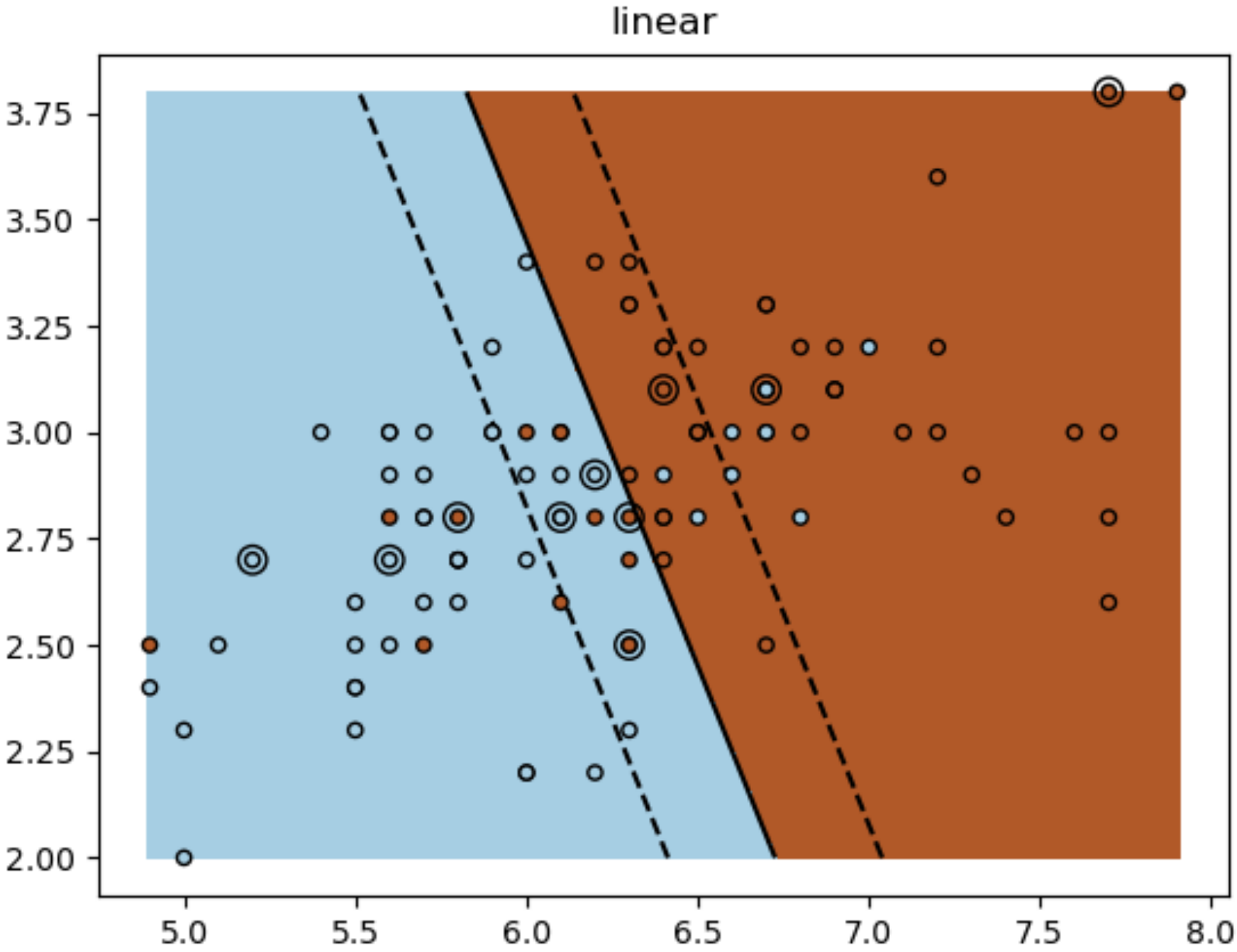
- Kernel Trick
  - $\Phi(x_i) \bullet \Phi(x_j) = K(x_i, x_j)$
  - $K(x_i, x_j)$ is a kernel function (expressed in terms of the coordinates in the original space)
    - Examples:
      - linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.
      - polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d, \gamma > 0$.
      - radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2), \gamma > 0$.
      - sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$.

Gaussian kernel

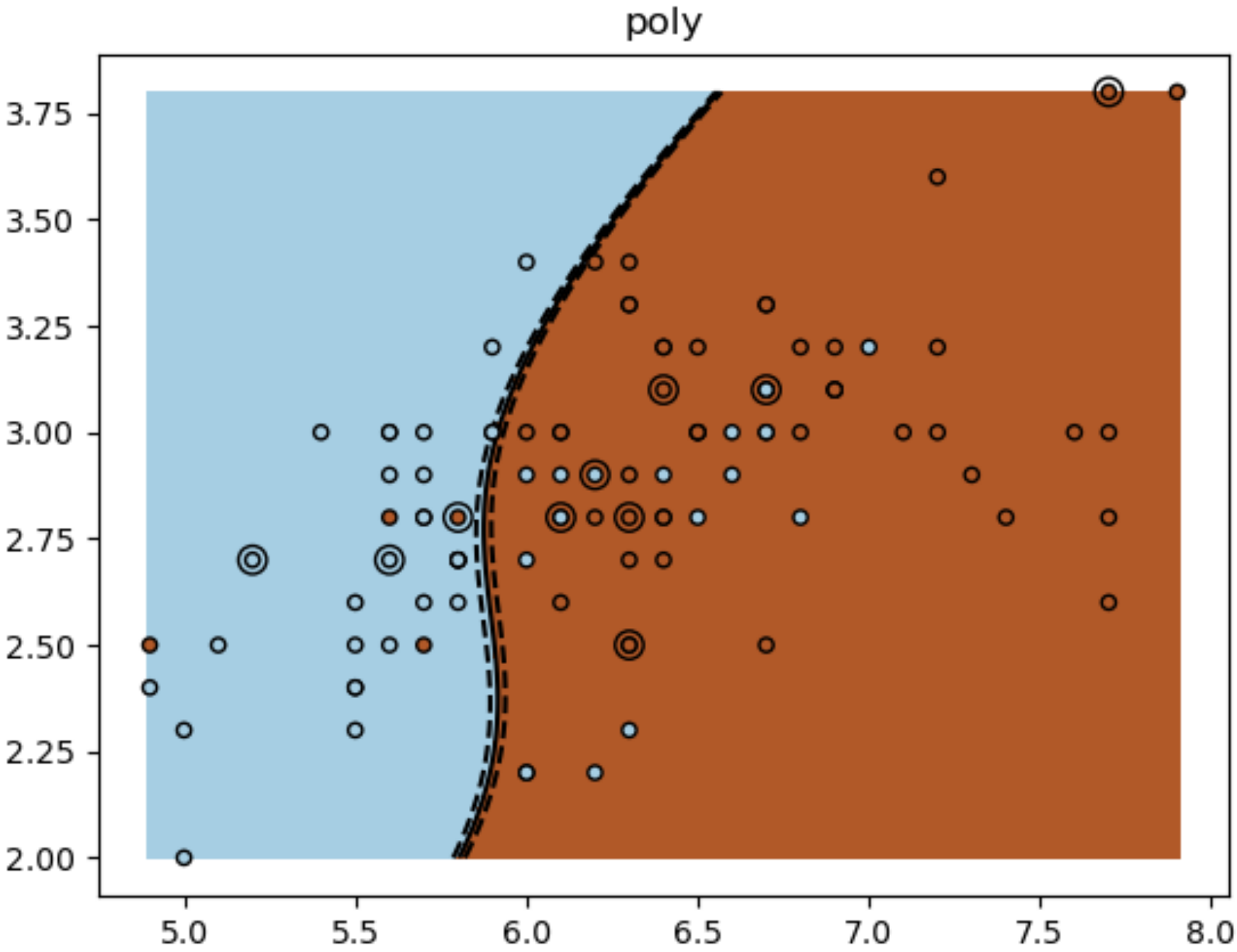
Example of Nonlinear SVM



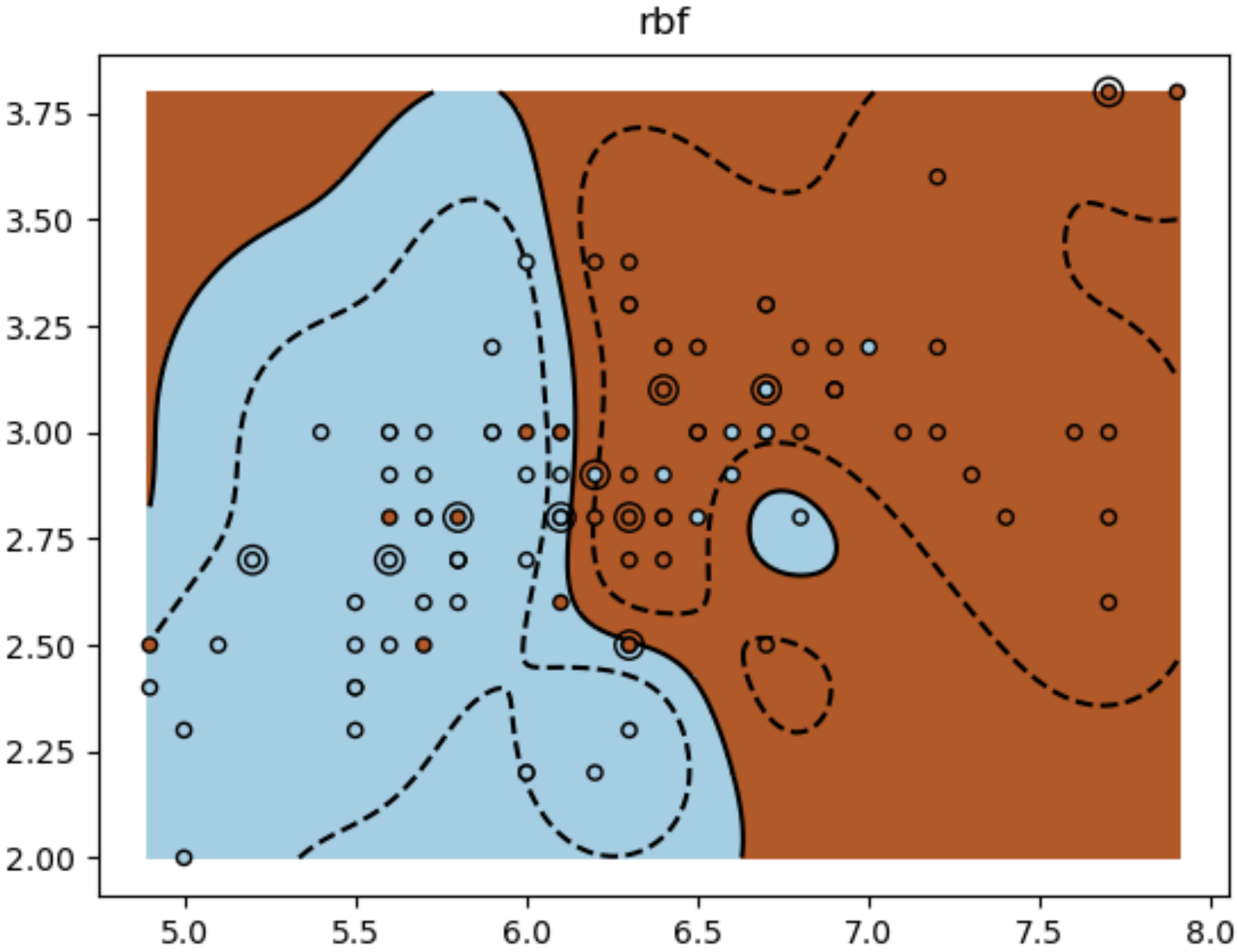
SVM with polynomial degree 2 kernel



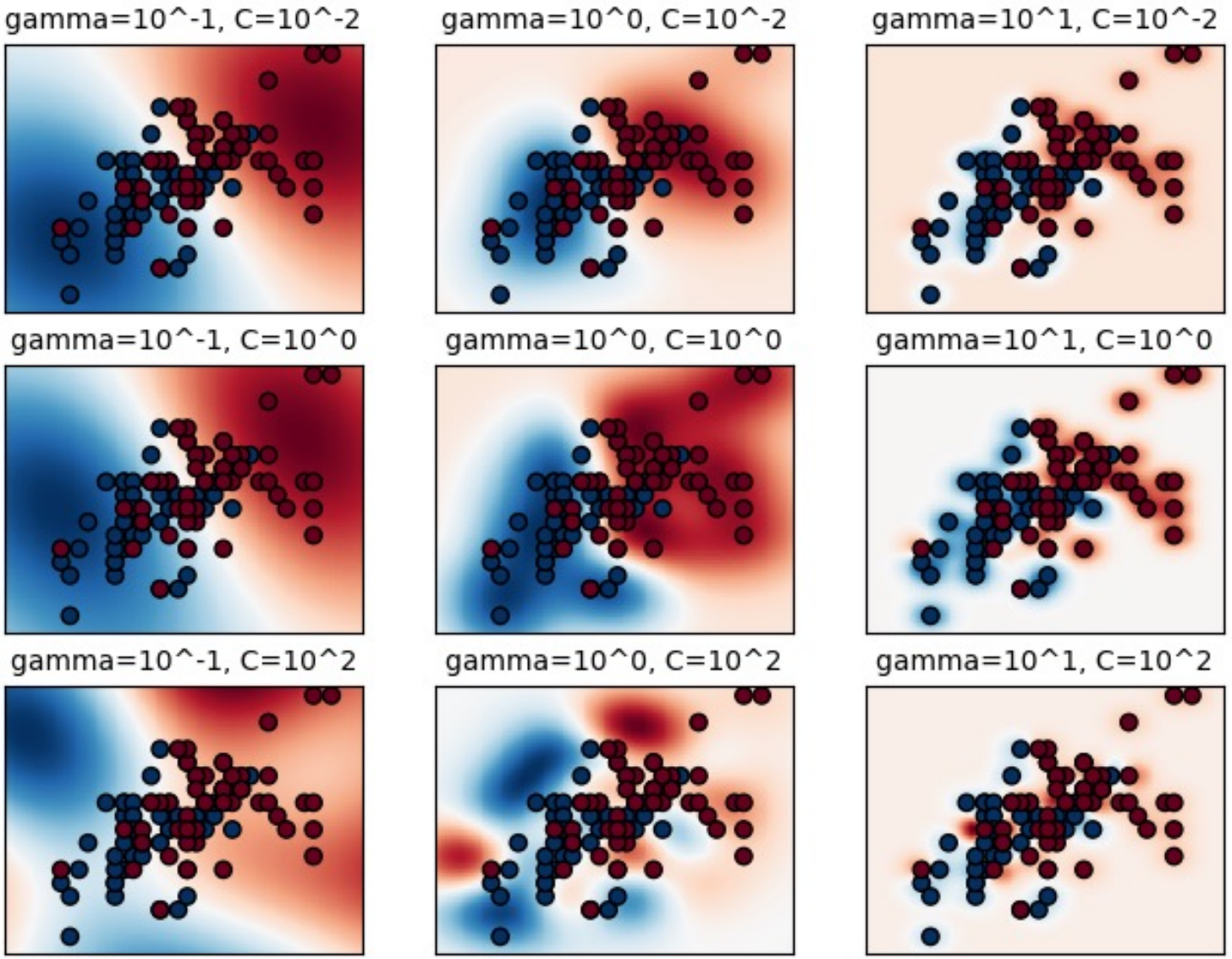
Ref: https://scikit-learn.org/stable/auto_examples/exercises
/plot_iris_exercise.html#sphx-glr-auto-examples-exercises-plot-iris-exercise-py



Ref: https://scikit-learn.org/stable/auto_examples/exercises
/plot_iris_exercise.html#sphx-glr-auto-examples-exercises-plot-iris-exercise-py



Ref: https://scikit-learn.org/stable/auto_examples/exercises
/plot_iris_exercise.html#sphx-glr-auto-examples-exercises-plot-iris-exercise-py



The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.

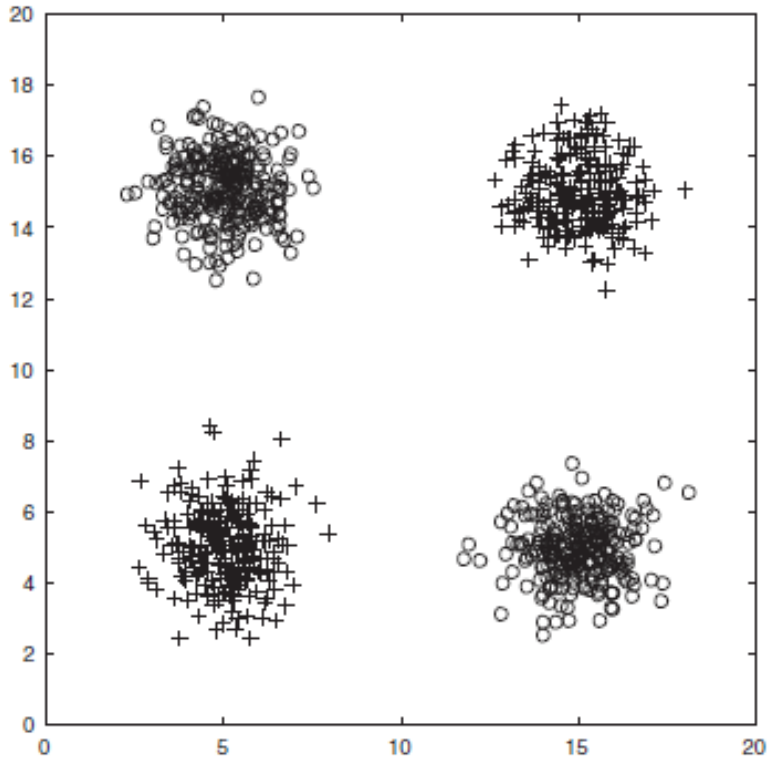
If gamma is too large, the radius of the area of influence of the support vectors only includes the support vector itself

When gamma is very small, the model is too constrained and cannot capture the complexity or “shape” of the data.

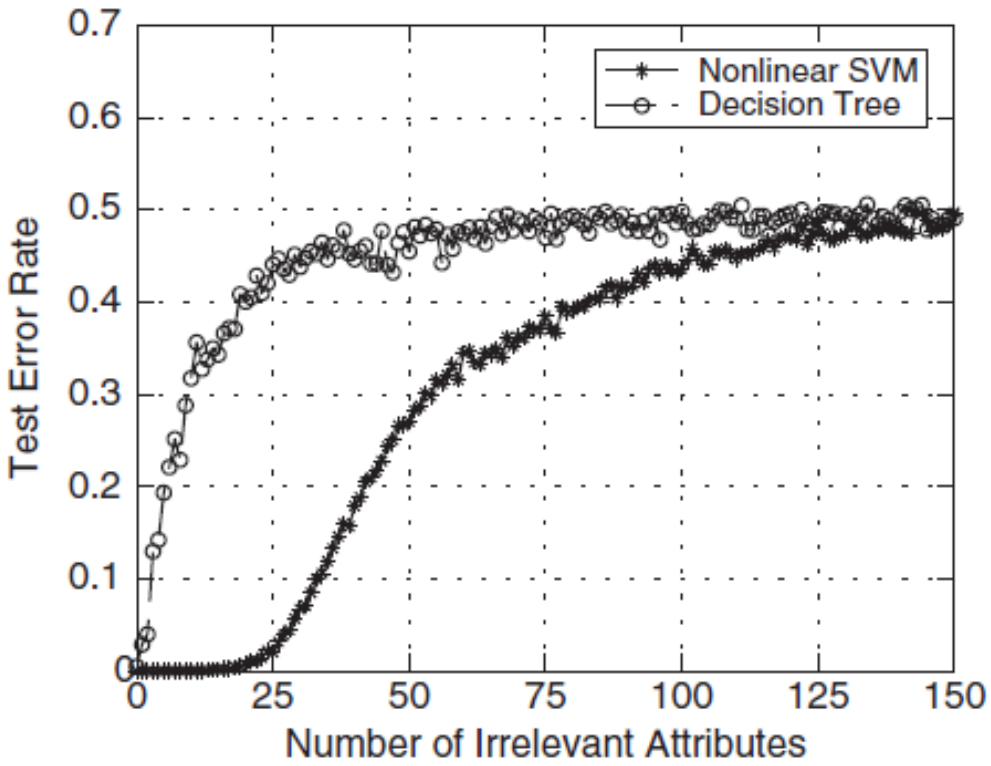
The region of influence of any selected support vector would include the whole training set.

Characteristics of SVM

- The 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 and provides an effective way of regularizing the model parameters by maximizing the margin of the decision boundary (overfitting)
- SVM can handle irrelevant and redundant attributes better than many other techniques
- The user needs to provide the type of kernel function and cost function



(a) Original data with two attributes.



(b) Test error rates after adding irrelevant attributes to the original data.

Figure 4.41. Comparing the effect of adding irrelevant attributes on the performance of nonlinear SVMs and decision trees.

Characteristics of SVM (cont.)

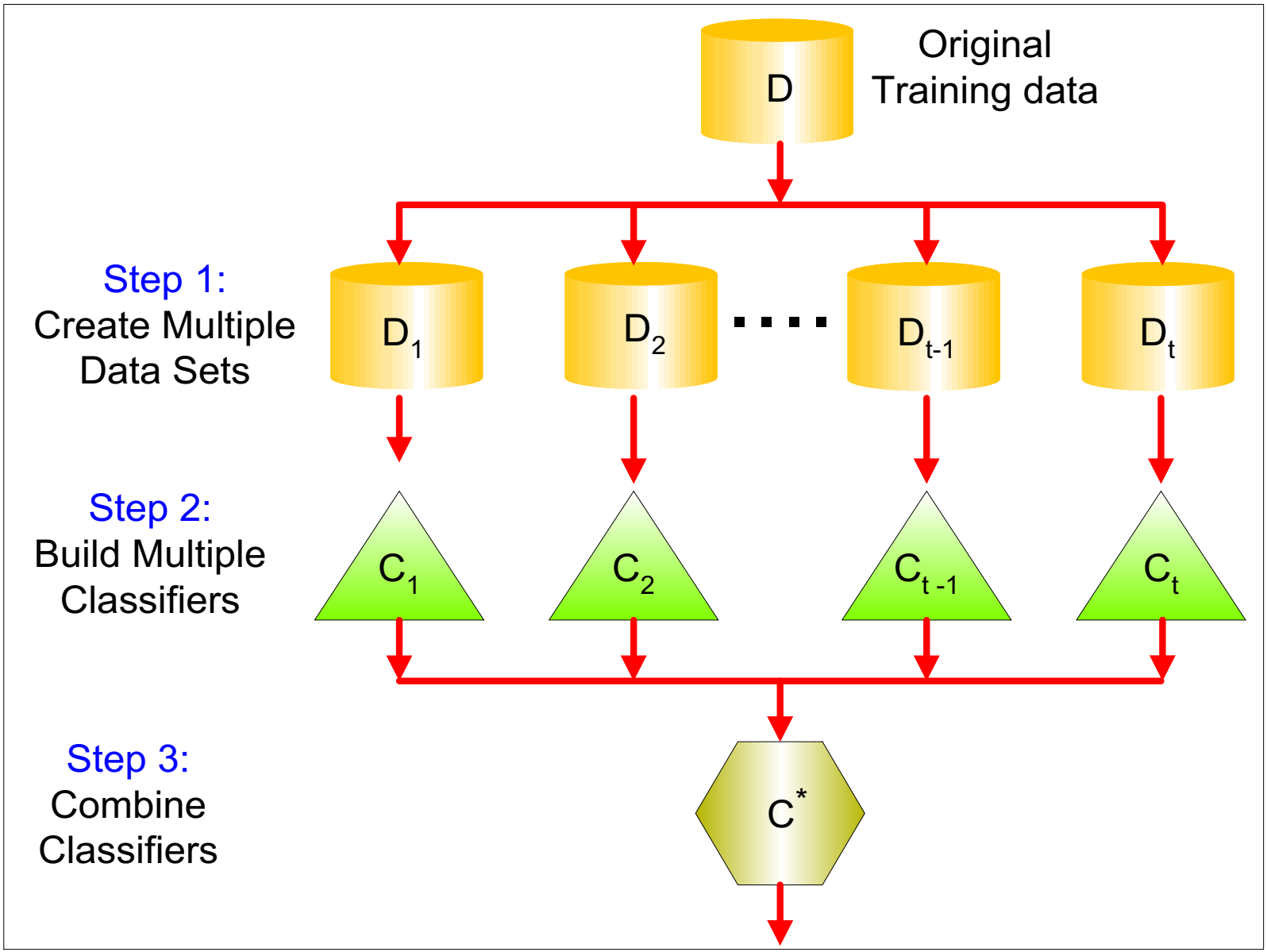
- Can be applied to categorical attribute by introducing dummies for each categorical attribute value present in the data (one hot encoding).
- Multi-class extensions of SVM have also been proposed.
- Although the training time of an SVM model can be large, the learned parameters can be succinctly represented with the help of a small number of support vectors, making the classification of test instances quite fast.

Ensemble Classification

Ensemble Classification

- Ensemble:
  - Two heads are better than one
  - Constructs a set of base classifier from training data
  - performs classification by taking a vote on the predictions made by each base classifier
  - The class can be obtained by taking a majority vote on the individual prediction or by weighting each prediction with the accuracy of the base classifier.

Logical View of Ensemble



Can Ensemble Performs Better ?

- Two necessary condition for an ensemble classifier to perform better than a single classifier
  - The base classifier should be independent
  - The base classifier should perform better than random guessing

- In practice, it is difficult to ensure total independence among the base classifier.

low correlation

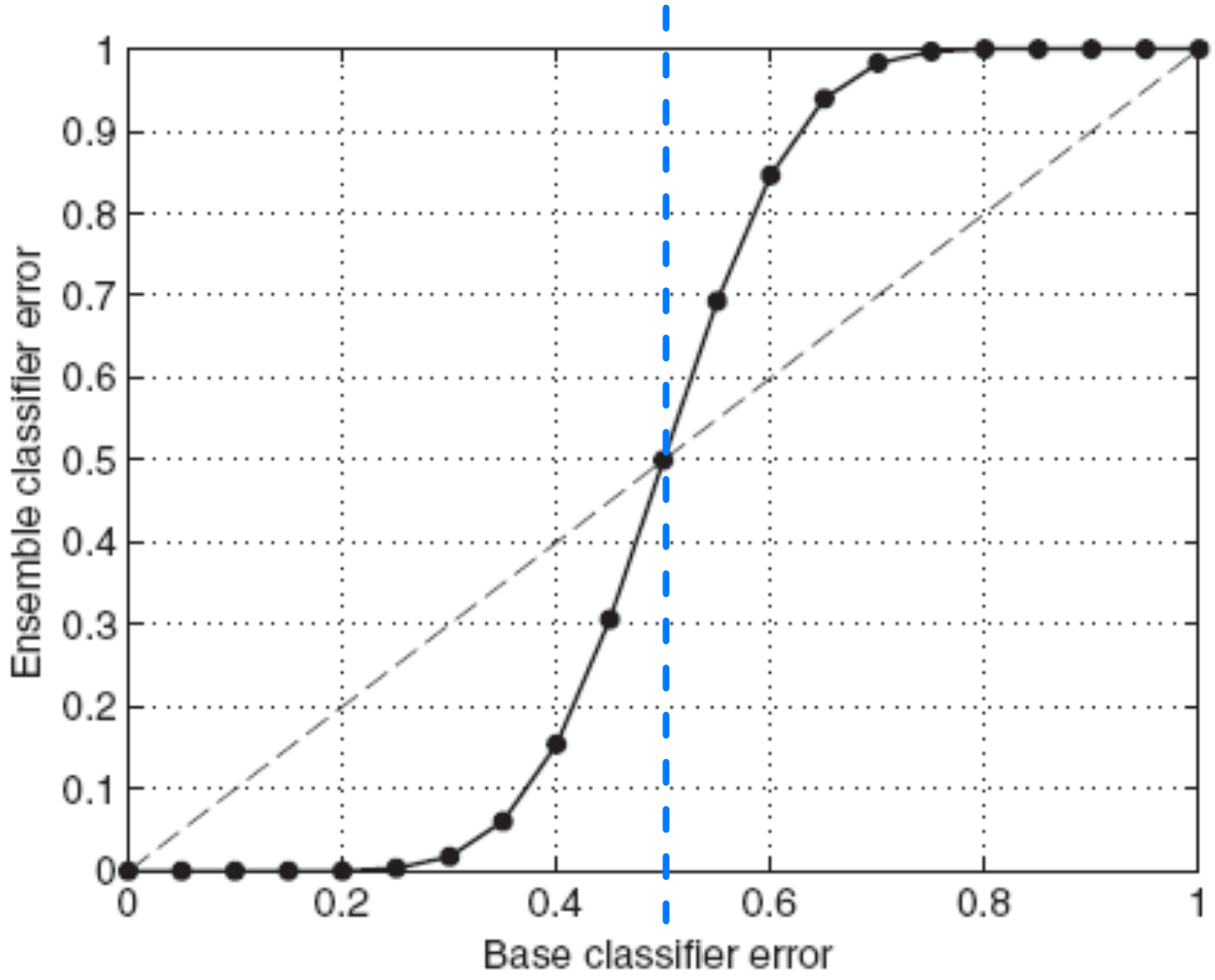
- Improvements in classification accuracies have been observed in ensemble in which the base classifiers are slightly correlated.

Can Ensemble Performs Better ?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06 \quad < 0.35 = \varepsilon$$

Can Ensemble Performs Better ?



Construction Methods of Ensemble

1. By manipulating the training set
2. By manipulating the input features
3. By manipulating the class labels
4. By manipulating the learning algorithm

Manipulating Training Set

- Multiple training sets are created by sampling the original data according to some sampling distribution.
- The sampling distribution determines how likely it is that an example will be selected for training & it may vary from one trial to another
- A classifier is built from each training set using a particular learning algorithm.
- Bagging, Boosting

Manipulating Input Features

- A subset of input features is chosen to form each training set.
- The subset can be either chosen randomly or based on the recommendation of domain experts.
- Works very well with data sets that contain highly redundant features.
- Random forest

Manipulating Class Labels

- Used when the number of classes is sufficiently large.
- Training data is transformed into a binary class problem by randomly partitioning the class labels into two disjoint subsets A0, & A1.
- Relabeled examples are then used to train a base classifier.
- By repeating the class relabeling and model building steps multiple times, an ensemble of base classifiers is obtained.
- When a test is presented, each base classifier is used to predict its class label.
- If the test is predict as class 0 (1), all the classes that belong to A0 (A1) will receive a vote.
- Error-correcting output coding (ECOC)

Manipulating Learning Algorithm

- Many learning algorithm can be manipulated in such a way that applying the algorithm several times on the same training data may result in different models.
- Ensemble of decision trees can be constructed by injecting randomness into tree-growing procedure.
- Instead of choosing the best splitting attribute at each node, we can randomly choose one of the top k attributes for splitting.

Ensemble classification

- General procedure for ensemble method

1.Let D denote the original training data , k denote the number of base classifiers and T be the test data

2.For i=1 to k do

Create training set,D_i from D

Build a base classifier C_i from D_i

end for

3.For each test record x T do

$C^*(x)=Vote(C_1(x),C_2(x),.....C_k(x))$

4.End for

Bagging (Bootstrap AGGregatING)

- Bagging is manipulated by training data
- Repeatedly samples (with replacement) from a data set according to a uniform probability distribution
- Each bootstrap sample has the same size as the original training data
- In each round, some instances appear several times, some may omitted from the training data : introduce variability
- A base classifier is built on each bootstrap sample
- Each sample has probability $1-(1 - 1/n)^n$ of being selected in each round (if n is sufficiently large, converges to 0.632)

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Bagging (Bootstrap AGGregatING)

- Bootstrap sampling: sampling with replacement

Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

- Build classifier on each bootstrap sample

- Probability of a training instance being selected in a bootstrap sample is:
  - $1 – (1 - 1/n)^n$ (n: number of training instances)
  - ~0.632 when n is large

Derivation:

1. **Dataset Size (n)**:
   - Assume the dataset contains n data points.
2. **Probability of Not Selecting a Specific Point in One Draw**:
   - Each data point has a $1/n$ chance of being selected. Thus, the probability of **not** selecting a specific point in one draw is:

$$1 - \frac{1}{n}$$

3. **Probability of Not Selecting a Specific Point in n Draws**:
   - A bootstrap sample involves n draws with replacement. The probability of **not selecting** a specific point in n draws is:

$$\left(1 - \frac{1}{n}\right)^n$$

4. **Probability of Selecting the Point at Least Once**:
   - The complement of the above probability is the probability of **selecting the point at least once** in n draws:

$$1 - \left(1 - \frac{1}{n}\right)^n$$

5. **Convergence as $n \to \infty$**:
   - As n becomes large, the term $\left(1 - \frac{1}{n}\right)^n$ approaches e^{-1}, where e is the base of the natural logarithm ($e \approx 2.718$).
   - Thus, the probability of selecting the data point at least once converges to:

$$1 - e^{-1} \approx 1 - 0.367 = 0.632$$

Bagging Algorithm

Algorithm 4.5 Bagging algorithm.

1: Let k be the number of bootstrap samples.
2: **for** $i = 1$ to k **do**
3: Create a bootstrap sample of size N, D_i.
4: Train a base classifier C_i on the bootstrap sample D_i.
5: **end for**
6: $C^*(x) = \underset{y}{\text{argmax}} \sum_i \delta\big(C_i(x) = y\big)$. // C^*: ensemble classifier

$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise.$\}$

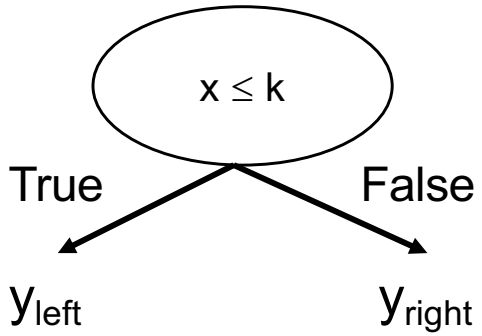
Bagging Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump (decision tree of size 1)
  - Decision rule: $x \leq k$ vs. $x > k$
  - Split point k is chosen based on entropy



Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.5	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

n

Bagging Example

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	1	1	1	1	1

x <= 0.65 ==> y = 1
x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

x <= 0.3 ==> y = 1
x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ==> y = 1
x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ==> y = -1
x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ==> y = -1
x > 0.05 ==> y = 1

Bagging Example

- majority vote (sign of sum of predictions)

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Boosting

- An iterative procedure used to adaptively change the distribution of training examples
- The based classifiers will focus on examples that are hard to classify
- Boosting assigns a weight to each training example & may adaptively change the weight at the end of each round.
- Weights can be used
  - as a sampling distribution to draw a set of bootstrap samples from the original data
  - by the base classifier to learn a model that is biased toward higher-weight examples.

Boosting (cont.)

- Initially, the examples are the same weights 1/N
- Drawn a sample according to the distribution to obtain a new training set
- A classifier is induced from the training set and used to classify all the examples
- Increase the weight of misclassified examples decrease the weight of correctly classy ones
- Update the weight, do again

Boosting (cont.)

- Records that are wrongly classified will have their weights increased in the next round
- Records that are classified correctly will have their weights decreased in the next round

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Boosting (cont.)

- The implementations of the boosting algorithms are different in two aspects:
  - How the weights of the training example are updated at the end of each boosting round
  - How the predictions made by each classifier are combined (i.e., AdaBoost is not a majority voting scheme)

AdaBoost

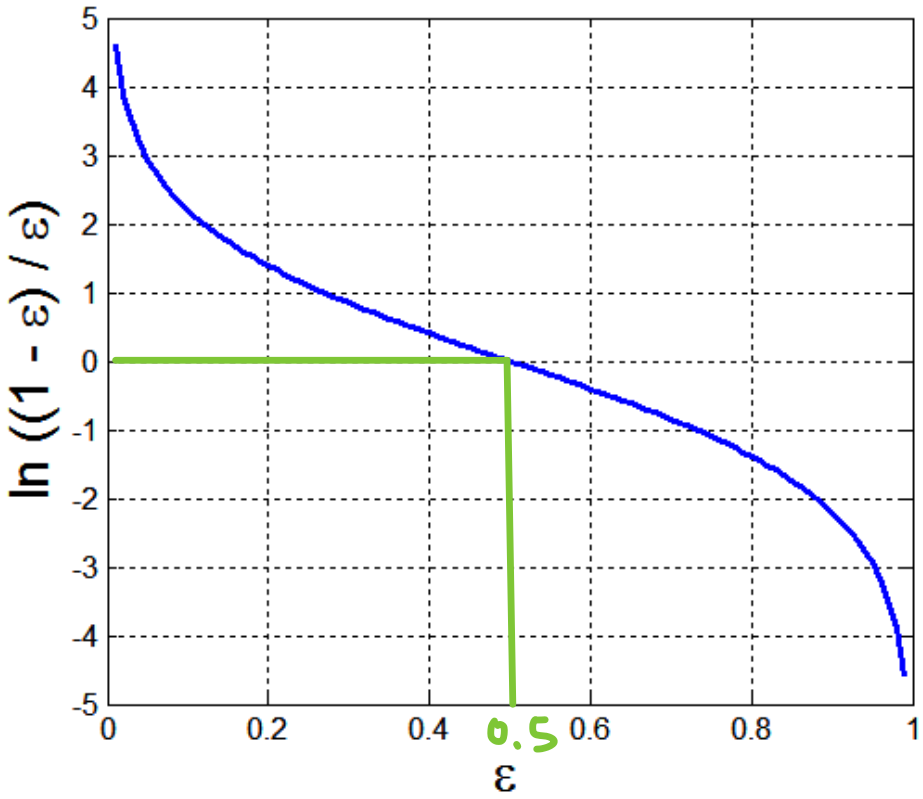
- Base classifiers: C_1, C_2, …, C_T
- Error rate of a base classifier:

$$\epsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j^{(i)}\,\delta(C_i(x_j) \neq y_j)$$

- Importance of a classifier:

(Amount of Say)

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Base Classifiers $C_1, C_2, ..., C_T$:

- Weak classifiers, often simple models like decision stumps, are trained iteratively on weighted datasets.

Error Rate of a Base Classifier (ϵ_i):

$$\epsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j^{(i)} \delta(C_i(x_j) \neq y_j)$$

- $w_j^{(i)}$: Weight of the j-th example at iteration i.
- $\delta(C_i(x_j) \neq y_j)$: Indicator function, 1 if the prediction $C_i(x_j)$ is incorrect, 0 otherwise.
- ϵ_i: Weighted proportion of misclassified examples.

Classifier Importance (α_i):

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$$

- Represents the weight/importance of classifier C_i.
- A smaller error rate ϵ_i results in a higher α_i, emphasizing the classifier's contribution.

AdaBoost Algorithm

- Weight update:

Comparison { w: sample weight; α: classifier importance

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

Where Z_i is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated
- Classification:

$$C^*(x) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

2. Weight Update Rule

After training a base classifier C_i, the sample weights are updated to focus more on misclassified examples.

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i}, & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i}, & \text{if } C_i(x_j) \neq y_j \end{cases}$$

Explanation:

- Z_i: Normalization factor ensuring that $\sum_j w_j^{(i+1)} = 1$.
- $e^{-\alpha_i}$: Reduces weight for correctly classified examples.
- e^{α_i}: Increases weight for misclassified examples.
- As a result, the next iteration will focus more on difficult examples.

---

3. Final Strong Classifier

At the end of T boosting iterations, the strong classifier is defined as a weighted combination of the weak classifiers:

$$C^*(x) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

- Each weak classifier C_i contributes to the final prediction based on its importance α_i.

AdaBoost Algorithm

Algorithm 4.6 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all examples in the original training set, D.
7: $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \, \delta\big(C_i(x_j) \neq y_j\big)\right]$ {Calculate the weighted error.}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each example according to Equation 4.103.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y)\big)$.

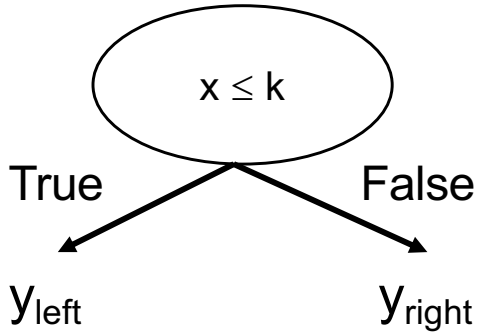
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \leq k$ vs. $x > k$
  - Split point k is chosen based on entropy



AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

- Weights

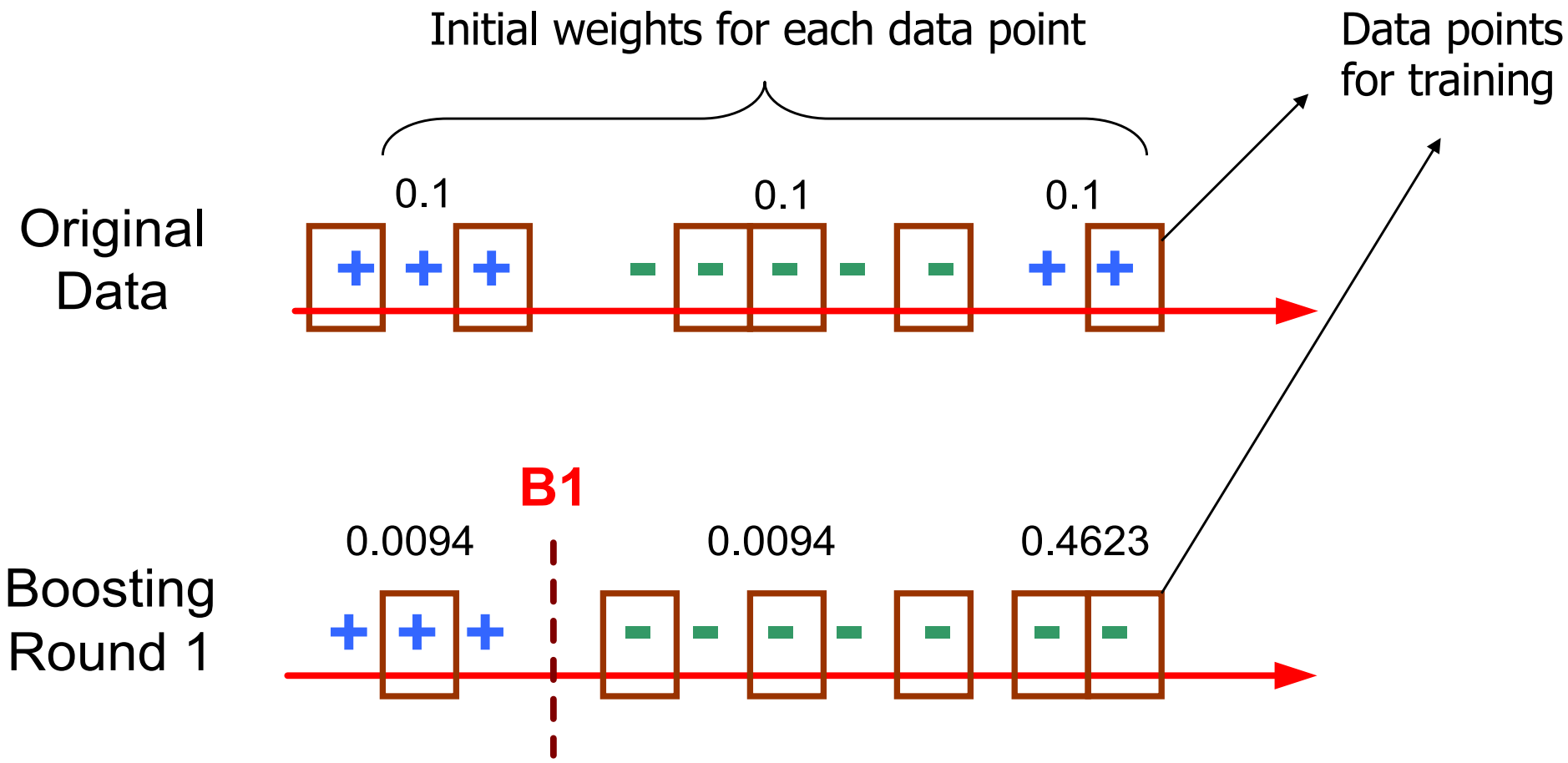
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- Classification

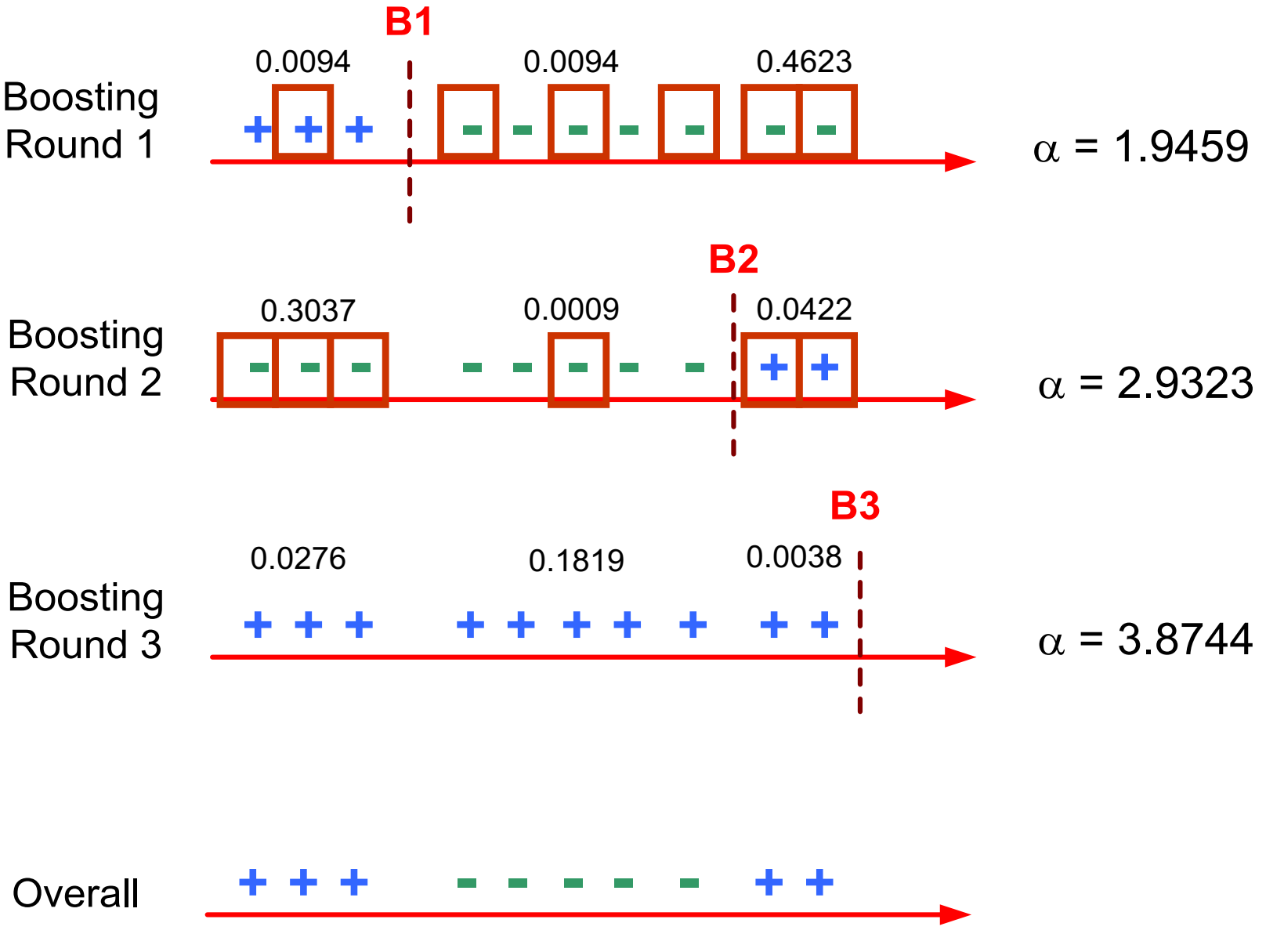
Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Illustrating AdaBoost



Illustrating AdaBoost



AdaBoost (cont.)

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

(a) Training records chosen during boosting

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

(b) Weights of training records

Figure 5.38. Example of boosting.

AdaBoost (cont.)

Round	Split Point	Left Class	Right Class	α
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

(a)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

(b)

Figure 5.39. Example of combining classifiers constructed using the AdaBoost approach.

Random Forest

- Construct an ensemble of decision trees by manipulating training set as well as features
  - Use bootstrap sample to train every decision tree (similar to Bagging)
  - Use the following tree induction algorithm:
    - At every internal node of decision tree, randomly sample *p* attributes for selecting split criterion
    - Repeat this procedure until all leaves are pure (unpruned tree)
- Randomness helps to reduce the correlation among decision trees so that the generalization error of the ensemble can be improved.

Random Forest

- Manipulated by the input feature

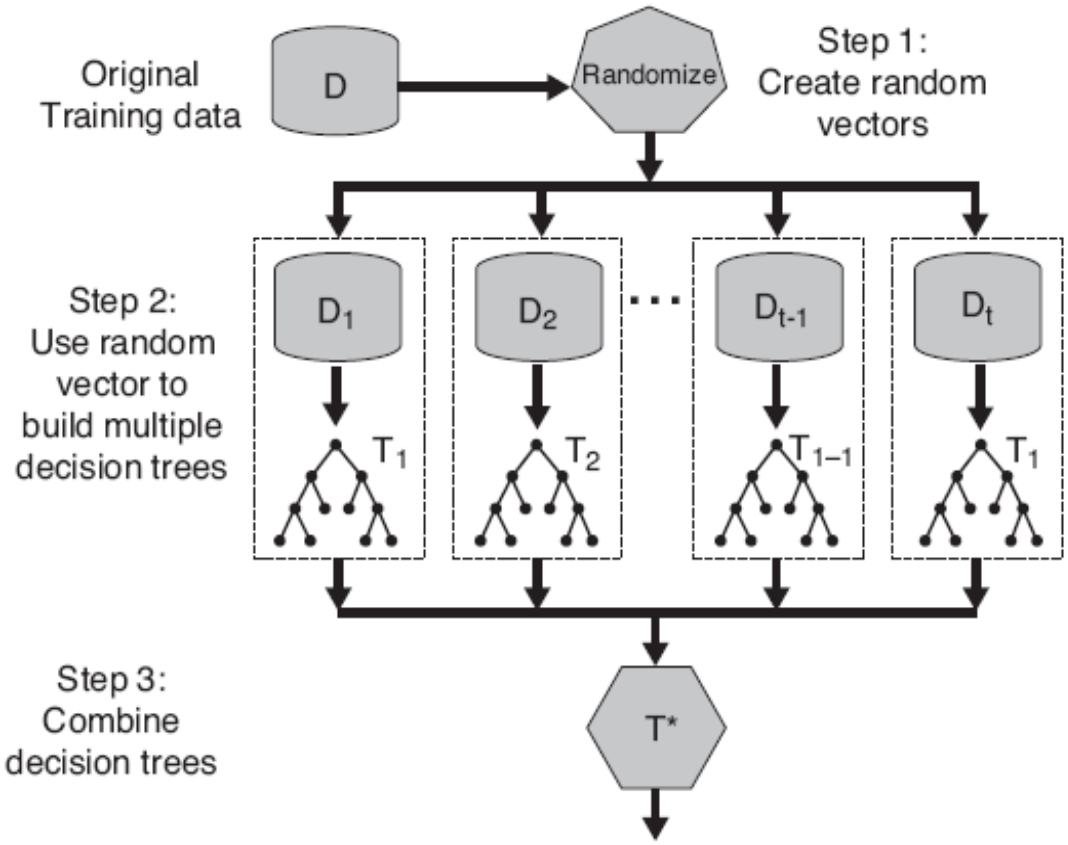


Figure 5.40. Random forests.

Characteristics of Random Forest

- Base classifiers are unpruned trees and hence are *unstable classifiers*
- Base classifiers are *decorrelated* (due to randomization in training set as well as features)
- Random forests reduce variance of unstable classifiers without negatively impacting the bias
- Selection of hyper-parameter p
  - Small value ensures lack of correlation
  - High value promotes strong base classifiers
  - Common default choices: $\sqrt{d}$, $\log_2(d+1)$

→ # (attributes) randomly selected at each split in the decision tree construction

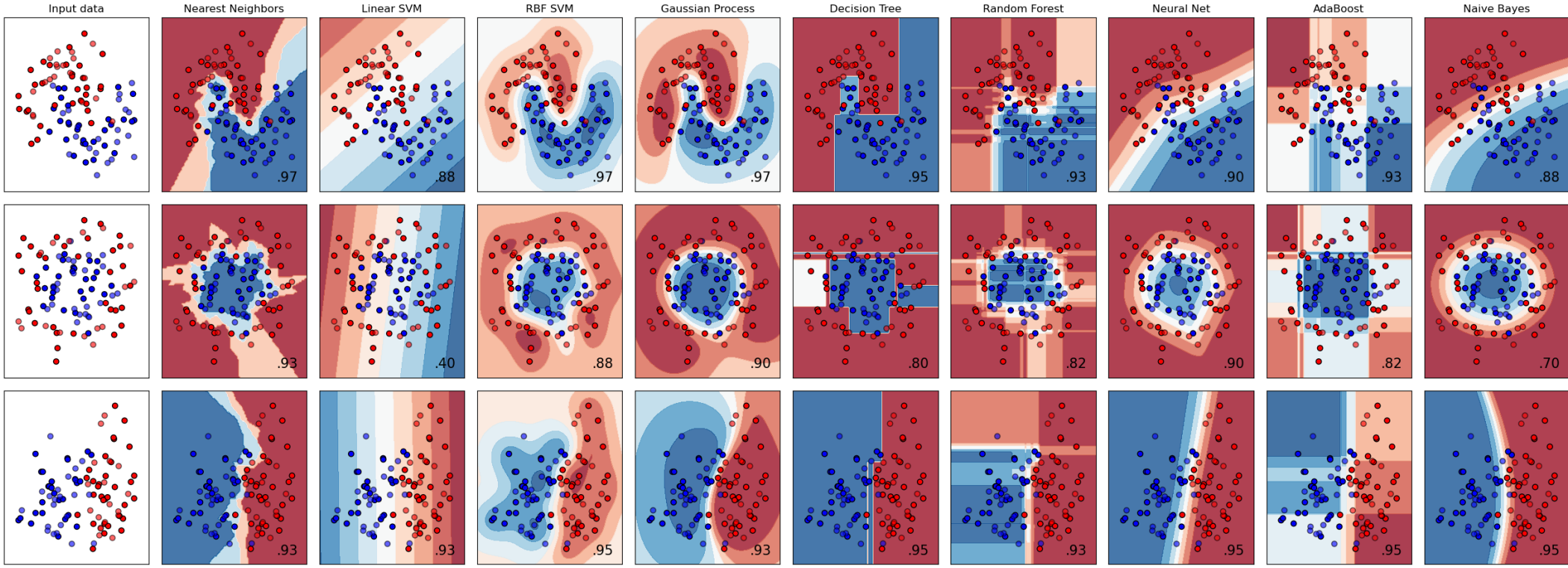
(d: total # of features)

classification tasks

↳ regression tasks

Conclusions

- Classification (Prediction): supervised learning
- Algorithms
  - Decision tree
  - Rule-based
  - Bayesian
  - K-NN
  - SVM
  - Bagging
  - AdaBoost
  - Random Forest



(Ref: https://scikit-learn.org/stable/auto_examples/classification
/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py)