Introduction to Machine Learning

Ensemble Methods

林彥宇 教授
Yen-Yu Lin, Professor
國立陽明交通大學 資訊工程學系
Computer Science, National Yang Ming Chiao Tung University

Some slides are modified from T.-W. Yue, T.-L. Liu, and D. Sontag

Outline

- AdaBoost
- Decision tree
- Bagging
- Random forests

Outline

- AdaBoost
- Decision tree
- Bagging
- Random forests

AdaBoost

[Freund and Schapire, 1995]

- AdaBoost = Adaptive Boosting
- Boosting: A category of machine learning algorithms that build a strong classifier by combining a set of weak classifiers
  - Weak classifier: weak learner
  - Strong classifier: a linear combination of weak learners
- AdaBoost is a boosting algorithm
- It maintains a weight distribution on training data for iterative weak learner selection
- The resultant strong classifier performs classification based on the weighted vote of the selected weak learners

AdaBoost: Formulation

$$\left.\begin{array}{c} h_1(x) \in \{-1,+1\} \\ h_2(x) \in \{-1,+1\} \\ \vdots \\ h_T(x) \in \{-1,+1\} \end{array}\right\} \quad H_T(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

weak learners

strong classifier

- Weak learners and strong classifier perform binary classification
- Each weak learner performs better than random guess
- The strong classifier is a weighted combination of weak learners

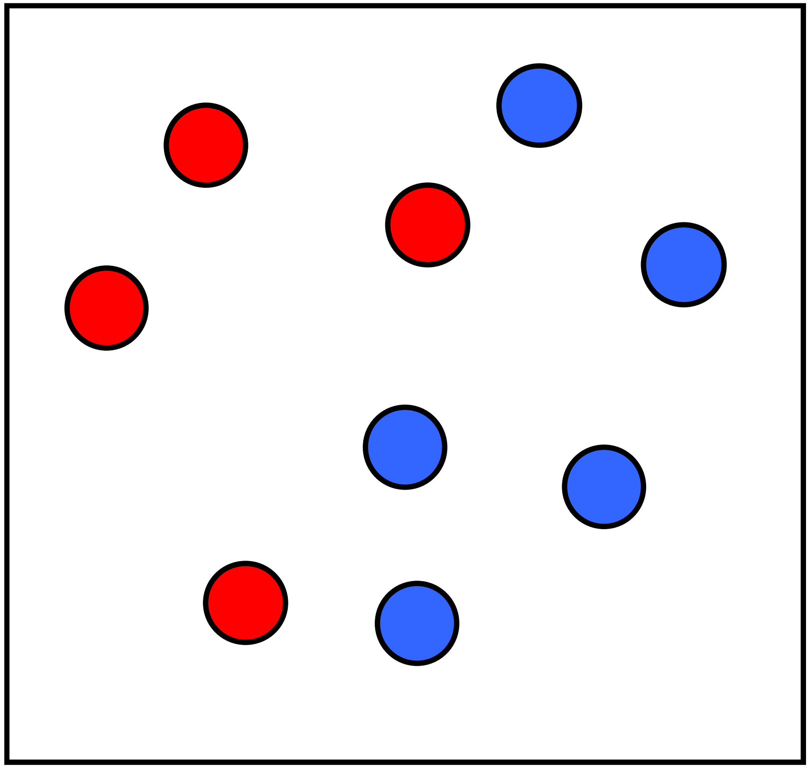
Weak learners

- Each weak learner learns by considering one simple feature
- T most beneficial features for classification should be selected

- How to
  - define features?
  - select beneficial features?
  - train weak learners?
  - manage the weights of training samples?
  - associate a weight to each weak learner?

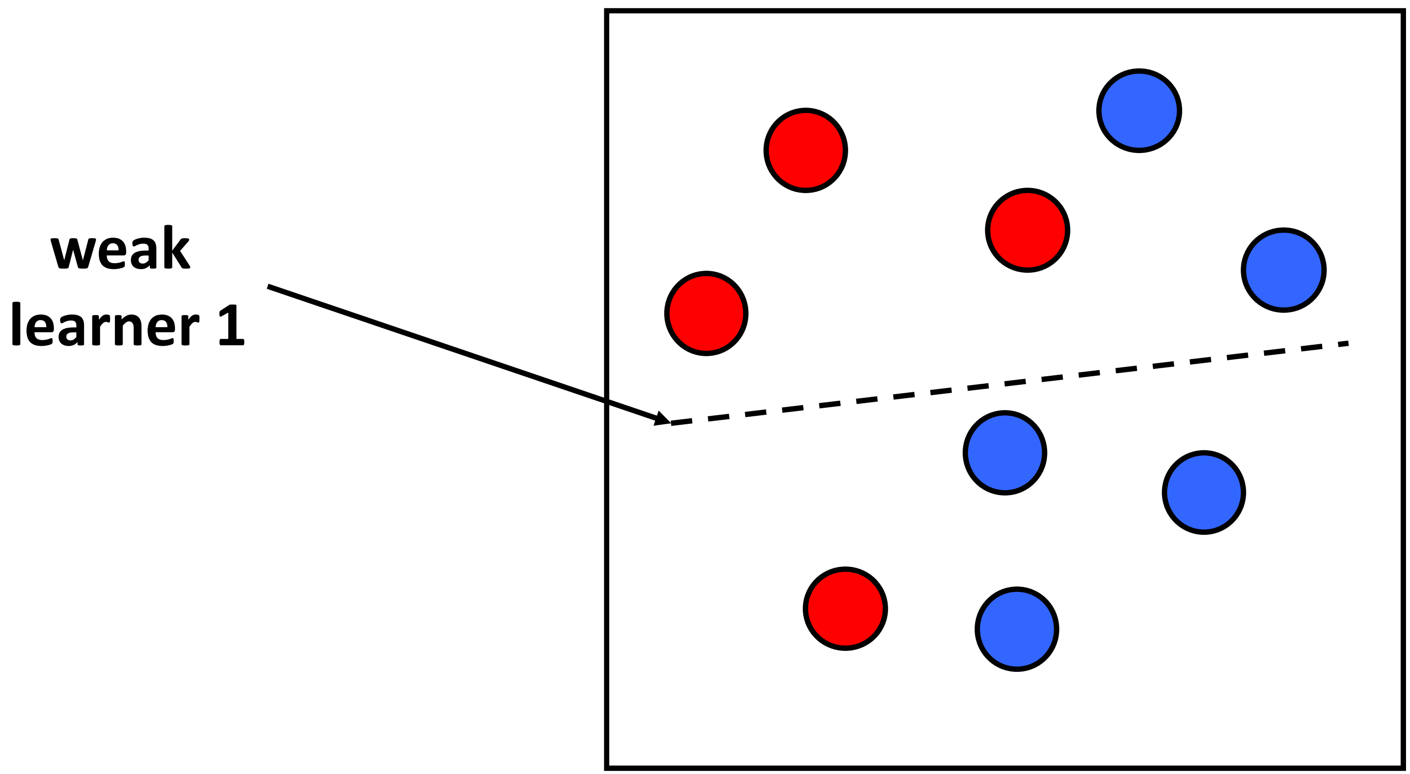
AdaBoost: An illustration

- At the first iteration, we initialize the weight distribution of training data as a uniform distribution



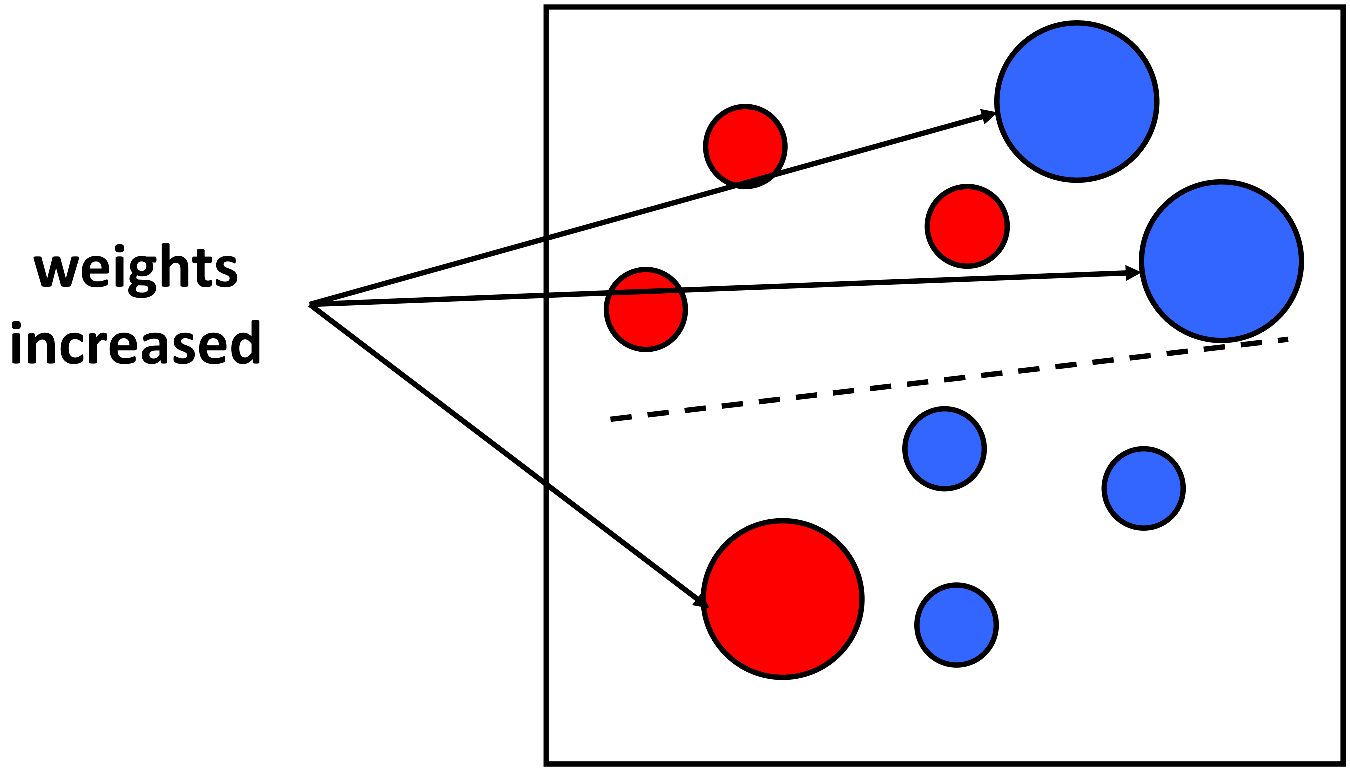
AdaBoost: An illustration

- Select the first weak learner that has the minimal weighted error



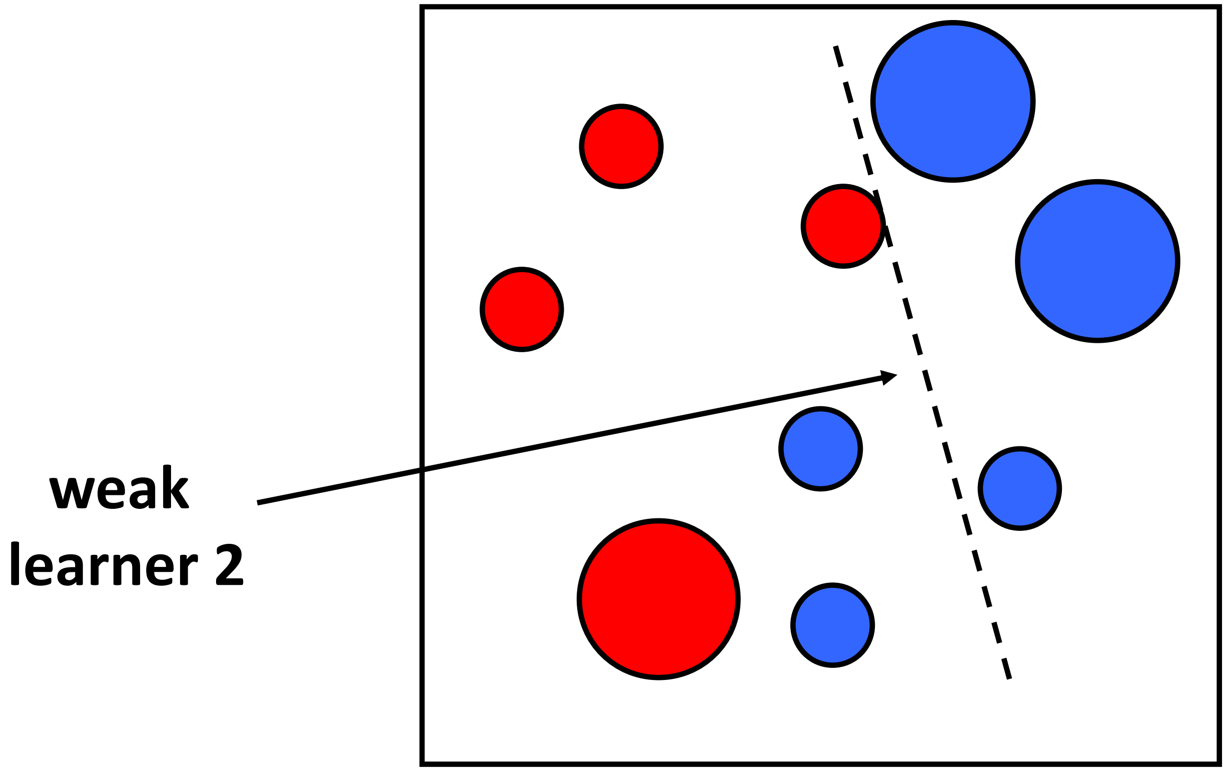
AdaBoost: An illustration

- Increase the weights of training data that are wrongly classified



AdaBoost: An illustration

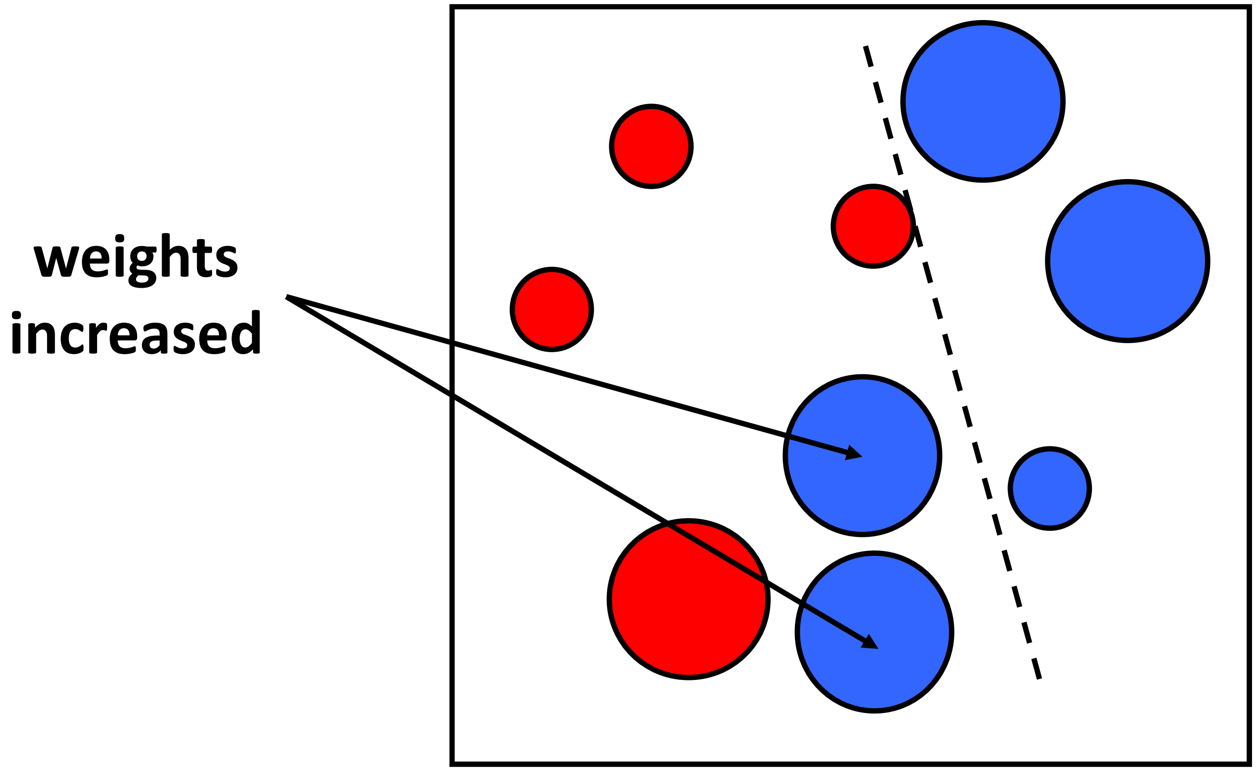
- Select the second weak learner that has the minimal weighted error



National Chiao Tung University

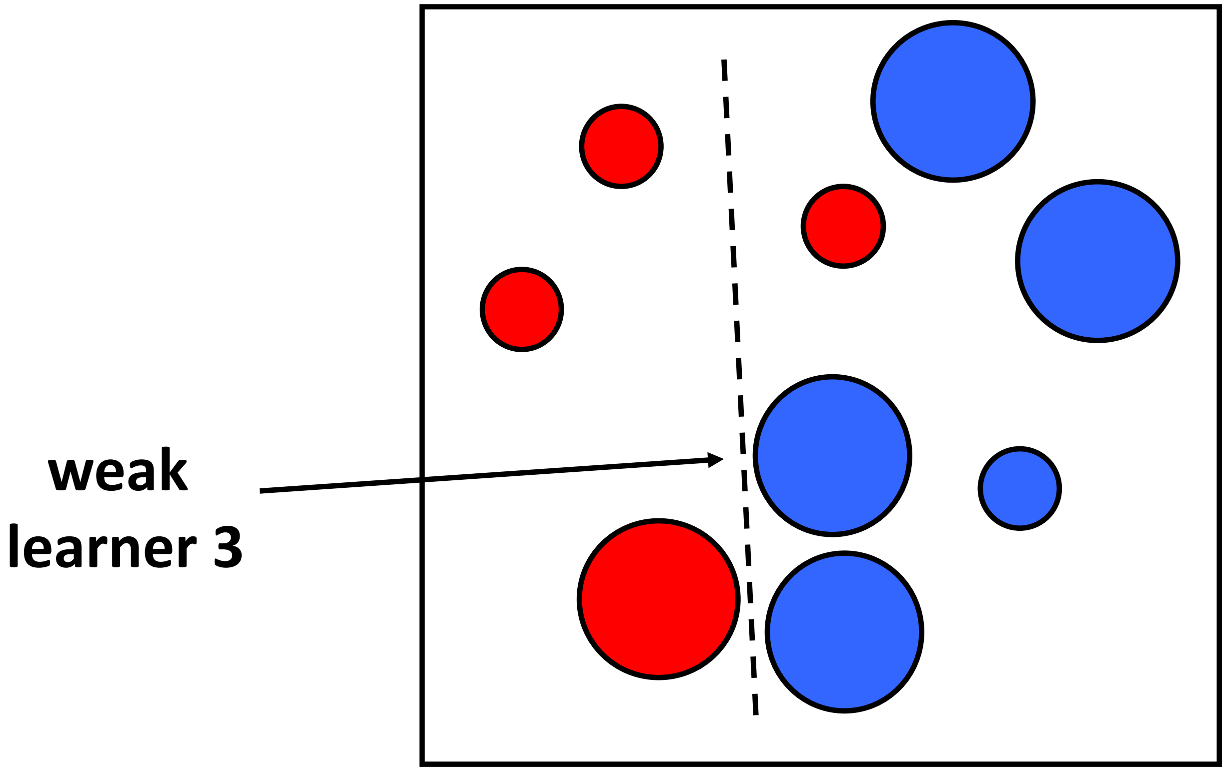
AdaBoost: An illustration

- Increase the weights of training data that are wrongly classified by the second weak learner



AdaBoost: An illustration

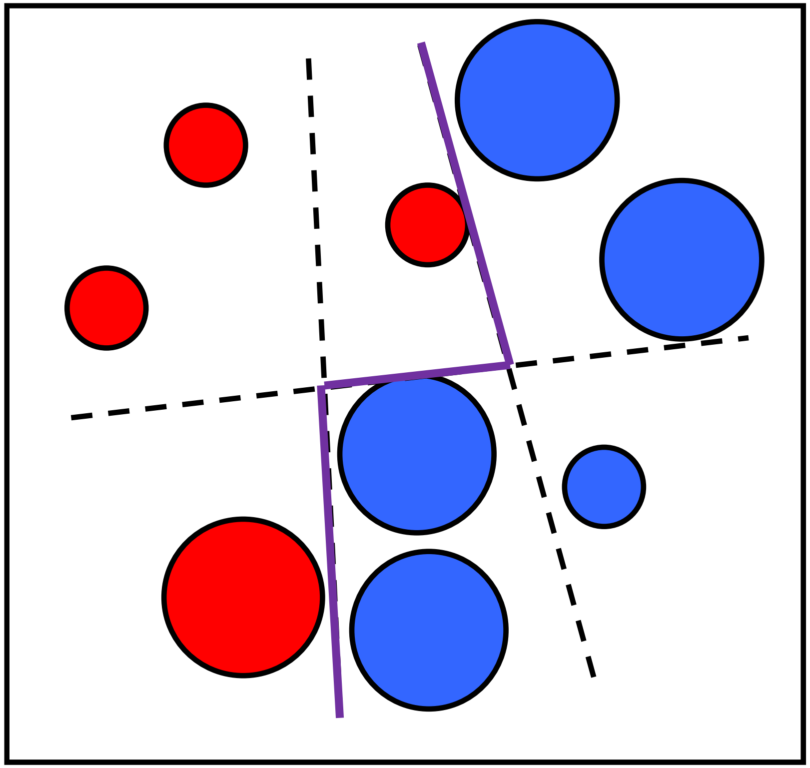
- Select the third weak learner that has the minimal weighted error



AdaBoost: An illustration

- The resultant strong classifier consisting of the three selected weak learners

strong classifier



国立交通大學
National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t,i.e.,

  $$h_t = \arg\min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

國立交通大學
National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

uniform weight distribution

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \rightarrow \{-1, +1\}$ which minimizes error wrt D_t ,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

國立交通大學
National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

uniform weight distribution

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t ,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \text{ where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

minimize weighted error

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

國立交通大學
National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

uniform weight distribution

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t ,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

minimize weighted error

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

minimize exponential loss

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \ Z_t \text{ is for normalization}$$

國立交通大學
National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

uniform weight distribution

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t ,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

minimize weighted error

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

minimize exponential loss

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

give wrongly classified data more chance

國立交通大學 National Chiao Tung University

The AdaBoost algorithm

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1 - \varepsilon_t}{\varepsilon_t}$

- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i) \exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

Output final classifier: $sign\left(H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \right)$

Goal of AdaBoost

- The decision made by the learned strong classifier on sample *x*

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

- The margin of the strong classifier on a sample (*x*, *y*): $yH(x)$
- Margin points out
  - Whether the sample is correctly classified, i.e., $yH(x) >= 0$
  - The classification confidence
- Given a set of training data, AdaBoost aims to maximize the margins of all training data via minimizing the exponential loss

$$loss_{\exp}\left[H(x)\right] = E_{x,y}\left[e^{-yH(x)}\right]$$

Goal of AdaBoost at timestamp t

- Final classifier: $sign\left(H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \right)$
- Exponential loss: $loss_{\exp}\left[H(x)\right] = E_{x,y}\left[e^{-yH(x)}\right]$
- According to the definition, we have $H_t(x) = H_{t-1}(x) + \alpha_t h_t(x)$

$$E_{x,y}\left[e^{-yH_t(x)}\right] = E_x\left[E_y\left[e^{-yH_t(x)} \mid x\right]\right]$$

$$= E_x\left[E_y\left[e^{-y[H_{t-1}(x)+\alpha_t h_t(x)]} \mid x\right]\right]$$

$$= E_x\left[E_y\left[e^{-yH_{t-1}(x)} e^{-y\alpha_t h_t(x)} \mid x\right]\right]$$

$$= E_x\left[e^{-yH_{t-1}(x)}\left[e^{-\alpha_t} P(y = h_t(x)) + e^{\alpha_t} P(y \neq h_t(x))\right]\right]$$

Goal of AdaBoost at timestamp t: $h_t = ?$

- The derivation of the exponential loss at timestamp *t*:

$$E_{x,y}\left[e^{-yH_t(x)}\right] = E_x\left[e^{-yH_{t-1}(x)}\left[e^{-\alpha_t}P(y = h_t(x)) + e^{\alpha_t}P(y \neq h_t(x))\right]\right]$$

- Note that
  - $e^{-yH_{t-1}(x)}$ is proportional to the weight of sample *x*
  - α_t is larger than 0: $e^{-\alpha_t}$ lies in [0, 1], while e^{α_t} is larger than 1.

- At timestamp *t*, we select the weak learner h_t that is with the minimal weighted error.

Goal of AdaBoost at timestamp t: $\alpha_t = ?$

- The derivation of the exponential loss at timestamp *t*:

$$E_{x,y}\left[e^{-yH_t(x)}\right] = E_x\left[e^{-yH_{t-1}(x)}\left[e^{-\alpha_t}P(y=h_t(x)) + e^{\alpha_t}P(y \neq h_t(x))\right]\right]$$

- Once weak learner h_t has been selected, its weight α_t is determined by minimizing the exponential loss.
- Set the partial derivative of the exponential loss of α_t to 0, i.e.,

$$\frac{\partial}{\partial \alpha_t} E_{x,y}\left[e^{-yH_t(x)}\right] = 0$$

- We have $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$

國立交通大學
National Chiao Tung University

Goal of AdaBoost at timestamp t: $D_{t+1} = ?$

- Update the data weight distribution for next timestamp, t+1
- The weight of sample *x* at *t* is proportional to $e^{-yH_{t-1}(x)}$
- The weight of sample *x* at *t+1* proportional to $e^{-yH_t(x)}$
- This is equivalent to

$$D_{t+1}(i) = \frac{D_t(i)\exp[-\alpha_t y_i h_t(x_i)]}{Z_t},\ Z_t \text{ is for normalization}$$

The AdaBoost algorithm revisited

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$

Initialization: $D_1(i) = \frac{1}{m}, i = 1, \ldots, m$

For $t = 1, \ldots, T$:

- Find classifier $h_t : X \to \{-1, +1\}$ which minimizes error wrt D_t ,i.e.,

$$h_t = \arg\min_{h_j} \varepsilon_j \quad \text{where } \varepsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$$

- Weight classifier: $\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$

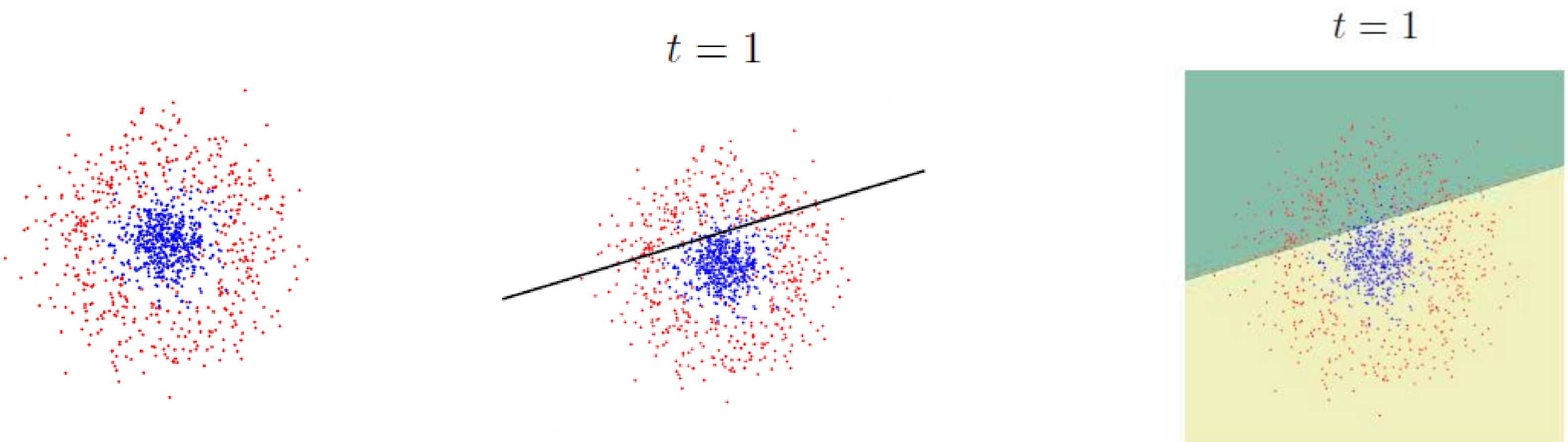
- Update distribution:

$$D_{t+1}(i) = \frac{D_t(i)\exp[-\alpha_t y_i h_t(x_i)]}{Z_t}, \; Z_t \text{ is for normalization}$$

Output final classifier: $sign\left(H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \right)$

An example

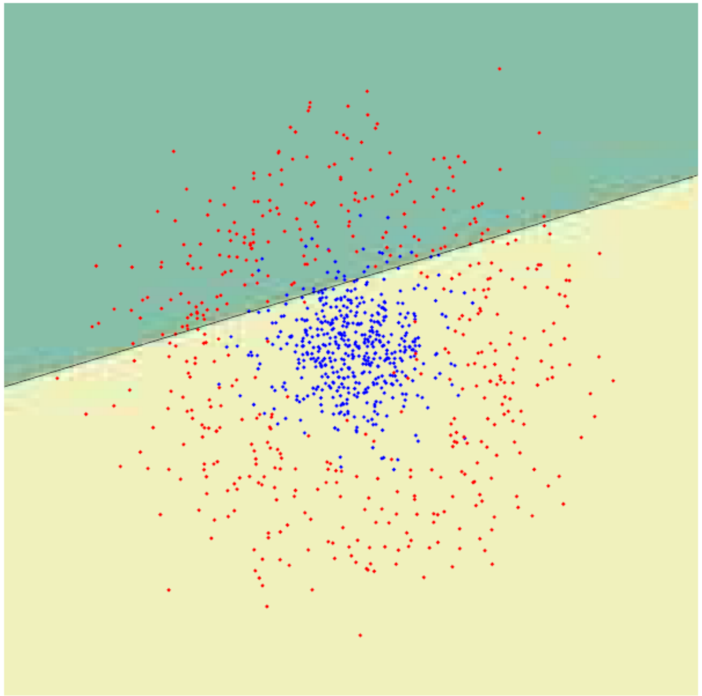
- AdaBoost on a two-class dataset



An example

- The decision boundary

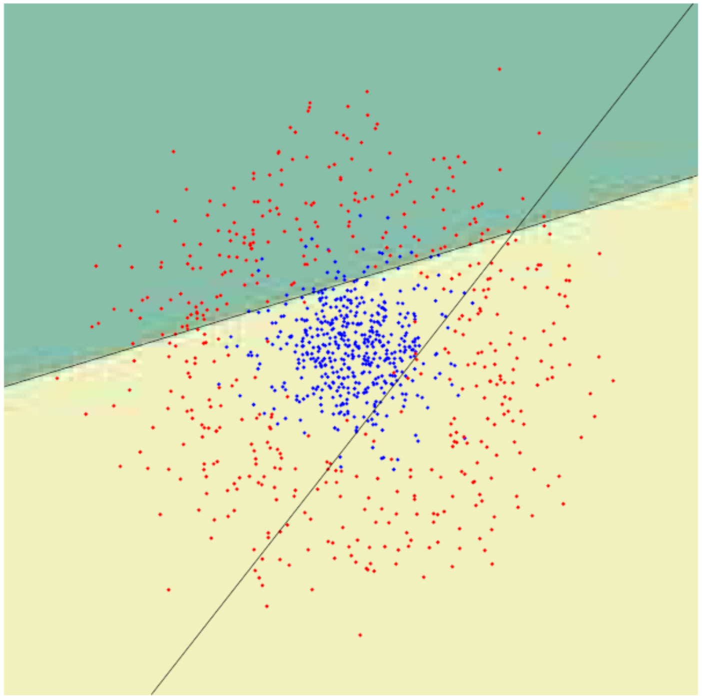
$$t = 1$$



An example

- The decision boundary

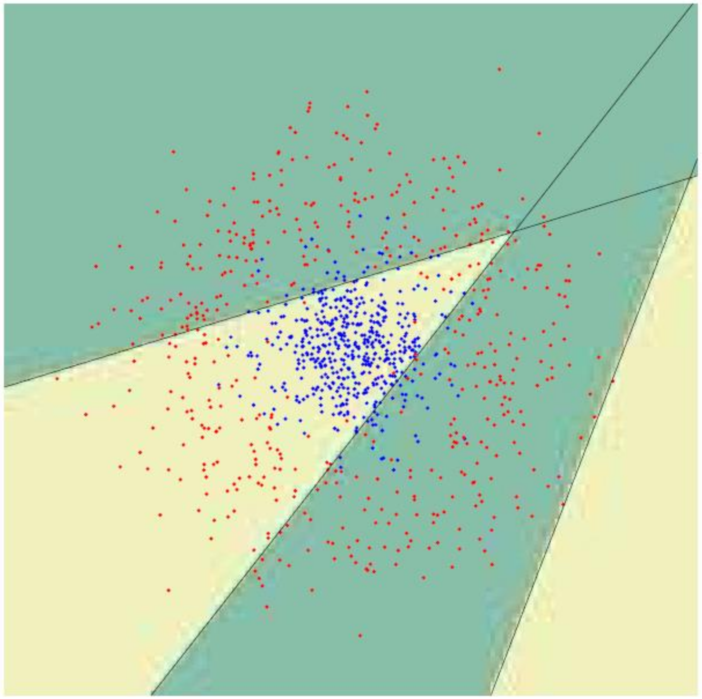
$$t = 2$$



An example

- The decision boundary

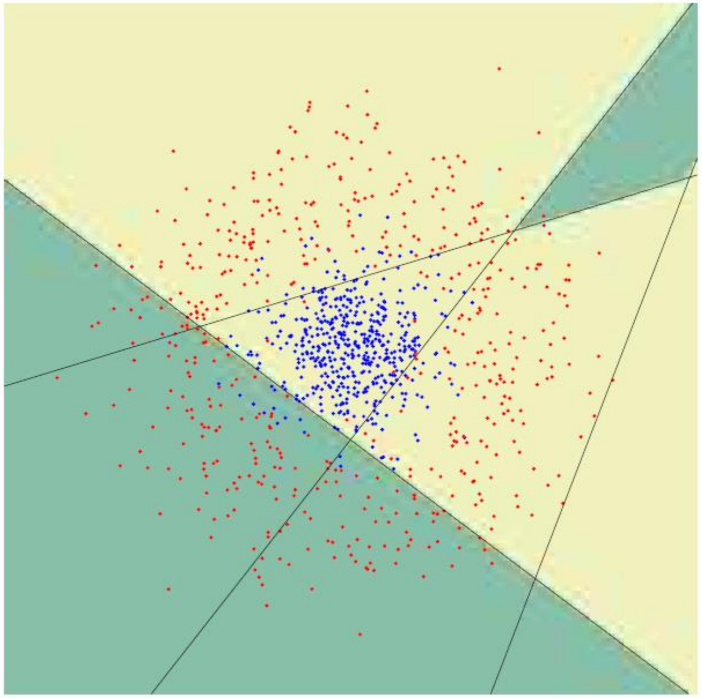
$$t = 3$$



An example

- The decision boundary

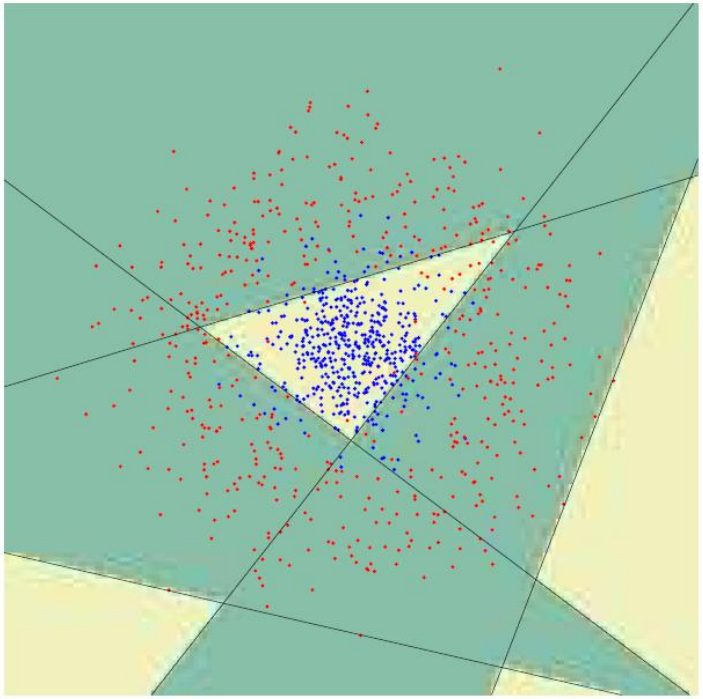
$$t = 4$$



An example

- The decision boundary

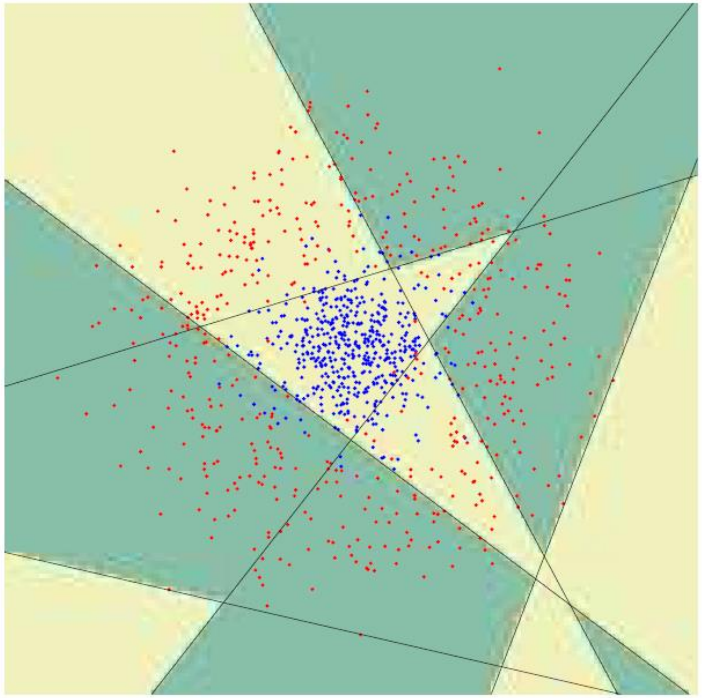
$$t = 5$$



An example

- The decision boundary

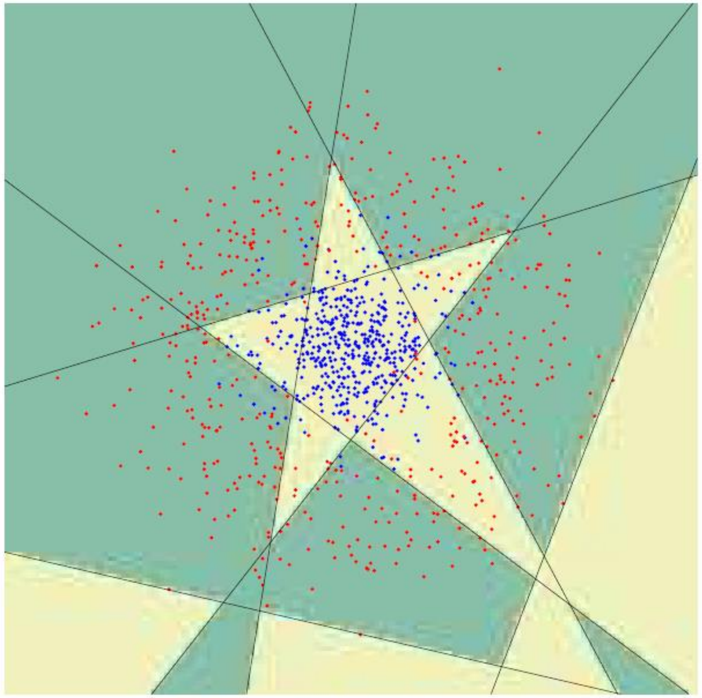
$$t = 6$$



An example

- The decision boundary

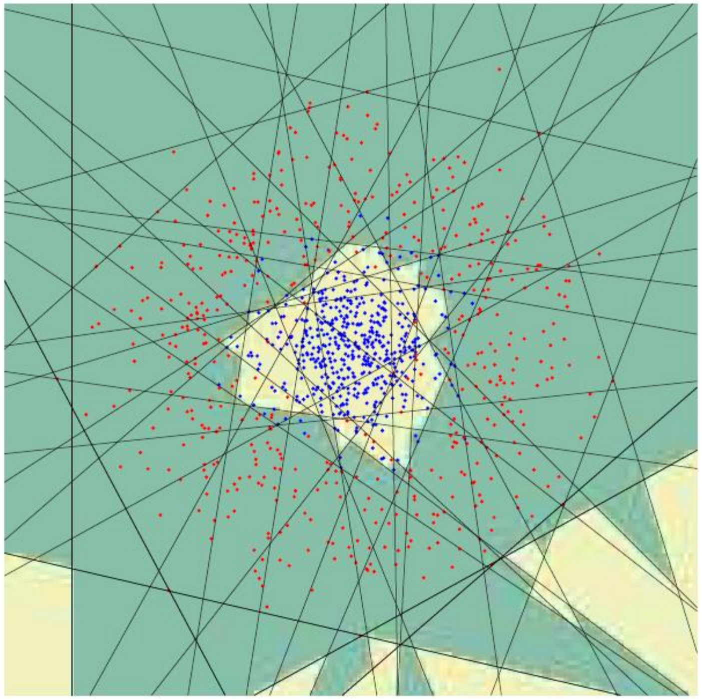
$$t = 7$$



An example

- The decision boundary

$$t = 40$$



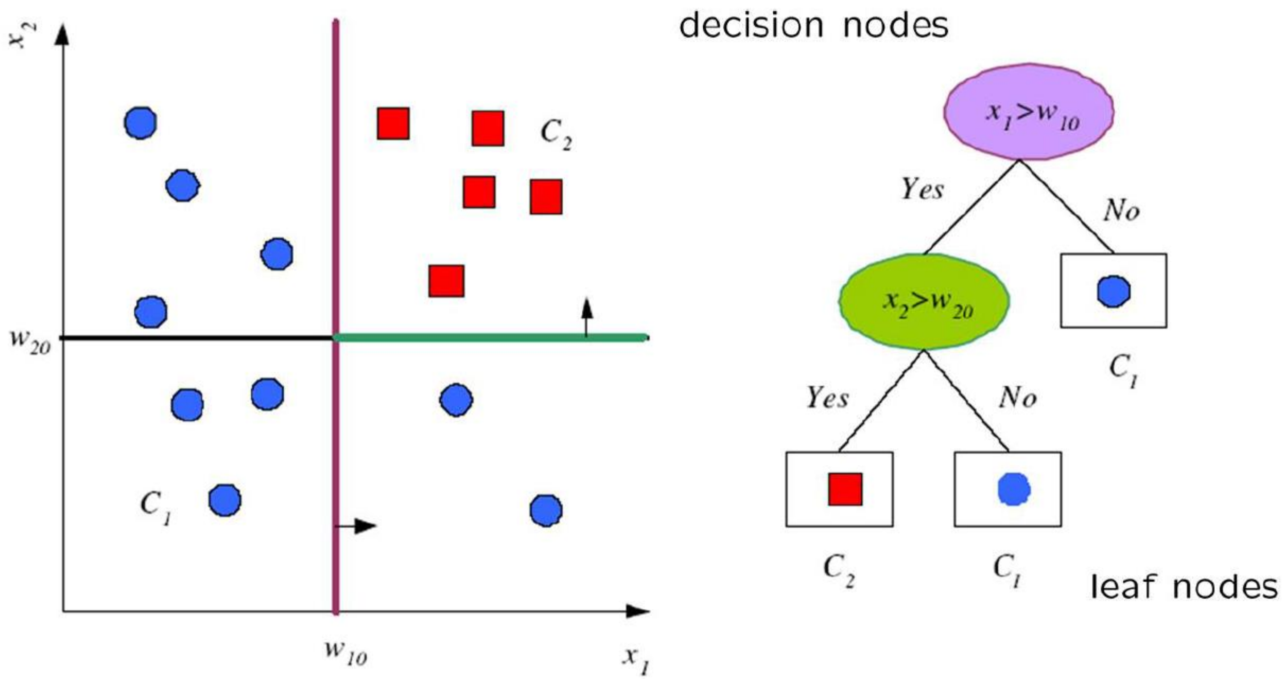
Outline

- AdaBoost
- Decision tree
- Bagging
- Random forests

Decision tree

- A decision tree is a hierarchical model for supervised learning
  - It implements a divide-and-conquer strategy for classification and regression
- A decision tree has internal decision nodes and leaf nodes
  - Each decision node m implements a decision function $f_m(\mathbf{x})$ with discrete outcomes labeling the branches
  - Each leaf node is associated with a constant (regression) or a label (classification)
  - Given an input, a test is applied at a decision node at root, and then one of the branches is taken depending on the outcome
  - This process starts at the root and is repeated recursively until a leaf node is reached

An example: A decision tree for classification



- A decision tree with two decision nodes, which partition the input space into three cuboid regions

Decision tree induction

- Tree induction is the construction of a decision tree given a training dataset
  - Training data $\{\mathbf{x}_n\}_{n=1}^N$ and the target labels $\{t_n\}_{n=1}^N$
- There exist many trees that code the dataset with no error
  - We are interested in finding the smallest one among them, where the tree size is measured as the number of nodes in the tree and the complexity of the decision nodes.
- However, finding the smallest tree is NP-complete
- In practice, greedy algorithms based on local search are used to yield reasonable trees in reasonable time

Decision tree induction

- Starting at the root with the complete training data, we look for the best split that divides the training data into two or multiple parts (branches/subsets)
- The splitting is applied recursively with the corresponding subset of training data until a certain criterion is met, at which point a leaf node is created and labeled
  - This data subset is pure
  - Maximum tree depth is reached
  - The minimum number of data per leaf
  - The information gain is less than a threshold
  - …

Decision (internal) node

- Decision (internal) nodes
  - Univariate: Uses a single attribute $\mathbf{x}_i$
    - Numeric $\mathbf{x}_i$: Binary split with $\mathbf{x}_i > \mathrm{w}_m$
    - Discrete $\mathbf{x}_i$: n-way split for n possible categorical values
  - Multivariate: Uses all attributes $\mathbf{x}$

Measure of Impurity

- The goodness of a split in a classification tree is quantified by an impurity (or uncertainty) measure
- A split is pure if after the split, for all branches, all the instances choosing a branch belong to the same class
- Suppose N_m data instances reaching node m, and N_m^k of them belong to class C_k
- Then the estimate for the probability of class C_k is given by

$$p_m^k = \frac{N_m^k}{N_m}$$

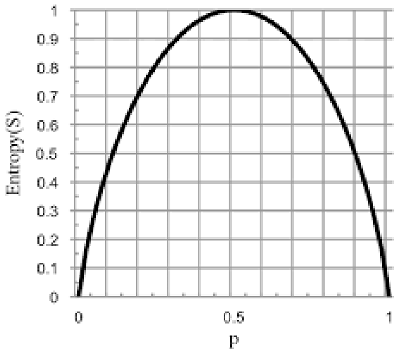
- Node m is pure if all data points it has belong to the same class. In this case, node m is a leaf with that class label

Measure of Impurity

- For node m and its data points, we can use the entropy function to measure the impurity/uncertainty

$$I_m = -\sum_{k=1}^{K} p_m^k \log p_m^k$$

  - $0\log 0 \equiv 0$
  - Larger entropy means that data are more uniformly distributed over classes
  - For 2-class classification, entropy $I(p_m^1, p_m^2)$ is a nonnegative function measuring the impurity of a node m
    - $I(0.5,0.5)$ has the maximum value
    - $I(1,0) = I(0,1) = 0$: minimum value
    - $I(p, 1-p)$ is increasing when $0 \leq p \leq 0.5$ and is decreasing when $0.5 \leq p \leq 1$



High or low entropy

- High entropy
  - Data are from a uniform like distribution
  - Flat histogram
  - Values sampled from it are less predictable

- Low entropy
  - Data are from a varied (peaks and valleys) distribution
  - Histogram has many lows and highs
  - Values sampled from it are more predictable

Best split

- If node m is pure, generate a leaf and stop, otherwise split and continue recursively
- Impurity after split: N_{mj} of N_m data take branch j, and N_{mj}^k belong to class C_k
  - The estimate for the probability of class C_k for branch j

$$p_{mj}^k = \frac{N_{mj}^k}{N_{mj}}$$

  - The entropy after split

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{k=1}^{K} p_{mj}^k \log p_{mj}^k = \sum_{j=1}^{n} \frac{N_{mj}}{N_m} I_{mj}$$

Best split

- Entropy before split

$$I_m = -\sum_{k=1}^{K} p_m^k \log p_m^k$$

- Entropy after split

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{k=1}^{K} p_{mj}^k \log p_{mj}^k = \sum_{j=1}^{n} \frac{N_{mj}}{N_m} I_{mj}$$

- Pick the split that maximizes the information gain

$$I_m - I'_m$$

- Pick the split (variable + threshold) that minimizes the impurity

How many splits

- For a binary tree, a decision function splits on attribute $\mathbf{x}_i$
  - Branch 1: $\mathbf{x}_i > \mathrm{w}_m$
  - Branch 2: $\mathbf{x}_i \leq \mathrm{w}_m$
- A split (decision function) is composed of a feature (input variable) and a threshold
  - How many features?
    - The data dimension M
  - For each feature, how many thresholds are there?
    - Infinite?
    - Only a few threshold values are representative
    - Sort N data according to their values on this feature
    - Try $N-1$ threshold values, where the *i*-th threshold is the average of the *i*-th and *(i+1)*-th sorted values

Decision tree algorithm

- **GenerateTree**($\mathcal{X}$)

  1. `if` **NodeEntropy** $(\mathcal{X} < \theta_I)$
  2. `  Create leaf labeled by majority class in` $\mathcal{X}$
  3. `  return`
  4. $i \longleftarrow$ **SplitAttribute**($\mathcal{X}$)
  5. `for each branch of` x_i
  6. `  Find` $\mathcal{X}_i$ `falling in branch`
  7. **GenerateTree** $(\mathcal{X}_i)$

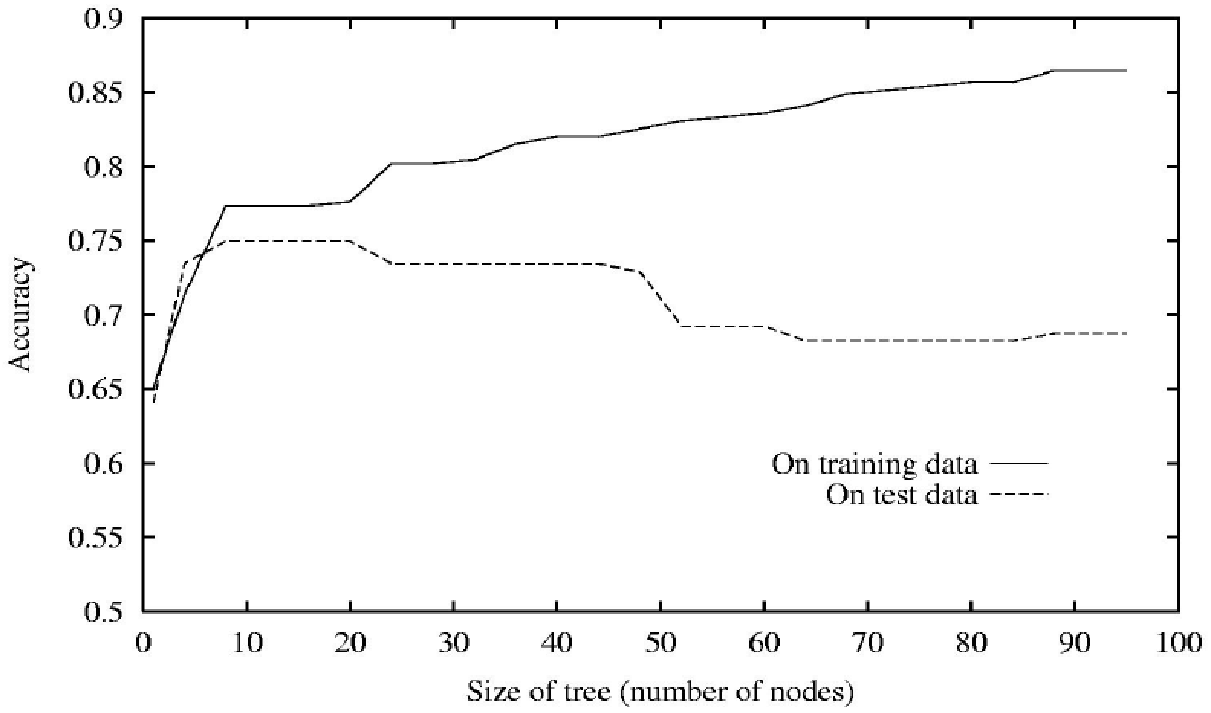
Decision tree algorithm

- **SplitAttribute**$(\mathcal{X})$

MinEnt $\longleftarrow$ MAX
for all attributes $i = 1, \ldots, d$
  if x_i is discrete with n values
    Split $\mathcal{X}$ into $\mathcal{X}_1, \ldots, \mathcal{X}_n$ by x_i
    $e \longleftarrow$ **SplitEntropy**$(\mathcal{X}_1, \ldots, \mathcal{X}_n)$
    if $e <$ MinEnt MinEnt $\longleftarrow e$; bestf $\longleftarrow i$
  else
    for all possible splits
      split $\mathcal{X}$ into $\mathcal{X}_1$, $\mathcal{X}_2$ on x_i
      $e \longleftarrow$ **SplitEntropy**$(\mathcal{X}_1, \mathcal{X}_2)$
      if $e <$ MinEnt MinEnt $\longleftarrow e$; bestf $\longleftarrow i$
return bestf

Decision trees tend to overfit

- When the size of a decision tree increases, the risk of overfitting also increases



National Chiao Tung University

Some practical strategies

- Some strategies for picking simpler trees
  - Fixed depth
  - Minimum number of data points per leaf
  - Minimum information gain
  - Tree pruning
  - Combination of them

- Bagging and random forests are two methods to
  - Improve the performance of a single decision tree
  - Make decision trees more robust to noisy data
  - Reduce the risk of overfitting of decision trees

Summary

- Decision trees are one of the most popular ML and PR tools
  - Easy to understand, implement, and use
  - Computationally cheap
    - Training: greedy strategy for training
    - Testing: a serial of decision functions, one with a single data feature
- Information gain (or Gini index) is used to select useful attributes/features
- Not only for classification, decision tress can be used for regression and density estimation

Outline

- AdaBoost
- Decision tree
- Bagging
- Random forests

Bagging: Bootstrap aggregation

- It was proposed by Leo Breiman in 1994
- Bagging is a method for generating multiple versions of a predictor and using them to get an aggregated predictor
- The aggregation averages the versions for regression and does a majority vote for classification
- It takes repeated bootstrap samples from a given training set D
- Bootstrap sampling: Given set D containing N training examples, create another training set D' by drawing N examples at random with replacement from D

Bagging

- Examples of bootstrap sampling with replacement

Training Data

Data ID

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

- Bagging algorithm
  - Create T bootstrap sample datasets, $D_1, D_2, \dots, D_T$
  - Train a distinct classifier on each dataset D_t
  - Classify a new instance by majority vote or average

Bagging

- Examples of bootstrap sampling with replacement

Training Data

Data ID

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

- Bagging algorithm
  - Create T bootstrap sample datasets, $D_1, D_2, \ldots, D_T$
  - Train a distinct classifier on each dataset D_t
  - Classify a new instance by majority vote or average

Bagging

- Examples of bootstrap sampling with replacement

Training Data

Data ID

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

- Bagging algorithm
  - Create T bootstrap sample datasets, D_1, D_2, …, D_T
  - Train a distinct classifier on each dataset D_t
  - Classify a new instance by majority vote or average

Bagging

- Examples of bootstrap sampling with replacement

Training Data

Data ID

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 algorithm
  - Create T bootstrap sample datasets, $D_1, D_2, \dots, D_T$
  - Train a distinct classifier on each dataset D_t
  - Classify a new instance by majority vote or average

Bagging

- Examples of bootstrap sampling with replacement

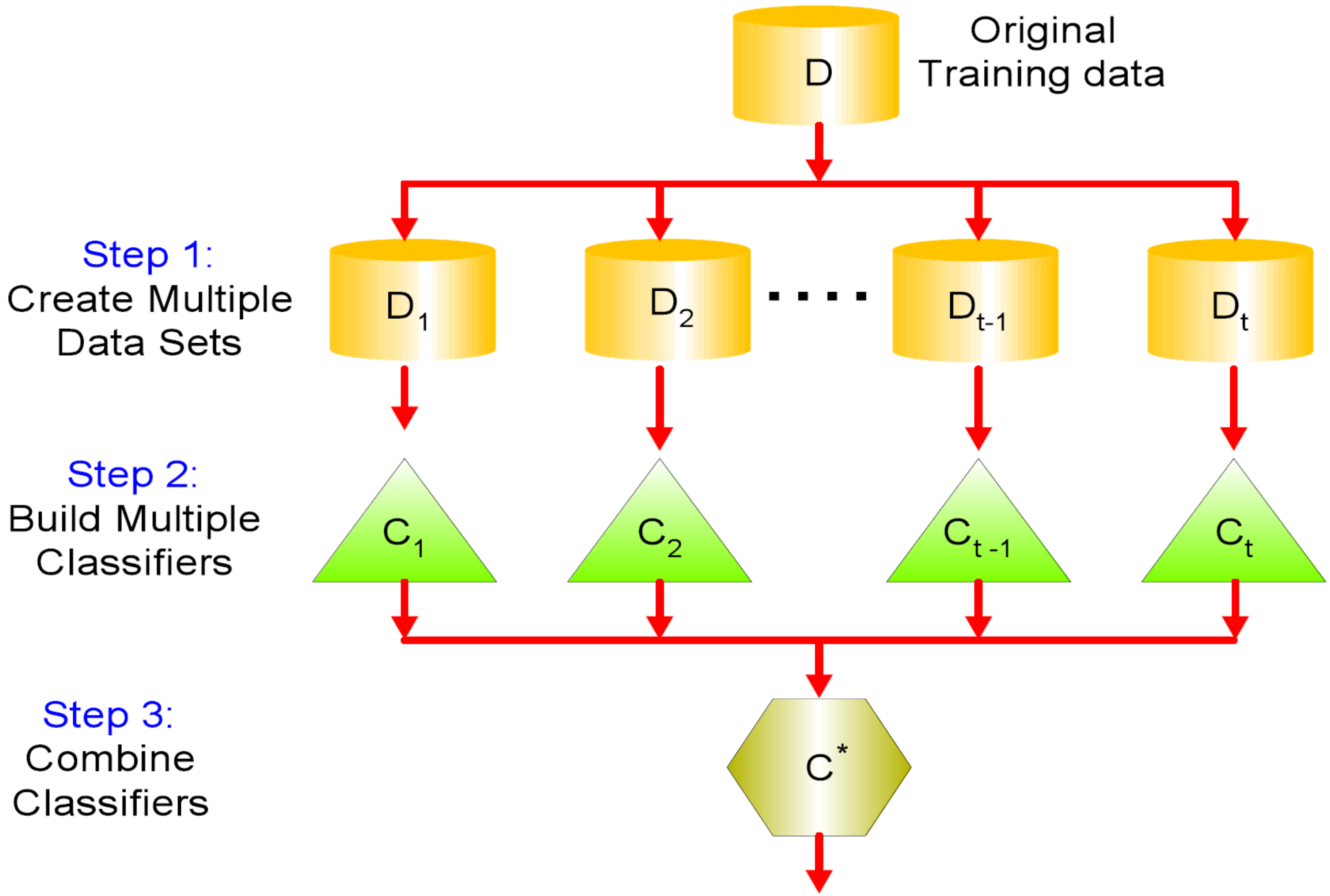
Data ID

Training Data

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 algorithm
  - Create T bootstrap sample datasets, $D_1, D_2, \ldots, D_T$
  - Train a distinct classifier on each dataset D_t
  - Classify a new instance by majority vote or average

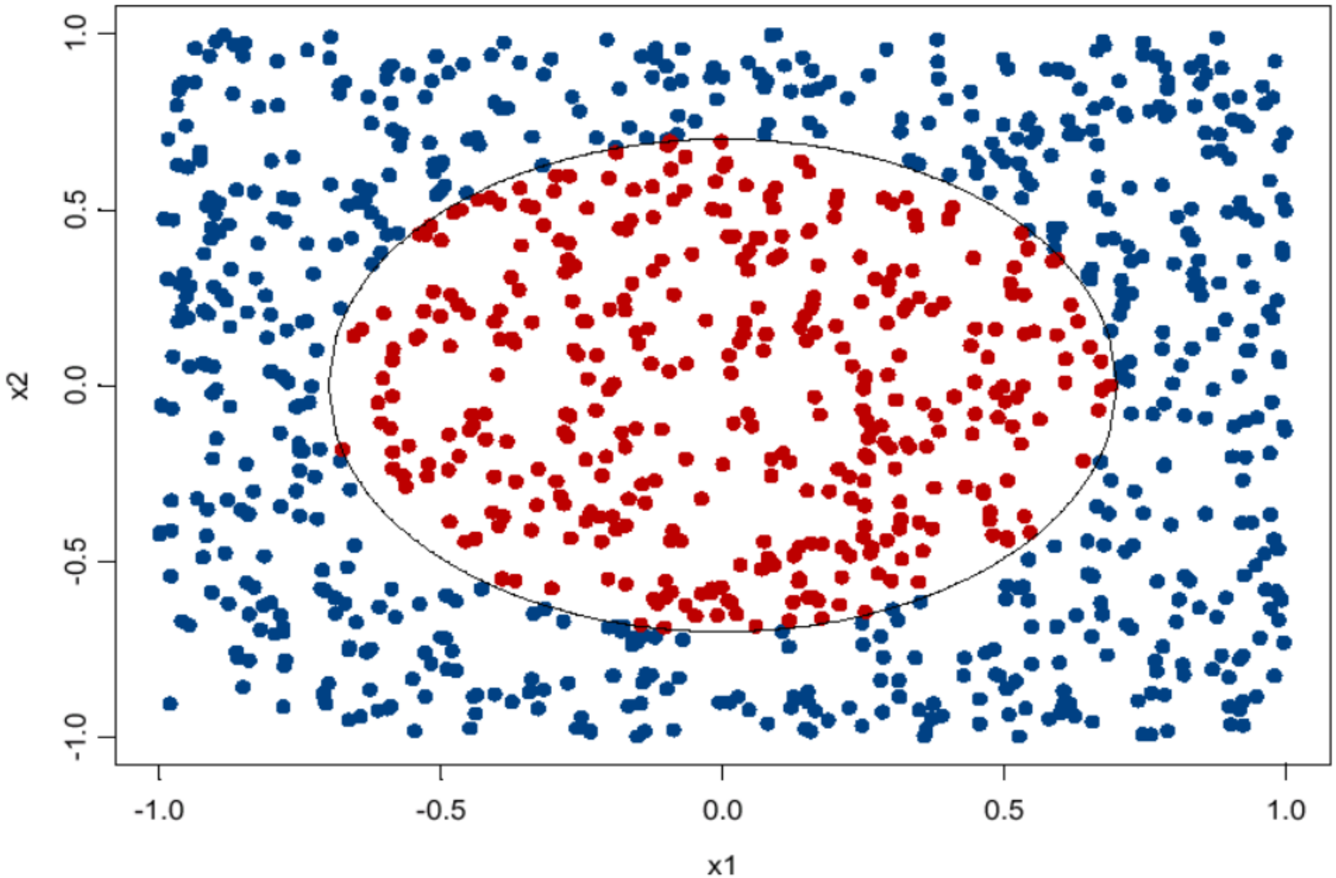
Bagging



國立交通大學
National Chiao Tung University

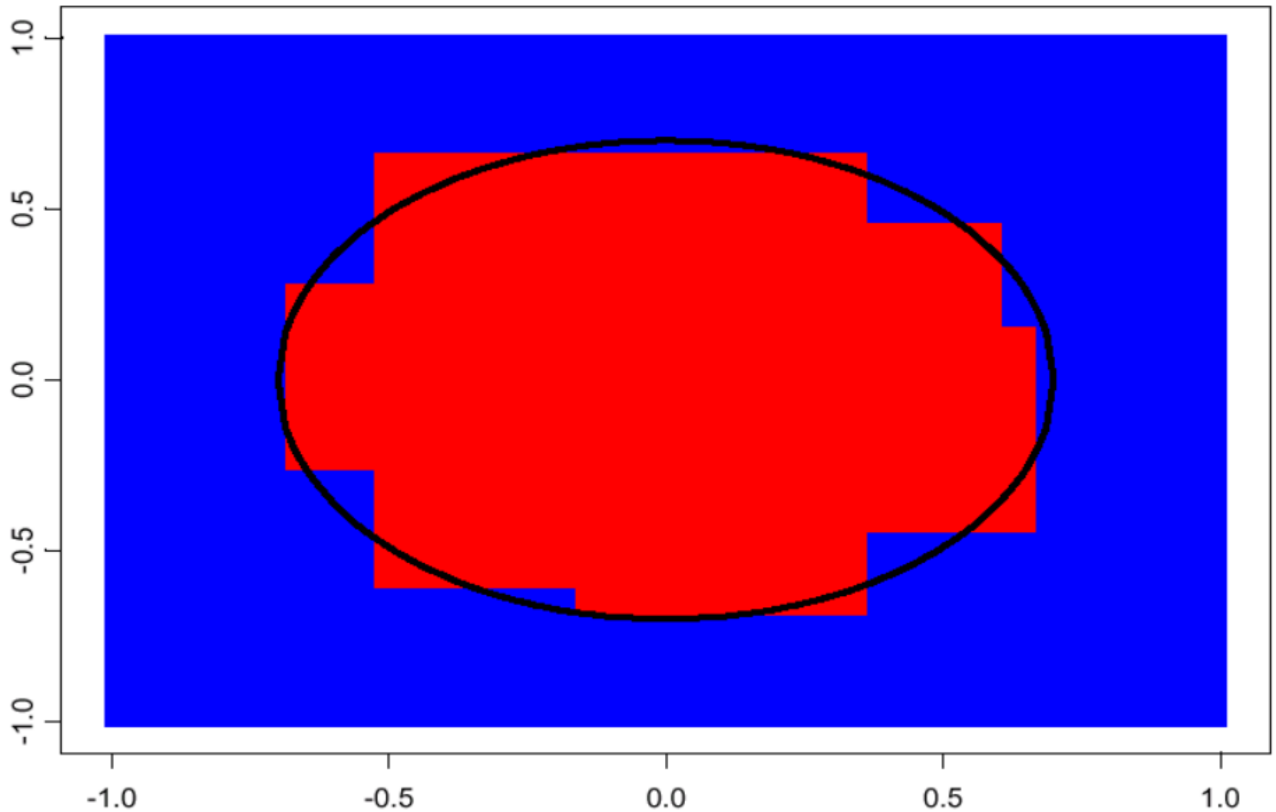
A two-class classification example

- Training data for binary-class classification



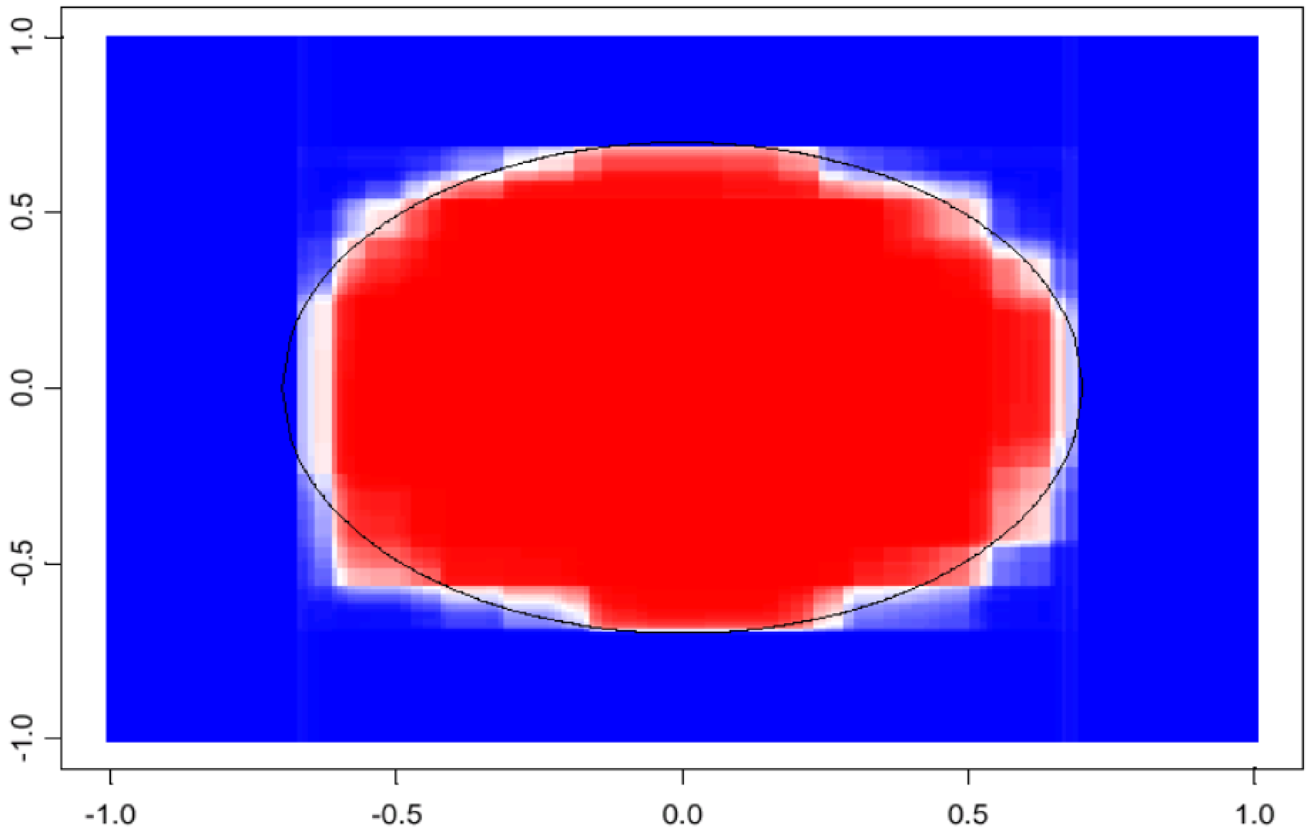
A two-class classification example

- CART: classification and regression tree, a decision tree
- Decision boundary derived by a CART



A two-class classification example

- 100 bagged CARTs



Experimental results

- Seven datasets for evaluation

Data Set	**# Samples**	**# Variables**	**# Classes**
waveform	300	21	3
heart	1395	16	2
breast cancer	699	9	2
ionosphere	351	34	2
diabetes	768	8	2
glass	214	9	6
soybean	683	35	19

Experimental results

- CART vs. 50 bagged CARTs

Misclassification Rates (%)

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

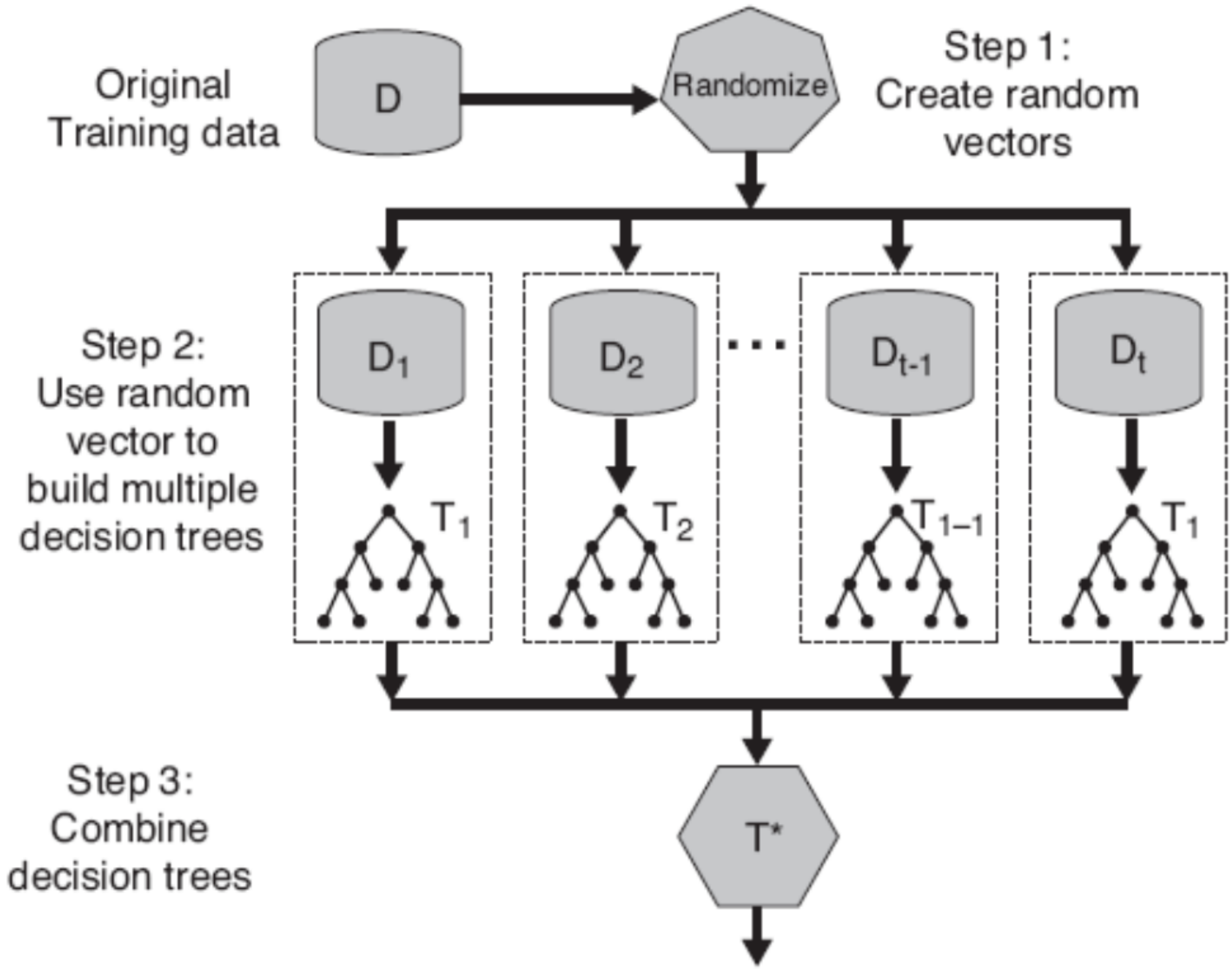
Outline

- AdaBoost
- Decision tree
- Bagging
- Random forests

Random forests

- Random forests were proposed by Leo Breiman in 2001
- An ensemble method specifically designed for decision tree classifiers
- It introduces two sources of randomness: Bagging and random input vectors
  - Bagging: each tree is grown using a bootstrapped dataset
  - Random vector: At each decision node, best split is chosen from a random sample of m attributes/features instead of all attributes

Random forests



Three methods for constructing a decision node

- M: data dimension, $m < M$: a constant, $L \leq M$: a constant
- Method 1:
  - Choose m attributes randomly, compute their information gains, and choose the attribute with the largest gain to split
- Method 2:
  - When M is small, randomly select L of the attributes. Compute a linear combination of the L attributes using weights generated from [-1,+1] randomly
- Method 3:
  - Compute the information gain of all M attributes. Select the top m attributes by information gain. Randomly select one of the m attributes as the split.

Random forests algorithm with method 1

- 1. For $b = 1$ to B
  - a. Draw a bootstrapped dataset D_b of size N from dataset D
  - b. Grow a decision tree T_b based on D_b by recursively repeating the following steps for each decision node until reaching a leaf
    - Randomly select m attributes from the M attributes
    - Pick the best attribute and its threshold as the split
    - Split the node into two daughter nodes
- 2. Output the ensemble of trees $\{T_b\}_{b=1}^{B}$

- Given a testing point $\mathbf{x}$, we feed it to all the trees, and use the majority vote as the classification result

Experimental results

- Datasets
  - 20 sets
  - First 13 from UCI
    - 90% for training
    - 10% for testing
  - Three sets are large
  - The last four are synthetic

Data set	Train size	Test size	Inputs	Classes
Glass	214	–	9	6
Breast cancer	699	–	9	2
Diabetes	768	–	8	2
Sonar	208	–	60	2
Vowel	990	–	10	11
Ionosphere	351	–	34	2
Vehicle	846	–	18	4
Soybean	685	–	35	19
German credit	1000	–	24	2
Image	2310	–	19	7
Ecoli	336	–	7	8
Votes	435	–	16	2
Liver	345	–	6	2
Letters	15000	5000	16	26
Sat-images	4435	2000	36	6
Zip-code	7291	2007	256	10
Waveform	300	3000	21	3
Twonorm	300	3000	20	2
Threenorm	300	3000	20	2
Ringnorm	300	3000	20	2

Experimental results

- One tree
  - A single decision tree
- Random forests (Forest-RI single input)
  - Use method 1 with $m = 1$ for decision node construction
  - 100 trees (except 200 trees in dataset ``zip-code'')
- Random forests (Selection)
  - Use method 1 with $m = 1$ and $m = \log_2(M + 1)$ for decision node construction
  - For each round, we use out-of-bag error to select the better tree
  - 100 trees (except 200 trees in dataset zip-code)
- AdaBoost
  - 50 trees (except 100 trees in dataset zip-code)

Test set errors (%).

Data set	Adaboost	Selection	Forest-RI single input	One tree
Glass	22.0	20.6	21.2	36.9
Breast cancer	3.2	2.9	2.7	6.3
Diabetes	26.6	24.2	24.3	33.1
Sonar	15.6	15.9	18.0	31.7
Vowel	4.1	3.4	3.3	30.4
Ionosphere	6.4	7.1	7.5	12.7
Vehicle	23.2	25.8	26.4	33.1
German credit	23.5	24.4	26.2	33.3
Image	1.6	2.1	2.7	6.4
Ecoli	14.8	12.8	13.0	24.5
Votes	4.8	4.1	4.6	7.4
Liver	30.7	25.1	24.7	40.6
Letters	3.4	3.5	4.7	19.8
Sat-images	8.8	8.6	10.5	17.2
Zip-code	6.2	6.3	7.8	20.6
Waveform	17.8	17.2	17.3	34.0
Twonorm	4.9	3.9	3.9	24.7
Threenorm	18.8	17.5	17.5	38.4
Ringnorm	6.9	4.9	4.9	25.7

References

- AdaBoost
  - Y. Freund and R. Schapire: A decision-theoretic generalization of on-line learning and an application to boosting, in EuroCOLT, 1995.
- Decision tree
  - Chapter 14.4 in the PRML textbook.
- Bagging
  - L. Breiman: Bagging predictors, Machine Learning, 1996.
- Random forests
  - L. Breiman: Random forests, Machine Learning, 2001.

Thank You for Your Attention!

Yen-Yu Lin (林彥宇)

Email: lin@cs.nycu.edu.tw

URL: https://www.cs.nycu.edu.tw/members/detail/lin