Classification: Alternative Classification Methods

Cam Tu Nguyen

阮锦绣

Software Institute, Nanjing University
nguyenct@lamda.nju.edu.cn
ncamtu@gmail.com

Outline

- Support Vector Machines
- Artificial Neural Network (ANN)
- Lazy Learners
- Ensemble Learning

Support Vector Machines

- Linear Binary Classification
  - The case where the data are linearly separable
  - The case where the data are linearly inseparable

- Non-linear Classification
- Multi-class classification

Linearly Separable Cases

- Let $D = \{(X_1, y_1), (X_2, y_2), ..., (X_{|D|}, y_{|D|})\}$ be the training set with associated labels, $y_i \in \{+1, -1\}$

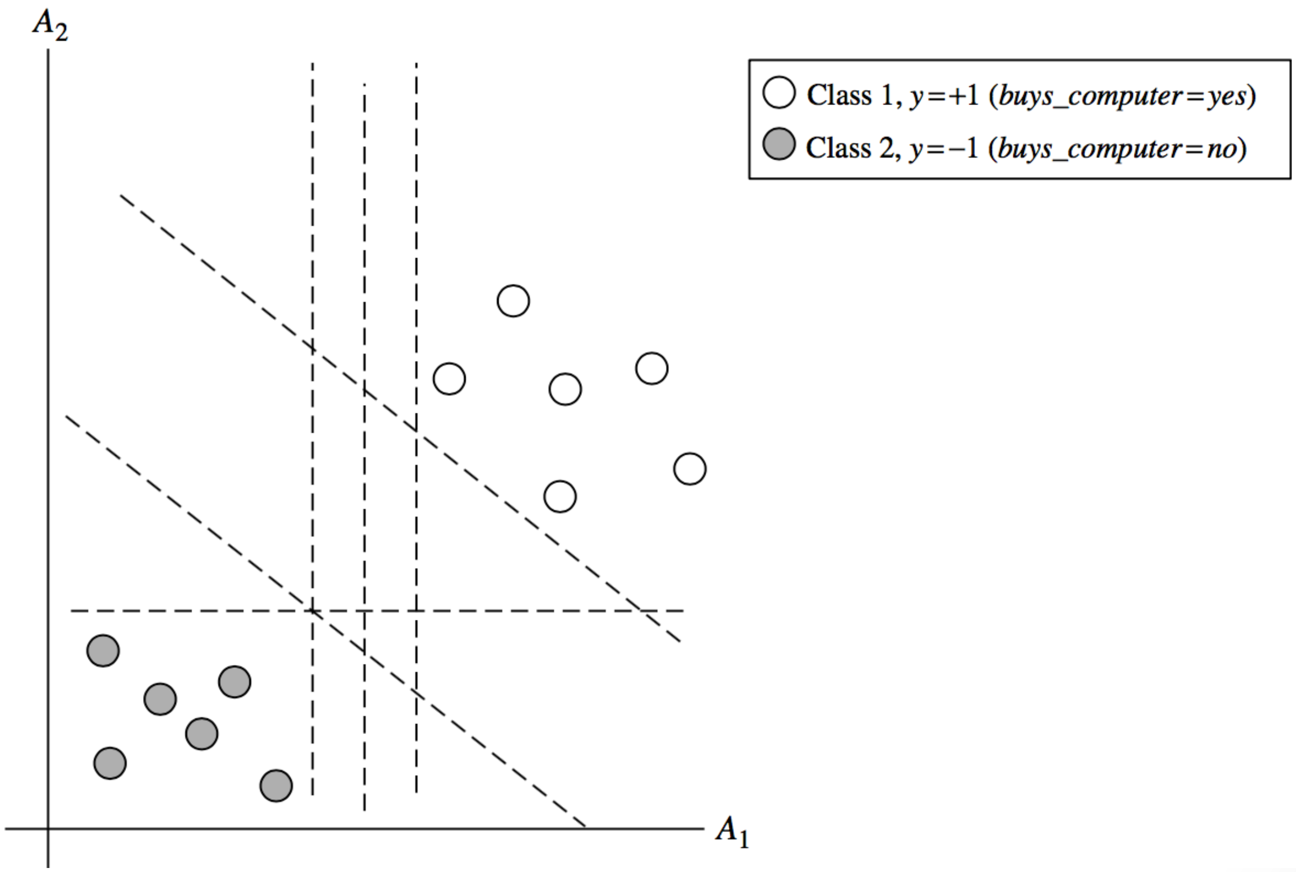
- Hyperplane equation $W^T X + b = 0$

  - A hyperplane in R^2 is a line
  - A hyperplane in R^3 is a space
  - Generalization: a hyperplane in R^n is a (n-1) dimensional affine supbspace of R^n.

- We consider the case in which there exists a hyperplane that separates positive and negative points in the training set.
  - The objective is to find that hyperplane.

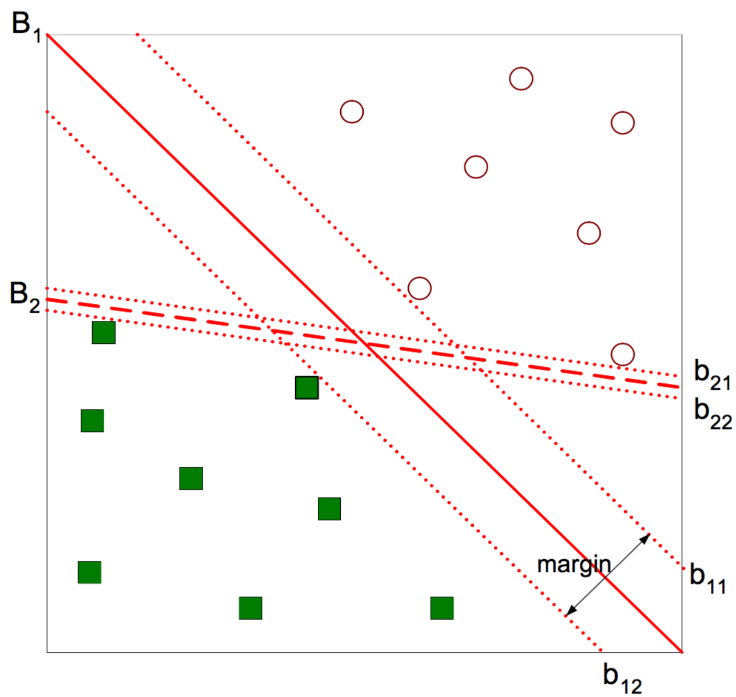
Linearly Separable Cases

- There are infinite number of separating hyperplanes.



Linearly Separable Cases

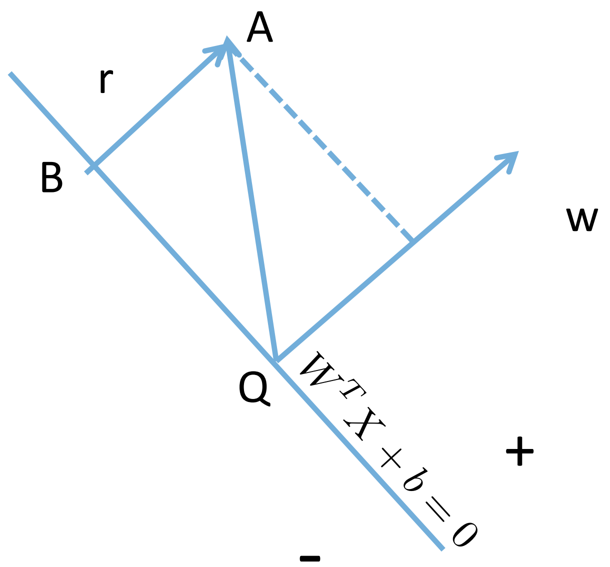
- Support Vector Machine:
  - Find the separating hyperplane with **maximum margin**
  - Why? Maximum marginal hyperplane has better chance to be more accurate to predict future data (better generalization).



Margin: The shortest distance to the closest training point of either class.

Linearly Separable Cases

- Distance from the nearest point A in the training set (assume that it lies on the positive region) to the hyperplane:



$$r = \frac{W^T(X_A - X_Q)}{||W||}$$

Since Q lies in the separating hyperplane:

$$W^T X_Q + b = 0$$

$$r = \frac{W^T X_A + b}{||W||}$$

Generalize to both sides of the hyperplane:

$$r = \frac{y_A(W^T X_A + b)}{||W||}$$

Positive region: $W^T X + b \geq 0$

Negative region: $W^T X + b < 0$

Linearly Separable Cases

- Recall: distance from a nearest point A in the training set to the separating hyperplane:

$$r = \frac{y_A(W^T X_A + b)}{||W||}$$

- Note: this distance is invariant to any scaling of W and b; i.e; r for 5W and 5b is the same with W and b.

- To facilitate calculation, we can scale W and b so that:

$$y_A(W^T X_A + b) = 1$$

Linearly Separable Cases

- Since A is the nearest point to the hyperplane, all the points in the training set satisfy:

$$y(W^T X + b) \geq 1$$

- The margin is twice the distance from the nearest point to the hyperplane, and thus it is:

$$\gamma = \frac{2}{||W||}$$ (We want maximize this margin)

- Put these together, we have the optimization problem:

$$\min_{W,b} \frac{1}{2} W^T W \qquad \text{s.t.} \quad y_i(W^T X_i + b) \geq 1 \forall i$$

Linearly Separable Cases

- Primal problem (restated)

$$\min_{W,b} \frac{1}{2} W^T W \qquad \text{s.t.} \qquad y_i(W^T X_i + b) \geq 1 \forall i$$

  - Numerical methods to solve (constrained) quadratic optimization
  - Support vectors: training points that satisfy $y_i(W^T X_i + b) = 1$
  - Prediction: classify new data point X' using $sign(W^T X' + b)$

Linearly Separable Cases

- Dual Problem: Use Lagrange method, we convert the primal problem to a dual problem

$$\max_{\alpha_1,\alpha_2,\ldots,\alpha_N} \sum \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j Y_i Y_j X_i^T X_j$$

Subject to: $\sum_i \alpha_i Y_i = 0$

$$\alpha_i \geq 0 \forall 1 \leq i \leq N$$

- Solution is of the form

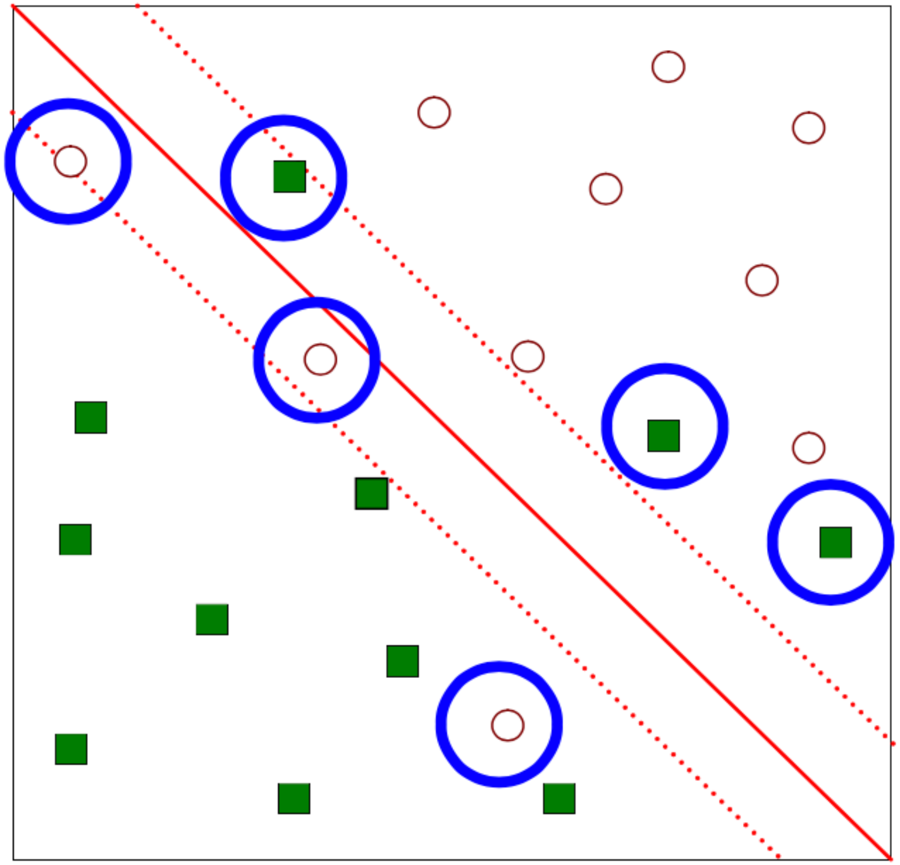
$$W = \sum \alpha_i Y_i X_i$$
$$b = Y_k - W^T X_k \text{ for any } X_k \text{ such that } \alpha_k \neq 0$$

  - Most of α_i are zeros, those that are nonzero corresponds to support vectors.
  - Classify new X': $\text{sign}(\sum_i \alpha_i Y_i X_i^T X' + b)$

See more: http://cs229.stanford.edu/notes/cs229-notes3.pdf

Linearly Inseparable Cases

- What if the problem is not linearly separable



Linearly Inseparable Cases

- What if the problem is not linearly separable
  - Introduce slack variables
    - Need to minimize

$$\min_{\xi, W, b} \frac{1}{2} W^T W + C(\sum_{i=1}^{N} \xi_i)$$

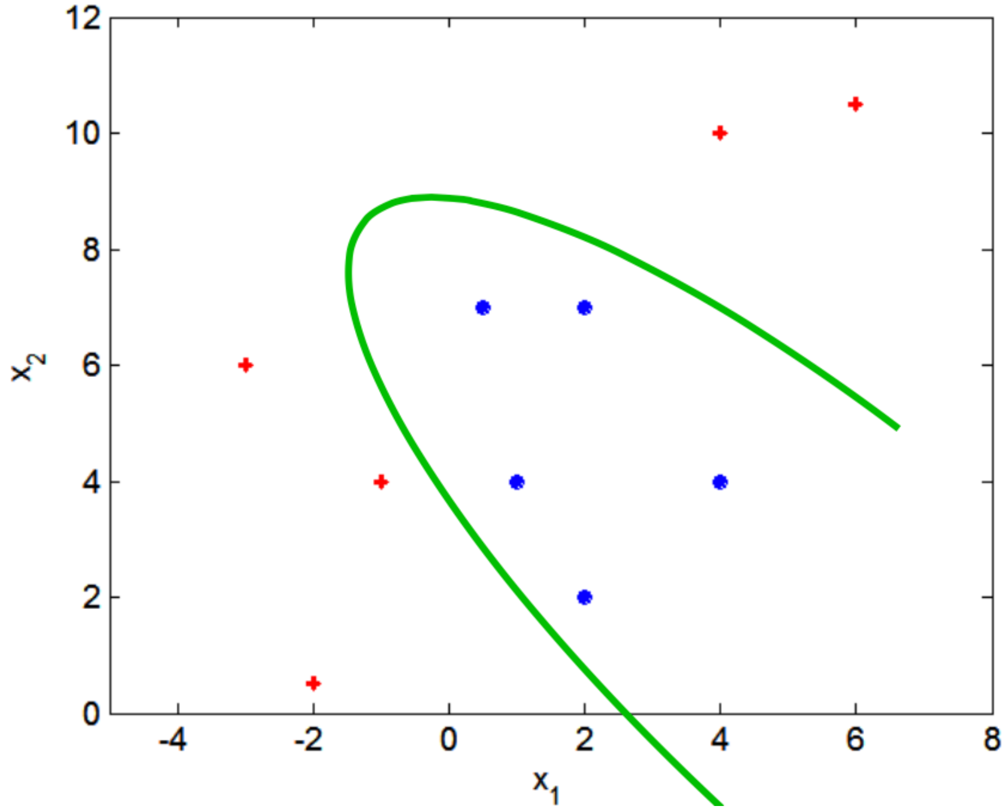
    - Subject to:

$$Y_i(W^T X_i + b) \geq 1 - \xi_i, i = 1, ..., N$$

$$\xi_i \geq 0, i = 1, ..., N$$

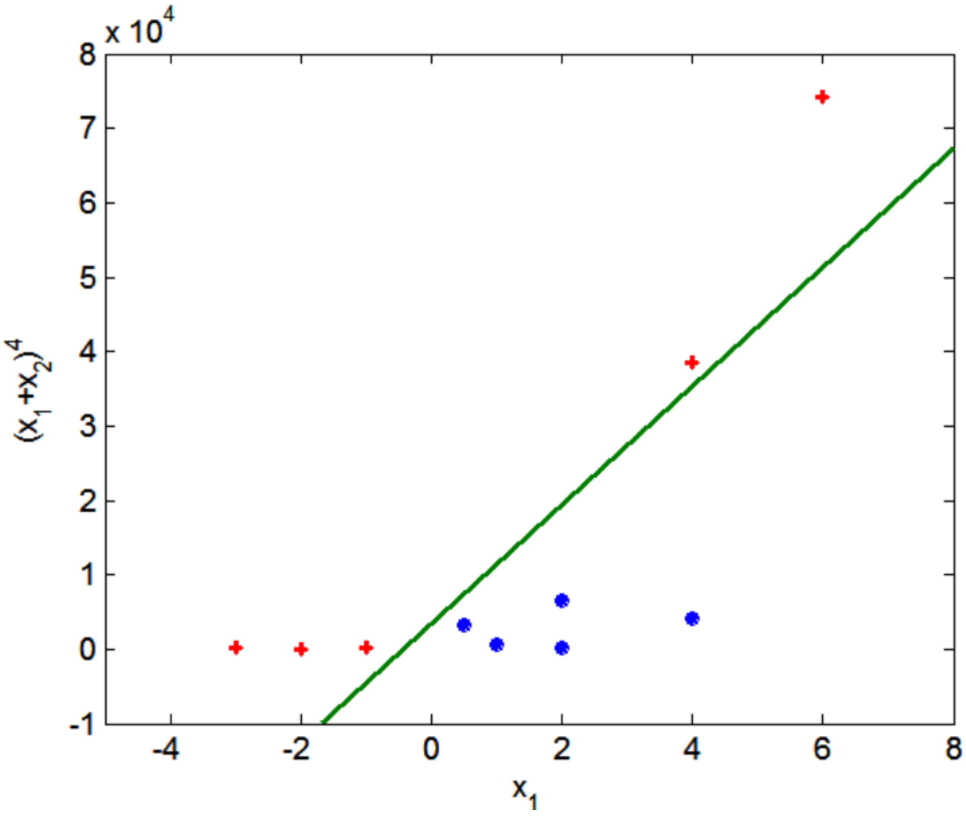
Nonlinear Support Vector Machines

- What if decision boundary is not linear?



Nonlinear Support Vector Machines

- Transform data into higher dimensional space
  - Using Kernel Trick



SVM for Multi-class classification

- Multi-class Classification
  - Decompose the problem into binary classification problems
    - One-vs-one
    - One-vs-rest

Practical Considerations for SVM

- SVMs are integrated into most of machine learning and data mining libs, tools
  - Weka
  - Python scikit-learn
  - LibSVM (C/C++)
  - SVMLight

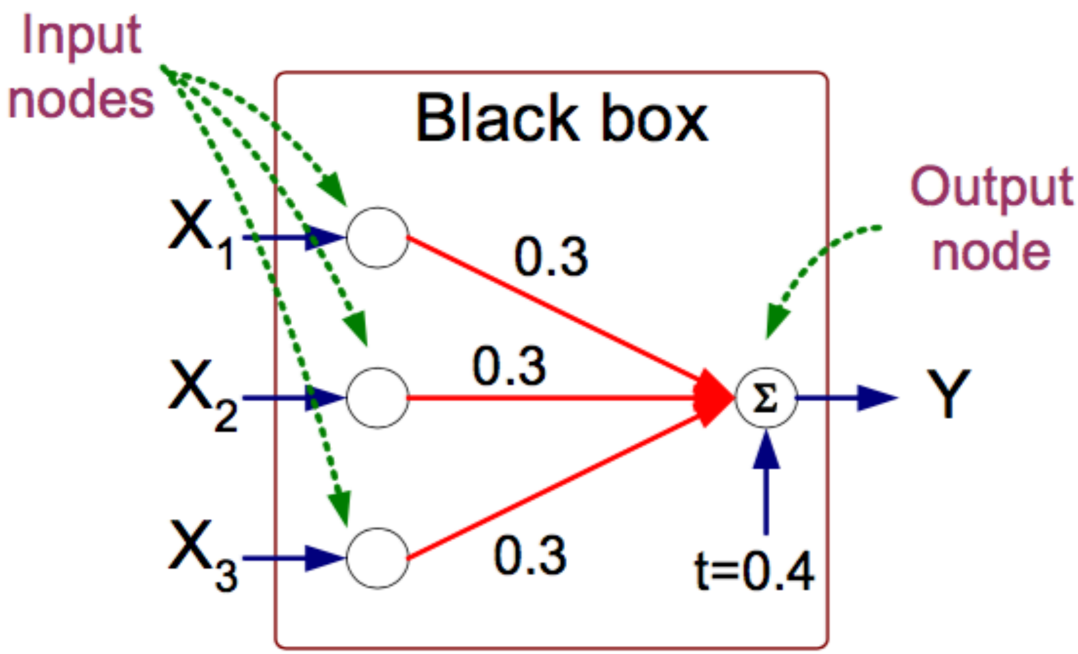
- Some notes on training with SVM
  - Normalization helps improve performance
  - Tune parameters (change constants C, kernel parameters) using validation set.

Outline

- Support Vector Machines
- Artificial Neural Network
- Lazy Learners
- Ensemble Learning

Artificial Neural Network (ANN)

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



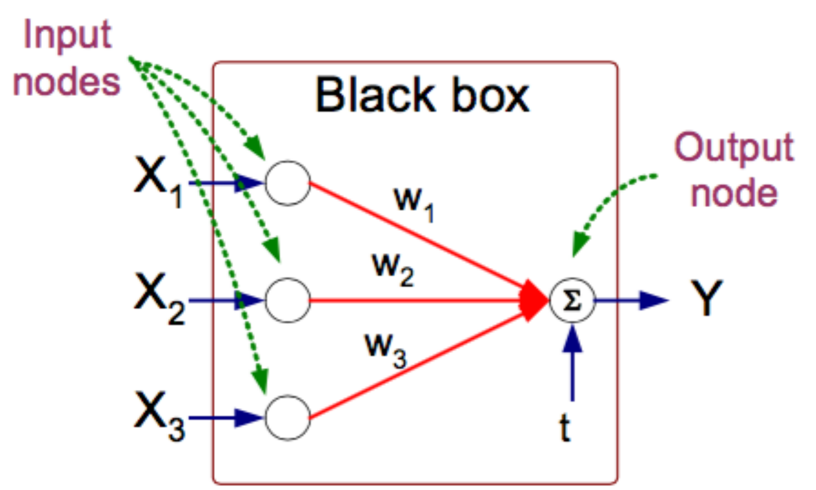
$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where} \quad I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

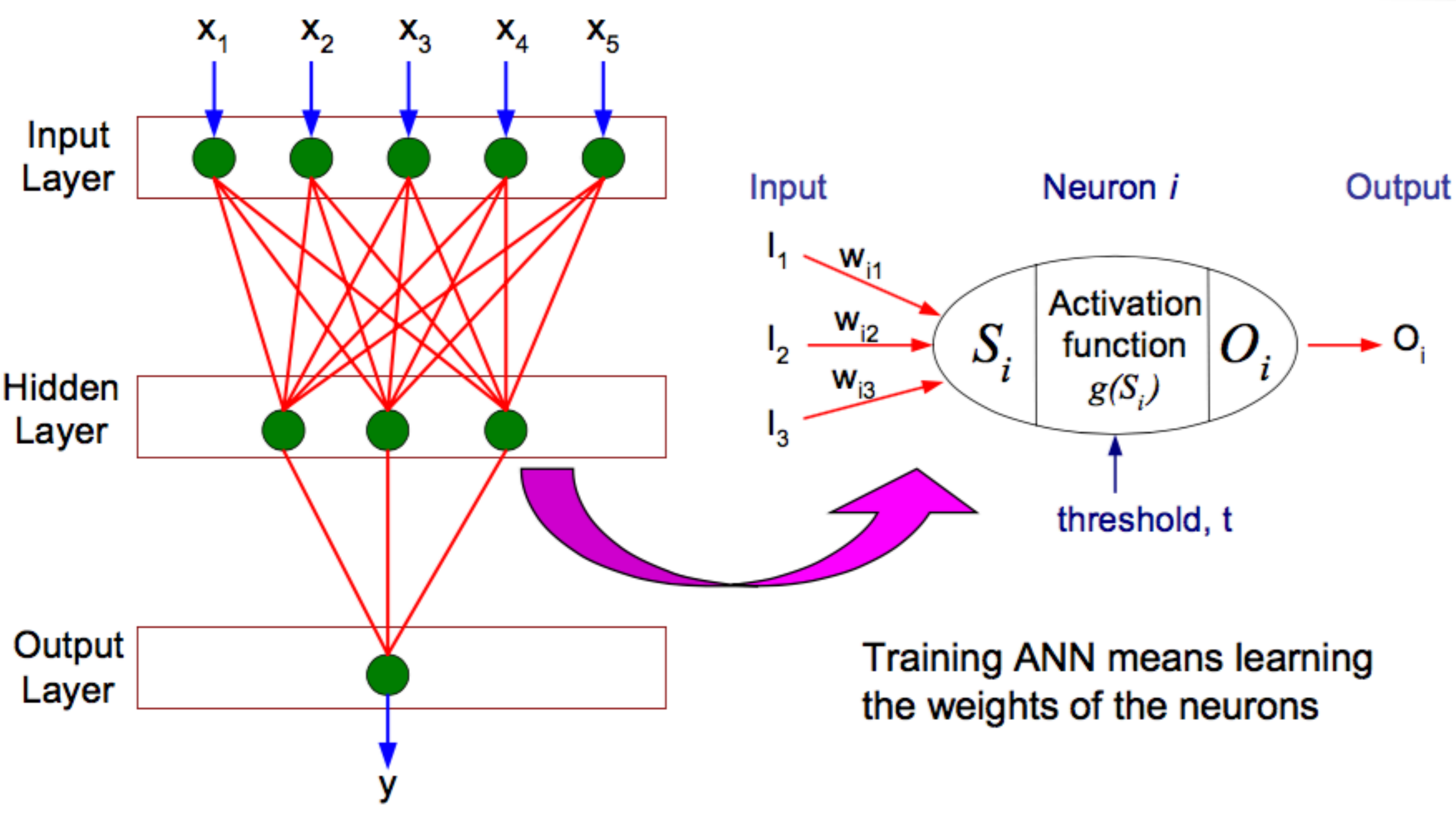
Artificial Neural Network (ANN)

- Model is an assembly of inter-connected nodes and weighted links.

- Output node sum up each of its input value according to the weights of its links



General Structure of ANN

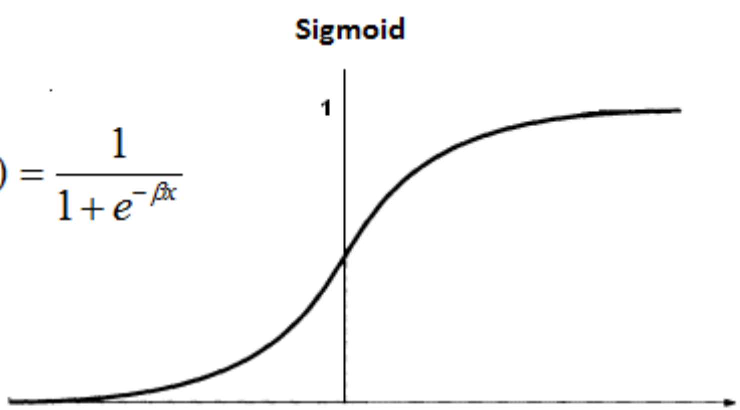


General Structure of ANN

- Define a network
  - Define the network topology
  - Define the activation functions

- Common Activation function
  - Logistic or sigmoid function

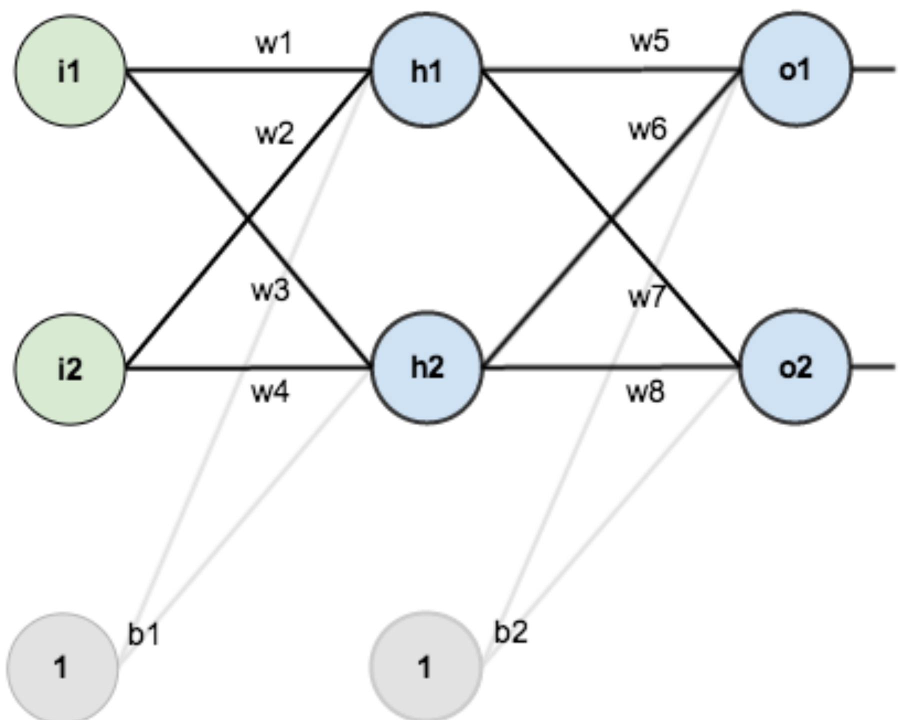
$$f(x) = \frac{1}{1+e^{-\beta x}}$$



$$O_i = g(S_i) = \frac{1}{1+e^{-S_i}}$$

Learning ANN with Backpropagation

- Learning objective:
  - Find weights **w** for links in ANN to minimize the network error.



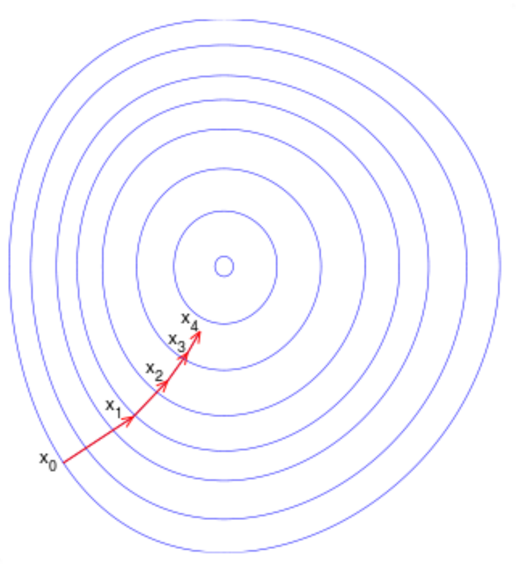
Gradient Descent (brief intro)

- Main idea:
  - If the multi-variable function F(x) is **defined** and **differentiable** in a neighborhood of a point **a,** then F(x) decreases fastest if one goes from **a** in the direction of the negative gradient of F(x) at **a**
  - If one starts with a guess x0 for a local minimum of F, and considers the sequence x0, x1, x2, …., such that

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n),\ n \ge 0.$$

  - We have

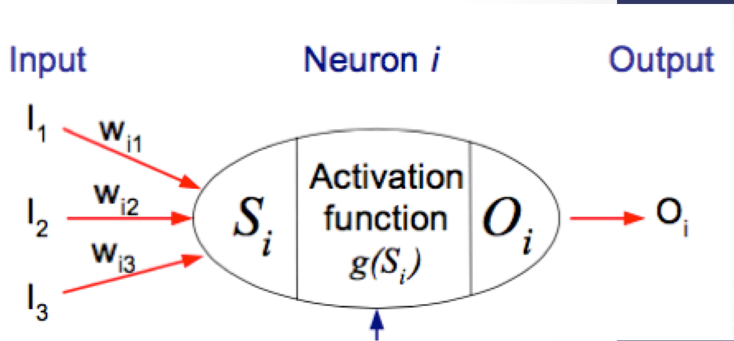
$$F(\mathbf{x}_0) \ge F(\mathbf{x}_1) \ge F(\mathbf{x}_2) \ge \cdots,$$



Backpropagation

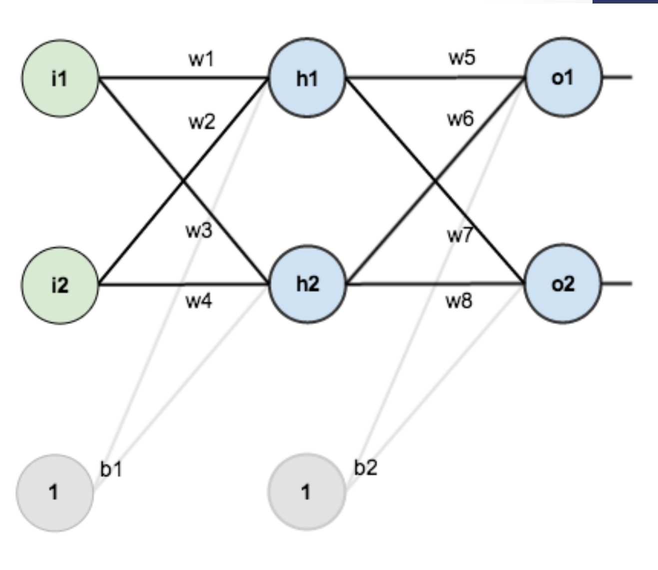
- The Forward Pass
  - Hidden node h1

$$S_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$
$$O_{h1} = \frac{1}{1 + e^{-S_{h1}}}$$



  - Hidden node h2 (similar to h1)
  - Output node o1

$$S_{o1} = w_5 * O_{h1} + w_6 * O_{h2} + b_2 * 1$$
$$O_{o1} = \frac{1}{1 + e^{-S_{o1}}}$$



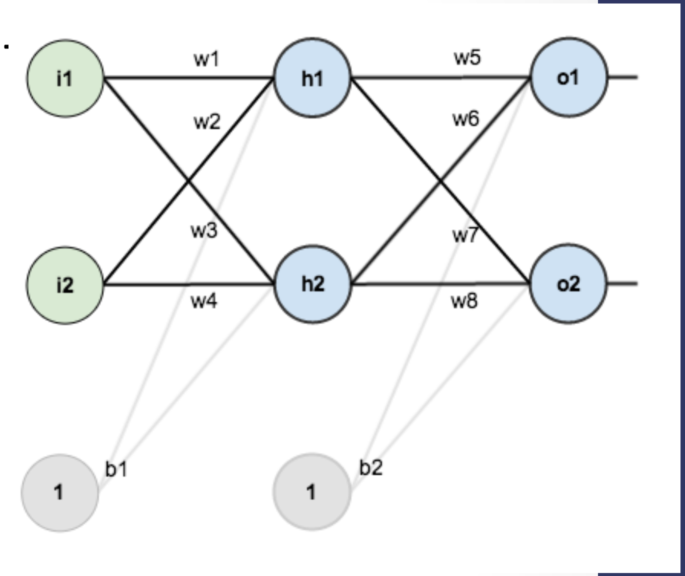
Backpropagation

- Least Square Error Method: total error of
training data set.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

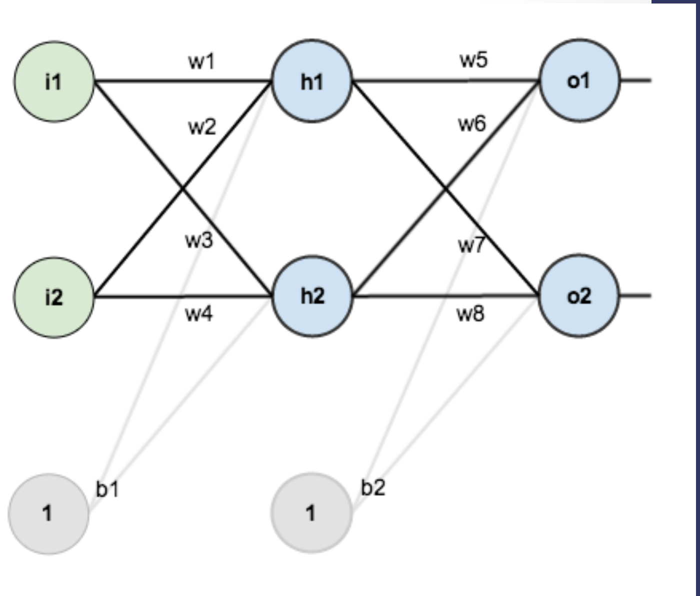
- Total Error (cont.)

$$\begin{aligned} E_{total} &= E_{o1} + E_{o2} \\ E_{o1} &= 1/2(\text{target}_{o1} - O_{o1})^2 \\ E_{o2} &= 1/2(\text{target}_{o2} - O_{o2})^2 \end{aligned}$$



Backpropagation

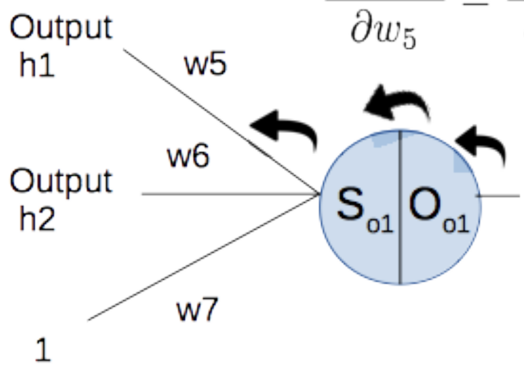
- The Backward Pass
  - The goal of backpropagation is to update link weights so that they cause **the actual output to be closer to the target out put.**
    - Minimize the error for each output neuron and for the whole network.



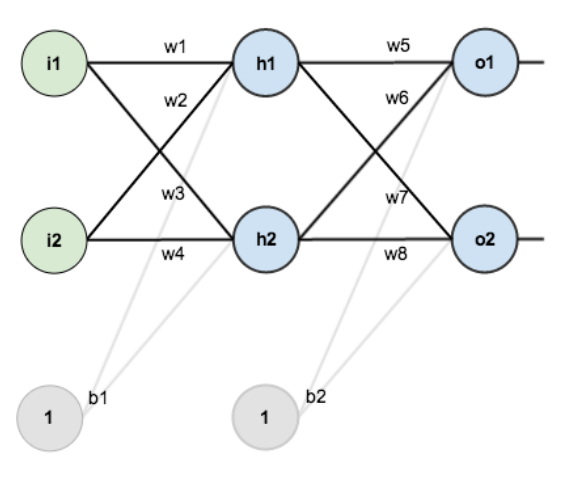
Backpropagation

- The Backward Pass
  - **The Output Layer**

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial O_{o1}} * \frac{\partial O_{o1}}{\partial S_{o1}} * \frac{\partial S_{o1}}{\partial w_5}$$



$$\begin{aligned} E_{total} &= E_{o1} + E_{o2} \\ E_{o1} &= 1/2(\text{target}_{o1} - O_{o1})^2 \end{aligned}$$



$$\begin{aligned} \frac{\partial E_{total}}{\partial w_5} &= -(target_{o1} - O_{o1}) * O_{o1} * (1 - O_{o1}) * O_{h1} \\ \text{Let } \delta_{o1} &= \frac{\partial E_{total}}{\partial O_{o1}} * \frac{\partial O_{o1}}{\partial S_{o1}} = -(target_{o1} - O_{o1}) * O_{o1} * (1 - O_{o1}) \\ \rightarrow \frac{\partial E_{total}}{\partial w_5} &= \delta_{o1} O_{h1} \end{aligned}$$

Backpropagation

- The Backward Pass
  - **The Output Layer**

$$
\begin{aligned}
\frac{\partial E_{total}}{\partial w_5} &= -(target_{o1} - O_{o1}) * O_{o1} * (1 - O_{o1}) * O_{h1} \\
\text{Let } \delta_{o1} &= \frac{\partial E_{total}}{\partial O_{o1}} * \frac{\partial O_{o1}}{\partial S_{o1}} = -(target_{o1} - O_{o1}) * O_{o1} * (1 - O_{o1}) \\
\rightarrow \frac{\partial E_{total}}{\partial w_5} &= \delta_{o1} O_{h1}
\end{aligned}
$$

  - To decrease the error, we then update w5 as follows

$$w_5^* = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

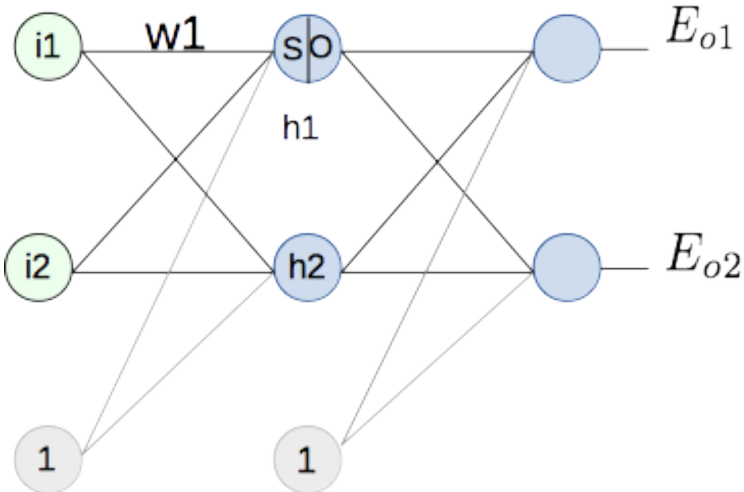
  - Where η is the learning rate.
  - Similar process can be made to derive updates for w6, w7, w8

Backpropagation

- The Backward Pass
  - **The Hidden Layer**

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial O_{h1}} * \frac{\partial O_{h1}}{\partial S_{h1}} * \frac{\partial S_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial O_{h1}} = \frac{\partial E_{o1}}{\partial O_{h_1}} + \frac{\partial E_{o2}}{\partial O_{h_1}}$$



$E_{total} = E_{o1} + E_{o2}$

Backpropagation

- The Backward Pass
  - **The Hidden Layer**

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \frac{\partial E_{total}}{\partial O_o} * \frac{\partial O_o}{\partial S_o} * \frac{\partial S_o}{\partial O_{h1}}) * \frac{\partial O_{h1}}{\partial S_{h1}} * \frac{\partial S_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = (\sum_o \delta_o * w_{h_o}) * O_{h1}(1 - O_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} * i_1$$

- **Update** w1 $w_1^* = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1}$
- **Similar** calculations can be made to update w2, w3, w4

Backpropagation

- Training ANN using Backpropagation
  - Initialize weights (w1, w2, …)
  - Until **terminating condition** is satisfied
    - For each training data point
      - Propagate the inputs forward
      - Calculate errors in the output layer
      - Backpropagate the errors and update weights
  - Note: **learning rate** η (domain [0,1]) can be selected by 1/t where t is the number of iterations through the training set so far.

- Terminating Conditions
  - The change in weights are small
  - The percentage of misclassified data points in the previous epoch (iteration) is below some threshold
  - A specified number of epochs (iterations) has been reached.

Outline

- Support Vector Machines
- Artificial Neural Network
- Lazy Learners
- Ensemble Learning

Lazy Learners

Set of Stored Cases

Atr1	………	AtrN	Class
			A
			B
			B
			C
			A
			C
			B

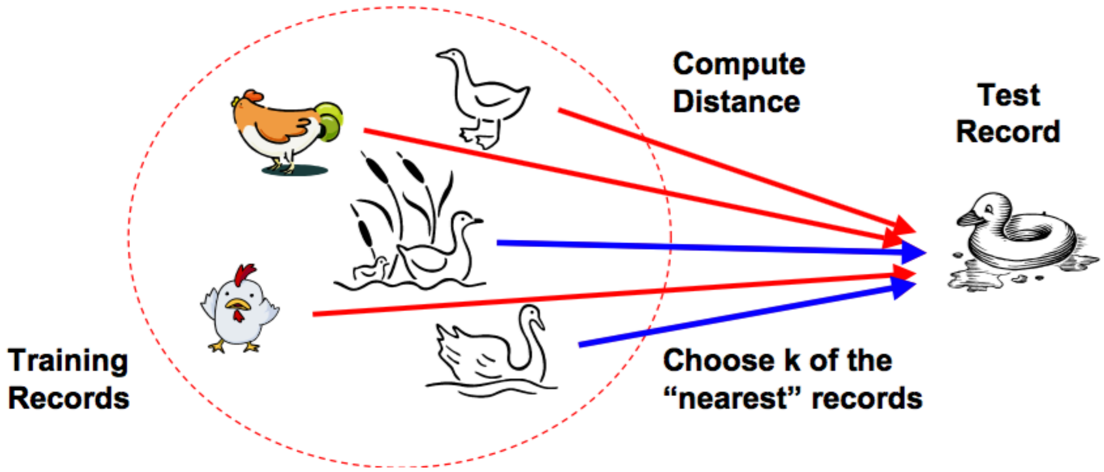
- Store the training records
- Use training records to predict the class label of unseen cases

Unseen Case

Atr1	………	AtrN

Lazy Learners

- Examples
  - K-Nearest Neighbor Classifiers

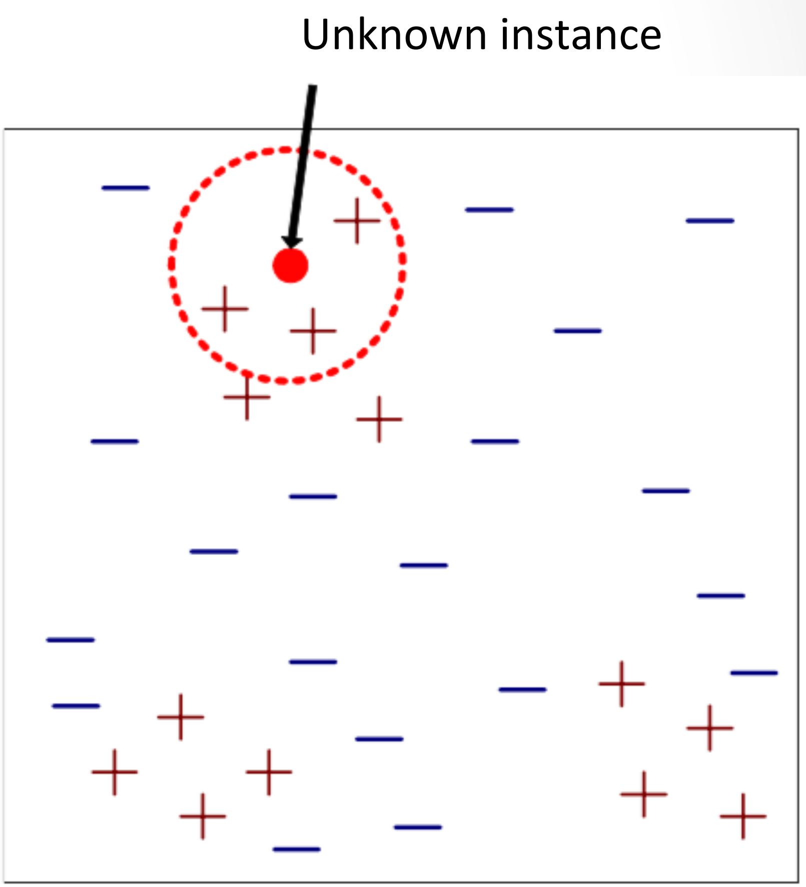


  - Case-based Reasoning

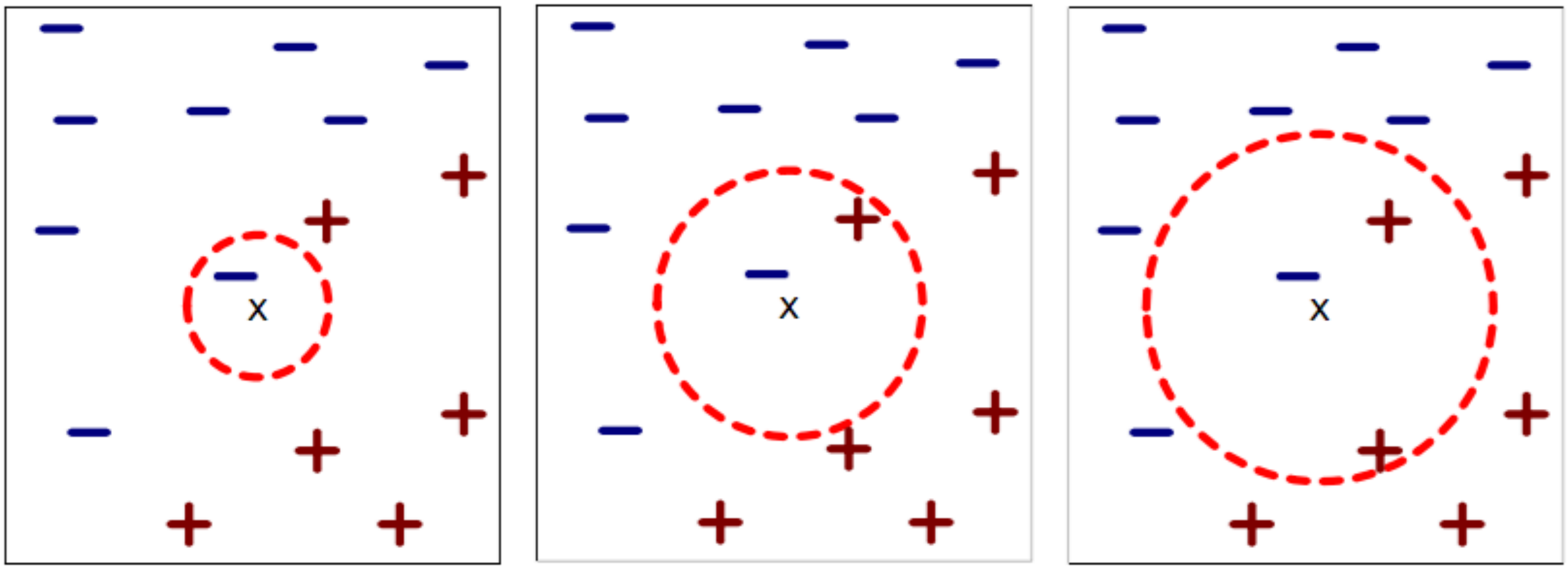
Nearest Neighbor Classifiers

- Requires three things
  - The set of stored instances
  - Distance metric to compute distance between instances
  - The value of **k,** the number of nearest neighbors to retrieve

- To classify an unknown record:
  - Compute distance to other training instances
  - Identify **k** nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown instance (by taking majority vote).



Definition of Nearest Neighbor



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

K-nearest neighbors of an instance **X** are training instances that have the **k** smallest distance to **X**

Nearest Neighbor Classification

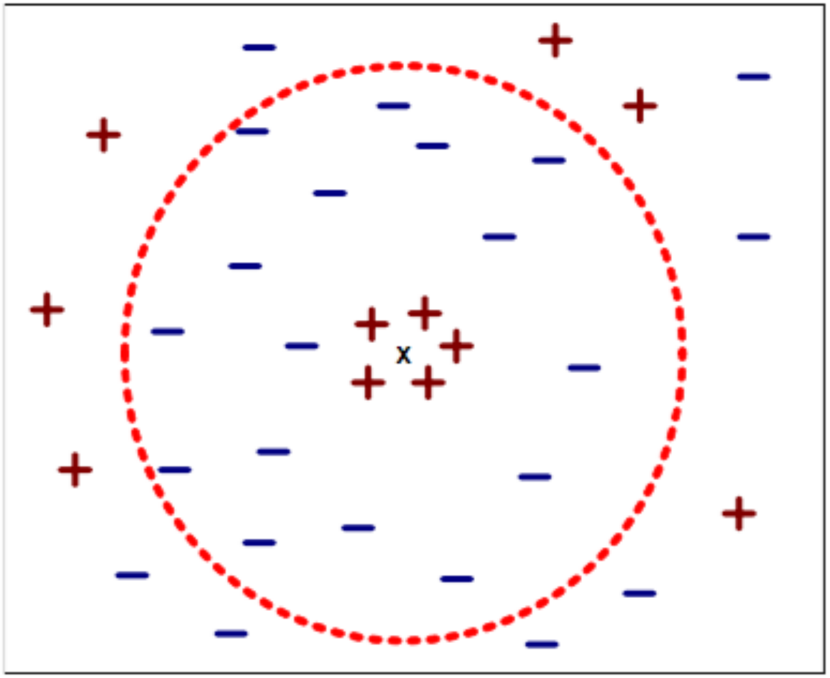
- Compute distance between two instances
  - Euclidean distance

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

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

Nearest Neighbor Classification

- Choosing the value of k
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes.



Nearest Neighbor Classification

- Scaling Issues
  - Attributes may have to be normalized to prevent distance measures from being dominated by one of the attributes

  - Example
    - Height of a person may vary from 1.5m to 1.8m
    - Weight of a person may vary from 40kg to 120kg
    - Income of a person may vary from \$10K to \$1M

Nearest Neighbor Classification

- K-NN classifiers are lazy learners
  - It does not build model explicitly
  - Classifying unknown instances are relatively expensive

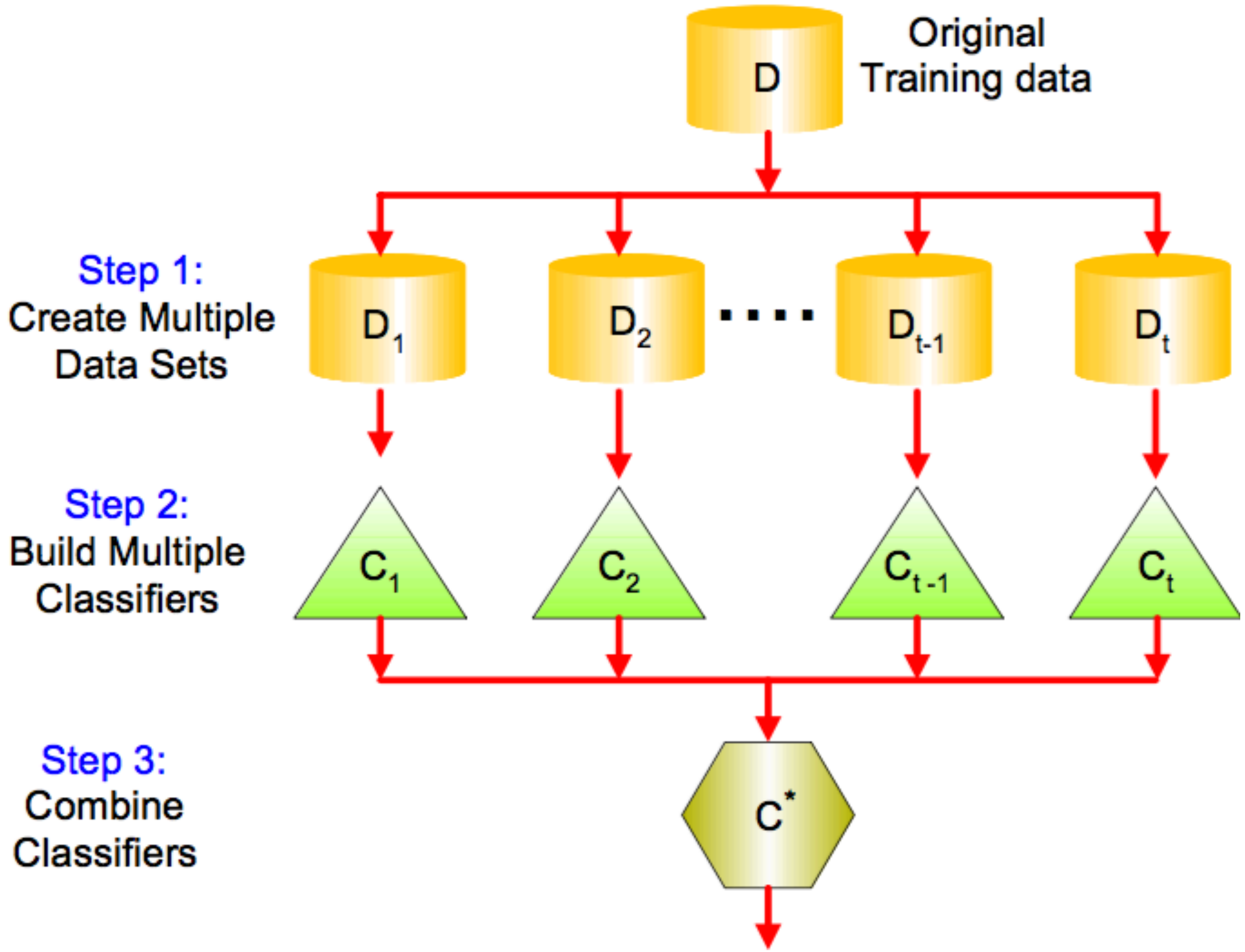
Outline

- Support Vector Machines
- Artificial Neural Network
- Lazy Learners
- Ensemble Learning

Ensemble Methods

- Construct a set of classifiers from the training data
- Predict class label of previously unseen records by aggregating predictions made by multiple classifiers

General Ideas



Why does it work?

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

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

Examples of Ensemble Methods

- How to generate an ensemble of classifiers
  - Bagging
  - Boosting

Bagging

- Sampling with replacement

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

- Build classifier on each bootstrap sample

- Each sample has probability $(1-1/n)^n$ of being selected.

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.

  - Initially, all N records are assigned equal weights

  - Unlike bagging, weights may change of the end of boosting round.

Boosting

- Records that are wrongly classified will have their weights increased.
- Records that are classified correctly will have their weights decreased.

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

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

Example: AdaBoost

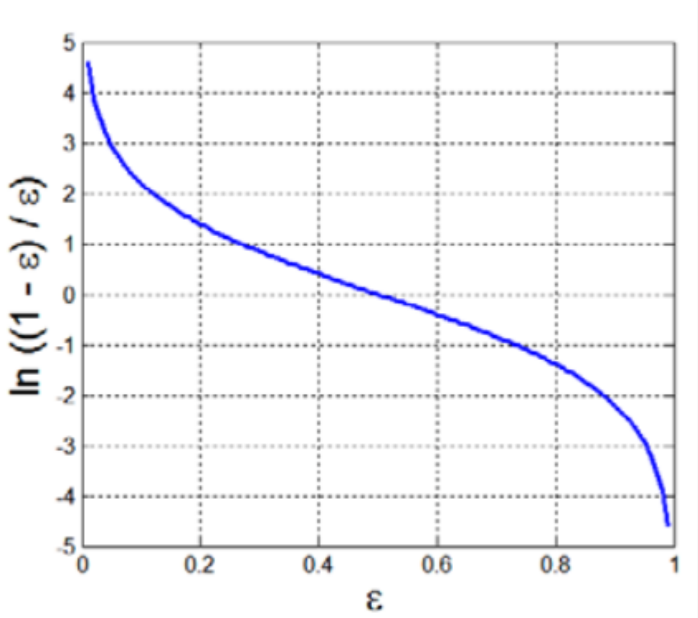
- Base Classifiers: C_1, C_2, …, C_T

- Error rate:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier

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



Example Adaboost

- Weight update:

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

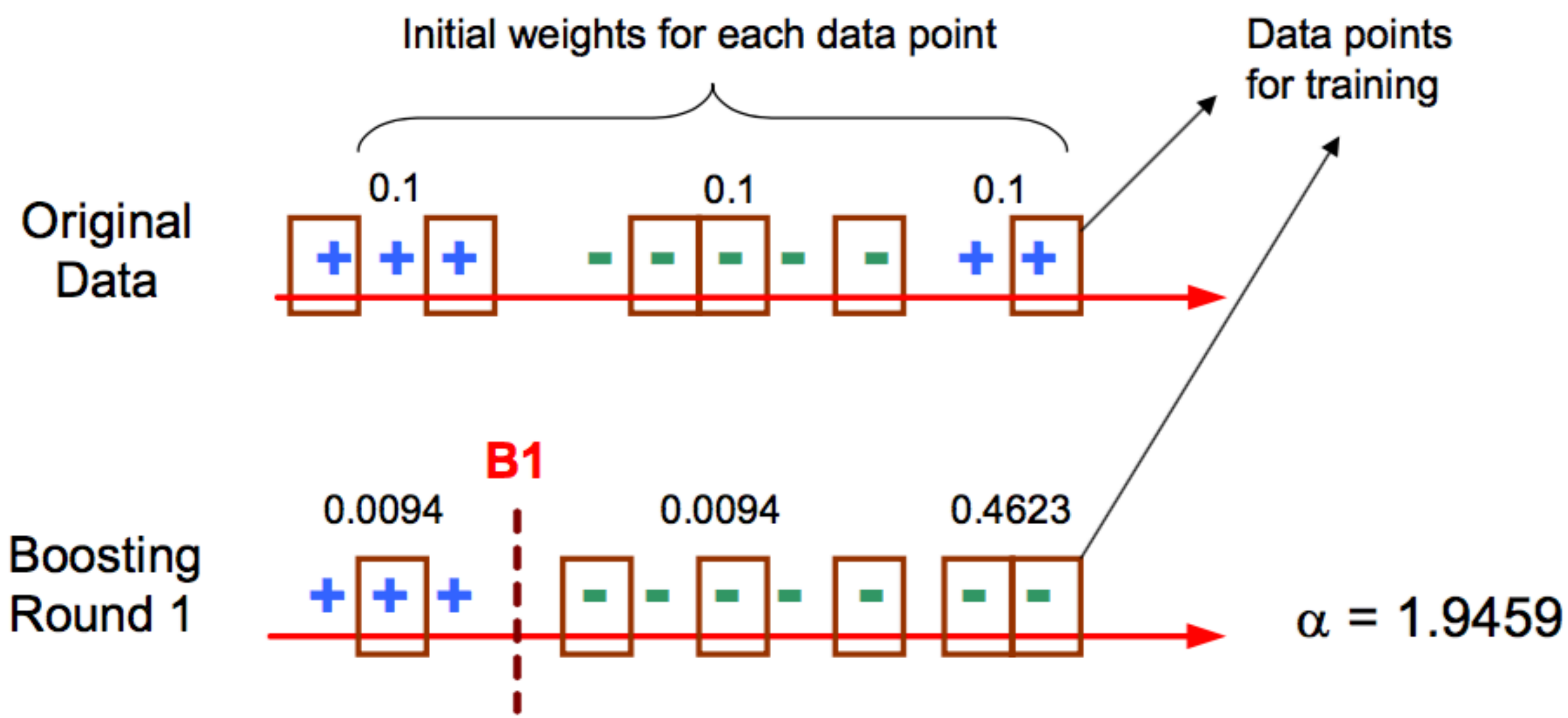
where Z_j is the normalization factor

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

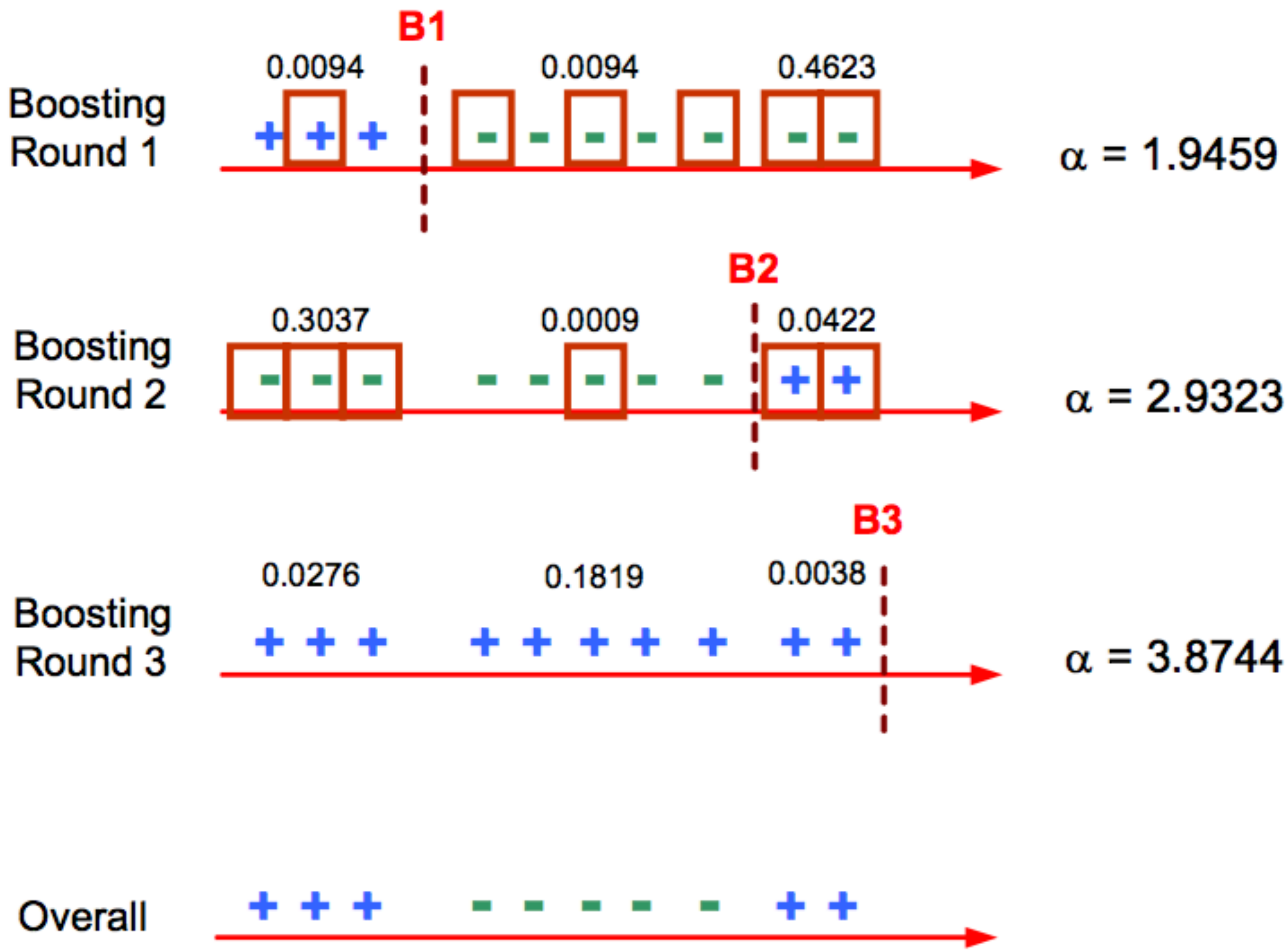
- Classification

$$\arg\max_y \sum_{j=1}^{T} \alpha_j \delta\left(C_j(x) = y\right)$$

Illustrating Adaboost



Illustrating AdaBoost



Summary

- Support Vector Machines
- Artificial Neural Network
- Lazy Learners
- Ensemble Learning