

高等机器学习

# 机器学习基础

秦涛 张卫强



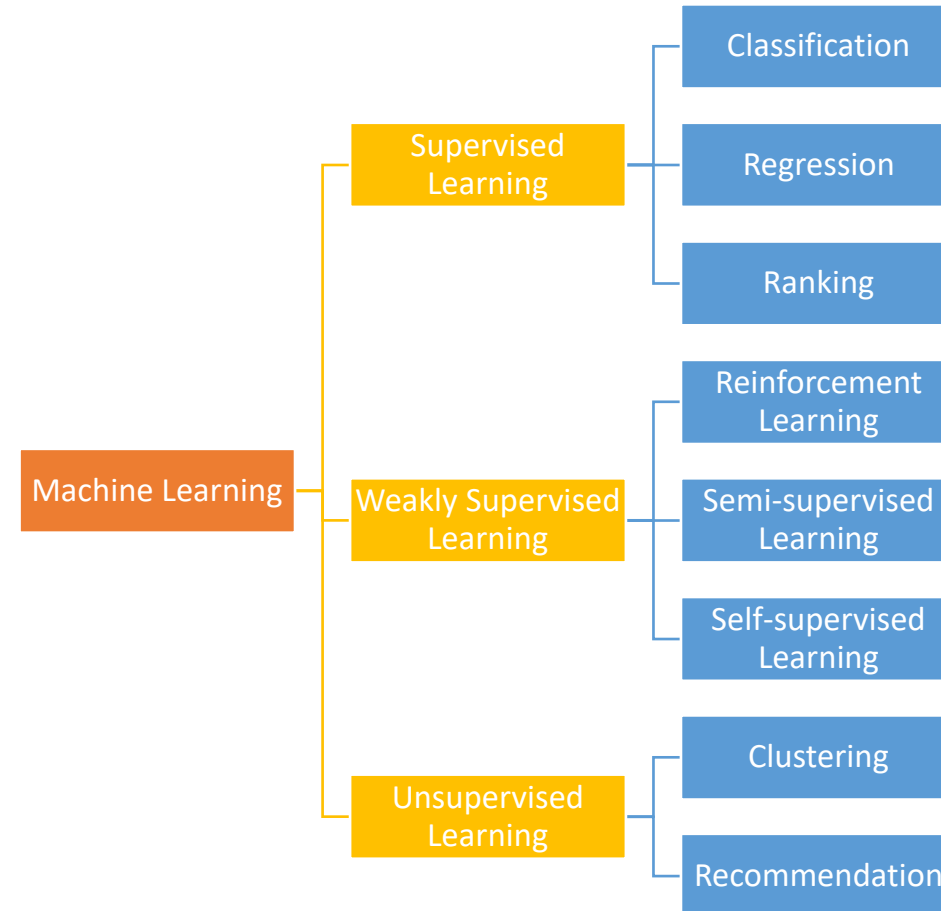
清华大学  
Tsinghua University

# Outline

- Taxonomy of Machine Learning
- Framework of Supervised Learning
- Machine Learning Models
- Model Ensemble

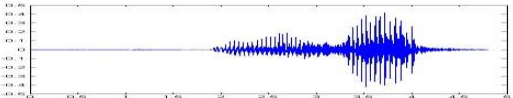
# Taxonomy of Machine Learning

# Machine Learning Taxonomy



# Supervised Learning $\approx$ Looking for a Mapping


- Speech Recognition

$$f(\text{  }) = \text{"How are you"}$$

- Image Recognition

$$f(\text{  }) = \text{"Cat"}$$

- Playing Go

$$f(\text{  }) = \text{"5-5"}_{\text{(next move)}}$$

- Chatbot

$$f(\text{ "Hi" }) = \text{ "Hello" }$$

(what the user said) (system response)





ALL

WORK

IMAGES

VIDEOS

TOOLS

About 43,700,000 results



Microsoft

<https://www.microsoft.com/en-us/research/>

## Microsoft Research AI for Science - Microsoft Research

Nov 14, 2024 · Whether that be through pushing the boundaries of AI to design and optimise new molecules for drug discovery, through to the development of models that will underpin advances in our ability to discover new materials with ...



Microsoft

<https://www.microsoft.com/en-us/research/>

## AI4Science to empower the fifth paradigm of scientific ...

Jul 7, 2022 · AI4Science is an effort deeply rooted in Microsoft's mission, applying the full breadth of our AI capabilities to develop new tools for scientific discovery so that we and others in the scientific community can confront some of ...



Generative Chemistry



Nature

<https://www.nature.com/articles>

## Scientific discovery in the age of artificial intelligence

Aug 2, 2023 · Artificial intelligence (AI) is being increasingly integrated into scientific discovery to augment and accelerate research, helping scientists to generate hypotheses, design...

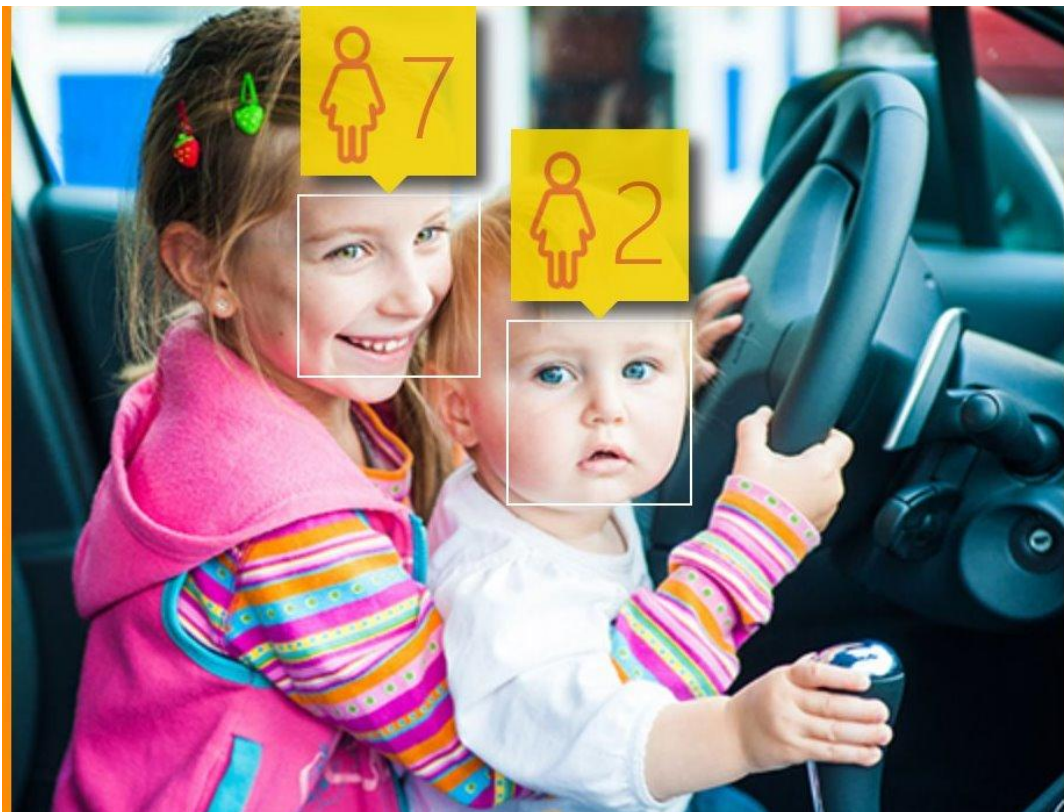


Elsevier

<https://www.elsevier.com/connect/ai-fo...>

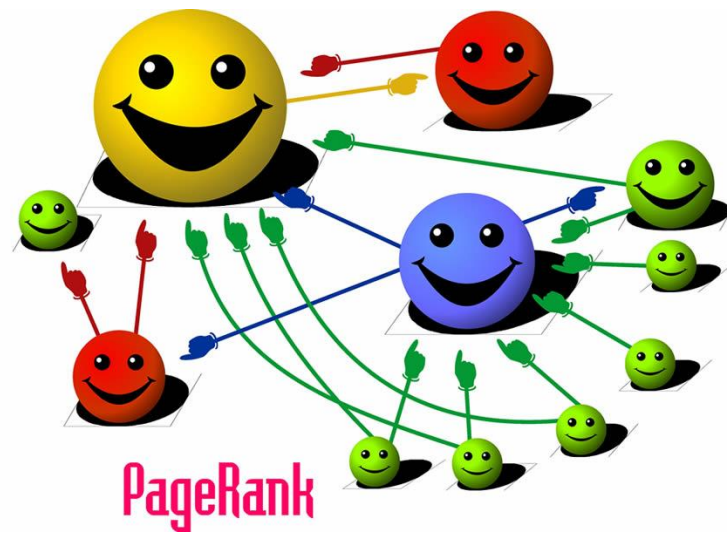
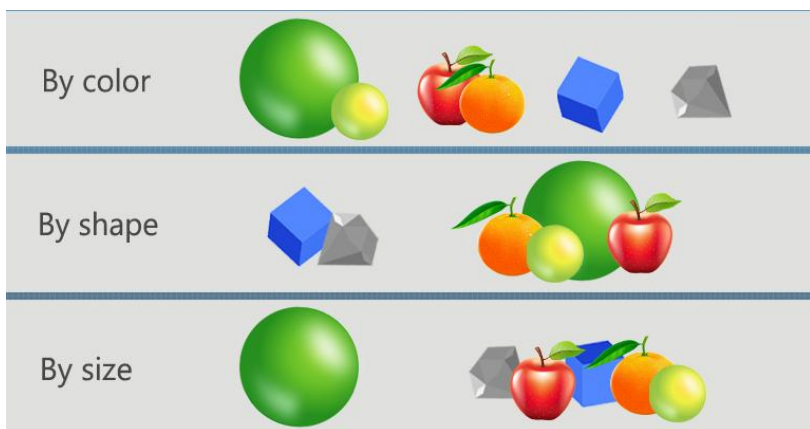
## AI for Science: a paradigm shift for scientific discovery ...

Apr 15, 2024 · New AI algorithms and models will bring unparalleled capabilities to assist scientists with analyzing huge and complex datasets and identifying patterns that guide their decisions and experimental designs.



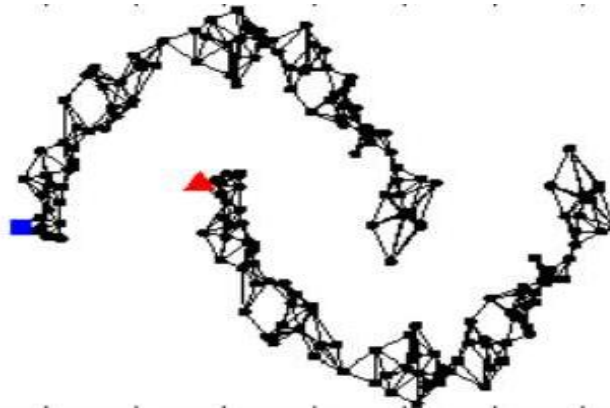
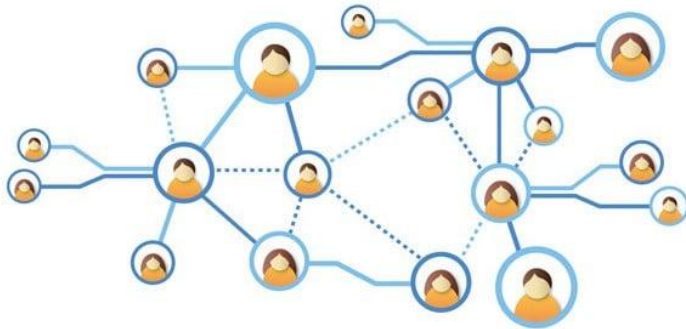
# Unsupervised Learning

- Discover patterns in unlabeled data
  - Unnecessary to label massive training data, however, usually have difficulty in learning desirable decision functions or dealing with very complex data
  - Clustering, graph ranking, anomaly detection, etc.



# Semi-supervised Learning

- Learn from both labeled and unlabeled data
  - Unlabeled data help establish the distributional landscape
  - Labeled data help determine the decision boundary

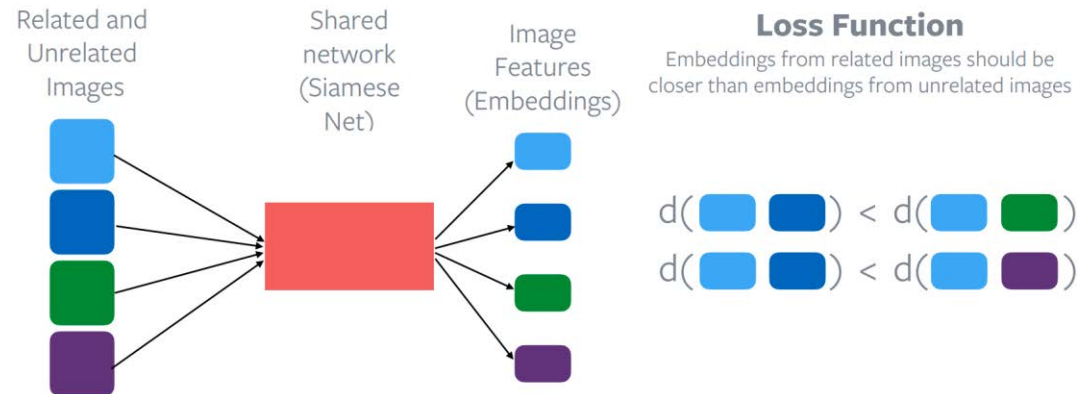




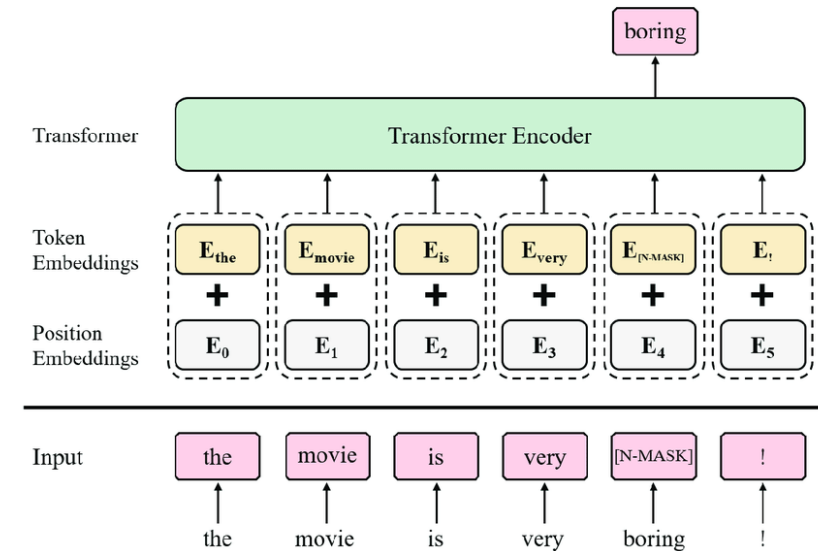
# Self-supervised Learning

- Learning from unlabeled data, but utilizing some signals naturally embedded in the data.

Contrastive Learning

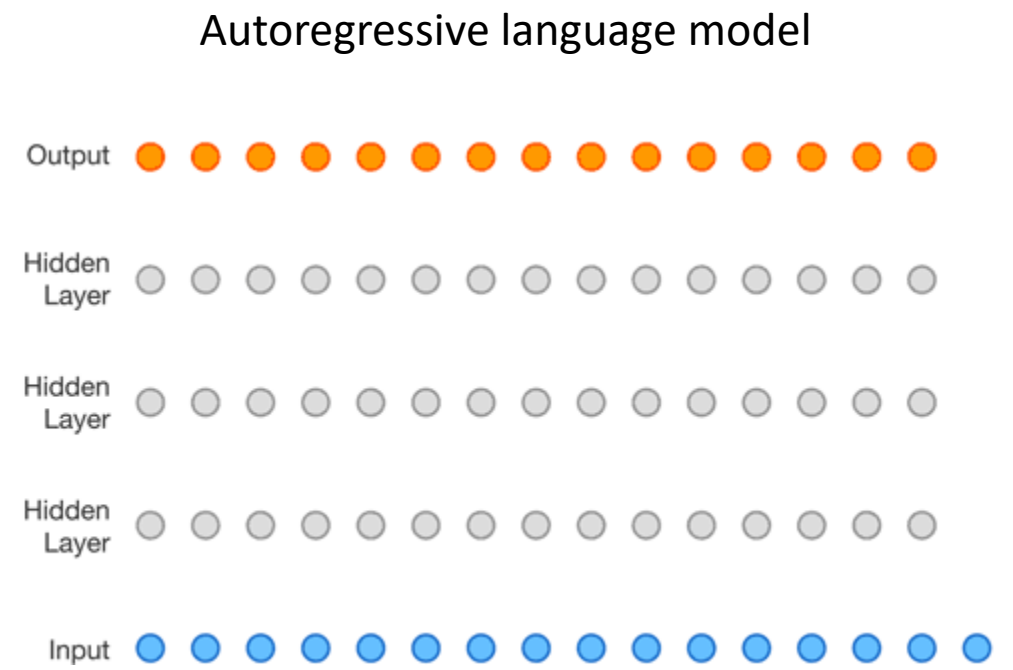
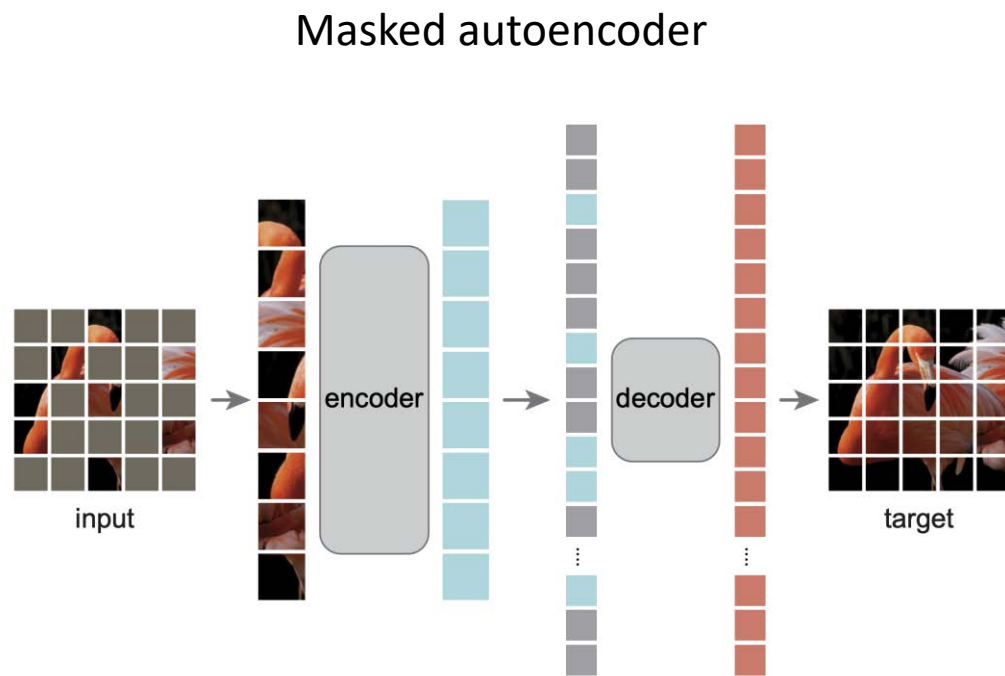


Masked language model



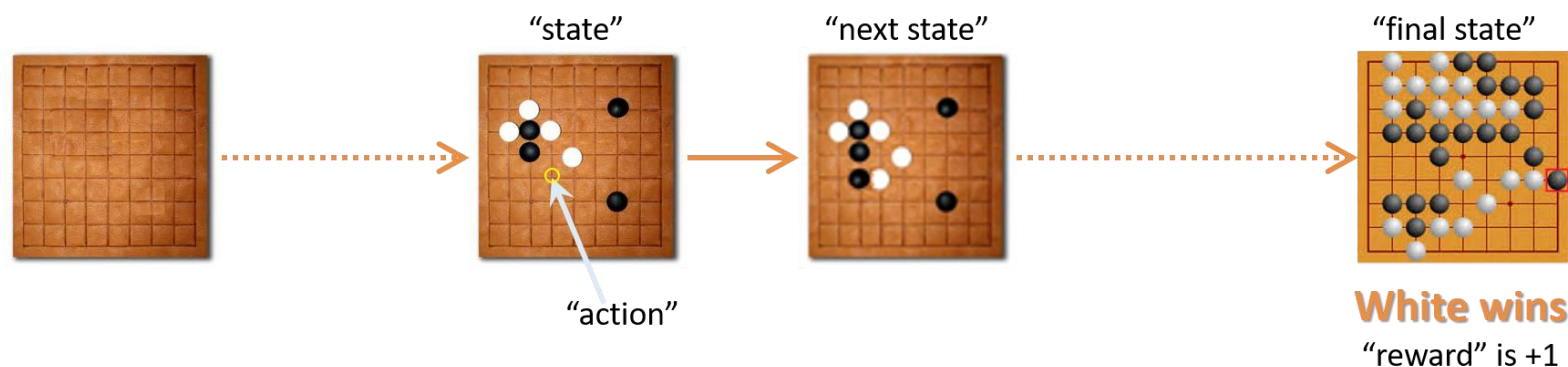
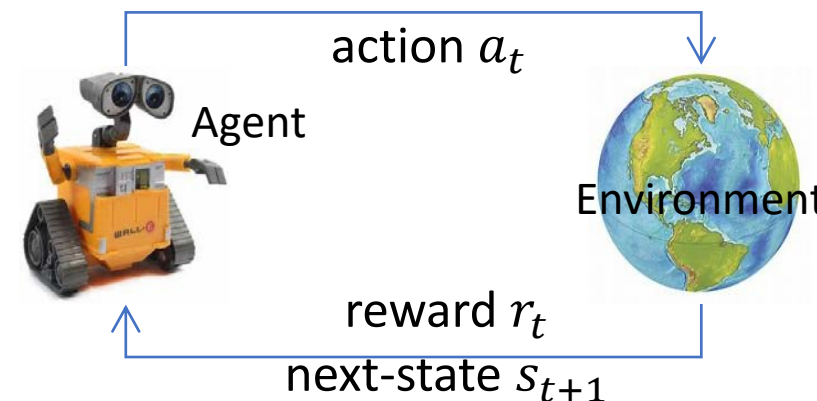
# Self-supervised Learning

- Learning from unlabeled data, but utilizing some signals naturally embedded in the data.



# Reinforcement Learning

- A dynamic learning approach that interacts with environment and identifies optimal actions according to delayed rewards
  - Feedback/reward from the environment is weaker than human labels, however, the interactions could be dynamic and more informative than i.i.d. training data



# Generative Learning

- Learn the joint probability  $p(x, y)$  for generation
  - Modeling the data distribution instead of making decisions
  - Require relatively more data since joint probability contains more information than conditional probability
- In contrast, discriminative learning learns conditional probability  $p(y|x)$ 
  - More direct decision-making power
  - Require relatively less data as compared to generative learning

# Generative AI

- The use of AI models to generative new content/data that is similar to some existing input or training data
  - GANs, VAEs, Sora, Dall E 1/2/3,
  - GPT 1/2/3/4, Gemini 1/1.5/2, DeepSeek 1/2/3, ...
- A subset of generative learning
- Generative learning: learn and understand the underlying structure or distribution of data



# Framework of Supervised Learning

# One Formula for Supervised Learning

The diagram illustrates the formula for supervised learning with the following components and annotations:

- Optimization:** Points to the  $\arg \min$  operator.
- Generalization:** Points to the summation index  $i=1, \dots, N \rightarrow \infty$ .
- Loss function:** Points to the  $L$  in the loss term  $L(f_\omega(x_i), y_i)$ .
- Hypothesis space:** Points to the domain  $\omega \in \Omega$  of the minimization.
- Input space:** Points to the input variable  $x_i \in X$ .
- Output space:** Points to the output variable  $y_i \in Y$ .

The formula is:

$$\omega^* = \arg \min_{\omega \in \Omega} \sum_{\substack{i=1, \dots, N \rightarrow \infty \\ x_i \in X, y_i \in Y \\ (x_i, y_i) \sim P}} L(f_\omega(x_i), y_i)$$

# Input Space

- Contains the objects under investigation.
- Objects can be represented by feature vectors (manually extracted according to different applications), or simply represented in its raw format (leaving the feature extraction task to the machine learning model).

# Output Space

- Contains the target of the prediction task
  - In supervised learning, ground truth labels are given in the output space
- Examples:
  - Regression: real-valued output
  - Classification: binary, multi-class, multi-label, hierarchical, etc.
  - Ranking: ranked list by preference

# Output Space

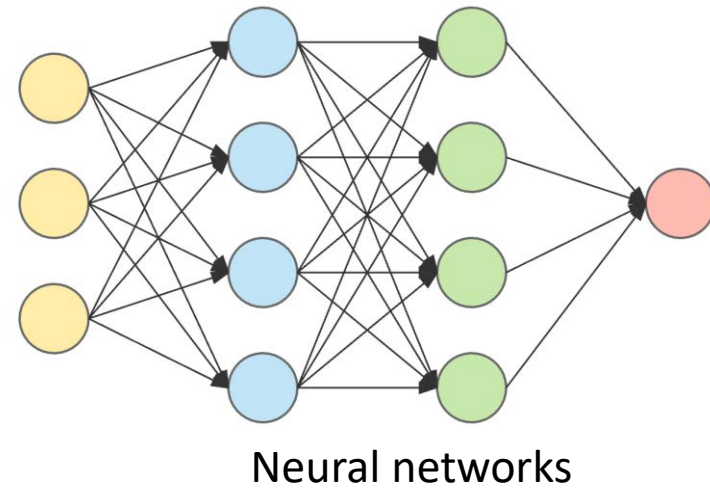
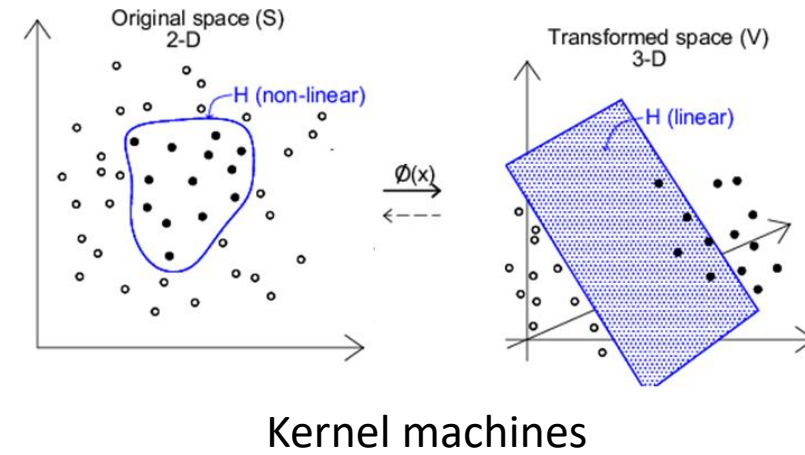
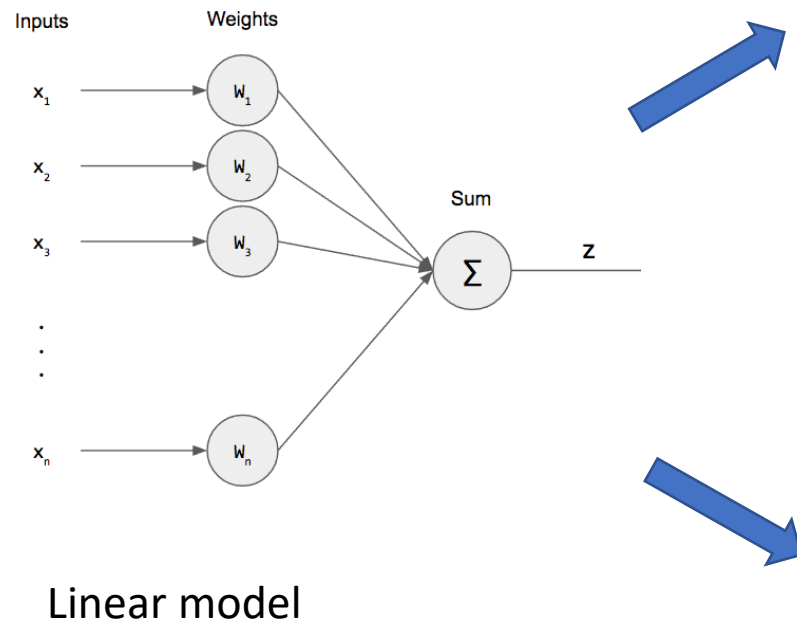
- Note: output space of a model might be different from data label space.
  - Label space of the task:  $\{+1, -1\}$  for classification
  - Model output space to facilitate learning: real numbers when using regression to solve classification/ranking tasks



# Hypothesis/function Space

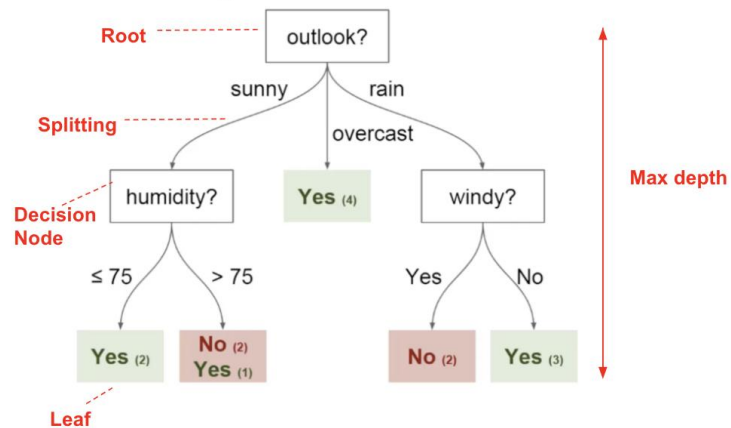
- Defines the class of functions (models) mapping the input space to the output space.
- The functions operate on the feature vectors of the input objects, and make predictions according to the format of the output space.
- Example
  - Linear models, trees, neural networks.

# Generalized Linear Model

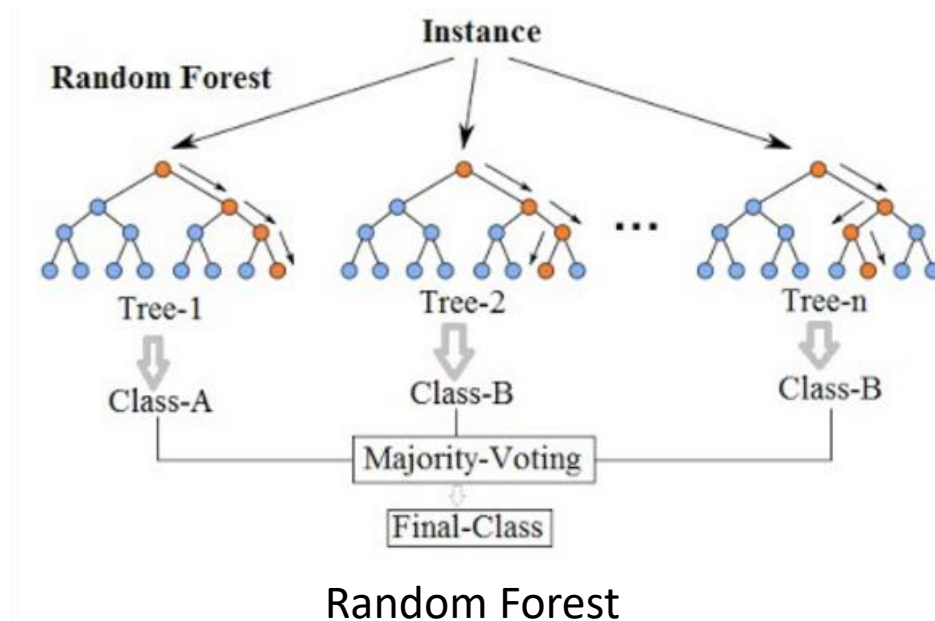


# Trees

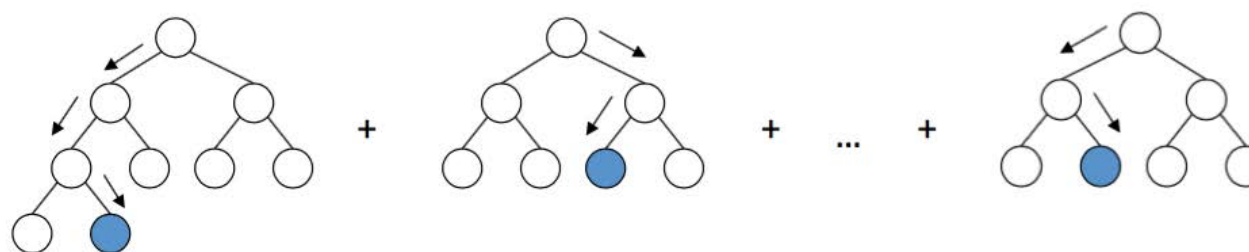
Decision Tree Diagram



Decision Tree



Random Forest



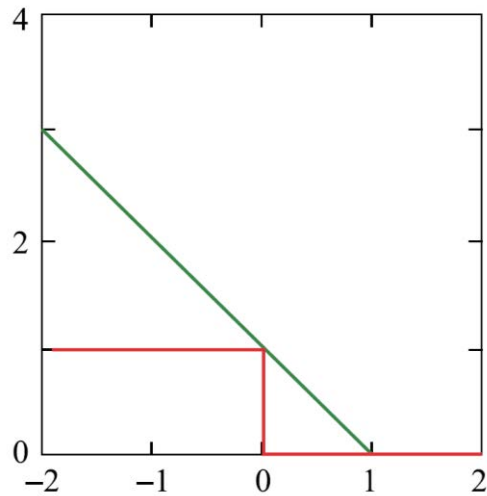
Boosting Trees

# Loss Function

- The loss function measures to what degree the prediction generated by the learned model is in accordance with the ground truth label.
  - With the loss function, an empirical risk can be defined on the training set, and the optimal model is usually (but not always) learned by means of empirical risk minimization.
- Example:
  - Square loss for regression
  - Exponential loss, hinge loss, and cross entropy loss for classification

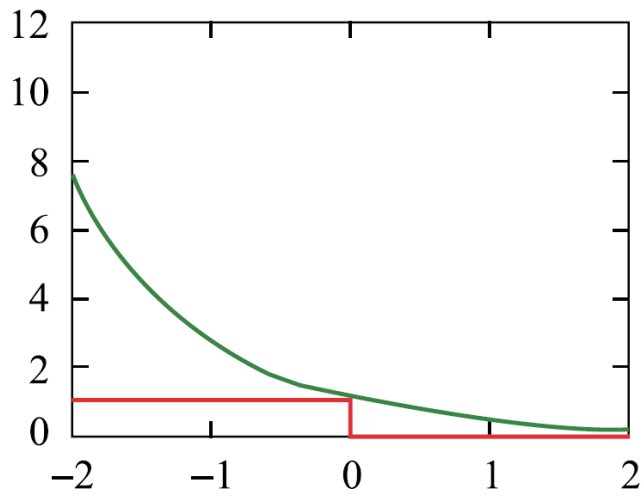
# Commonly used Loss Functions

## Hinge loss



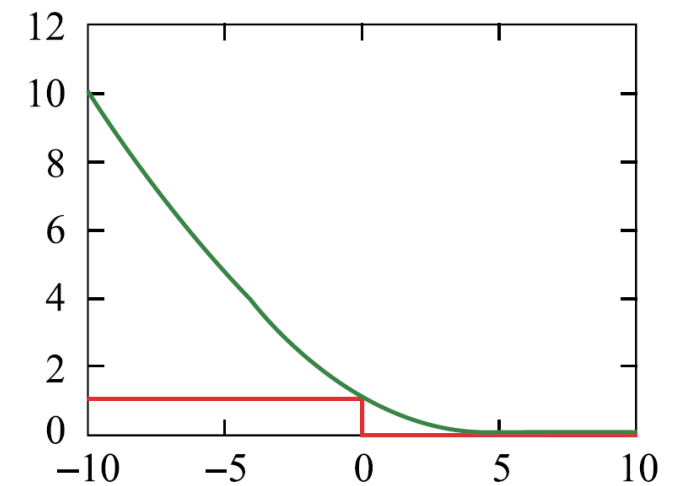
$$L(f_w; x, y) = \max\{0, 1 - yf_w(x)\}$$

## Exponential loss



$$L(f_w; x, y) = \exp(-yf_w(x))$$

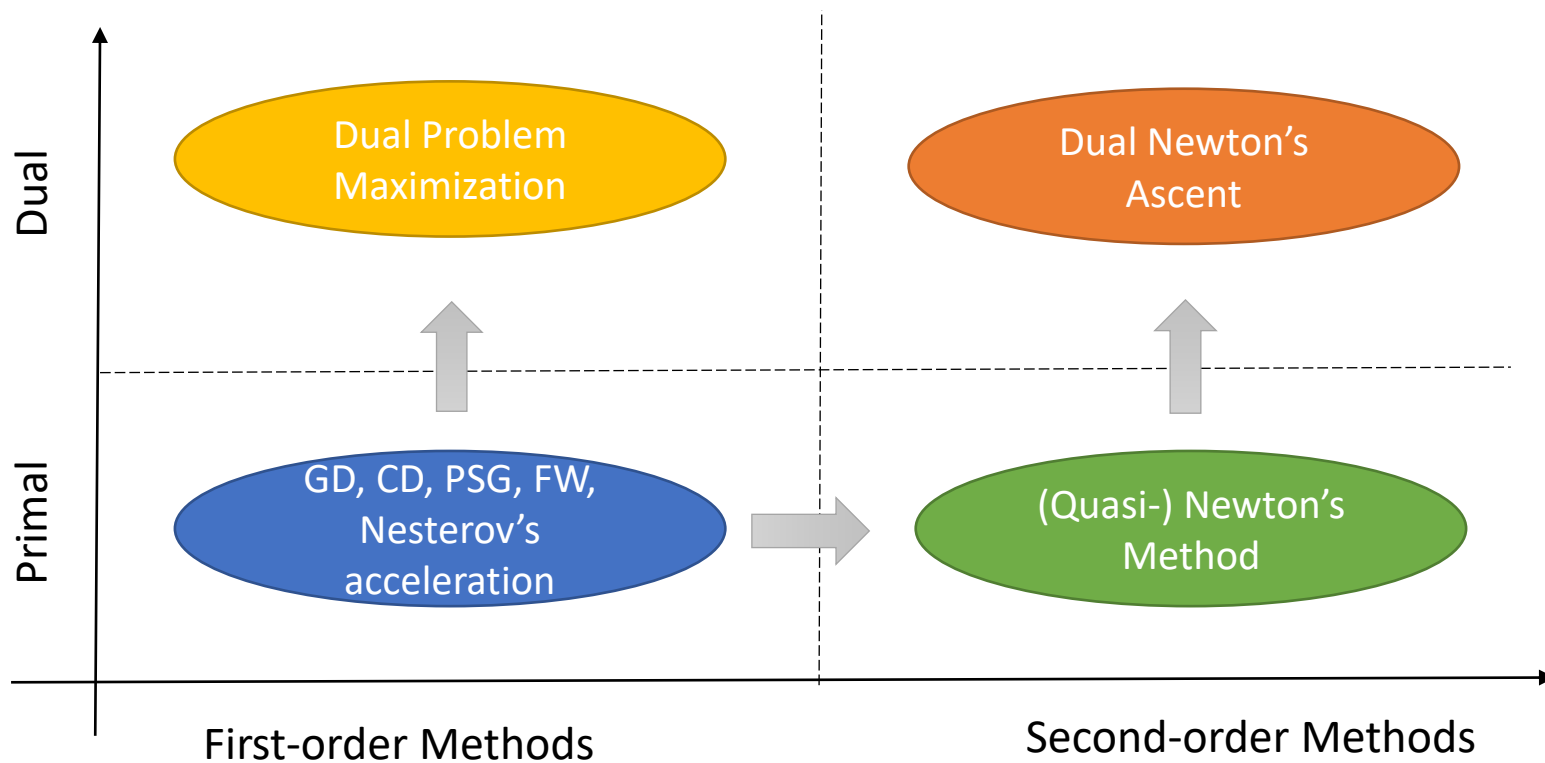
## Cross entropy loss



$$L(f; x, y) = -\log P(y|x; f_w)$$



# Optimization



# Gradient Descent [Cauchy 1847]

Unconstrained optimization

**Motivation:** to minimize the local first-order Taylor approximation of  $L$

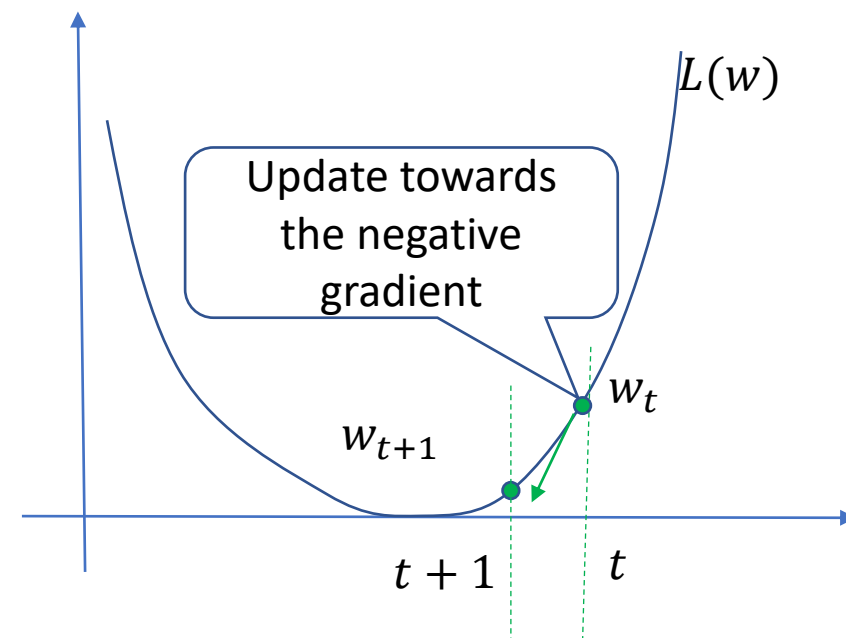
$$\min_w L(w) \approx \min_w L(w_t) + \nabla L(w_t)^\tau (w - w_t)$$

**Update rule:**

$$w_{t+1} = w_t - \eta \nabla L(w_t),$$

where  $\eta > 0$  is a fixed step-size.

$$L(w_{t+1}) = L(w_t) - \eta \nabla L(w_t)^\tau \nabla L(w_t) \leq L(w_t)$$



# Stochastic Gradient Descent (SGD)

Sweeps through the training set, computes gradient, and performs update for each training example

$$w \leftarrow w - \eta_k \nabla L_k(w)$$

- A sufficient condition to ensure convergence

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

- Step decay: decay by 0.5 every 5 epochs, by 0.1 every 20 epochs, or by validation error
- 1/t decay:  $\eta_k = \eta_0 / (1 + kt)$

Could be extended to mini-batch version

# Newton's Methods

Unconstrained optimization

## Motivation:

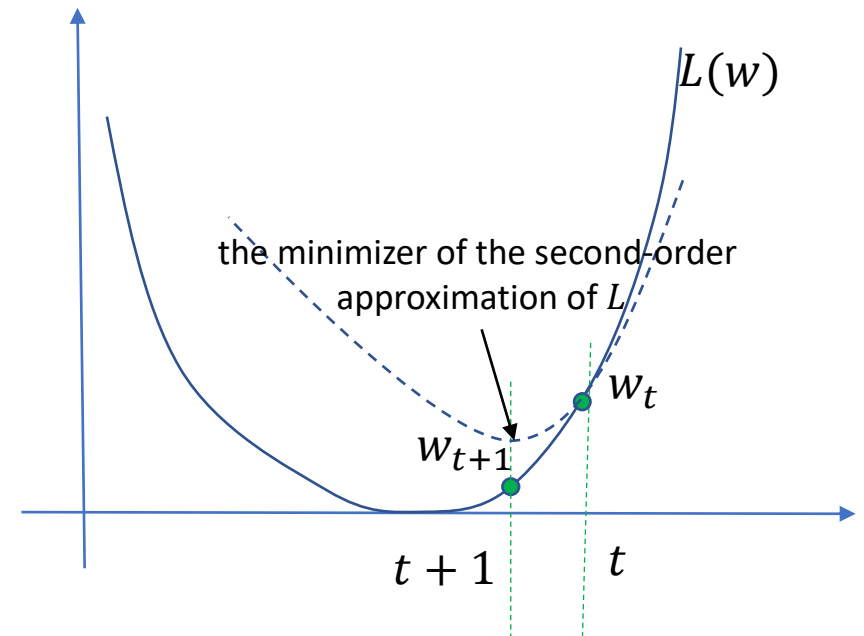
To minimize the local second-order Taylor approximation of  $L$ :

$$\min_w L(w) \approx \min_w L(w_t) + \nabla L(w_t)^\top (w - w_t) + \frac{1}{2} (w - w_t)^\top \nabla^2 L(w_t) (w - w_t).$$

$$\frac{dL(w)}{dw} = 0$$

**Update Rule:** suppose  $\nabla^2 L(w_t)$  is positive definite,  
$$w_{t+1} = w_t - [\nabla^2 L(w_t)]^{-1} \nabla L(w_t)$$

Could be approximated by Quasi Newton's methods



# Primal-Dual: Lagrangian

Constrained optimization

Primal Problem:

$$\begin{aligned} & \min_x f_0(x) \\ \text{s.t. } & f_i(x) \leq 0, i = 1, \dots, m \\ & h_j(x) = 0, j = 1, \dots, n \\ & \text{Optimal value: } p^* \end{aligned}$$

Linear approximation  
to indicator function



The Lagrangian:

$$\mathcal{L}(x, \lambda, \mu) \triangleq f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^n \mu_j h_j(x)$$

where  $\lambda_i \geq 0, \mu_j \in R$ .

*A problem with hard constraints  $\rightarrow$  A series of problems with soft constraints*



# Primal Function

Primal Problem:

$$\begin{aligned} & \min_x f_0(x) \\ & s.t. f_i(x) \leq 0, i = 1, \dots, m \\ & \quad h_j(x) = 0, j = 1, \dots, n \\ & \text{Optimal value: } p^* \end{aligned}$$

$$\theta_P(x) \triangleq \max_{\lambda \geq 0, \mu} \mathcal{L}(x, \lambda, \mu)$$

$$\theta_P(x) = \begin{cases} f_0(x), & \text{if } x \text{ is feasible} \\ +\infty, & \text{otherwise} \end{cases}$$

$$\min_x \theta_P(x) = \min_x \max_{\lambda \geq 0, \mu} \mathcal{L}(x, \lambda, \mu)$$

The Lagrangian:

$$\mathcal{L}(x, \lambda, \mu) \triangleq f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^n \mu_j h_j(x)$$

where  $\lambda_i \geq 0, \mu_j \in R$ .

$$p^* = \min_x \theta_P(x)$$

# Dual Function

Primal Problem:

$$\begin{aligned} & \min_x f_0(x) \\ & s.t. f_i(x) \leq 0, i = 1, \dots, m \\ & \quad h_j(x) = 0, j = 1, \dots, n \\ & \text{Optimal value: } p^* \end{aligned}$$

$$\theta_D(\lambda, \mu) \triangleq \min_x \mathcal{L}(x, \lambda, \mu)$$

$$\max_{\lambda \geq 0, \mu} \theta_D(\lambda, \mu) = \max_{\lambda \geq 0, \mu} \min_x \mathcal{L}(x, \lambda, \mu)$$

$$d^* = \max_{\lambda \geq 0, \mu} \theta_D(\lambda, \mu)$$

The Lagrangian:

$$\mathcal{L}(x, \lambda, \mu) \triangleq f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^n \mu_j h_j(x)$$

where  $\lambda_i \geq 0, \mu_j \in R$ .

# Primal-Dual: Duality

$$\theta_P(x) \triangleq \max_{\lambda \geq 0, \mu} \mathcal{L}(x, \lambda, \mu)$$

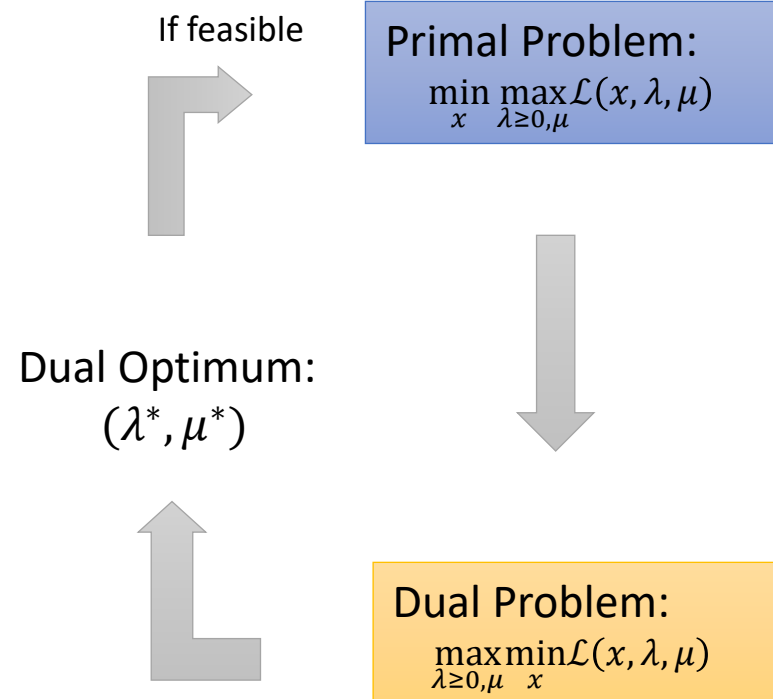
$$\theta_D(\lambda, \mu) \triangleq \min_x \mathcal{L}(x, \lambda, \mu)$$

$$d^* = \max_{\lambda \geq 0, \mu} \min_x \mathcal{L}(x, \lambda, \mu) \leq \min_x \max_{\lambda \geq 0, \mu} \mathcal{L}(x, \lambda, \mu) = p^*$$

Weak duality always holds:  $d^* \leq p^*$ .

When will strong duality  $d^* = p^*$  hold?

- Slater's condition (the primal problem is convex, and there is a feasible solution)



# Other Optimization Methods

- Projected Sub-gradient Descent
- Nesterov's Accelerated GD
- Frank-Wolfe
- Quasi Newton's Methods
- Adaptive optimization methods
- ... ..

# Machine Learning Models

# Outline

- Linear regression
- Neural networks
- Decision trees
- Support vector machines
- Boosting

\* Neural networks and decision trees will be introduced in detail later.

# Linear Regression

- Assumption:

- The target value can be approximated by the linear combination of input features.

$$f(x) = w^T x + b$$

- The combination weights can be learned by minimizing a square loss

$$L(w) = \sum_{i=1}^n (w^T x_i - y_i)^2$$

- Gradient descent can be leveraged to find the global optimum (square loss is convex)

# Probabilistic Explanation

- A noise model explanation

$$y_i = w^T x_i + \epsilon_i, \epsilon_i \sim N(0, \sigma^2)$$

- Then it is natural to obtain

$$p(y_i|x_i; w) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right)$$

- Conditional likelihood of the training data (i.i.d assumption)

$$l(w) = \prod_{i=1}^n p(y_i|x_i; w) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(w^T x_i - y_i)^2}{2\sigma^2}\right)$$



# Probabilistic Explanation

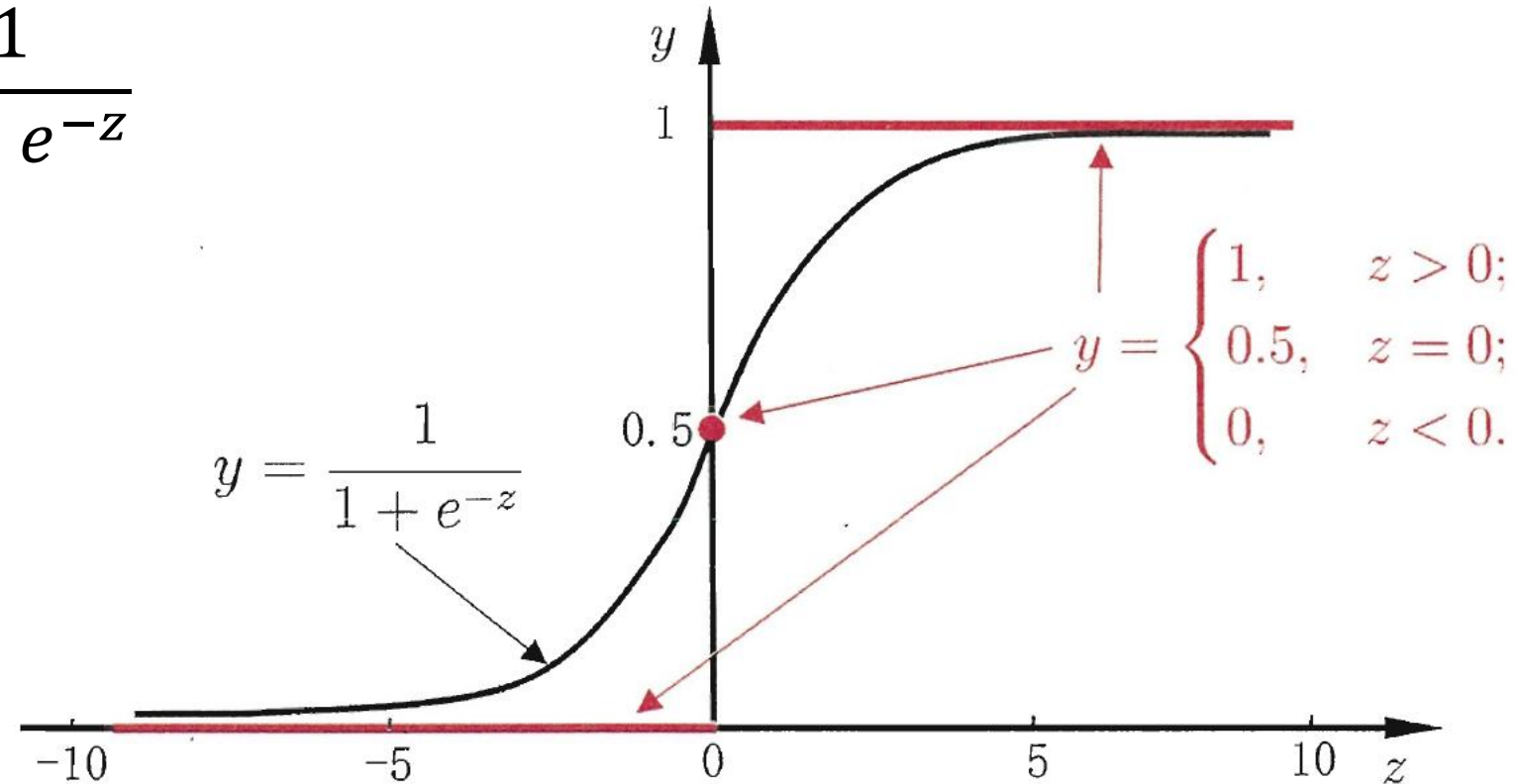
- The log likelihood

$$\log l(w) = n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_{i=1}^n (w^T x_i - y_i)^2$$

- Maximization of the log likelihood is equivalent to minimizing the least square loss function.
- Under certain probabilistic assumptions, least-square regression corresponds to finding the maximum likelihood estimation of  $w$ .

# From Regression to Classification

$$y = g(z) = \frac{1}{1 + e^{-z}}$$
$$z = w^T x$$



# Logistic Regression

$$y = g(z) = \frac{1}{1 + e^{-z}} \quad z = w^T x$$

$$y = \frac{1}{1 + e^{-w^T x}} = \frac{e^{w^T x}}{1 + e^{w^T x}}$$

$$p(y = 1|x) = \frac{e^{w^T x}}{1 + e^{w^T x}} \quad p(y = 0|x) = \frac{1}{1 + e^{w^T x}}$$

Maximum  
likelihood

$$L(w) = \sum_{i=1}^n \ln p(y_i|x_i; w)$$

# Further Extensions

- Multiclass classification, e.g.,  $K$ -class
  - 1 vs. 1: train  $\frac{K(K-1)}{2}$  binary classifiers and then vote
  - 1 vs. rest: train  $K$  binary classifiers and then choose maximum
  - Many vs. many: error correcting output codes
- Imbalanced classification, e.g. many more negative examples
  - Undersampling of negative examples
  - Oversampling of positive examples
  - Threshold moving
  - Cost-sensitive learning

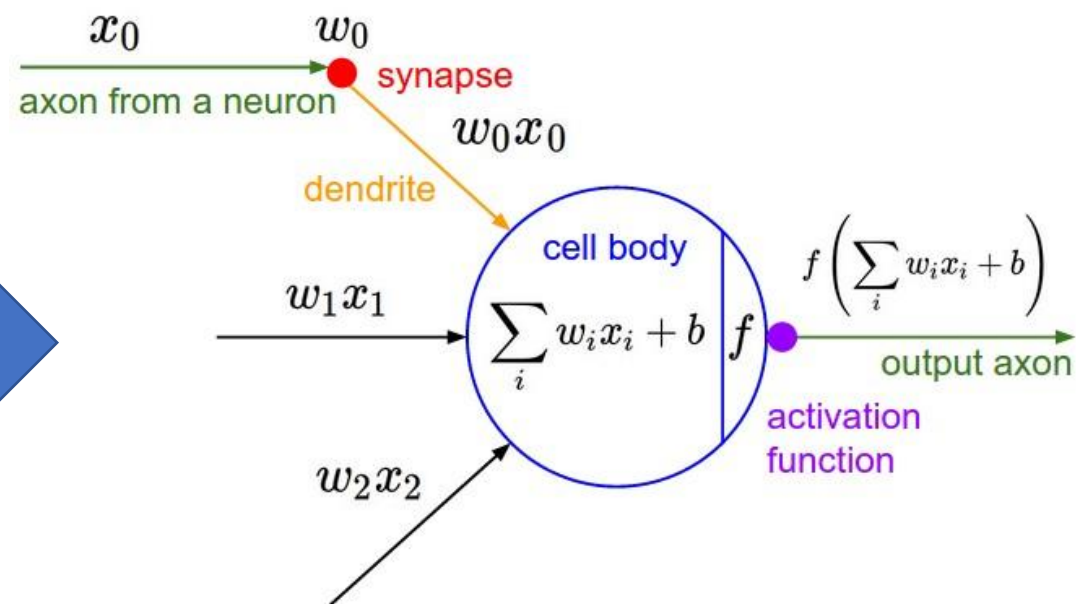
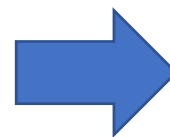
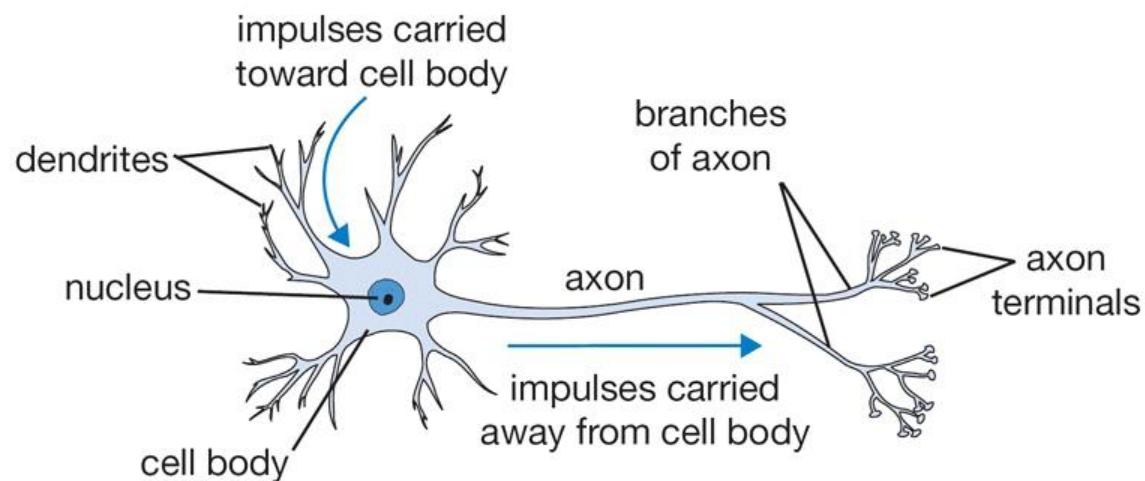
# Outline

- Linear regression
- Neural networks
- Decision trees
- Support vector machines
- Boosting

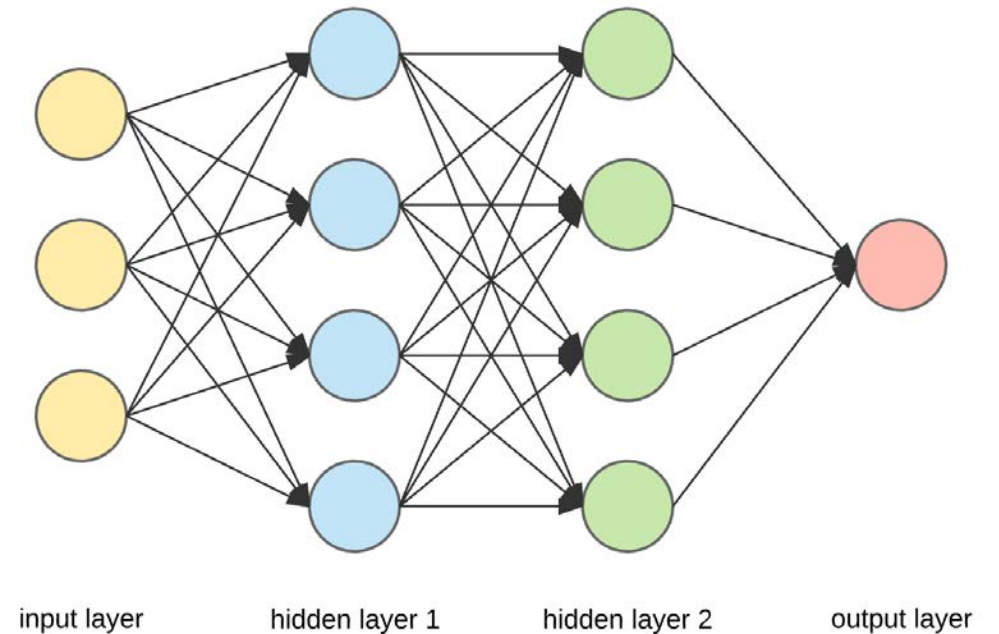
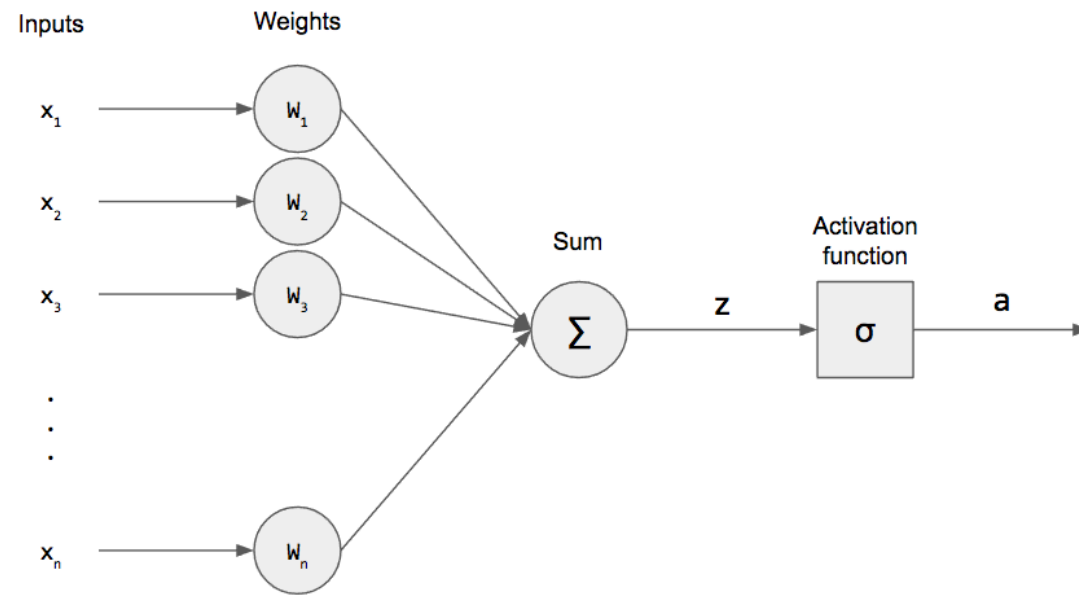
\* Neural networks and decision trees will be introduced in detail later.

# Perceptron

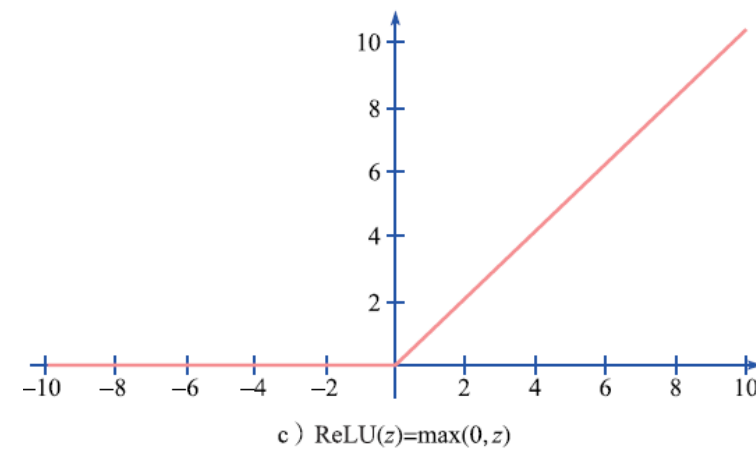
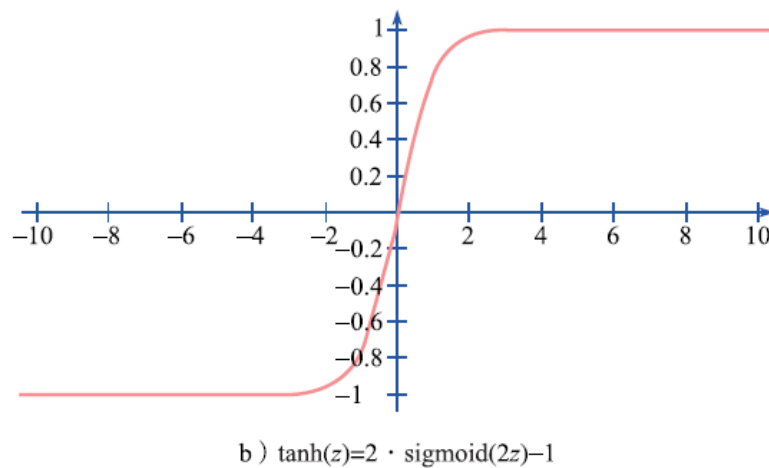
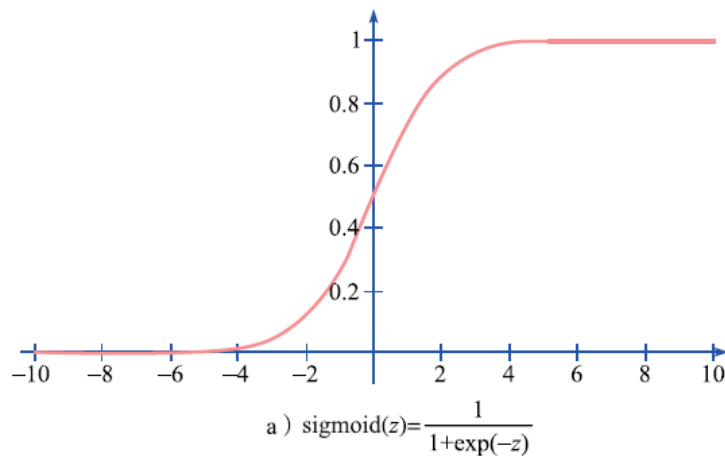
Dendrite, 树突, 接收信号  
Synapse, 突触, 放大信号  
Axon, 轴突, 输出信号



# From Perceptron to Neural Networks



# Activation Functions



Neurons have activation threshold



# Representation Power

## Universal Approximation Theorem

- A feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of  $\mathbb{R}^n$ , under mild assumptions on the activation function.

Let  $\varphi(\cdot)$  be a nonconstant, **bounded**, and **monotonically-increasing continuous** function. Let  $I_m$  denote the  $m$ -dimensional **unit hypercube**  $[0, 1]^m$ . The space of continuous functions on  $I_m$  is denoted by  $C(I_m)$ . Then, given any function  $f \in C(I_m)$  and  $\varepsilon > 0$ , there exists an integer  $N$  real constants  $v_i, b_i \in \mathbb{R}$  and real vectors  $w_i \in \mathbb{R}^m$ , where  $i = 1, \dots, N$ , such that we may define:

$$F(x) = \sum_{i=1}^N v_i \varphi(w_i^T x + b_i)$$

as an approximate realization of the function  $f$  where  $f$  is independent of  $\varphi$ ; that is,

$$|F(x) - f(x)| < \varepsilon$$

for all  $x \in I_m$ . In other words, functions of the form  $F(x)$  are **dense** in  $C(I_m)$ .

# Learning Model Parameters

- Minimize loss function
  - Regression: squared error
  - Classification: cross-entropy loss
- Back-propagation
  - Stochastic gradient descent to minimize the loss function
  - Sweep forward and backward over the network

# Design Choices

- Learning Algorithm
  - Issues: local minima, over-fitting, training time
  - Momentum, Nesterov's accelerated algorithm, Ada-series algorithms (to be introduced in the session of deep learning)
- Network architecture
  - Number of hidden layers, nodes per layer, skip connections
  - Feedforward, convolutional, recurrent, attention, etc.
  - Normalization

# Outline

- Linear regression
- Neural networks
- Decision trees
- Support vector machines
- Boosting

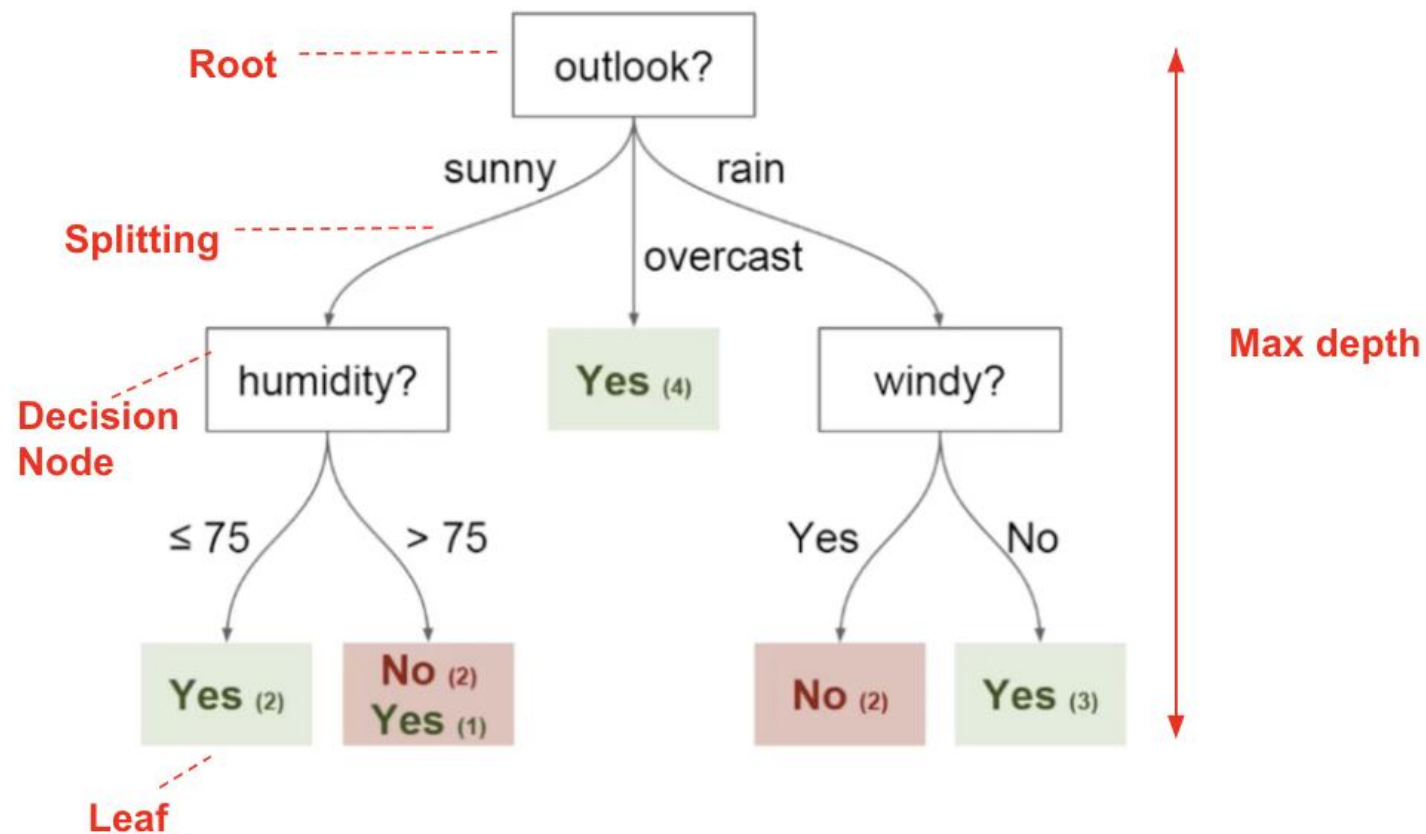
\* Neural networks and decision trees will be introduced in detail later.

# Decision Tree Learning

- Motivation:
  - “I can decide about a document by incrementally considering its properties.”
- Example:
  - This document contains the word “baseball.” So it is about sports.
  - If it does not, I would next check if it contains “finance”, then...
- Approach:
  - Construct a “tree of decisions” to follow, where a leaf applies a label to the document

# Decision Tree

Decision Tree Diagram



# Building a Decision Tree

- ID3 (Iterative Dichotomiser 3 – 迭代二分器)
  - Take all unused discrete attributes and calculate their entropy
  - Choose attribute for which entropy is minimum (or, equivalently, information gain  $Ent(D) - Ent(D|a)$  is maximal)
  - Make node containing that attribute

$$Ent(D) = - \sum_{k=1}^{|Y|} p_k \log_2 p_k$$

$$Ent(D|a) = \sum_v \frac{|D(a_v)|}{|D|} Ent(D(a_v))$$

- C4.5 Algorithm (improvement over ID3)
  - Information gain rate (normalized by intrinsic information)
  - Handling continuous attributes (discretization)
  - Handling training data with missing attribute values (fractional discount when computing IGR, beam search when testing)
  - Pruning trees after creation

# Design Choices

- Complexity of tree
  - The depth? The number of nodes?
- Pruning
  - Pre-pruning
  - Post-pruning
- Information criterion
  - Entropy (information Gain or IG rate)
  - Gini Index
  - Classification error



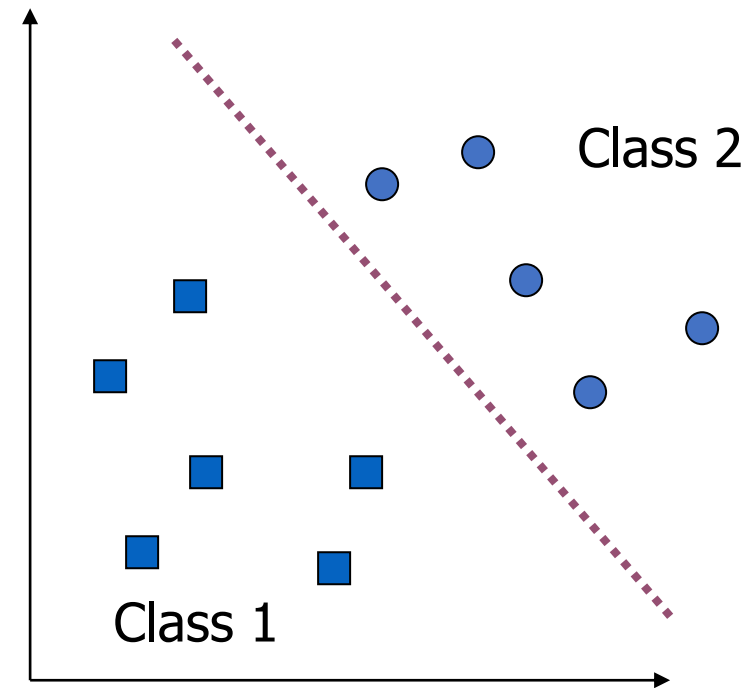
# Outline

- Linear regression
- Neural networks
- Decision trees
- Support vector machines
- Boosting

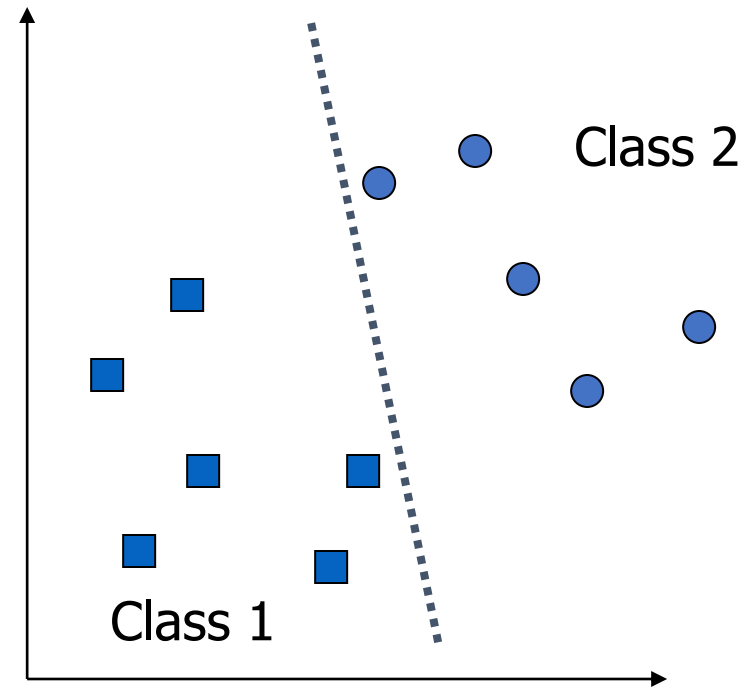
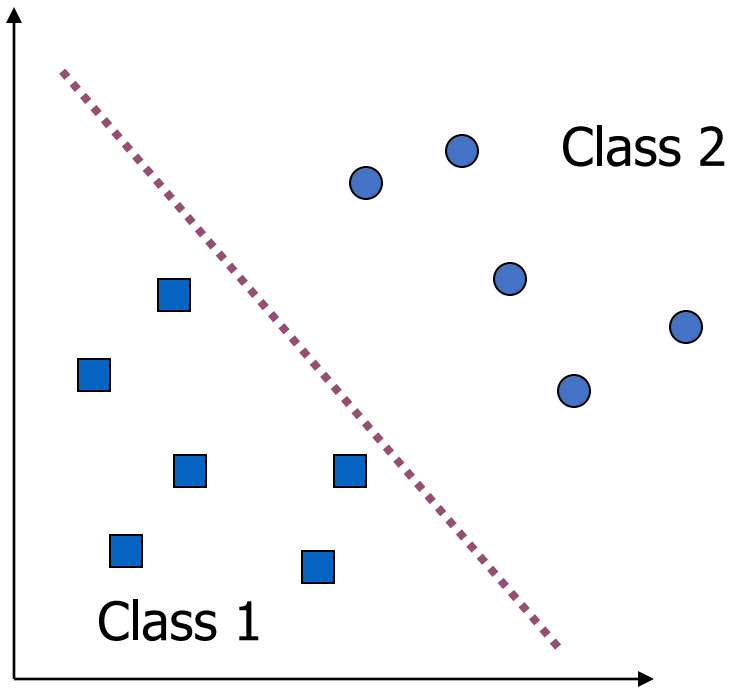
\* Neural networks and decision trees will be introduced in detail later.

# What is a Good Decision Boundary?

- Consider a two-class, linearly separable classification problem
- Many decision boundaries!
- Are all decision boundaries equally good?

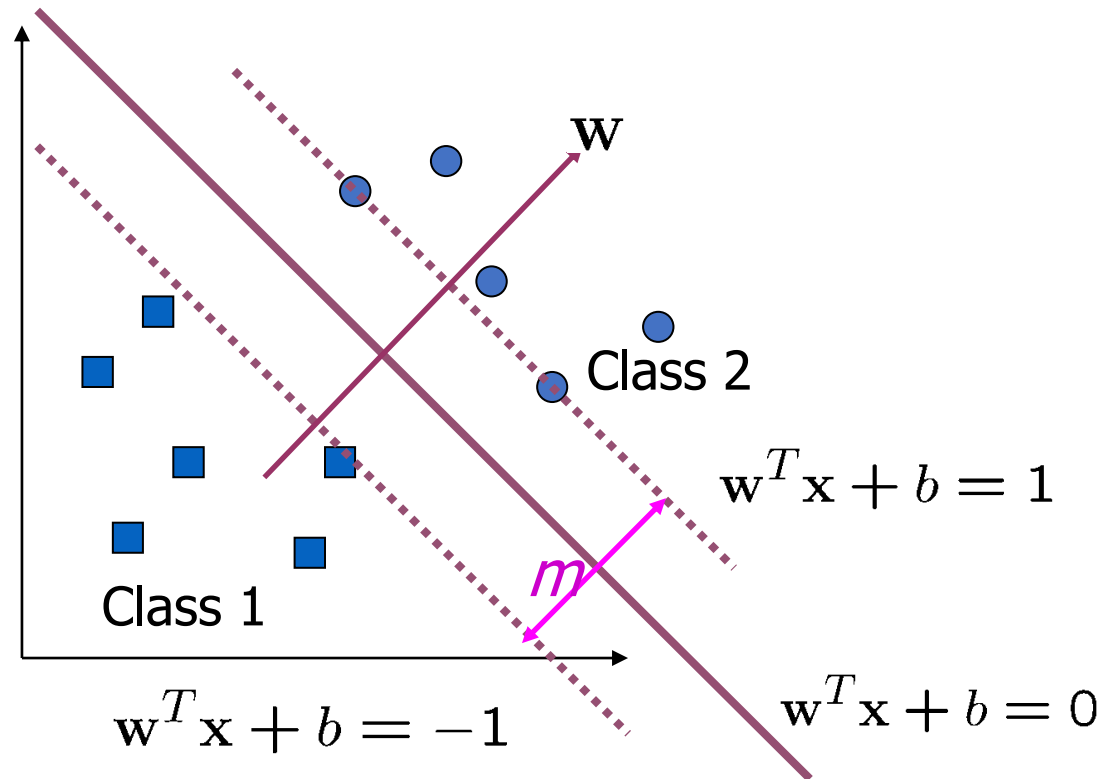


# Possibly Bad Decision Boundaries



# Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible (i.e., maximizing the margin,  $m$ )



$$d = \frac{|w^T x + b|}{\|w\|}$$
$$m = \frac{2}{\|w\|}$$

# Finding the Max-Margin Decision Boundary

- Let  $\{x_1, \dots, x_n\}$  be our data set and let  $y_i$  in  $\{1, -1\}$  be the class label of  $x_i$
  - The decision boundary should classify all points correctly
- $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall i$
- Margin maximization: The decision boundary can be found by solving the following constrained optimization problem

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \forall i \end{aligned}$$

- The Lagrangian of this optimization problem is

$$\mathcal{L} = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \alpha_i (y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1) \quad \alpha_i \geq 0 \quad \forall i$$

# The Dual Problem

- By setting the derivative of the Lagrangian to zero, we can get  $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$   
then the optimization problem can be written in its dual form (in terms of  $\alpha_i$ )

$$\max. W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - A global maximum of  $\alpha_i$  can always be found

- $\mathbf{w}$  can be recovered by  $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$

If the number of training examples is large, SVM training will be very slow because the number of parameters is very large in the dual problem.

# KKT Conditions

- Karush-Kuhn-Tucker Conditions are necessary conditions (sufficient as well for convex optimization) for a solution in nonlinear programming to be optimal.
  - Condition #1: the derivative of the Lagrange multipliers is zero
  - Condition #2: the multiplication of the non-equality constraints and their corresponding dual multipliers are zero.
- Checking the KKT conditions in the case of SVM, one finds that the QP problem of SVM is optimally solved when for all  $i$ ,

$$\alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0 \quad \forall i \quad \Longrightarrow \quad \begin{array}{ll} \alpha_i = 0 & \Rightarrow y_i f(\vec{x}_i) \geq 1 \\ \alpha_i > 0 & \Rightarrow y_i f(\vec{x}_i) = 1 \end{array}$$

# Characteristics of the Solution

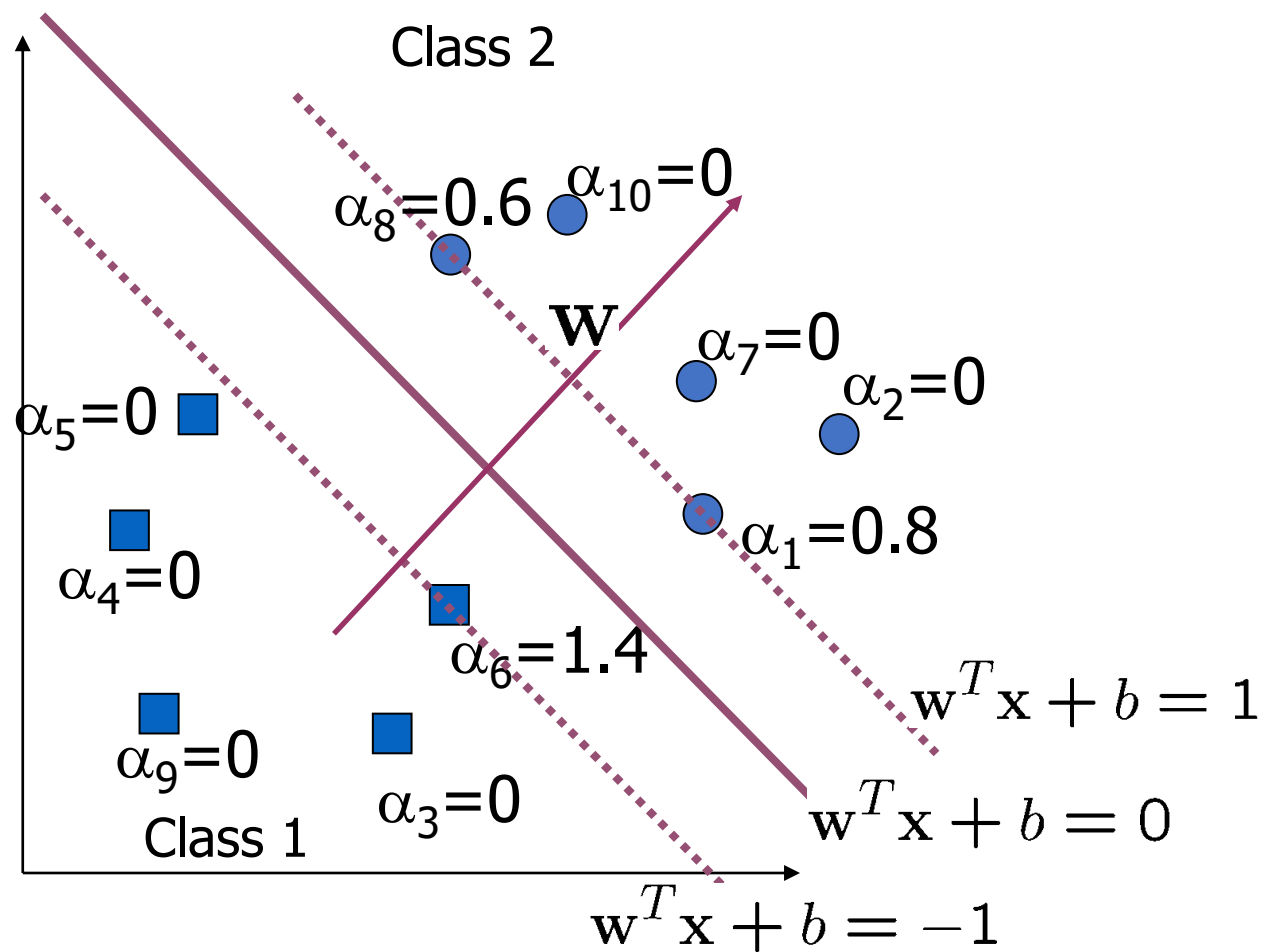
- KKT condition indicates many of the  $\alpha_i$  are zero
  - $w$  is a linear combination of a small number of data points
- $x_i$  with non-zero  $\alpha_i$  are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let  $t_j$  ( $j = 1, \dots, s$ ) be the indices of the  $s$  support vectors. We can write

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

- For testing with a new data  $\mathbf{z}$ 
  - Compute  $\mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$   
and classify  $\mathbf{z}$  as class 1 if the sum is positive, and class 2 otherwise.

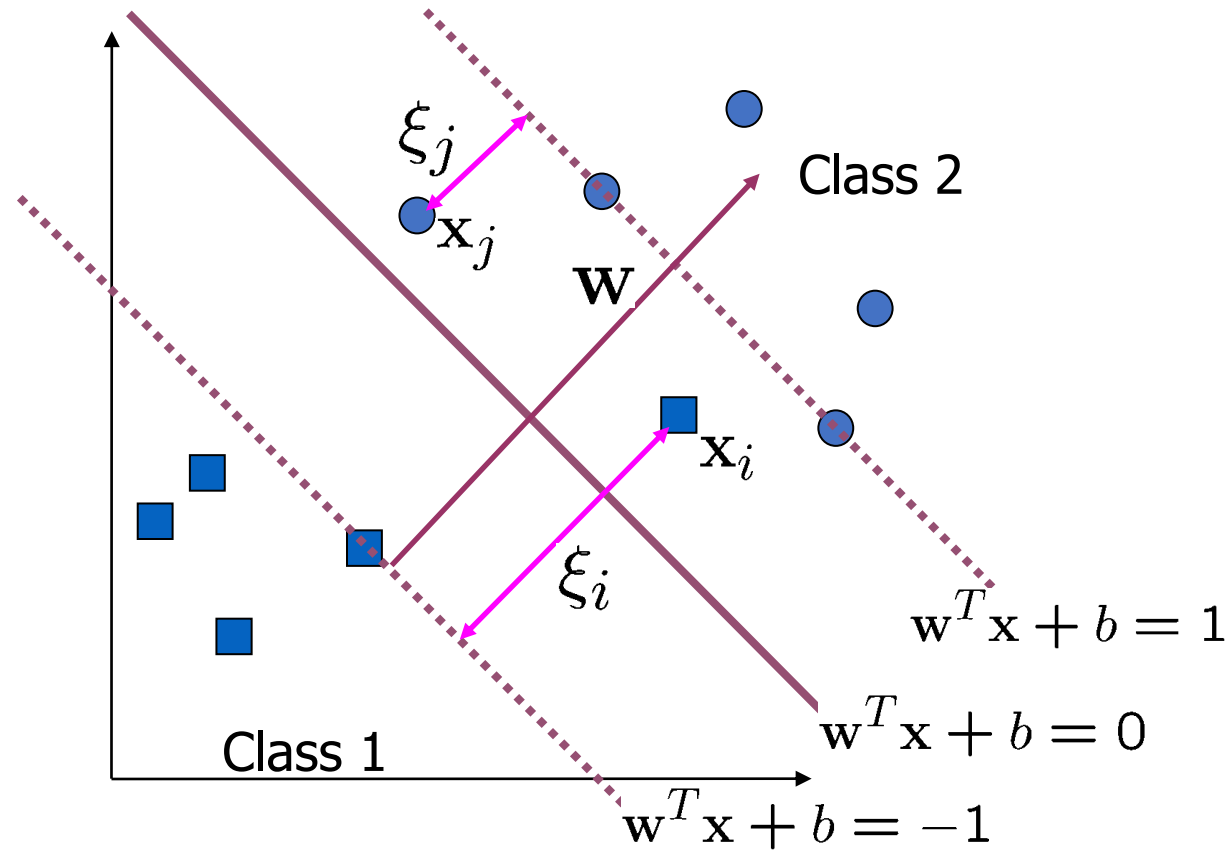


# A Geometrical Interpretation



# Linearly Inseparable Problems

- We allow “error”  $\xi_i$  in classification



# Soft Margin Hyperplane

- Error minimization

$\xi_i$  are “slack variables” in optimization  
 $\xi_i$  is the error for  $x_i$   
 $\xi_i = 0$  if there is no error for  $x_i$

- We want to minimize  $\frac{1}{2}||\mathbf{w}'||^2 + C \sum_i \xi_i$   
 $C$  : tradeoff parameter between error and margin
- The optimization problem becomes

Minimize  $\sum_i \xi_i$ , s.t.

$$\begin{cases} \mathbf{w}'^T \mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}'^T \mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

Structural  
risk

Empirical  
risk

Minimize  $\frac{1}{2}||\mathbf{w}'||^2 + C \sum_{i=1}^n \xi_i$   
subject to  $y_i(\mathbf{w}'^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

# The Optimization Problem

- The dual of the problem is

$$\begin{aligned} \max. \quad W(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to } C &\geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

- $w$  is recovered as  $\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound  $C$  on  $\alpha_i$  now
- Once again, a QP solver can be used to find  $\alpha_i$

# When to Use Primal/Dual Formulation

## Primal Problem

$$\begin{aligned} &\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ &\text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \end{aligned}$$

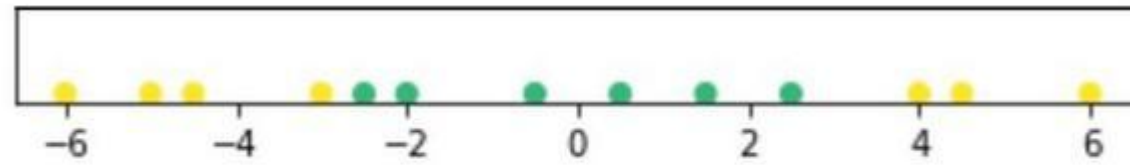
- If we have a few features and many datapoints, we should use the primal.
- Gives interpretable models

## Dual Problem

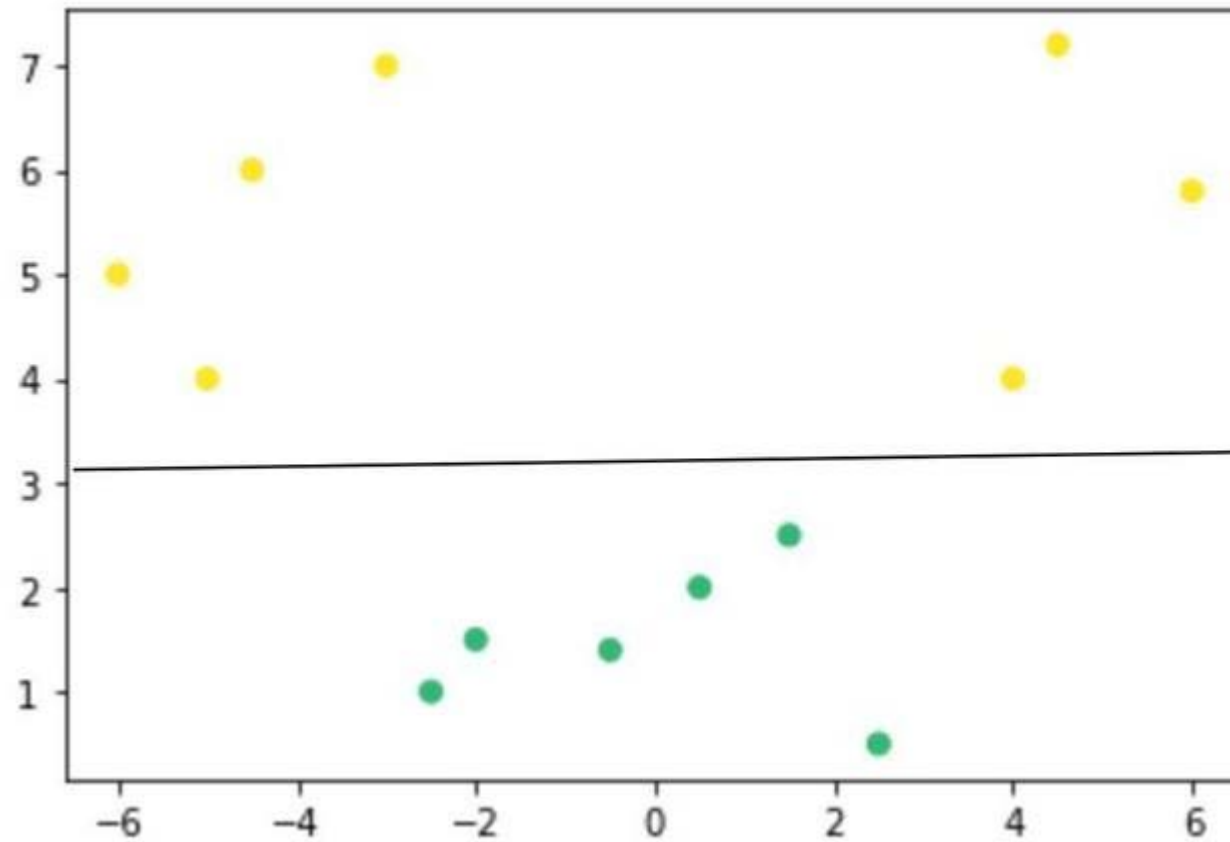
$$\begin{aligned} &\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ &\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

- if we have a lot of features, but fewer datapoints, we should use the dual.
- Can introduce kernel trick

Dataset in 1-D  
(Non-Separable)



Dataset in 2-D  
(Separable)

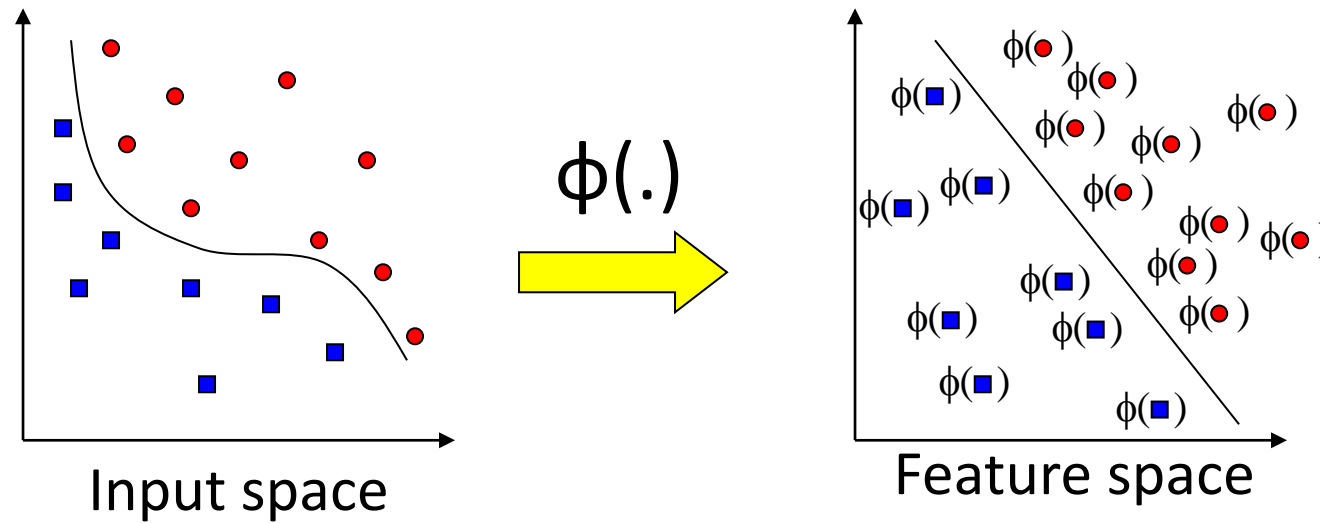


# Extension to Non-linear Decision Boundary

- Key idea: transform  $x_i$  to a higher dimensional space to “make life easier”
  - Input space: the space where point  $x_i$  are located
  - Feature space: the space of  $f(x_i)$  after transformation
- Why transformation?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Samples become much sparser in the high-dimensional space, and therefore classification can become easier with a proper transformation.

# Transforming the Data

- Computation in the feature space can be costly because it is high dimensional
  - The feature space might be of infinite dimensions!

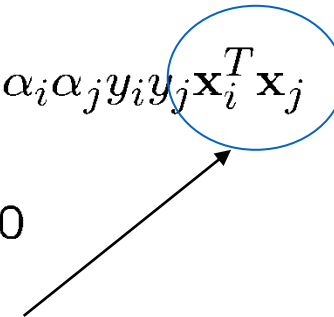


- The kernel trick comes to rescue



# The Kernel Trick

- Recall the SVM optimization problem

$$\begin{aligned} \max. \quad W(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to } C &\geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$


- The data points only appear as **inner product** in the feature space
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Define the kernel function  $K$  by  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

# Illustrating Example

- Suppose  $\phi(\cdot)$  is given as follows

$$\phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right), \phi\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out  $\phi(\cdot)$  explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out  $\phi(\cdot)$  explicitly is known as the [kernel trick](#)

# Kernel Functions

- In practical use of SVM, only the kernel function (and not  $\phi(\cdot)$ ) is specified.
- Kernel function can be thought of as a similarity measure between the input objects in the high-dimensional space
- Not all similarity measure can be used as kernel function, however Mercer's condition states that any **positive semi-definite** kernel  $K(x, y)$  can be expressed as an inner product in some high dimensional space.

# Examples of Kernel Functions

- Polynomial kernel with degree (up to)  $d$

$$K(x, y) = (x \cdot y)^d \qquad K(x, y) = (x \cdot y + 1)^d$$

- Radial basis function kernel with width  $\sigma$

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

- Closely related to radial basis function neural networks
- Kernel functions can also be learned and optimized
  - Kernel learning / metric learning

# Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

$$\begin{aligned} \max. \quad W(\boldsymbol{\alpha}) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to } C &\geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$



$$\begin{aligned} \max. \quad W(\boldsymbol{\alpha}) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1, j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to } C &\geq \alpha_i \geq 0, \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

# Modification Due to Kernel Function

- For testing, the new data  $\mathbf{z}$  is classified as class 1 if  $f > 0$ , and as class 2 if  $f < 0$

$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$



$$\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^s \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

# Outline

- Linear regression
- Neural networks
- Decision trees
- Support vector machines
- Boosting

\*Neural networks, random forest, gradient boosting trees will be introduced in separate sections.

# How to Make \$\$\$ In Horse Races?

- Ask professionals
- Suppose:
  - Professionals cannot give single highly accurate rule
  - But presented with a set of races, can always generate better-than-random rules
- Can you get rich?



# Basic Idea

- Ask expert for rule-of-thumb
  - Assemble set of cases where rule-to-thumb fails (hard cases)
  - Ask expert again for selected set of hard cases
  - And so on...
- 
- Combine all rules-of-thumb
  - Expert could be “weak” learning algorithm

# Questions

- How to choose races on each round?
  - Concentrate on “hardest” races  
(those most often misclassified by previous rules of thumb)
- How to combine rules of thumb into single prediction rule?
  - Take (weighted) majority vote of rules of thumb

# AdaBoost.M1

- Given training set  $X = \{(x_1, y_1), \dots, (x_m, y_m)\}$ 
  - $y_i \in \{-1, +1\}$  correct label of instance  $x_i \in X$
- Initialize distribution  $D_1(i) = \frac{1}{m}$ ; (How hard the case is)
- for  $t = 1, \dots, T$ :

- Find weak hypothesis (“rule of thumb”)

$$h_t : X \rightarrow \{-1, +1\}$$

with small error  $\epsilon_t$  on  $D_t$ :

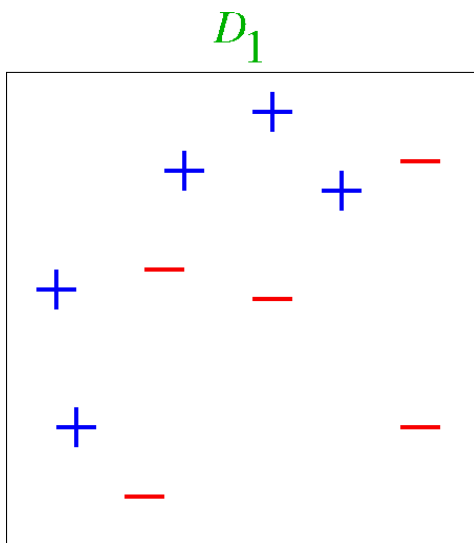
- Update distribution  $D_t$  on  $\{1, \dots, m\}$

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad \alpha_t = \frac{1}{2} \ln \frac{1 - \epsilon_t}{\epsilon_t}$$

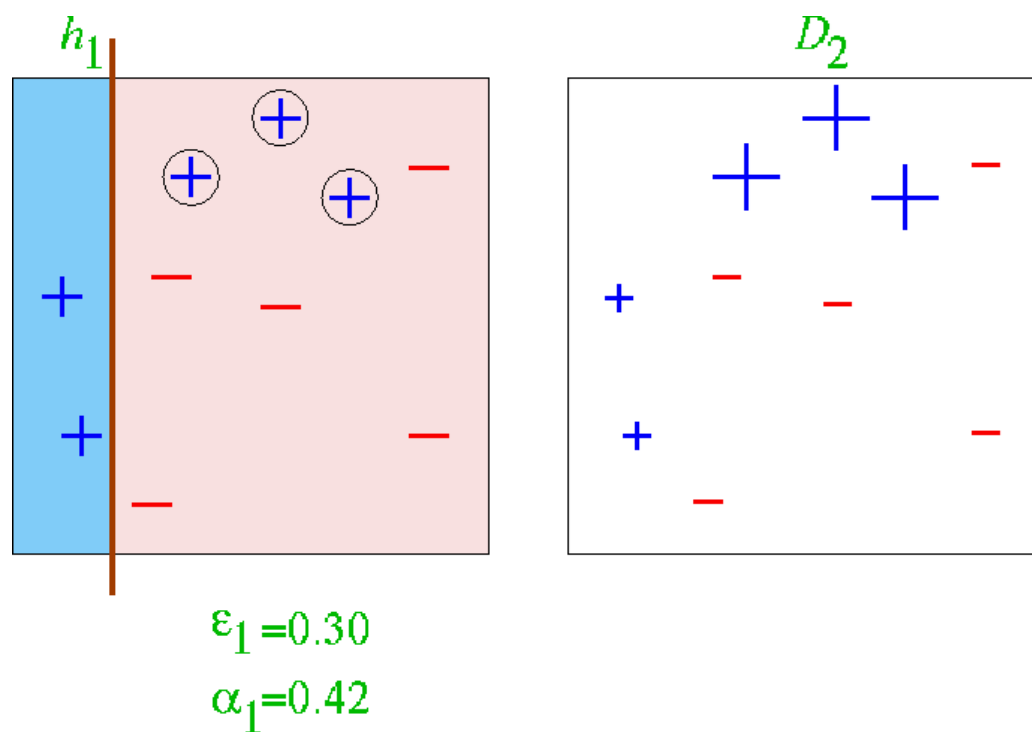
where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

- output final hypothesis  $H(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t h_t(x)\right)$

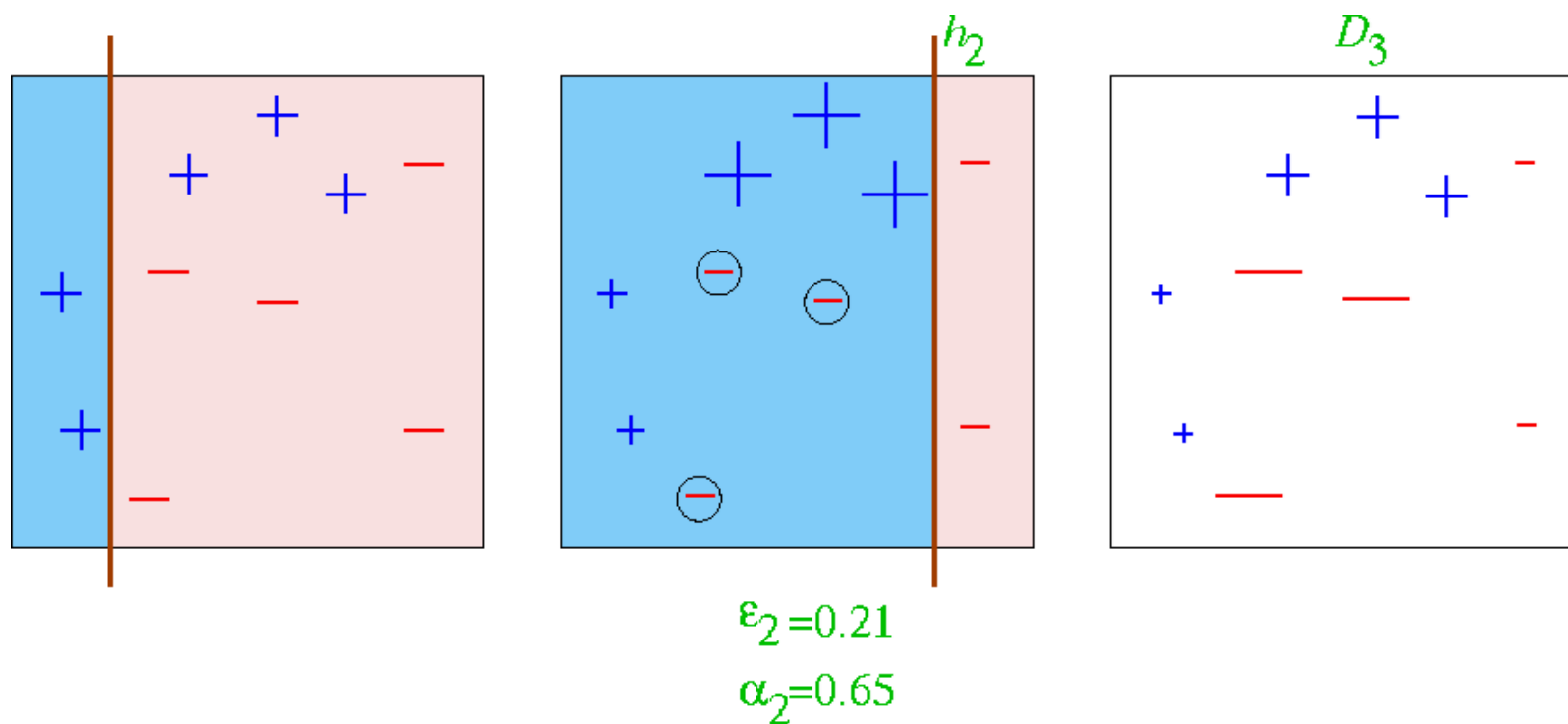
# Toy Example



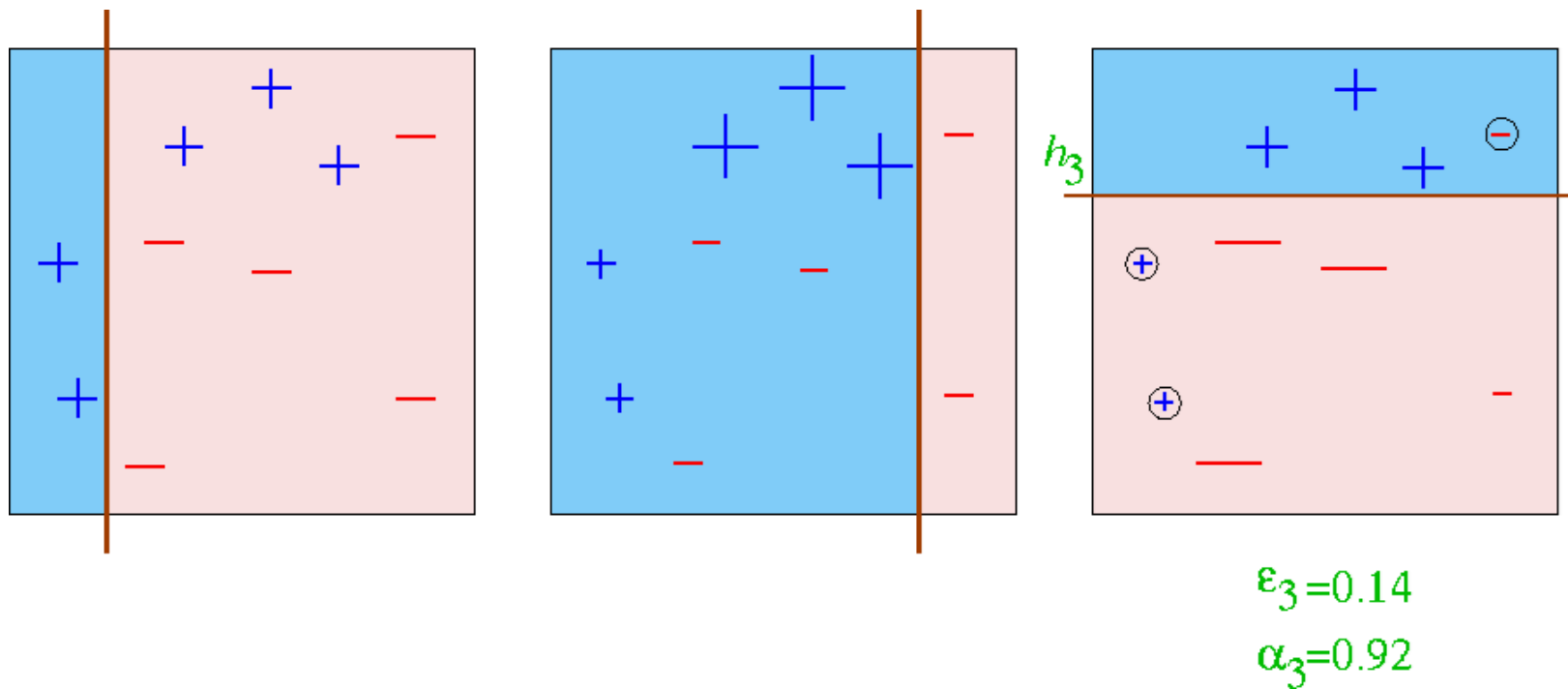
# Round 1



# Round 2

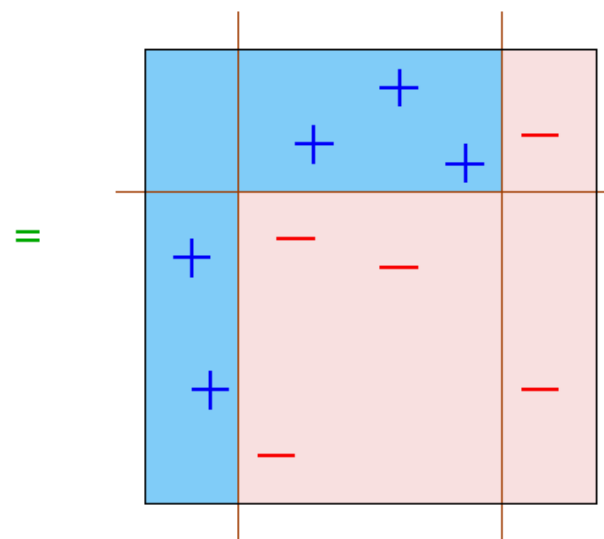


# Round 3



# Final Hypothesis

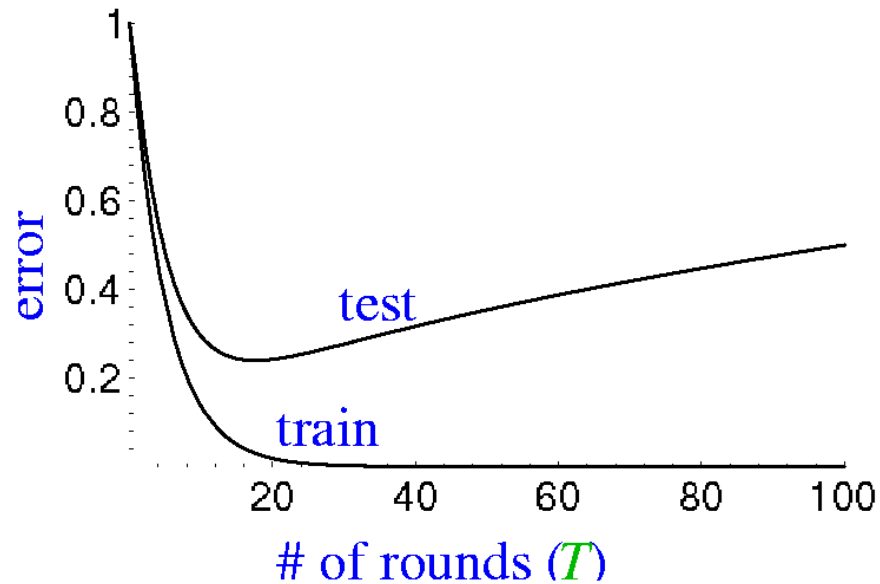
$$H_{\text{final}} = \text{sign} \left( 0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} \right)$$





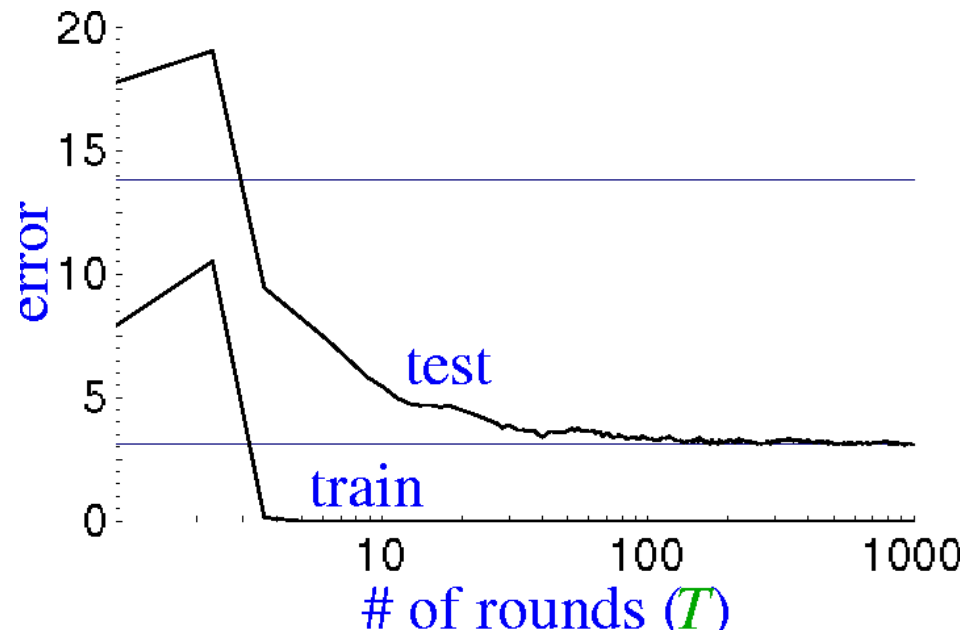
# Problem of Overfitting

- Training error to continue to drop (or reach zero)
- Test error to increase when  $H_{\text{final}}$  becomes “too complex”



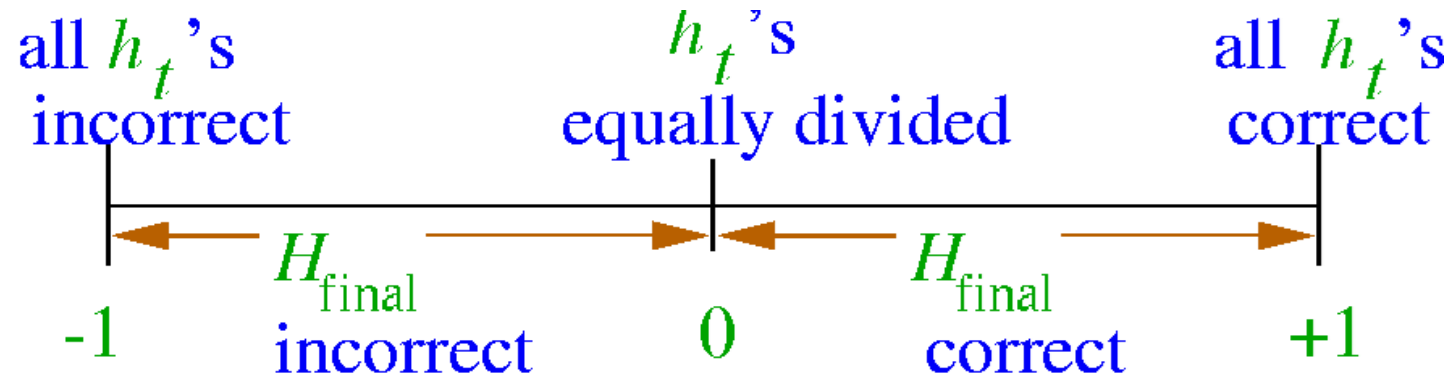
# Boosting Seems to Avoid Overfitting

- Test error does not increase even after 1,000 rounds ( $\sim 2,000,000$  nodes)
- Test error continues to drop after training error is zero!
- Occam's razor wrongly predicts “simpler” rule is better.



(boosting on C4.5 on  
“letter” dataset)

# A Margin Explanation



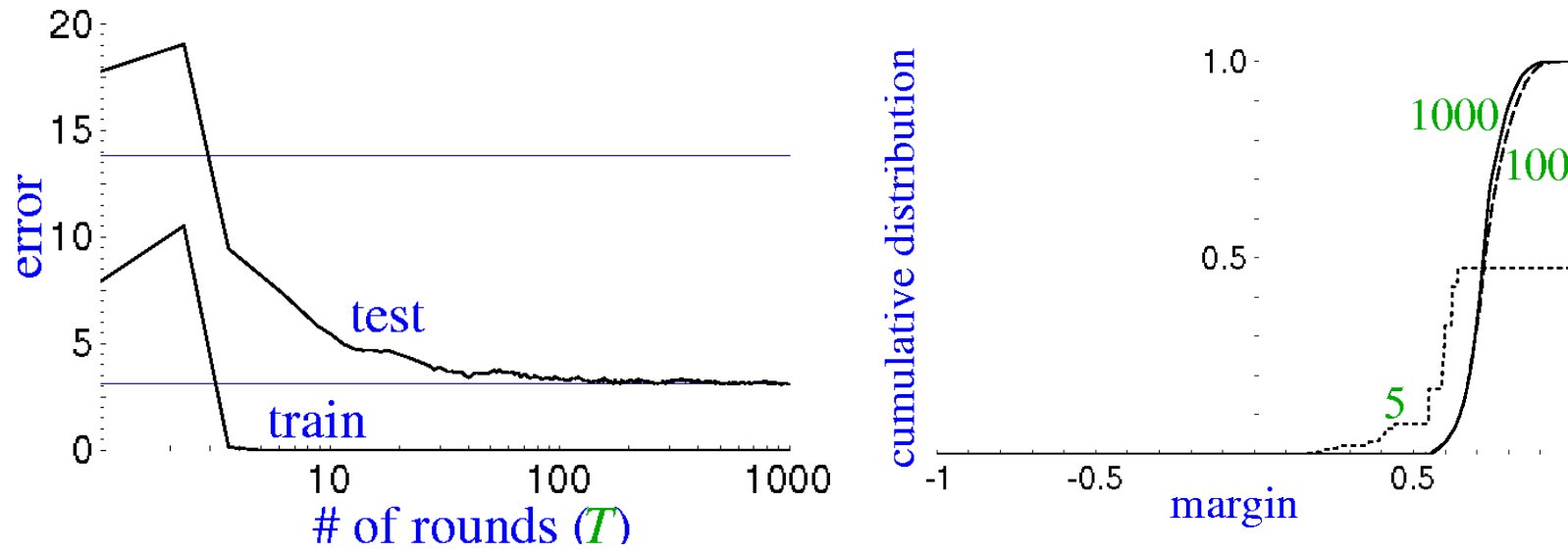
Key idea: Consider confidence (margin):

- with

$$H_{\text{final}}(x) = \text{sgn}(f(x)) \quad f(x) = \frac{\sum_t \alpha_t h_t(x)}{\sum_t \alpha_t} \in [-1, 1]$$

- define: margin of  $(x, y) = y \cdot f(x)$

# The Margin Distribution



epoch	5	100	1000
training error	0.0	0.0	0.0
test error	8.4	3.3	3.1
%margins $\leq 0.5$	7.7	0.0	0.0
Minimum margin	0.14	0.52	0.55

# Schapire's Margin Distribution Bound

- Schapire et al. demonstrated both theoretically and empirically that AdaBoost can generate good margin distribution.
- The margin distribution keeps improving even after the training error is zero. This accounts for AdaBoost's resistance to over-fitting.

**Theorem 1.** (Schapire et al., 1998) For any  $\delta > 0$  and  $\theta > 0$ , with probability at least  $1 - \delta$  over the random choice of sample  $S$  with size  $m$ , every voting classifier  $f \in \mathcal{C}(\mathcal{H})$  satisfies the following bound:

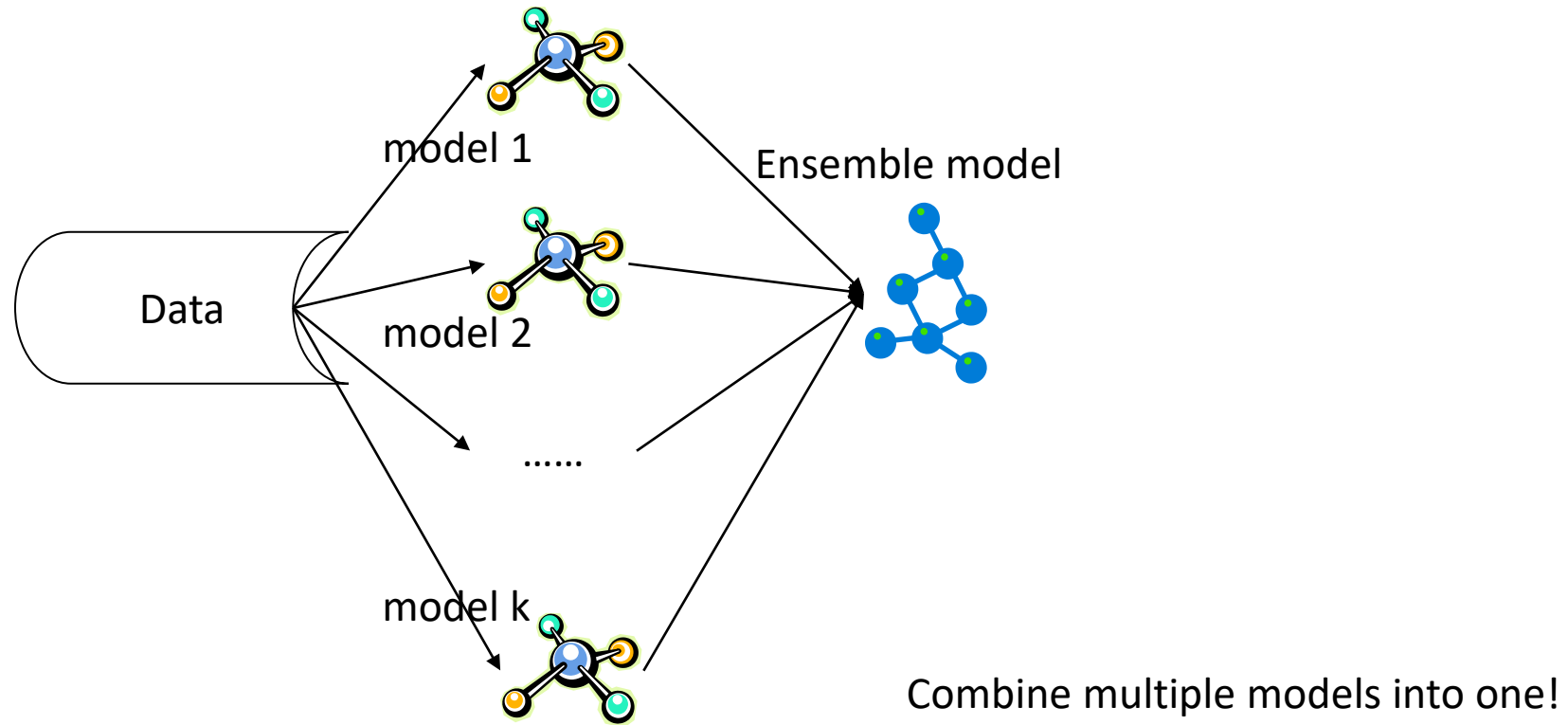
$$\Pr_D[yf(x) < 0] \leq \Pr_S[yf(x) \leq \theta] + O\left(\frac{1}{\sqrt{m}} \left(\frac{\ln m \ln |\mathcal{H}|}{\theta^2} + \ln \frac{1}{\delta}\right)^{1/2}\right).$$

Given any  $\theta$ , with increasing number of weak classifiers,  
 $p(yf(x) < \theta)$  exponentially decays to 0.  
Larger number of weak classifiers  $\rightarrow$  large  $\theta$  satisfying the inequality

For larger  $\theta$ , the second term also  
becomes smaller

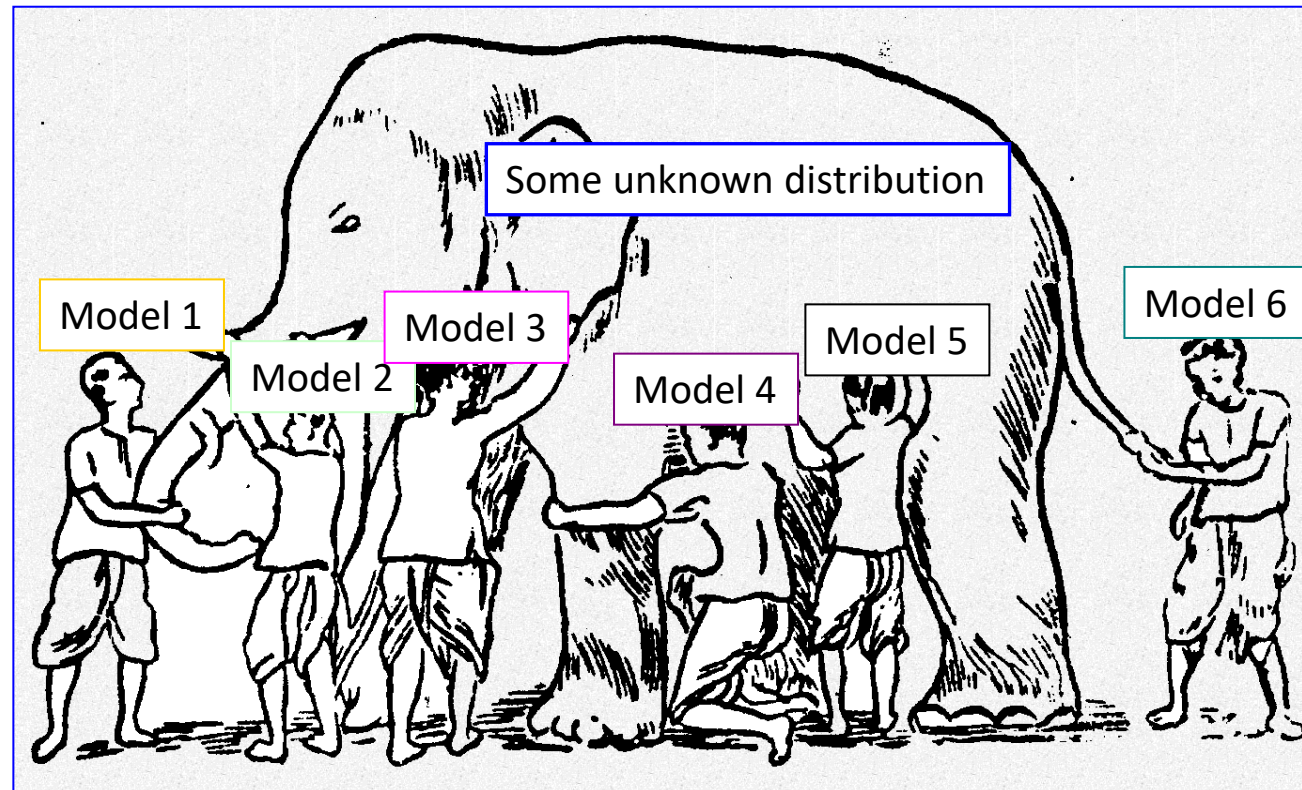
# Model Ensemble

# Ensemble



# Why Ensemble Works?

- Ensemble gives the global picture!



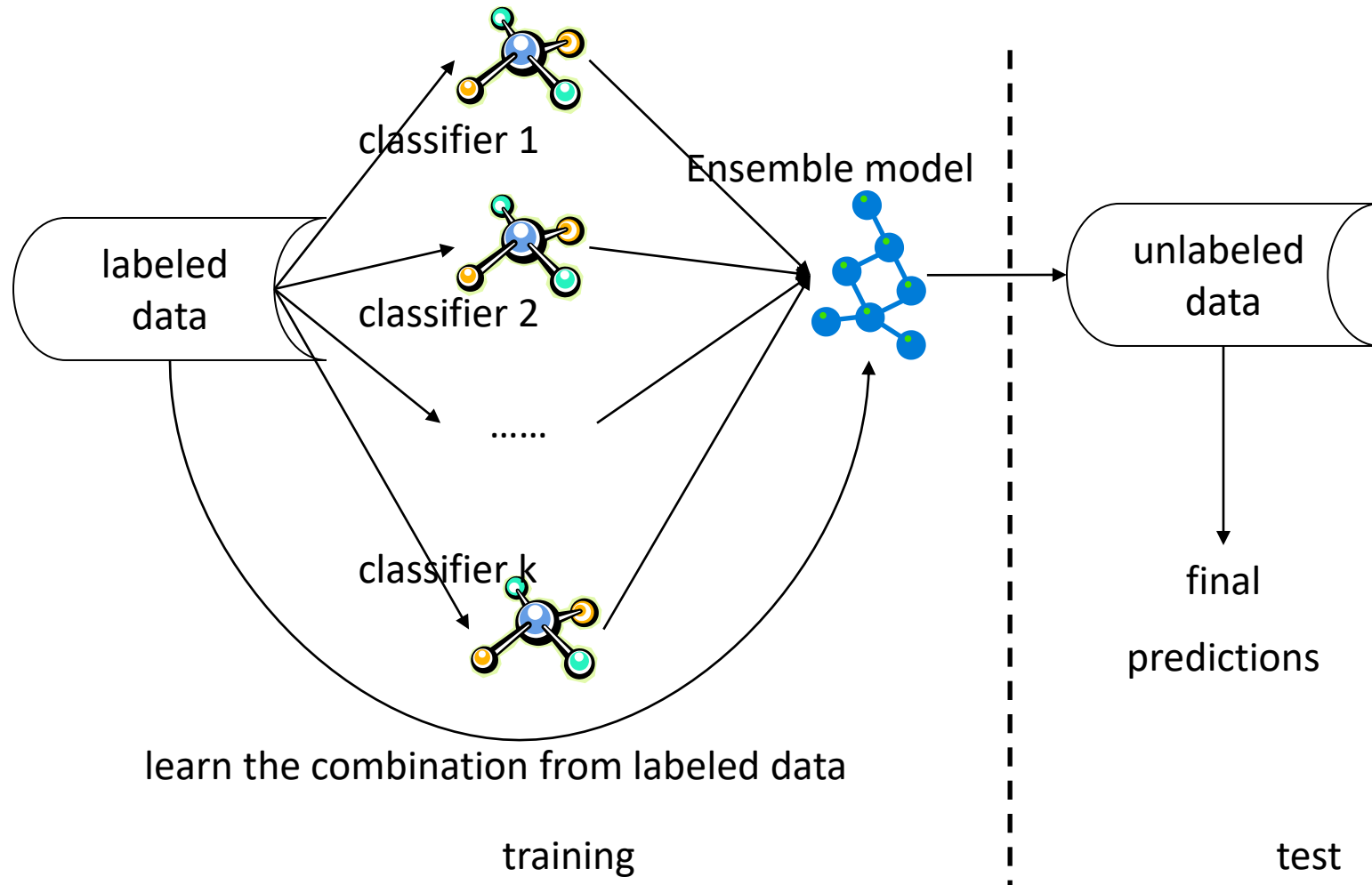


# Bagging: Generating Sub Models

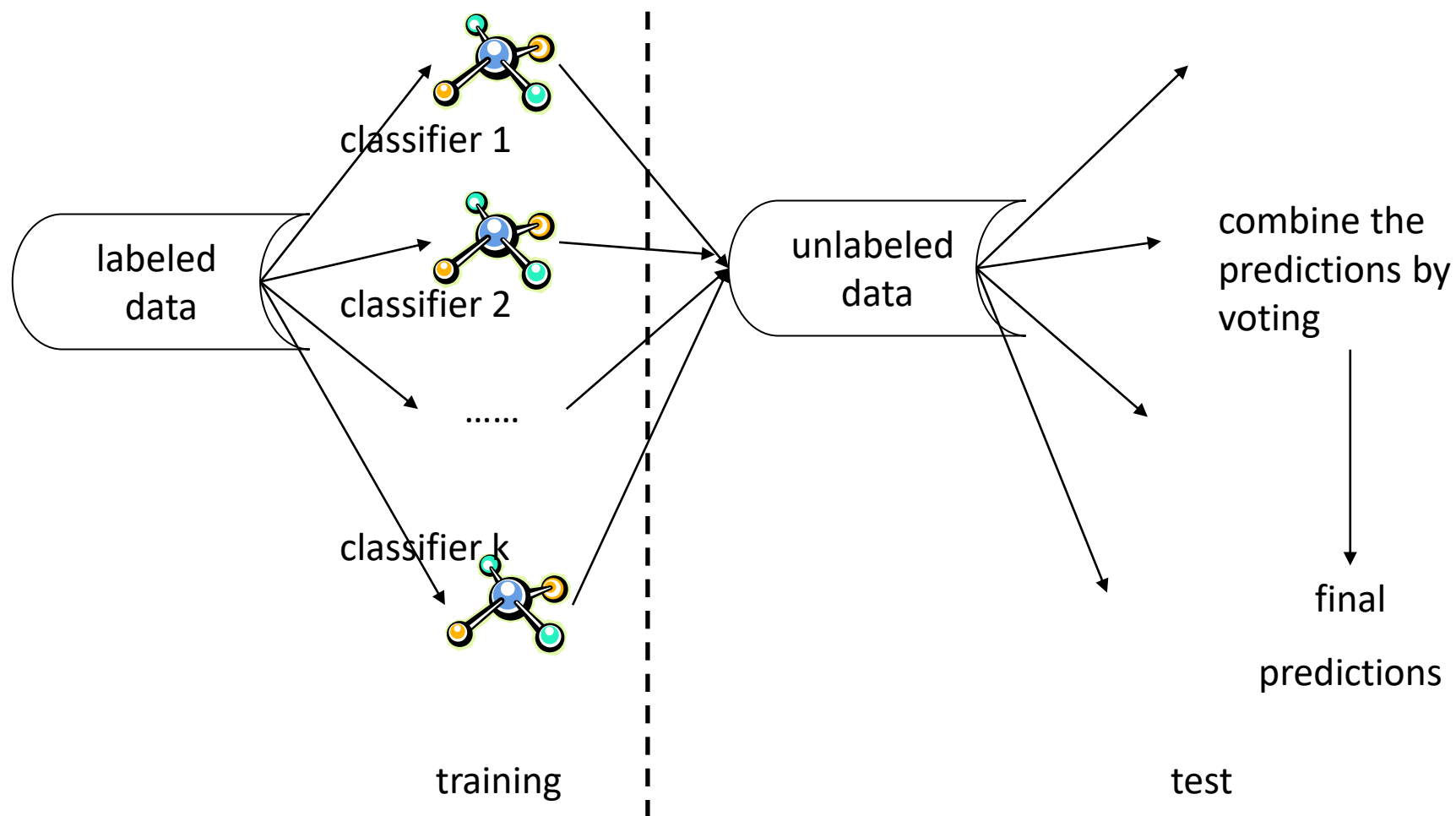
- Bootstrap
  - Sampling with replacement
  - Contains around 63.2% original records in each sampling
- Bootstrap Aggregation
  - Train a classifier on each bootstrap sampling
  - Combine the classifiers to obtain a stronger model

\*[Breiman96]

# Learn to Combine (Stacking)



# Majority Voting / Weighted Voting



# Pros and Cons

	Learning to combine	Majority voting
Pros	Get useful feedback from labeled data Can potentially improve accuracy	Do not need labeled data Can improve the generalization performance
Cons	Need to keep the labeled data to train the ensemble May overfit the labeled data Cannot work when no labels are available	No feedbacks from the labeled data Require the assumption that consensus is better

# Diversity in Ensemble Learning

- It is generally believed that to get a good ensemble, the base learners should be as more accurate as possible, and as more diverse as possible (Krogh and Vedelsby 1995).
  - However, no rigorous definition on what is intuitively perceived as diversity.
- In practice, the diversity of the base learners can be introduced from different channels, such as
  - Subsampling the training examples
  - Subsampling the features
  - Injecting randomness into learning algorithms (random seeds)
  - Or even using multiple mechanisms simultaneously

# Thanks!

<http://research.microsoft.com/users/taoqin/>

<http://web.ee.tsinghua.edu.cn/wqzhang>