

高等机器学习

机器学习框架

刘铁岩、张辉帅
微软亚洲研究院



清华大学
Tsinghua University

Outline

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

Taxonomy of Machine Learning

Machine Learning Taxonomy

Supervised Learning

- Learn prediction models from labeled data
 - Regression, classification, ranking
 - Can learn complex patterns or rules, but usually require many training examples
 - Usually assume i.i.d. distribution of the data, so as to ensure generalization ability

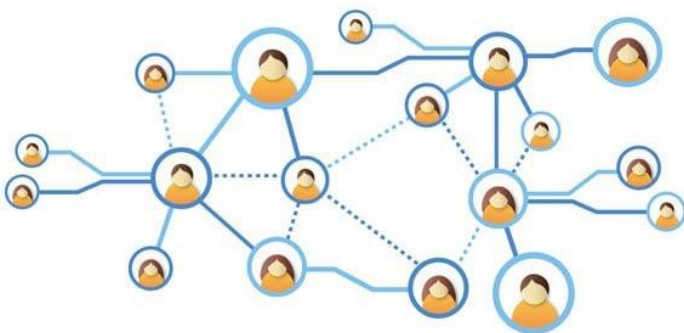
Unsupervised Learning

- Discover patterns in unlabeled data
 - Clustering, density estimation, importance ranking
 - Unnecessary to label massive training data, however, usually have difficulty in learning desirable decision functions or dealing with very complex data

Machine Learning Taxonomy

Semi-supervised Learning

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



Reinforcement Learning

- Learn from interactions with the environment
 - 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



Machine Learning Taxonomy

Discriminative Learning

- Learn the conditional probability $p(y|x)$ for discrimination
 - More direct decision-making power
 - Require relatively less data as compared to generative learning

Generative Learning

- Learn the joint probability $p(x, y)$ from generation
 - Modeling the data distribution instead of making decisions
 - Require relatively more data since joint probability contains more information than conditional probability

Framework of Supervised Learning

One Formula for Supervised Learning

The diagram illustrates the formula for supervised learning with several annotations in green text and lines pointing to specific parts of the equation:

- 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 itself is:

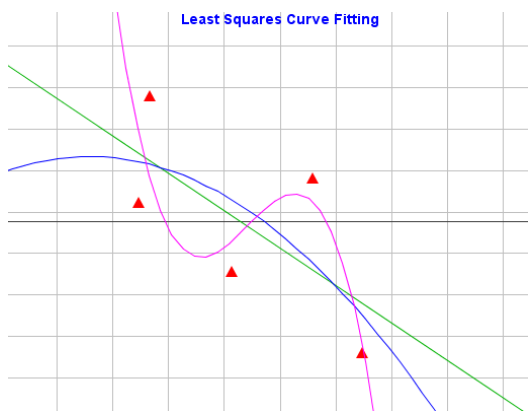
$$\omega^* = \arg \min_{\omega \in \Omega} \sum_{\substack{i=1, \dots, N \rightarrow \infty \\ x_i \in X, y_i \in Y \\ (x_i, y_i) \sim P}} L(f_\omega(x_i), y_i)$$

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

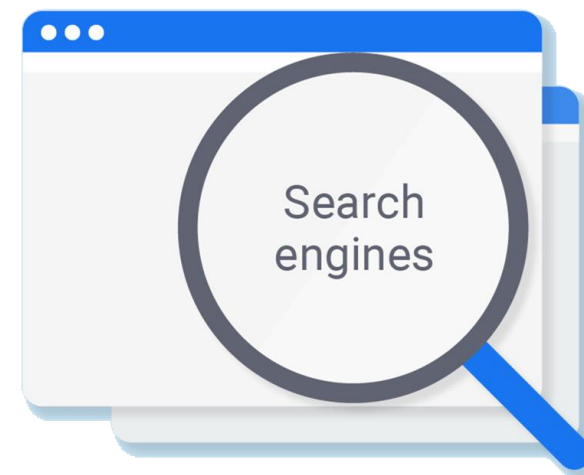


The Chicago Cubs
played a great game of
baseball.

Sports

The markets rallied
today sending the S&P
500 to a new 3 week
high to start the
financial year.

Finance



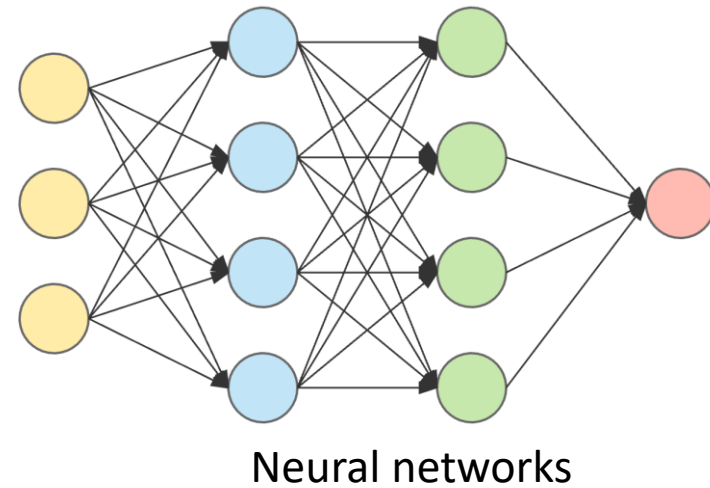
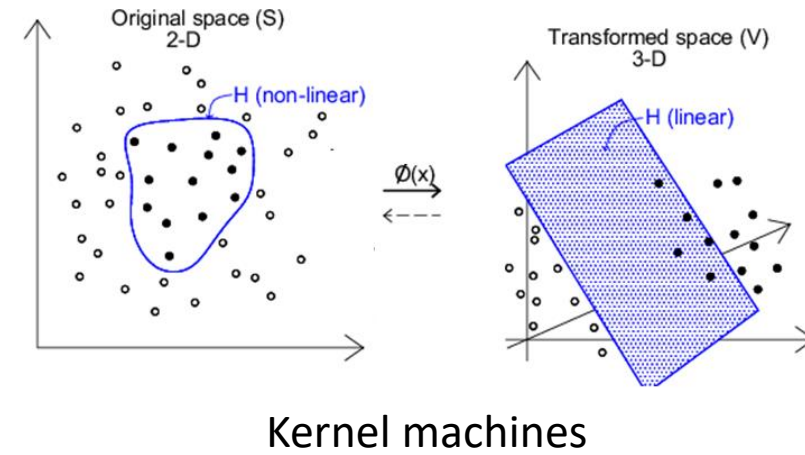
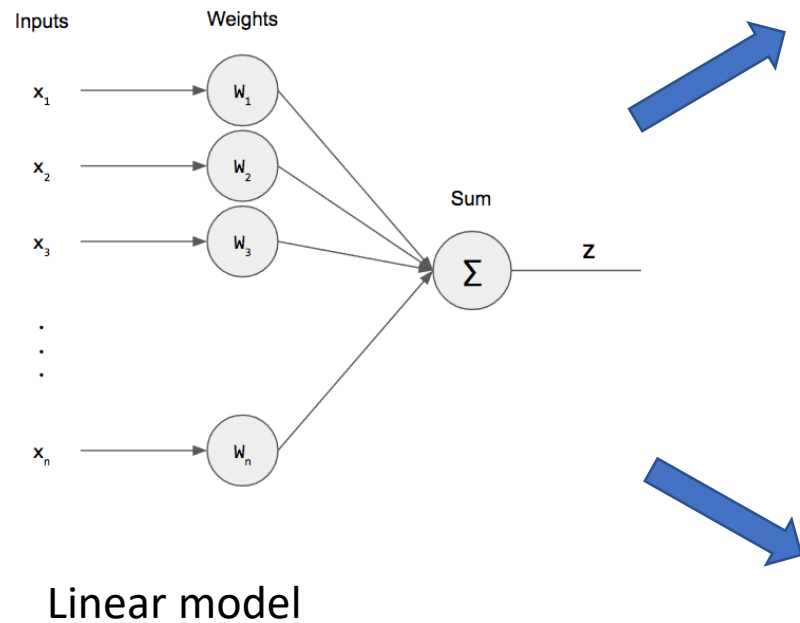
Output Space

- Note: there are two related but different definitions of output space.
 - Output space of the task (e.g., $\{+1, -1\}$ for classification)
 - Output space to facilitate learning (e.g., real numbers when using regression to solve classification tasks)

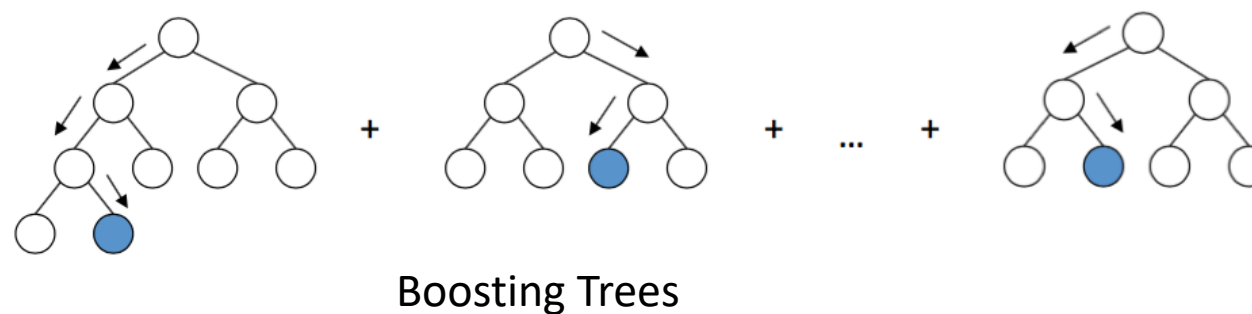
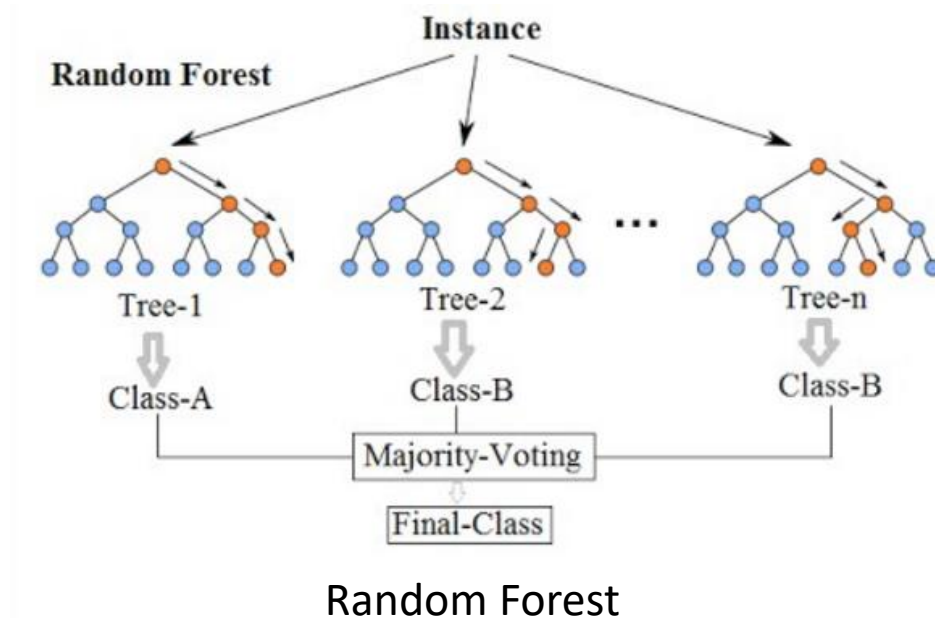
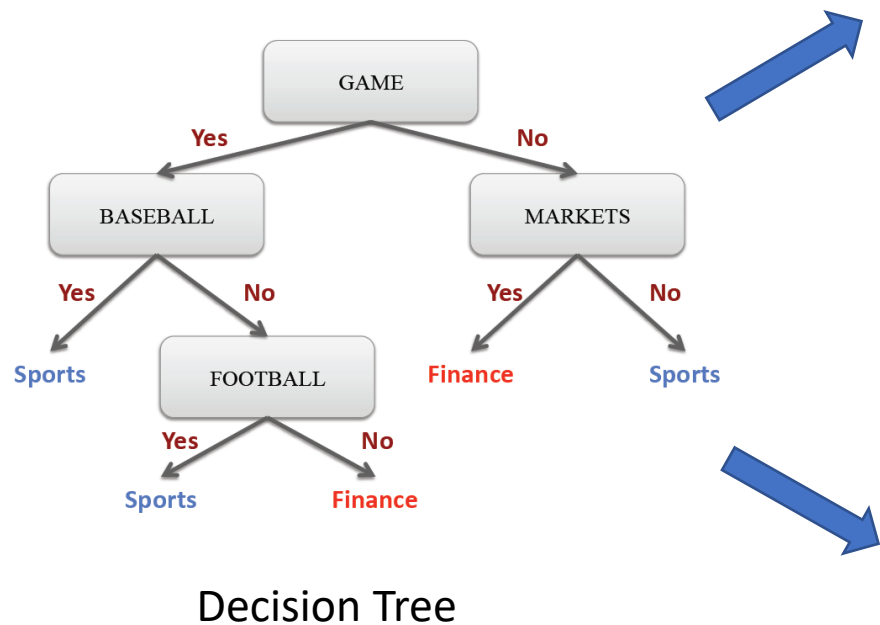
Hypothesis Space

- 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

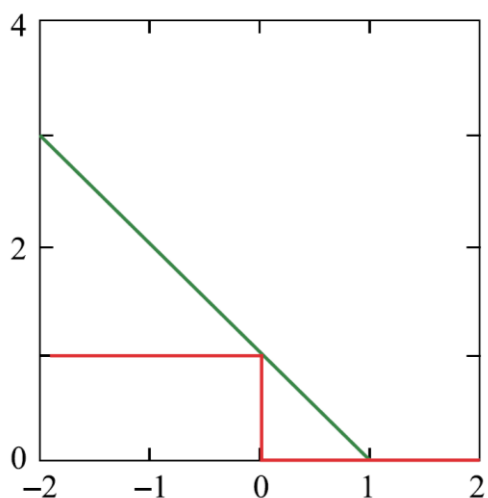


Loss Function

- 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 logistic loss for classification

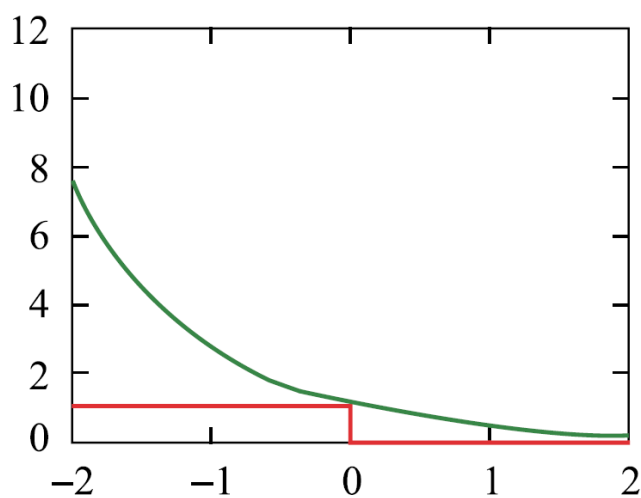
Commonly used Loss Functions

Hinge loss



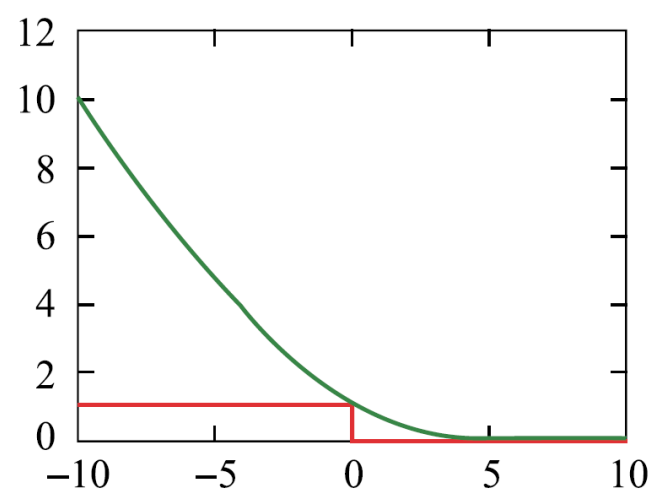
$$L(f_w; x, y) = \max\{0, 1 - yf_w(x)\}$$

Exponential loss



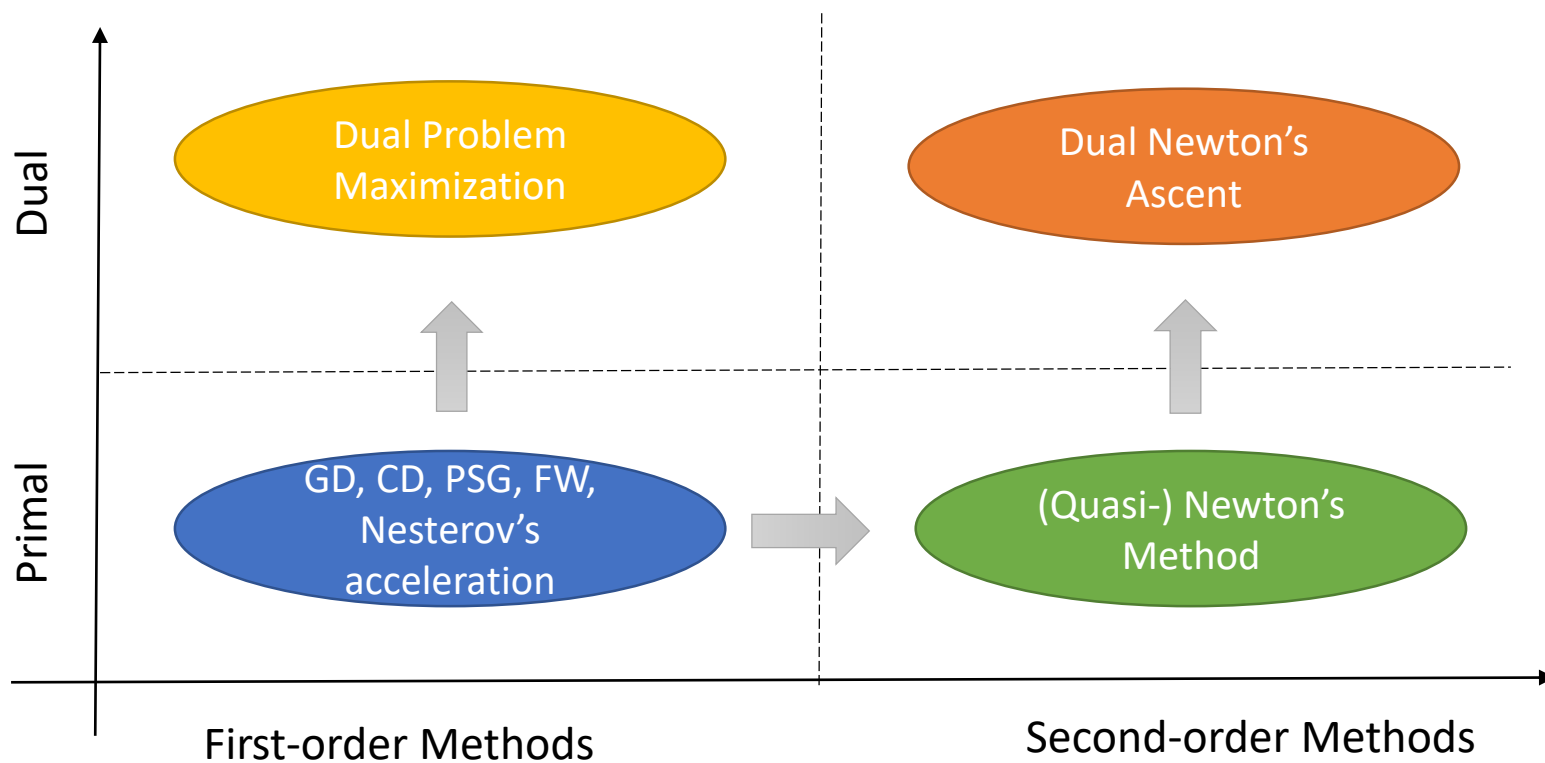
$$L(f_w; x, y) = \exp(-yf_w(x))$$

Cross entropy loss



$$L(f; x, y) = -\sum_{z \in \{-1, 1\}} I_{\{y=z\}} \log P(Y = z|x; f_w)$$

Optimization



Gradient Descent [Cauchy 1847]

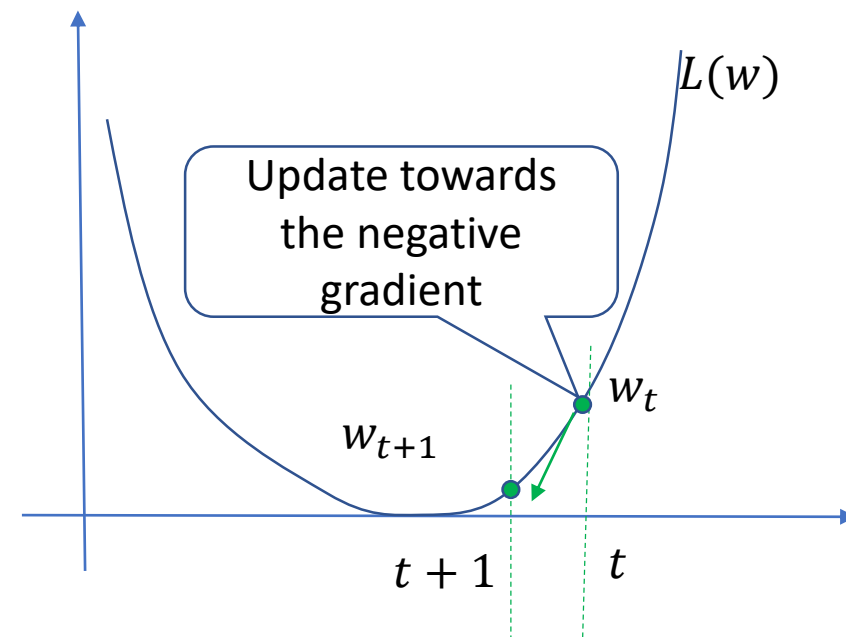
Motivation: to minimize the local first-order Taylor approximation of f

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

Update rule:

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

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



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

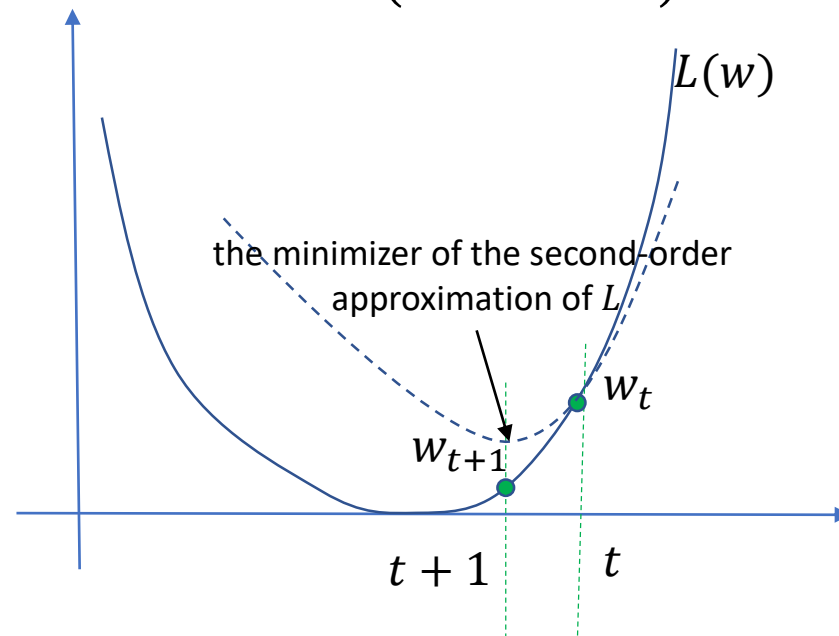
Motivation:

To minimize the local second-order Taylor approximation of f :

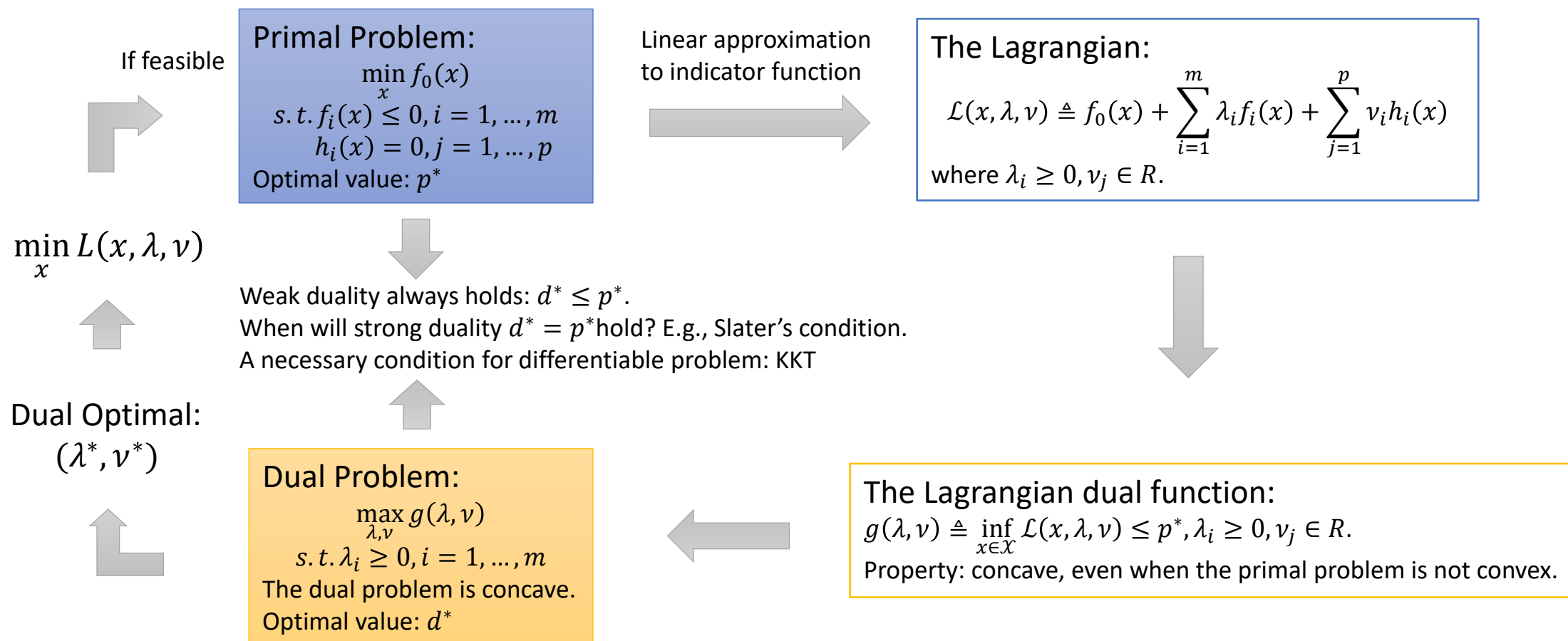
$$\begin{aligned} \min_w L(w) \\ \approx \min_w L(w_t) + \nabla L(w_t)^\top (w - w_t) + \frac{1}{2} (w - w_t)^\top \nabla^2 L(w_t) (w - w_t) + o(\|w - w_t\|^2). \end{aligned}$$

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 Methods



Machine Learning Models

Outline

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

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

Linear Regression

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

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

- 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

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

Probabilistic Explanation

- The log likelihood

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

- Minimization 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 .

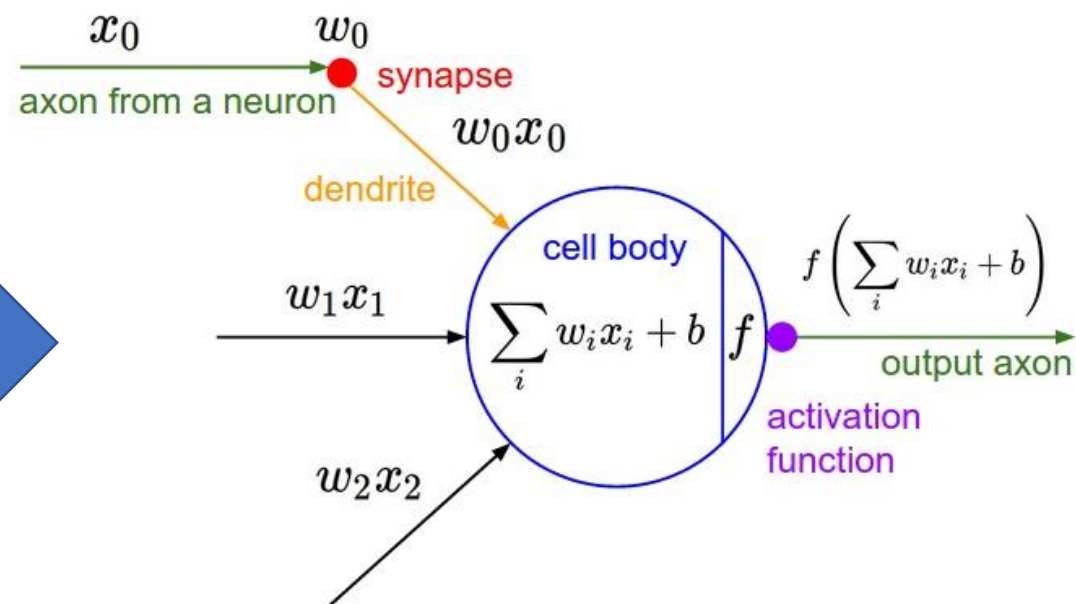
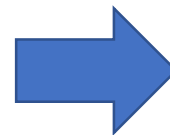
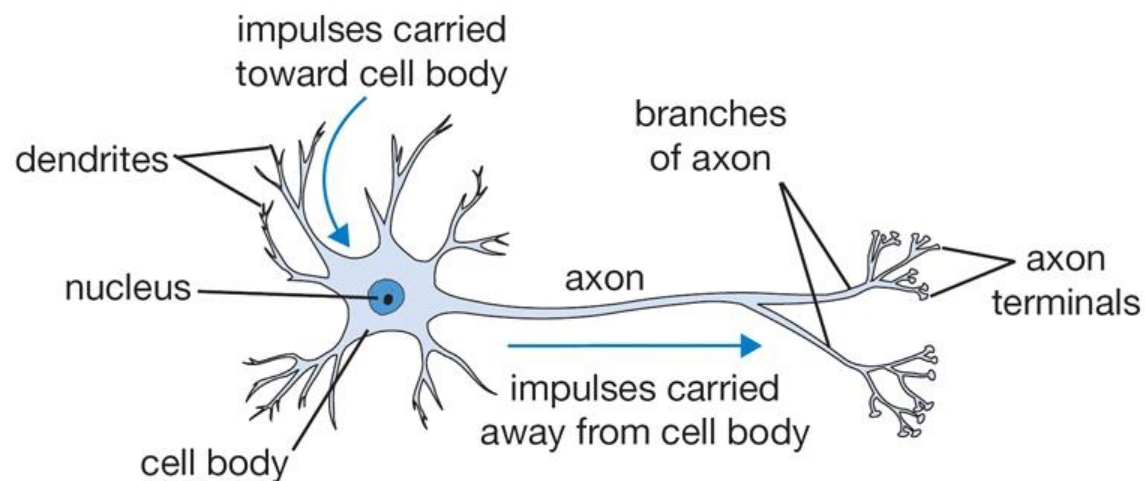
Outline

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

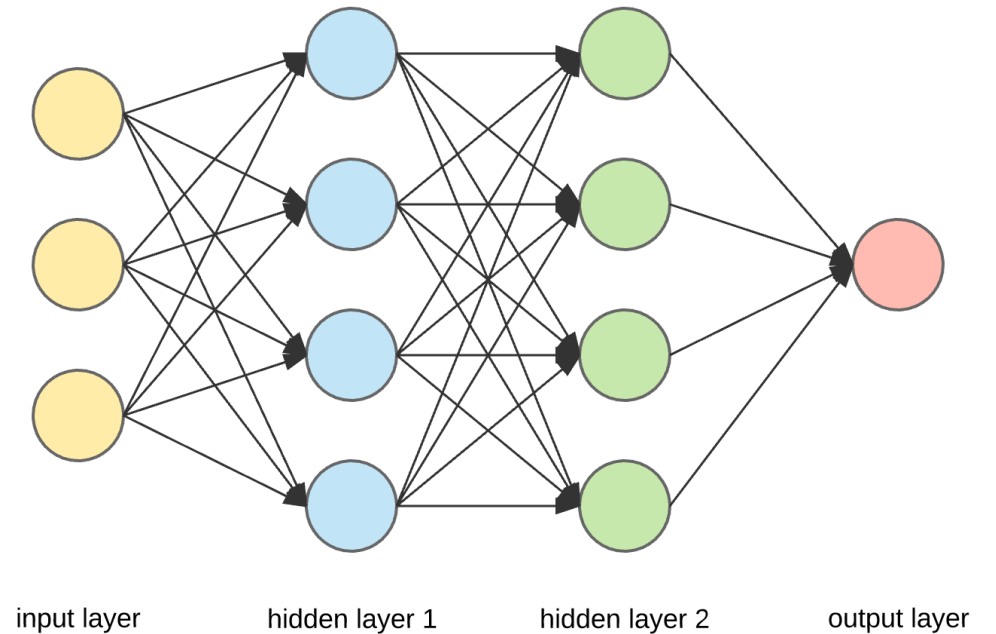
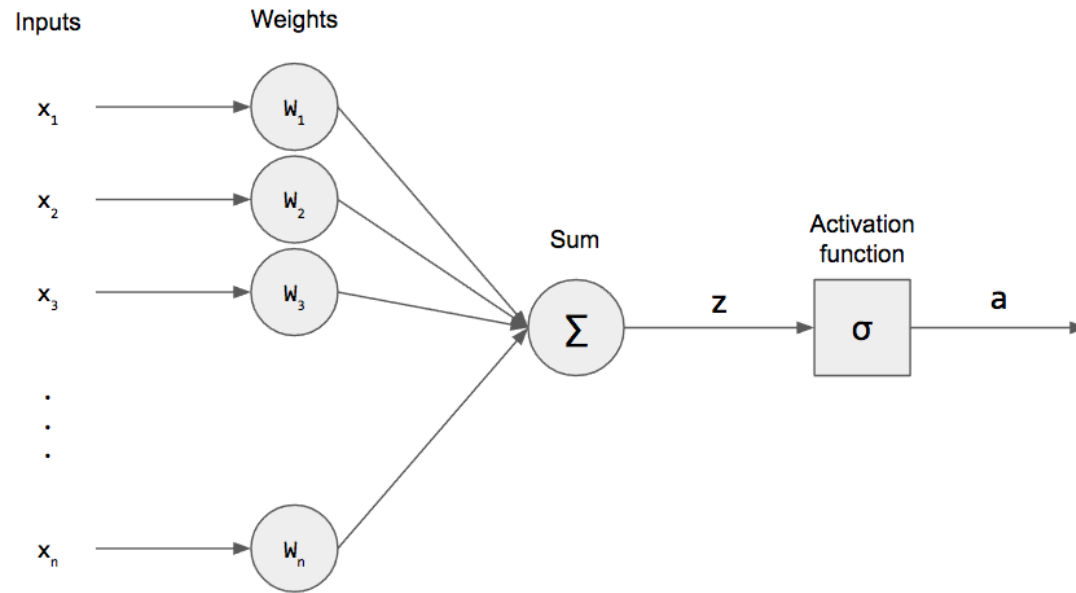
*Neural networks, random forest, gradient boosting trees will be introduced in separate sections.

Perceptron

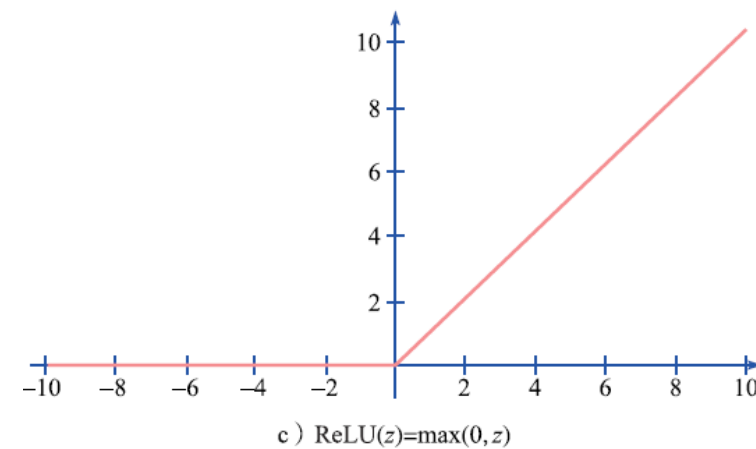
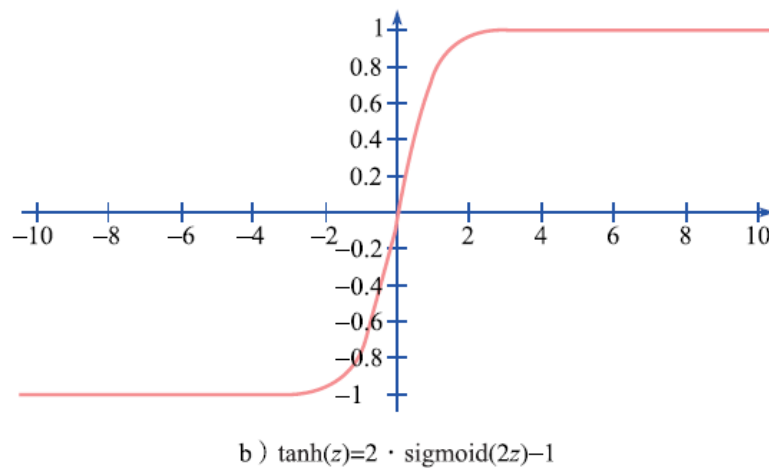
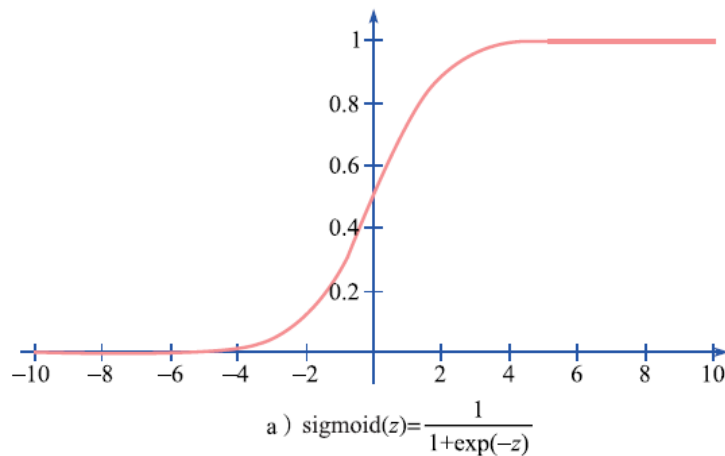
Dendrite, 树突, 接收信号
Synapse, 突触, 放大信号
Axon, 轴突, 输出信号



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 φ ; 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 Decisions

- Learning Algorithm
 - Issues: local minima, over-fitting, training time
 - Momentum, Nesterov's accelerated algorithm, Ada-series algorithms
- Network architecture
 - Number of hidden layers, nodes per layer, skip connections
 - Feedforward, convolutional, recurrent, etc.



End of 1st week

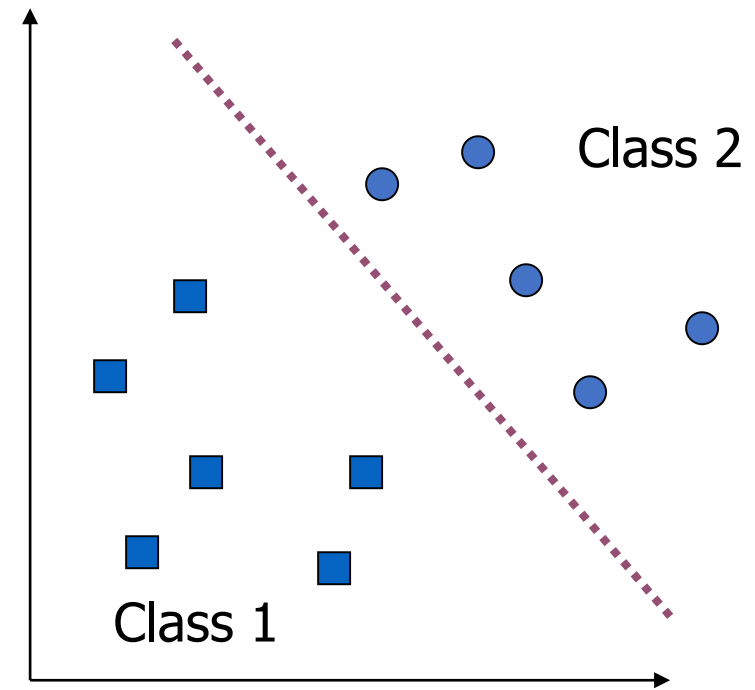
Outline

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

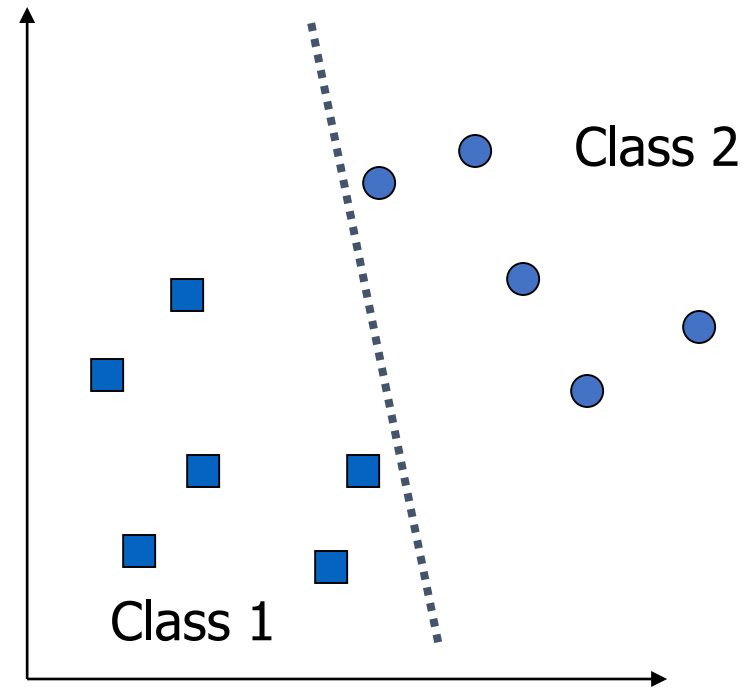
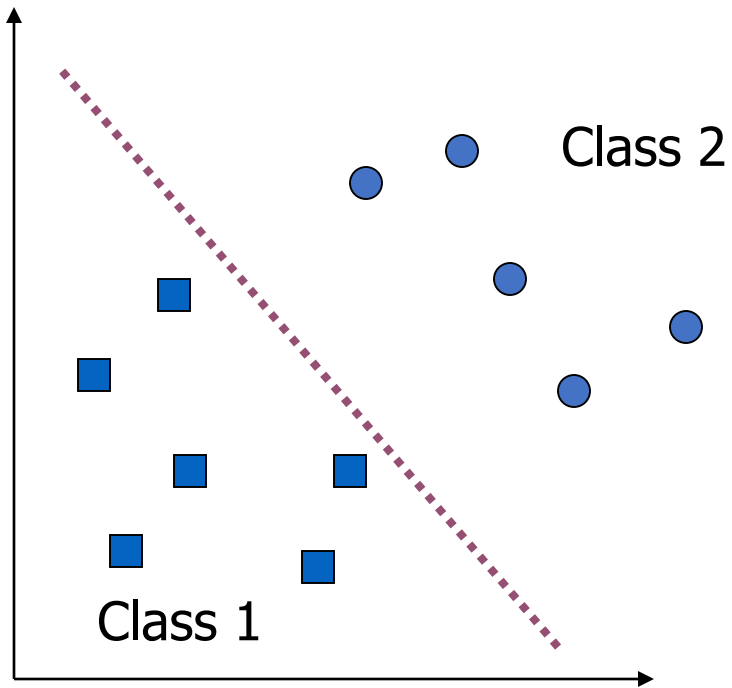
*Neural networks, random forest, gradient boosting trees will be introduced in separate sections.

What is a Good Decision Boundary?

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

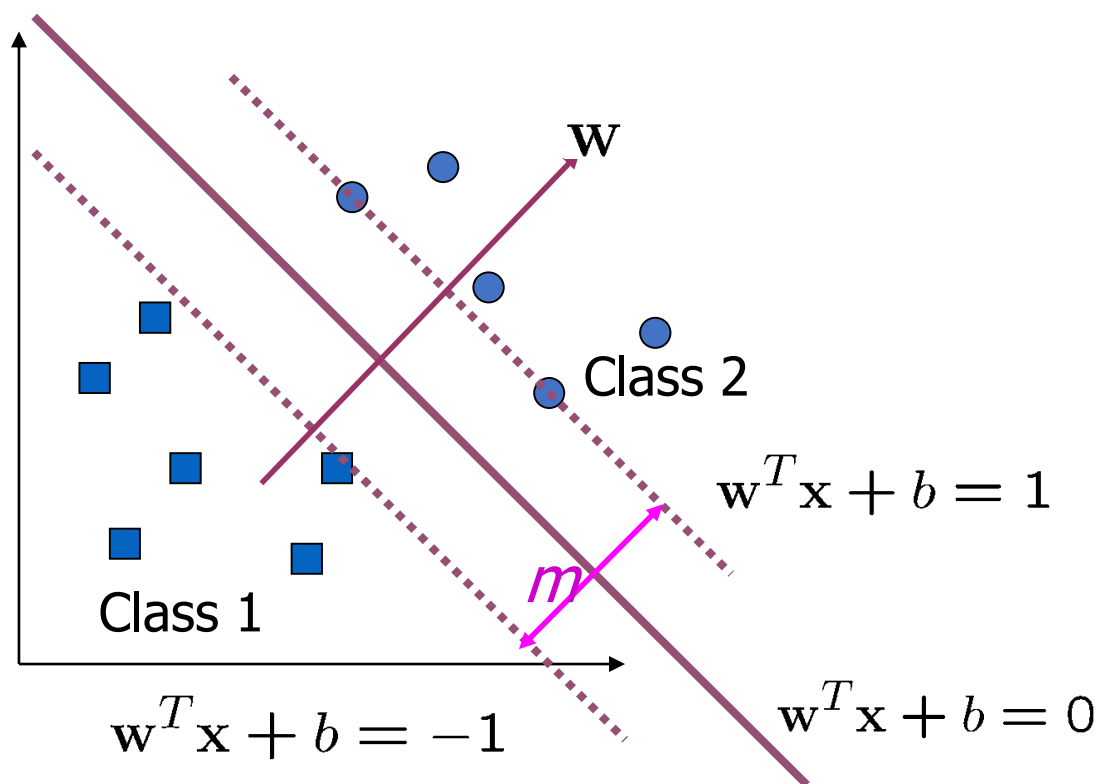


Possibly Bad Decision Boundaries



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)



$$m = \frac{2}{\|w\|}$$

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
→ $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall i$
- 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, the optimization problem can be written in terms of α_i (the dual problem)

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

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

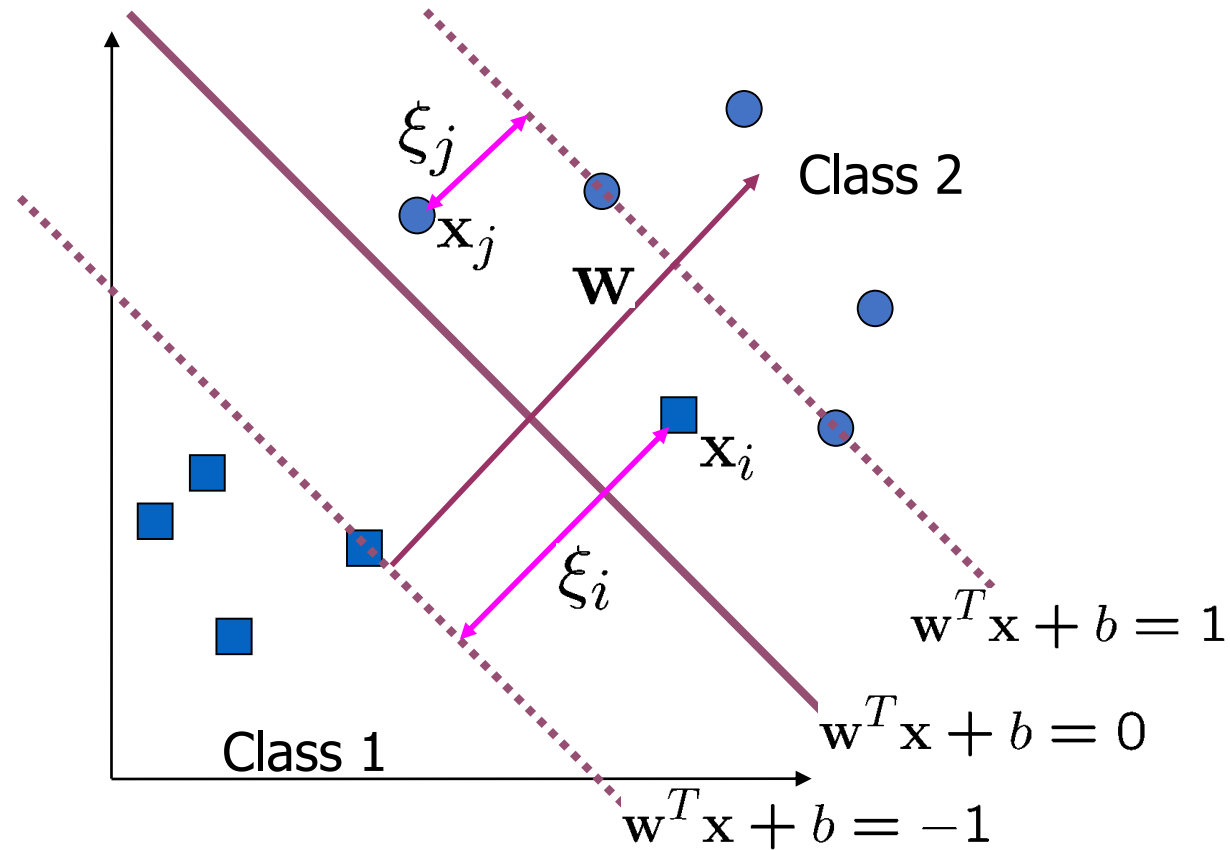
- This is a quadratic programming (QP) problem
 - A global maximum of α_i can always be found

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

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.

Non-linearly Separable Problems

- We allow “error” ξ_i in classification



Soft Margin Hyperplane

- By minimizing $\sum_i \xi_i$, ξ_i can be obtained by

$$\begin{cases} \mathbf{w}^T \mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T \mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

ξ_i are “slack variables” in optimization;
 $\xi_i = 0$ if there is no error for x_i ,
 ξ_i is an upper bound of the number of errors

- 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

$$\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{subject to } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Optimization Problem

- The dual of the problem is

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

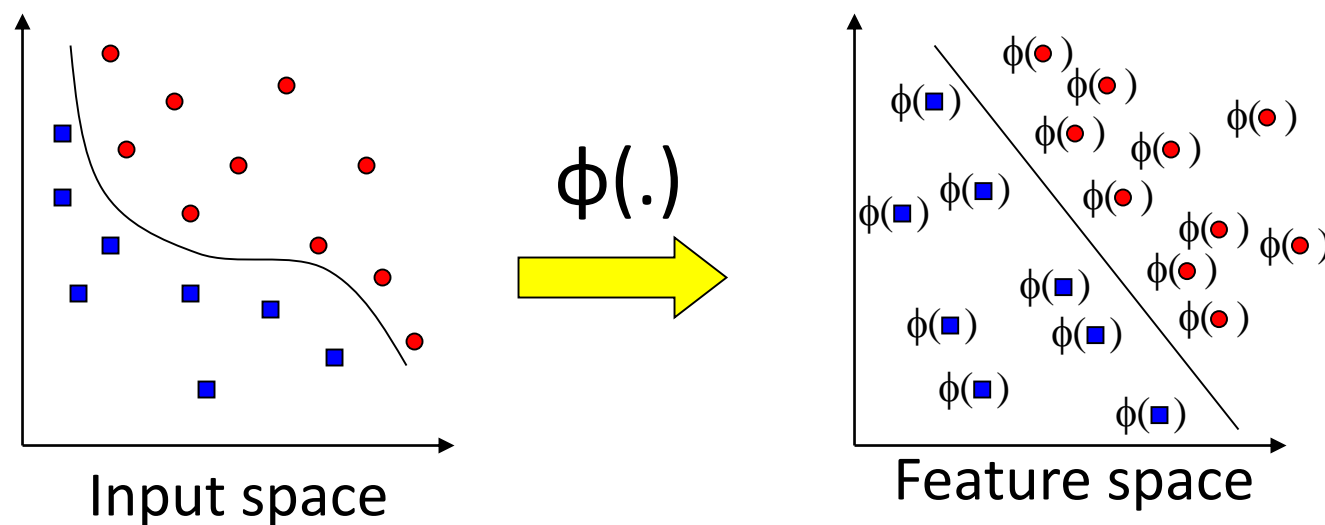
- \mathbf{w} is recovered as $\mathbf{w} = \sum_{j=1}^s \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound C on α_i now
- Once again, a QP solver can be used to find α_i

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 transform?
 - Linear operation in the feature space is equivalent to non-linear operation in input space
 - Classification can become easier with a proper transformation.

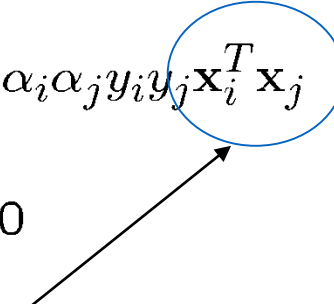
Transforming the Data

- Computation in the feature space can be costly because it is high dimensional
 - The feature space is typically infinite-dimensional!
- 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**
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function K by $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

An Example for $f(\cdot)$ and $K(\cdot, \cdot)$

- 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 $\varphi(\cdot)$) is specified. Kernel function can be thought of as a similarity measure between the input objects
- 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 a high dimensional space.

Examples of Kernel Functions

- Polynomial kernel with degree d

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

- Radial basis function kernel with width s

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

- Closely related to radial basis function neural networks

- Sigmoid with parameter k and q

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$$

- It does not satisfy the Mercer condition on all k and q

Modification Due to Kernel Function

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

Original

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



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

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 $f < 0$

Original

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

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



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

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

Outline

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

*Deep neural networks, random forest, gradient boosting trees will be introduced in separate sections.

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



Building a Decision Tree

- ID3 Algorithm
 - Take all unused attributes and count their entropy concerning test samples
 - Choose attribute for which entropy is minimum (or, equivalently, information gain is maximum)
 - Make node containing that attribute
- C4.5 Algorithm
 - Improvement over ID3

ID3 Algorithm

- ID3 (Examples, target_attribute, attributes)
 - If all target_attribute examples have the same label
 - return the single-node tree root with that label
 - Else
 - Create root using A = attribute that best classifies examples (in terms of entropy)
 - Add branches for each value of attribute A
 - Create a subtree for each value from: ID3(examples, target_attribute, attributes – A)

C4.5 Algorithm

- Handling both continuous and discrete attributes
 - In order to handle continuous attributes, C4.5 creates a threshold and then splits the list into those whose attribute value is above the threshold and those that are less than or equal to it.
- Handling training data with missing attribute values
 - C4.5 allows attribute values to be marked as ? for missing. Missing attribute values are simply not used in gain and entropy calculations.
- Pruning trees after creation
 - C4.5 goes back through the tree once it's been created and attempts to remove branches that do not help by replacing them with leaf nodes.

Design Choices

- Complexity of tree
 - The depth? The number of nodes?
- Pruning
 - Reduces over-fitting
- Selecting informative choices
 - Which features to select at each point?
 - “Attribute that best classifies examples”

Outline

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

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

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

- Ask professional.
- Suppose:
 - Professional 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 $D_1(i)=1/m$; (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 ε_t on D_t :

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

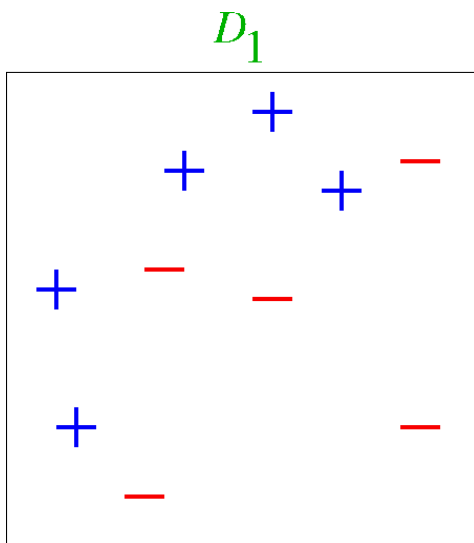
$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \quad \alpha_t = \log(1 - \varepsilon_t / \varepsilon_t)$$

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

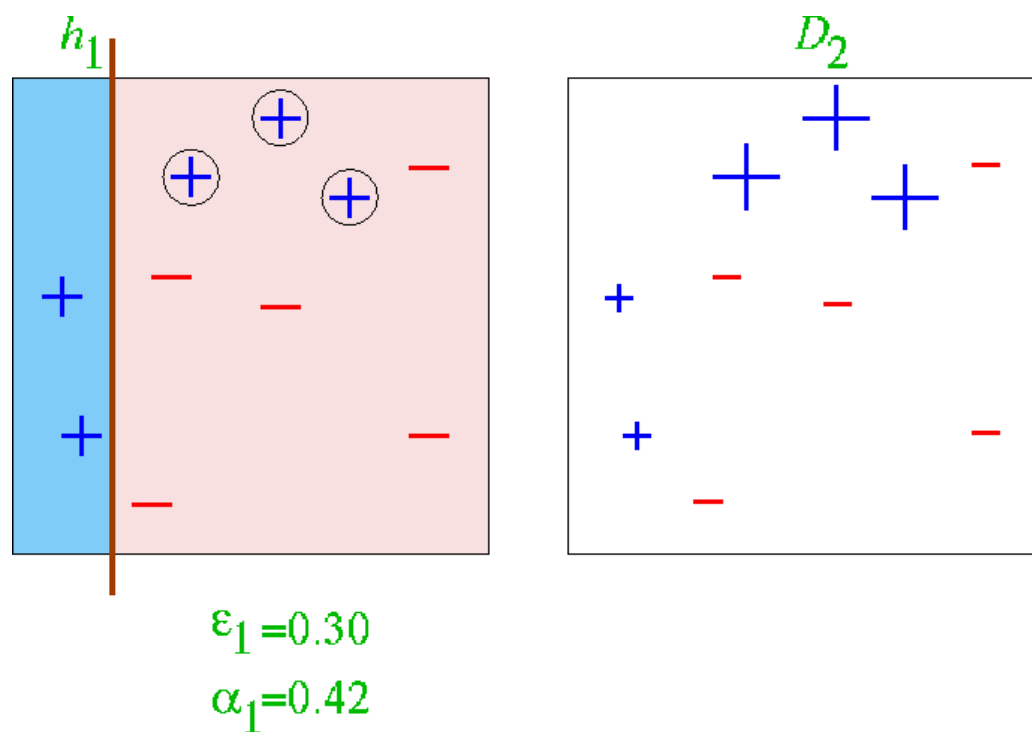
- output final hypothesis

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

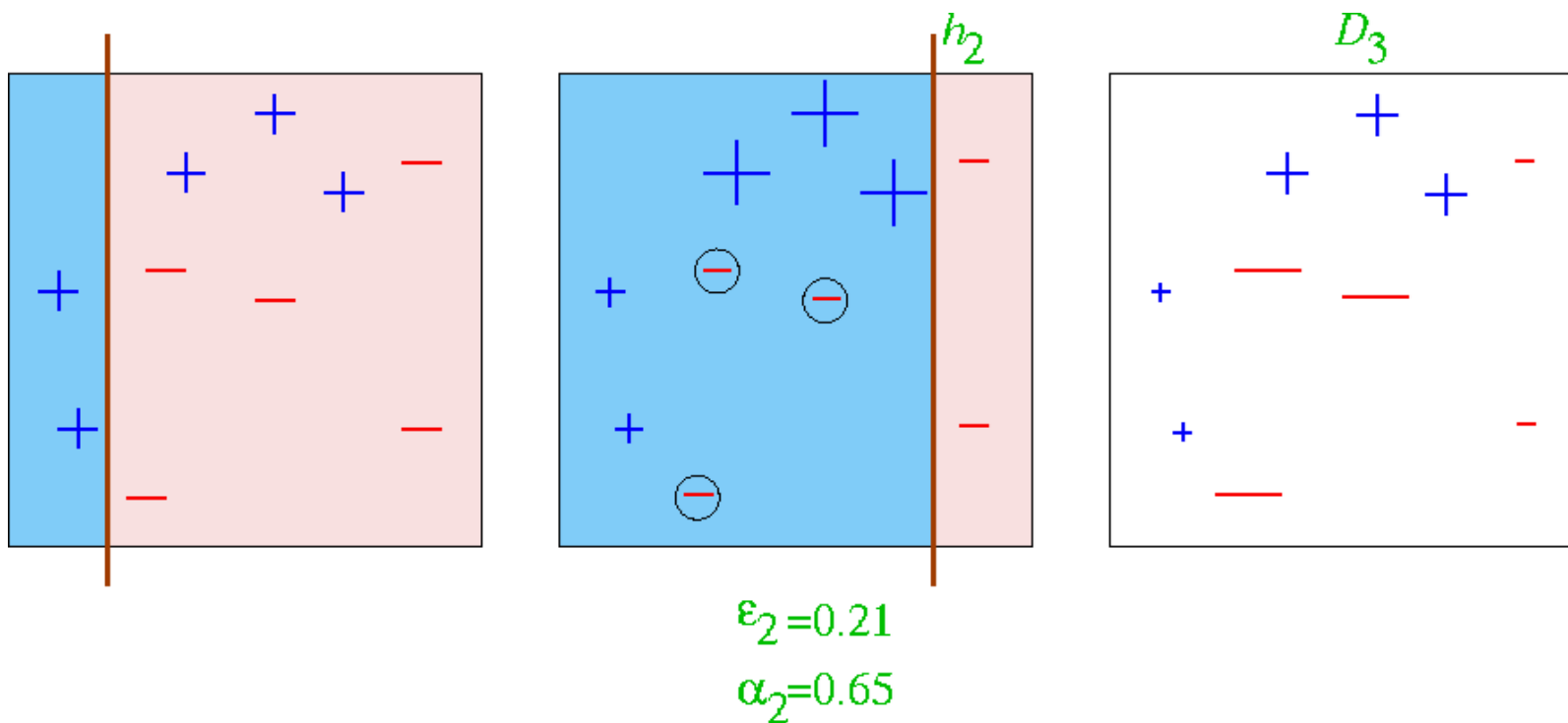
Toy Example



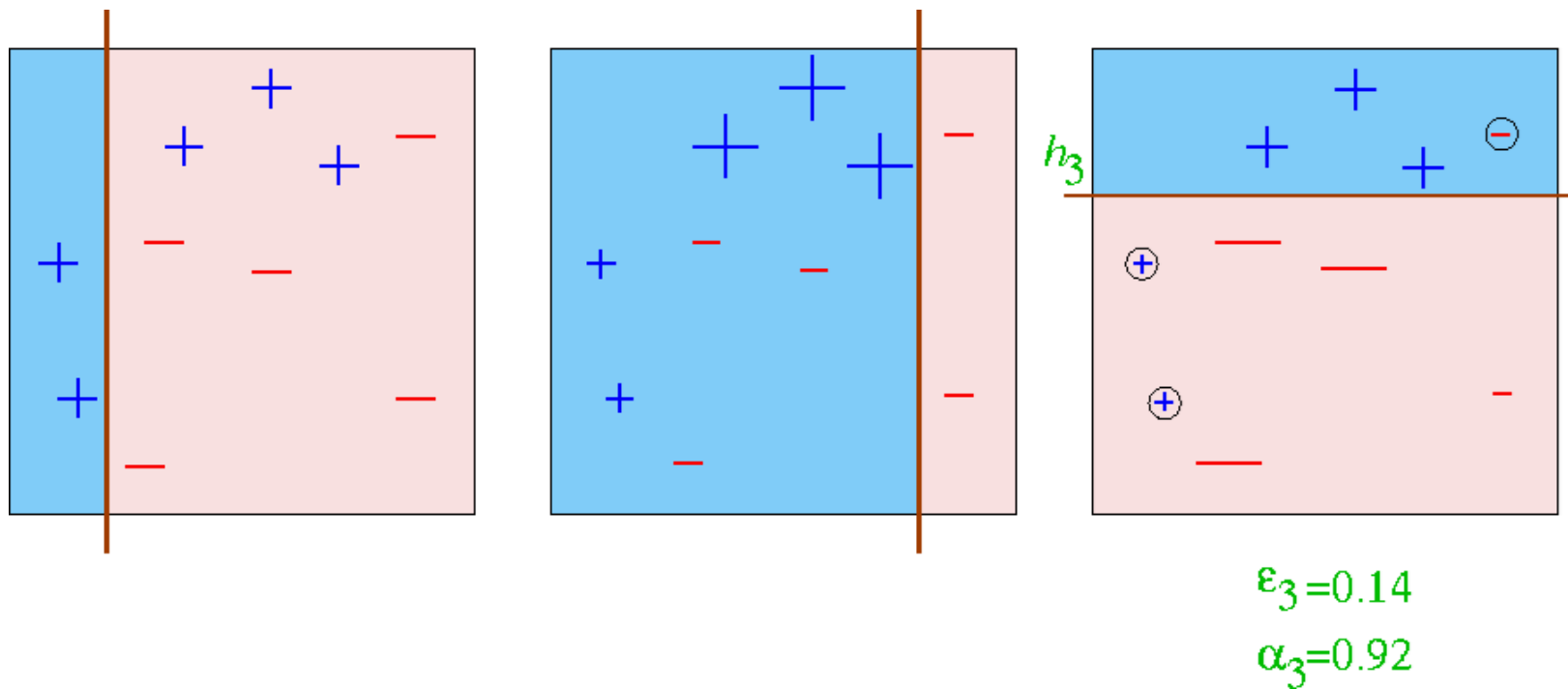
Round 1



Round 2

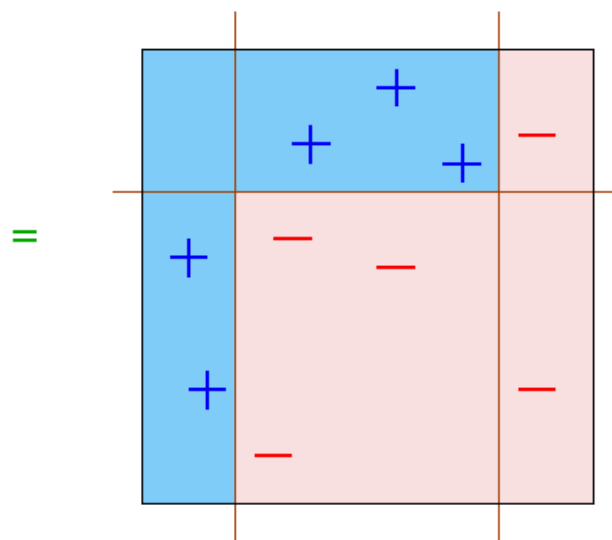


Round 3



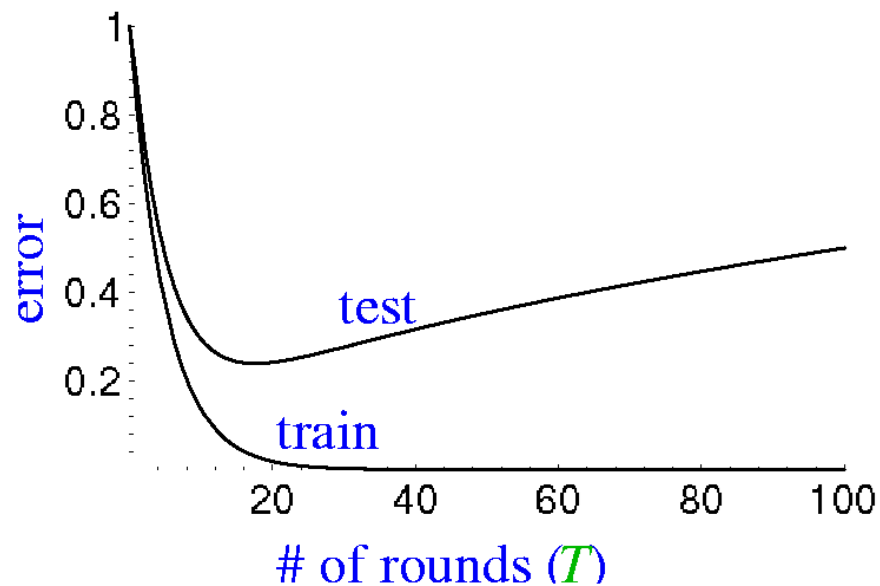
Final Hypothesis

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



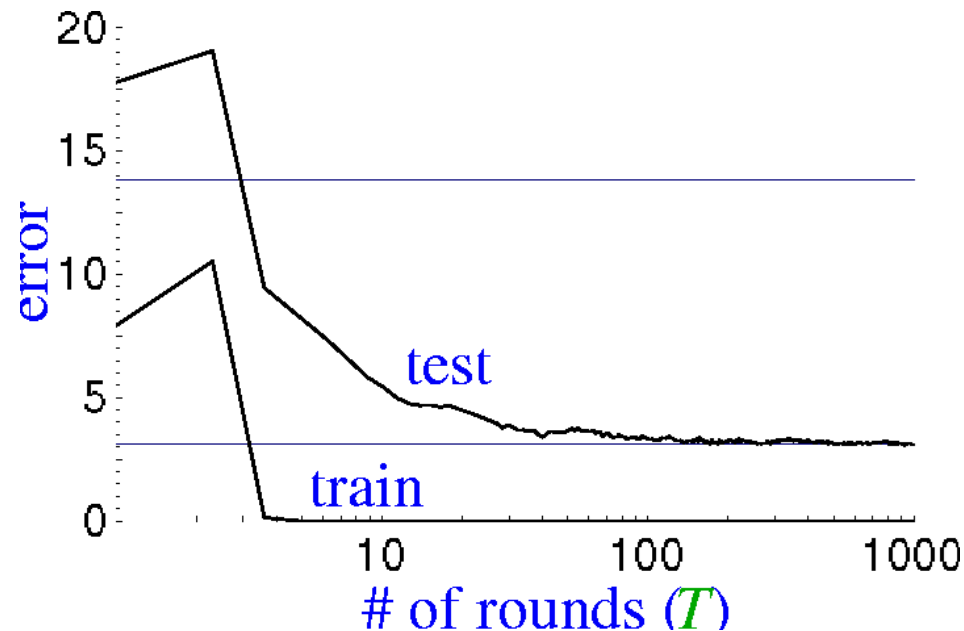
Problem of Overfitting

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



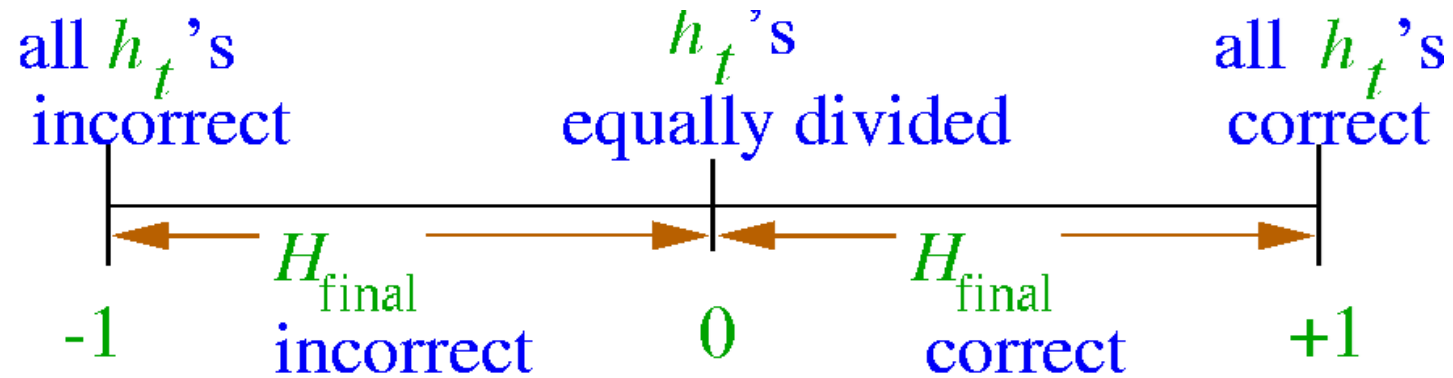
Boosting Seems to Avoid Overfitting

- 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)

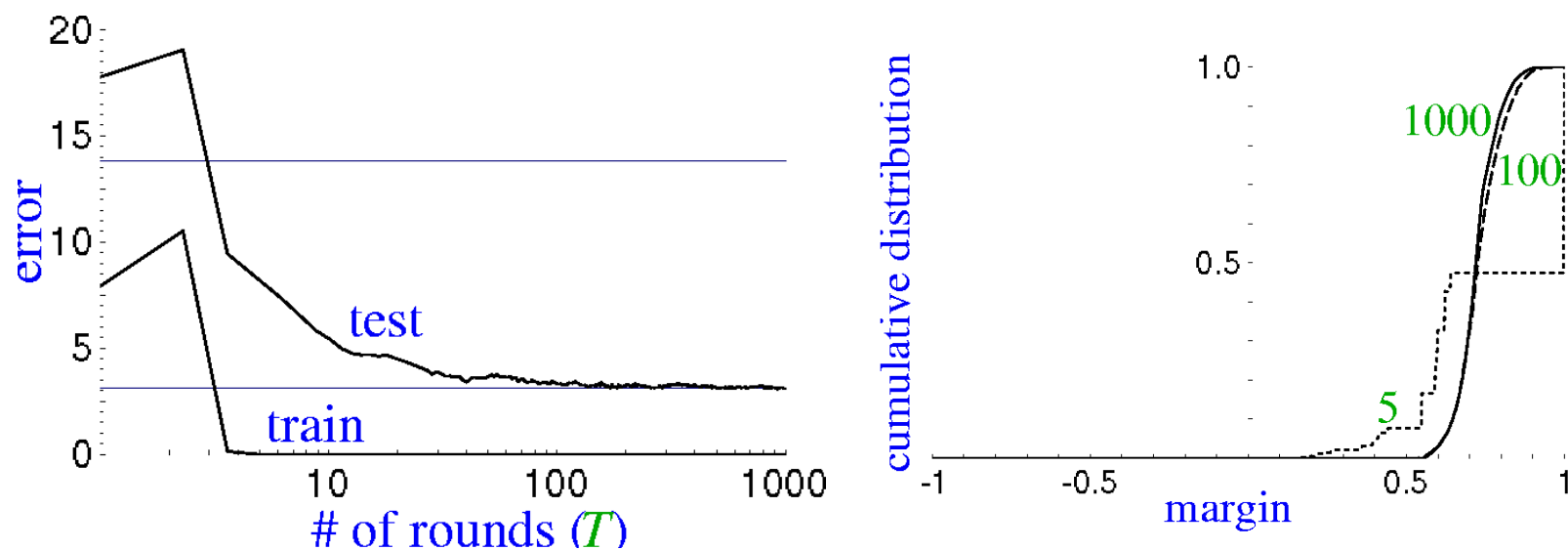
A Margin Explanation



In SVM: if the confidence score of all training point is high (far from the classification boundary), the margin is large.

Thus, we can use the average confidence score for the representation of margin.

The Margin Distribution



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

Schapire's Margin Distribution Bound

- Schapire et al. also demonstrated 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.

$$P_D(yf(x) \leq 0) \leq \inf_{\theta \in (0,1]} \left[P_S(yf(x) \leq \theta) + O\left(\frac{1}{\sqrt{n}} \left(\frac{\log n \log |H|}{\theta^2} + \log \frac{1}{\delta} \right)^{1/2} \right) \right]$$

Breiman's Doubt

- The minimum margin bound (Breiman1999)

$$\forall_{\delta} P_D(yf(x) \leq 0) \leq R \left(\log 2n + \log \frac{1}{R} + 1 \right) + \frac{1}{n} \log \left(\frac{|H|}{\delta} \right).$$

- Sharper than the margin distribution bound, $O\left(\frac{\log n}{n}\right)$ vs. $O\left(\sqrt{\frac{\log n}{n}}\right)$
- If the bound of Schapire et al. implies that the margin distribution is the key to the generalization error, Breiman's bound implies more strongly that the minimum margin governs the generalization error.

Breiman's Doubt

- Breiman proposed an algorithm named Arc-gv, which provably generates the largest possible minimum margin among all boosting type algorithms.
- However, arc-gv performs **consistently worse** than AdaBoost although it always generates larger minimum margins, as well as uniformly better margin distribution than AdaBoost.
- Breiman concluded that neither the margin distribution nor the minimum margin is the right explanation!

Schapire's Response

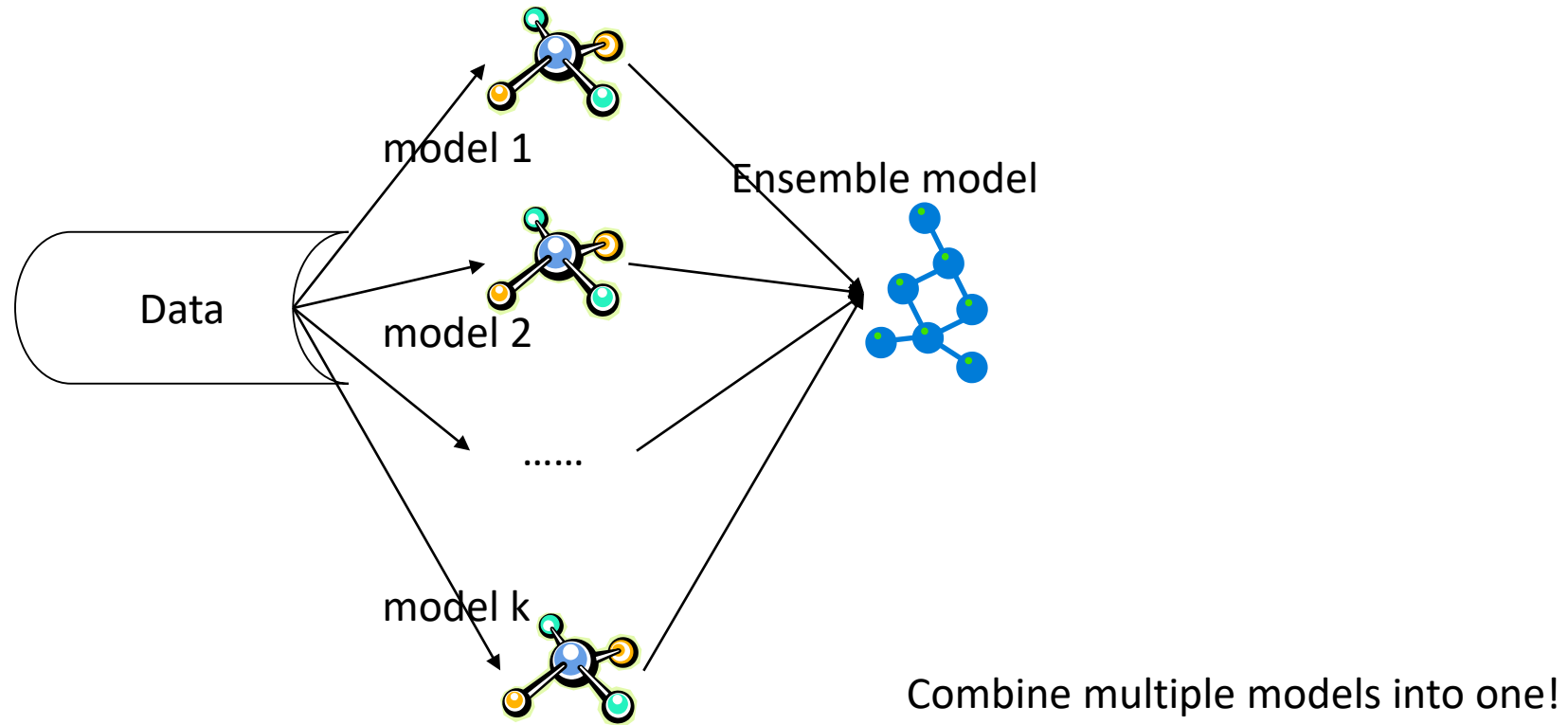
- Breiman's experiment:
 - Base classifiers: Using decision trees of a fixed number of leaves.
- Schapire's new discovery:
 - Trees generated by arc-gv are much deeper than those generated by AdaBoost!
 - Deeper trees are more complex even though the number of leaves are the same!
 - Breiman's experiment is not a fair comparison.
- With a fair comparison:
 - Base classifier: decision stump.
 - AdaBoost has better performance than arc-gv.
 - Arc-gv has larger minimum margins than AdaBoost, but a worse margin distribution.

The Story Continues

- Has Breiman's doubt been fully answered?
 - Arc-gv generates larger minimum margin yet has worse performance, which is contradicting to the (sharper) minimum margin bound!
 - What does it mean a “better” margin distribution?
- Recent theories
 - L. Wang, et al. On the Margin Explanation of Boosting Algorithms, COLT 2008 (a new margin bound based on E-margin, consistent with experiments).
 - W. Gao and Z.-H. Zhou. On the doubt about margin explanation of boosting. Artificial Intelligence, 2013 (a tighter margin bound, consistent with experiments)

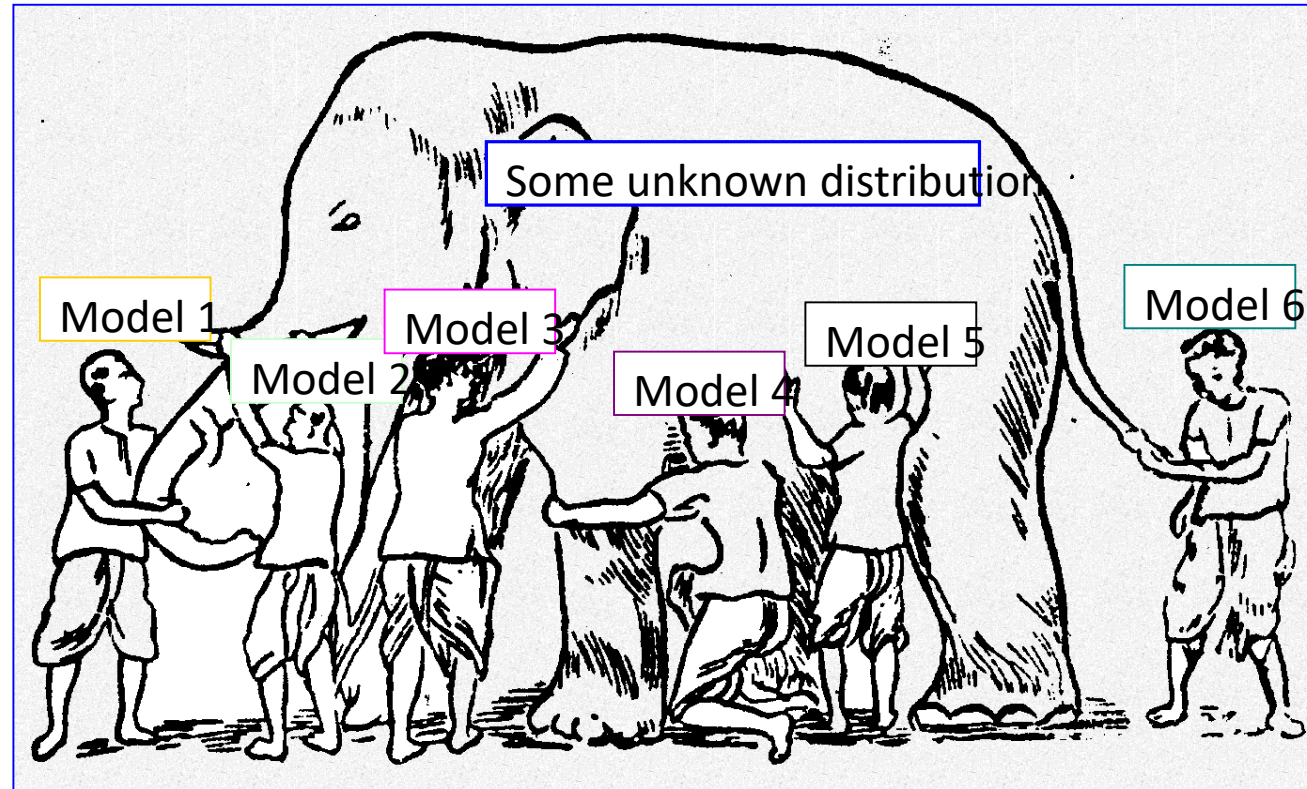
Model Ensemble

Ensemble



Why Ensemble Works?

- Ensemble gives the global picture!

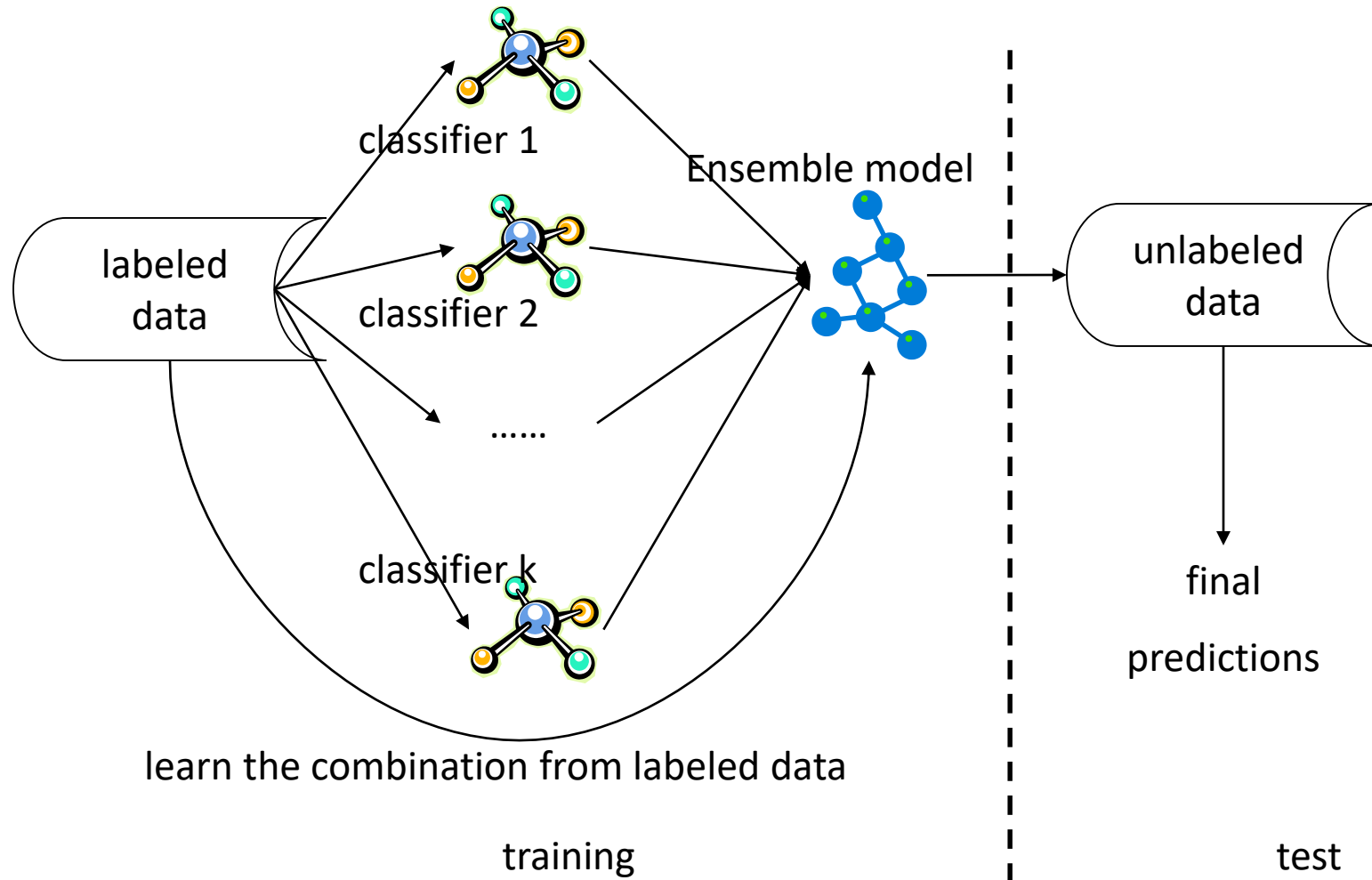


Bagging: Generating Sub Models

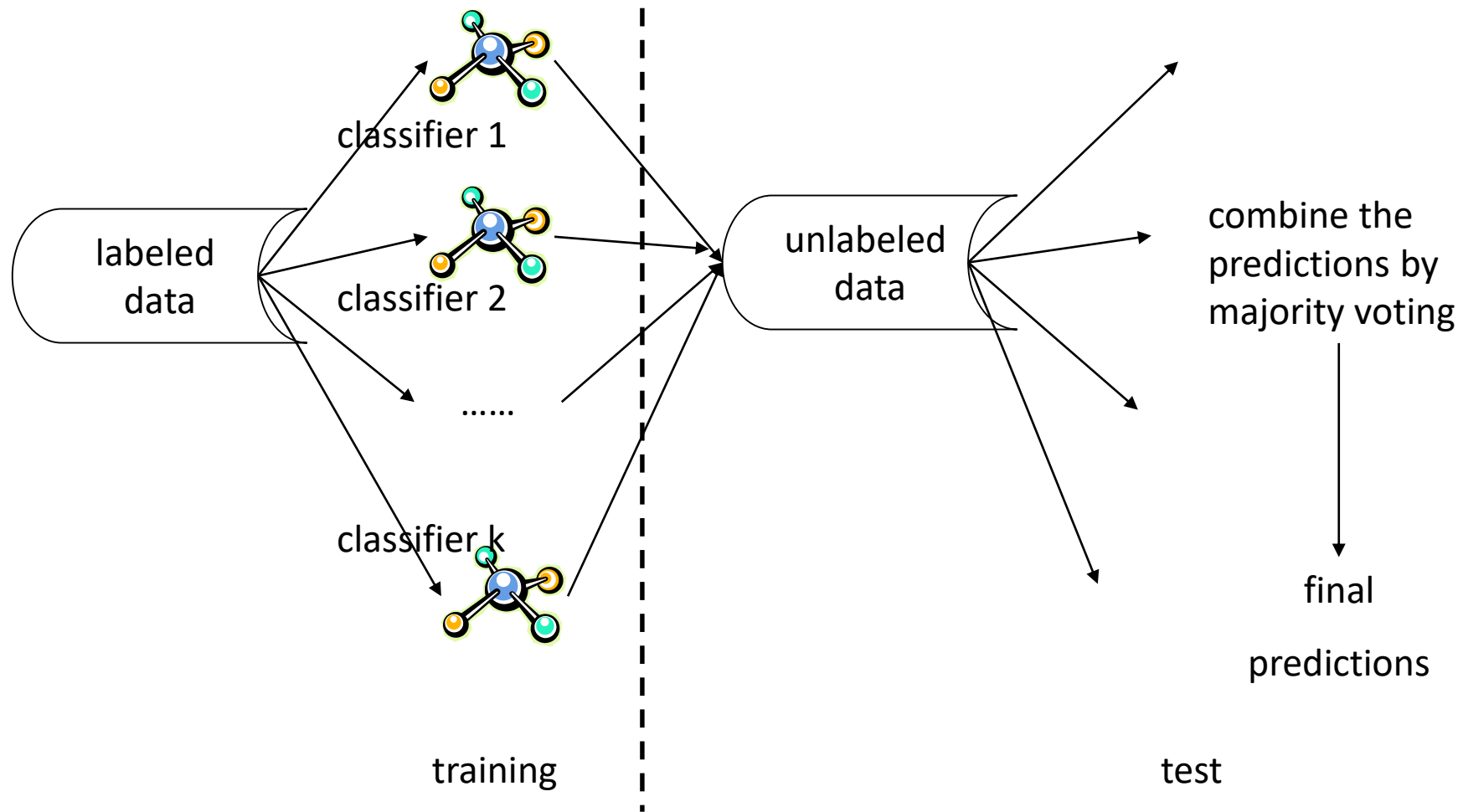
- **Bootstrap**
 - Sampling with replacement
 - Contains around 63.2% original records in each sample
- **Bootstrap Aggregation**
 - Train a classifier on each bootstrap sample
 - Combine the classifiers to obtain a stronger model

*[Breiman96]

Learn to Combine



Majority Voting



Pros and Cons

	Combine by learning	Combine by voting
Pros	<ul style="list-style-type: none">Get useful feedbacks from labeled dataCan potentially improve accuracy	<ul style="list-style-type: none">Do not need labeled dataCan improve the generalization performance
Cons	<ul style="list-style-type: none">Need to keep the labeled data to train the ensembleMay overfit the labeled dataCannot work when no labels are available	<ul style="list-style-type: none">No feedbacks from the labeled dataRequire the assumption that consensus is better

Reference

- 周志华, 机器学习, 清华大学出版社
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press
- Vapnik, The Nature of Statistical Learning Theory, Springer, 1999
- Yoav Freund and Robert E. Schapire, A Short Introduction to Boosting, 1999.

Thanks!

tyliu@microsoft.com

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