Data Mining:

Concepts and Techniques

(3rd ed.)

— Chapter 9 —
Classification: Advanced Methods

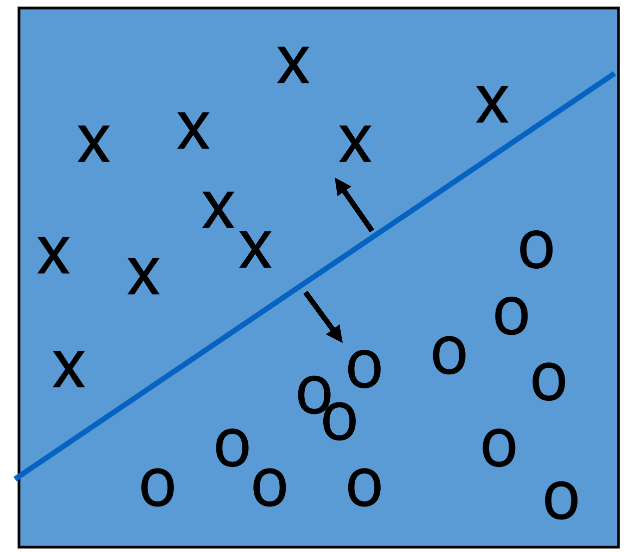
Slides Courtesy of Textbook

Chapter 9. Classification: Advanced Methods

- ~~Bayesian Belief Networks~~
- Perceptron, Backpropagation (Neural Network)
- Support Vector Machine
- ~~Classification by Using Frequent Patterns~~
- Lazy Learners (or Learning from Your Neighbors)
- ~~Other Classification Methods~~
- Additional Topics Regarding Classification
- Summary

Classification: A Mathematical Mapping

- Classification: predicts categorical class labels
  - E.g., Personal homepage classification
    - $x_i = (x_1, x_2, x_3, \ldots)$, $y_i = +1$ or -1
    - x_{i1} : # of word "homepage"
    - x_{i2} : # of word "welcome"
- Mathematically, $x \in X = \Re^n$, $y \in Y = \{+1, -1\}$,
  - We want to derive a function $f: X \rightarrow Y$
- Linear Classification
  - Binary Classification problem
  - Data above the red line belongs to class 'x'
  - Data below red line belongs to class 'o'
  - Examples: SVM, Perceptron, Probabilistic Classifiers



Linear binary classifier

- Suppose we work on binary classification($\{1, -1\}$), and the feature vector is d-dimensional vector.

- A linear classifier is determined by a (d+1)-dimensional vector $w = [w_0, w_1, \ldots, w_d]^T$. Given a feature vector $x = [x_1, \ldots, x_d]^T$, the linear classifier predict its label as:

$$\hat{y} = sign(w_0 + w_1 x_1 + \cdots + w_d x_d) = sign(w^T x)$$

where $sign(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & if\ z \geq 0 \end{cases}$

where we define $x_0 = 1$

Linear binary classifier

- Training data: $\{(x_i, y_i)\}_{i=1,\ldots,n}$
  - $x_i \in R^d$ feature vector
  - $y_i \in \{-1,1\}$ class label

- Suppose the training data is linearly separable, i.e. there exist a linear classifier such that
  - $y_i = sign(w^T x_i)$ for every training pair

- Our **goal** is to find such linear classifier.

Perceptron Algorithm

- Randomly initialize w and set η (learning rate) to be a small positive value
- Randomly pick a pair (x, y) from the training set:
  - If $y \neq sign(w^T x)$, update $w = w + \eta x y$
  - Otherwise, no action
- Repeat above procedure until the entire training set is classified correctly

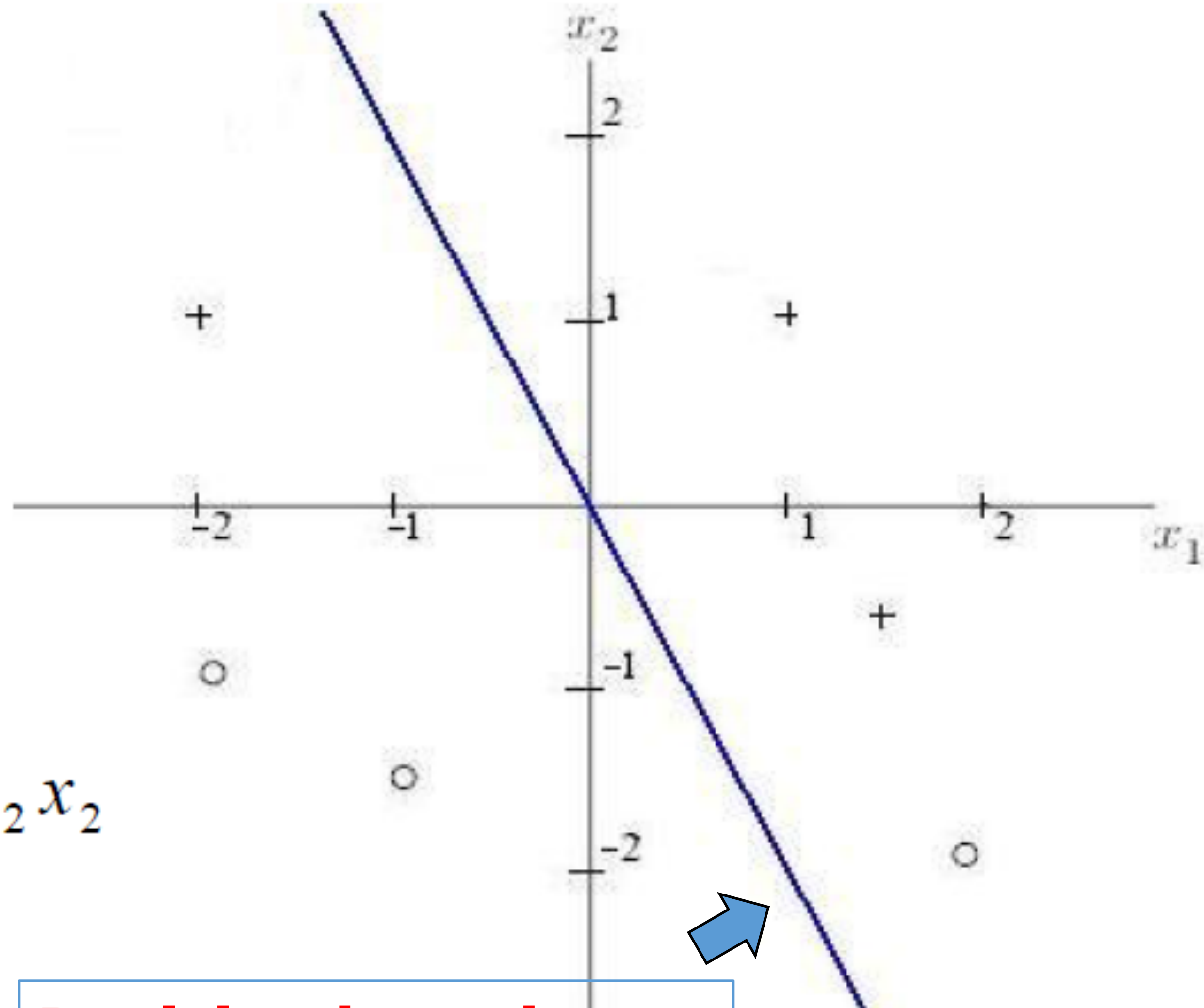
Learning Example

Initial Values:

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$0 = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0 + x_1 + 0.5 x_2$$
$$\Rightarrow \quad x_2 = -2 x_1$$

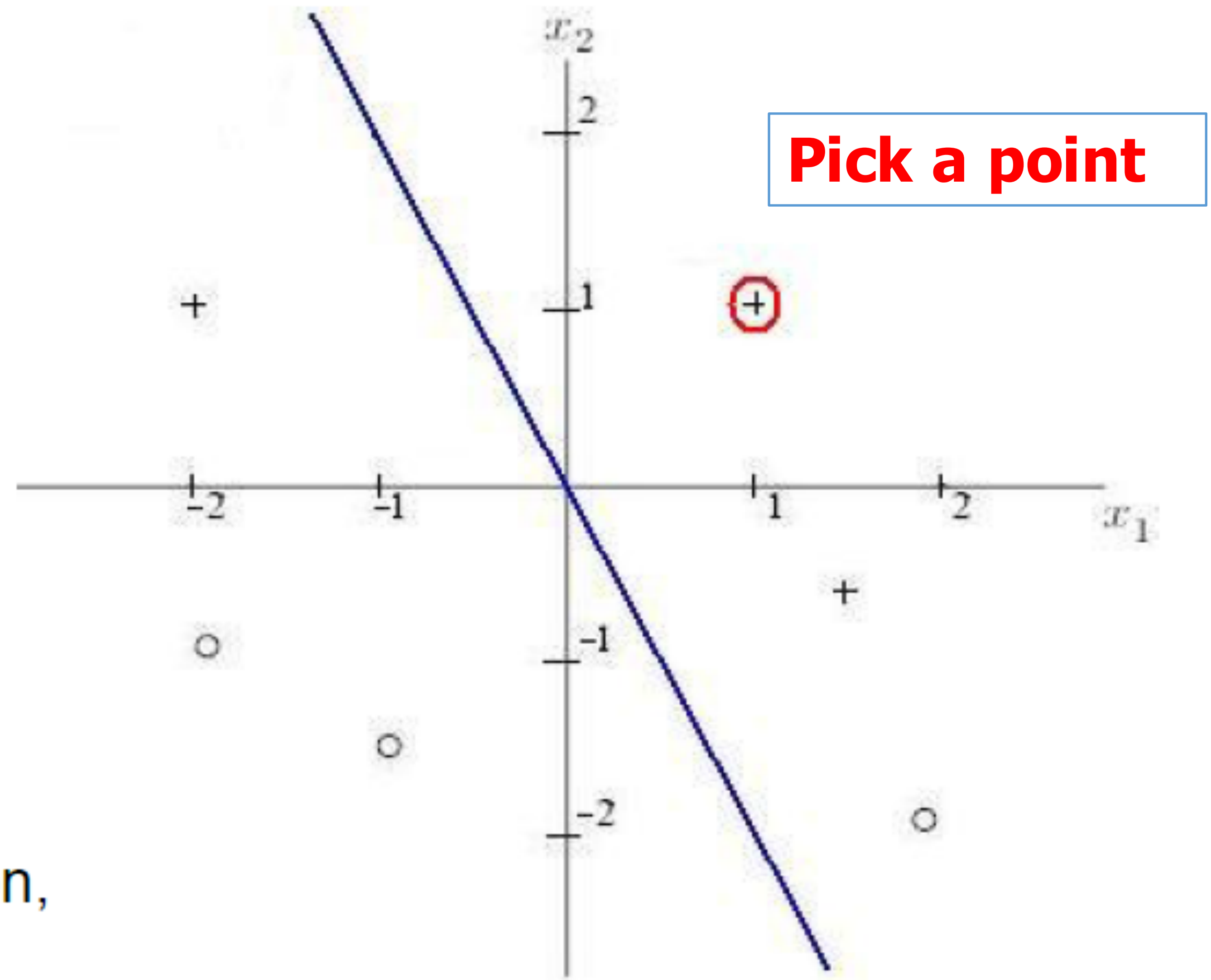


Decision boundary

Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$



$x_1 = 1, x_2 = 1$
$w^T x > 0$

Correct classification, no action

Learning Example

$\eta = 0.2$

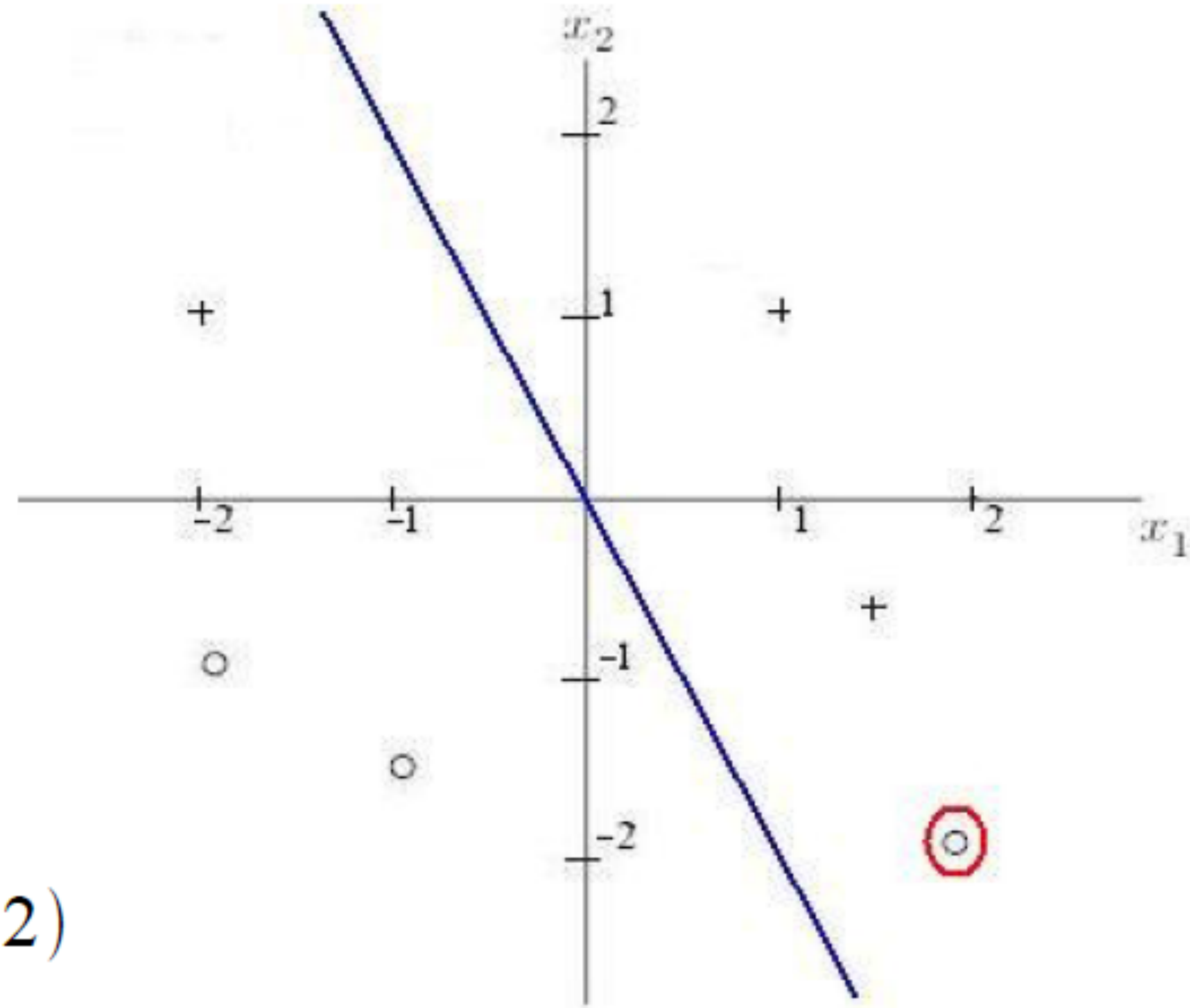
$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 2,\ x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

$$w_2 = w_2 - 0.2 * (-2)$$



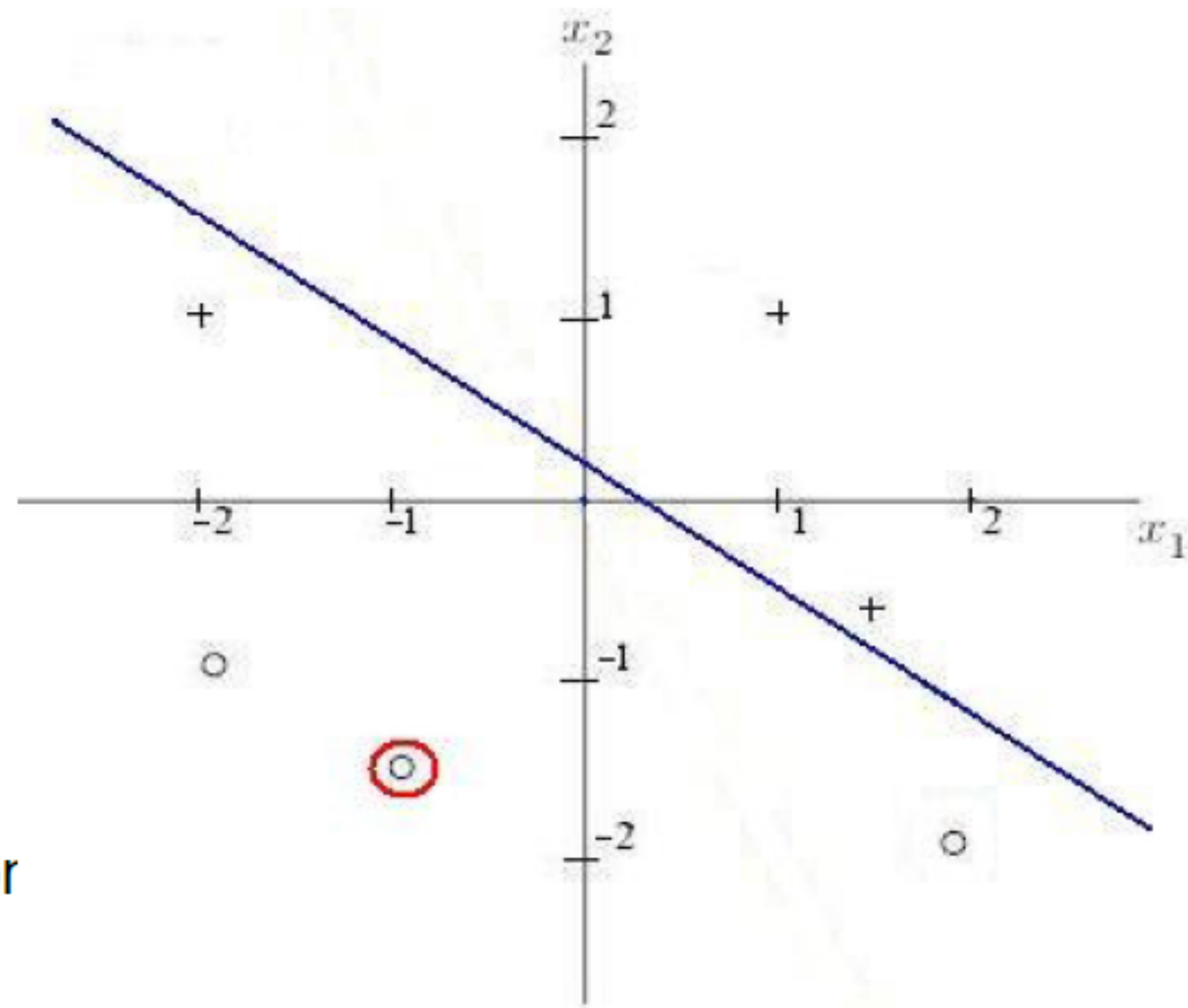
Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -1,\ x_2 = -1.5$
$w^T x < 0$

Correct classificatior
no action



Learning Example

$\eta = 0.2$

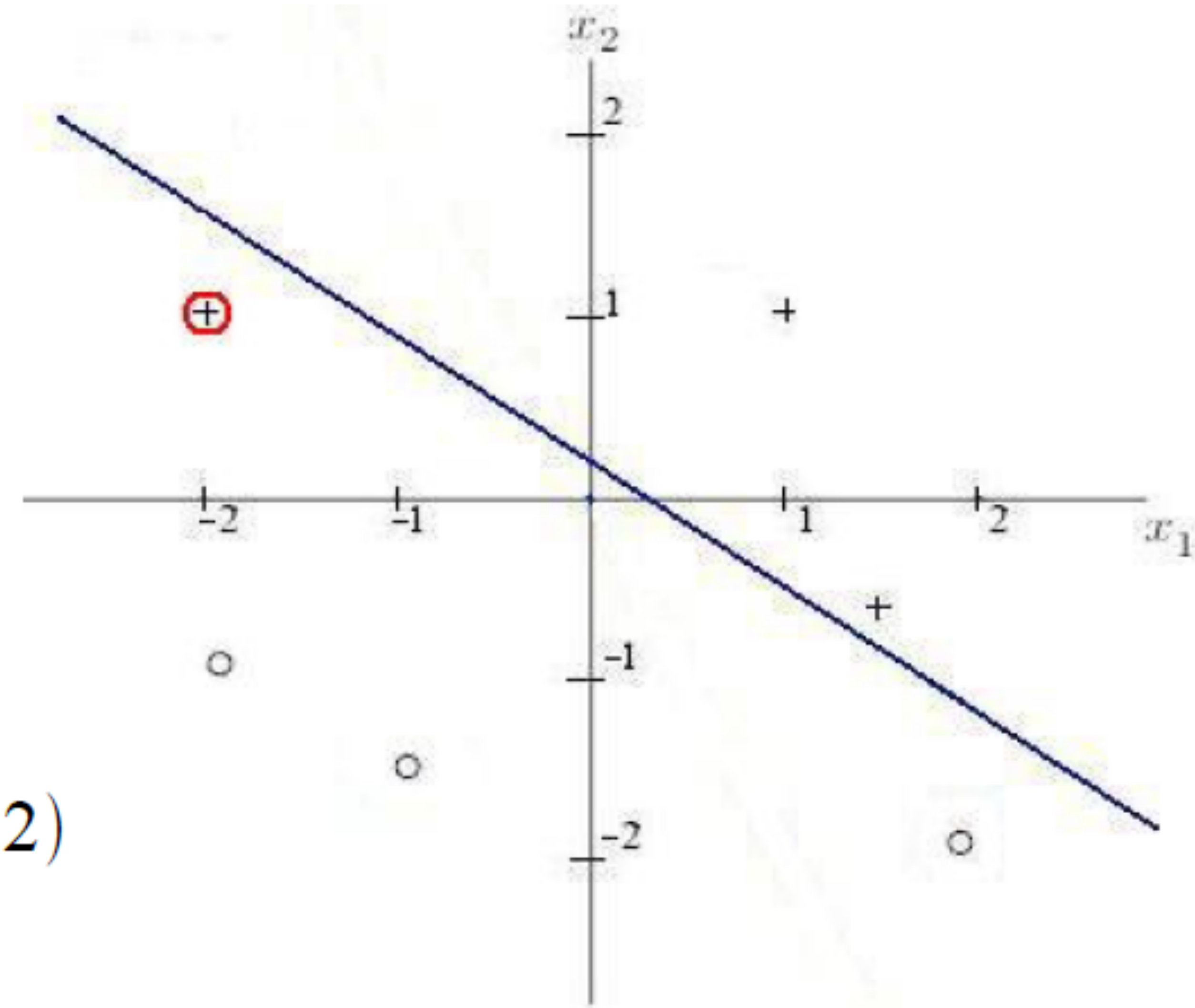
$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -2,\ x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$



Learning Example

$\eta = 0.2$

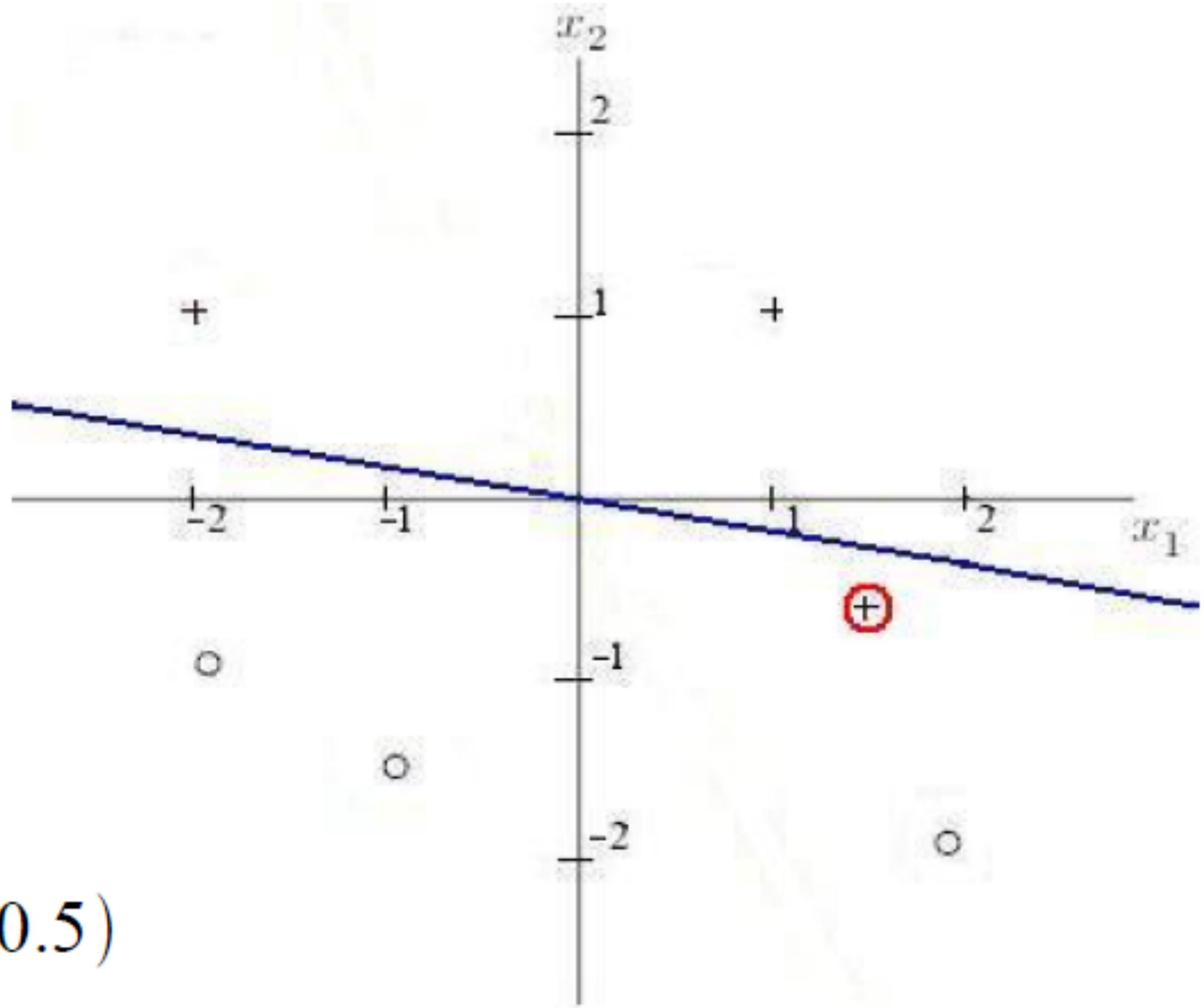
$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$x_1 = 1.5,\ x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

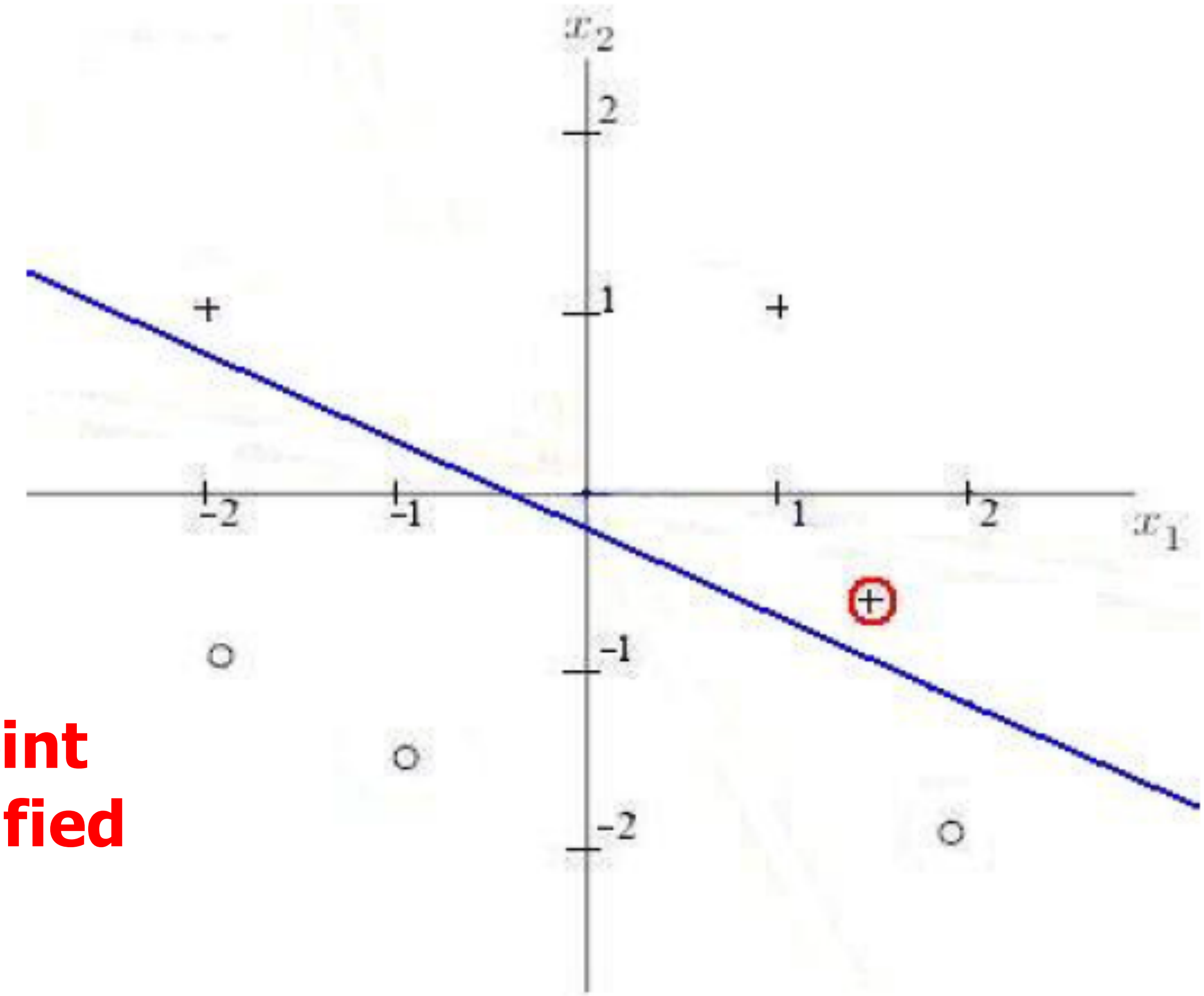
$$w_2 = w_2 + 0.2 * (-0.5)$$



Learning Example

$\eta = 0.2$

$$w = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$



Every training point is correctly classified

Perceptron algorithm

- Advantages
  - Simple and computationally efficient
  - Guaranteed to learn a linearly separable problem (convergence, global optimum)
- Limitations
  - Only linear separations
  - Only converges for linearly separable data
  - Not really "efficient with many features"

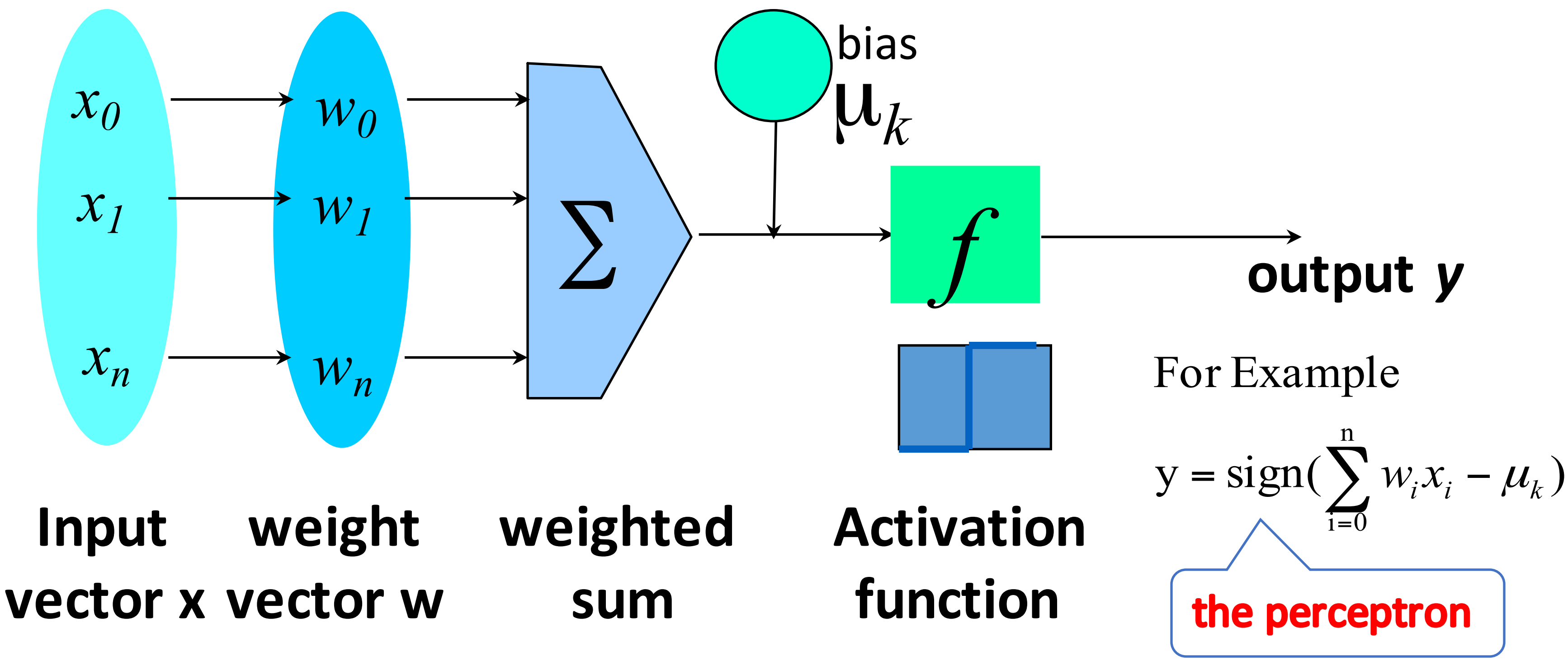
Speech recignication, Text translation

Classification by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a **weight** associated with it
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

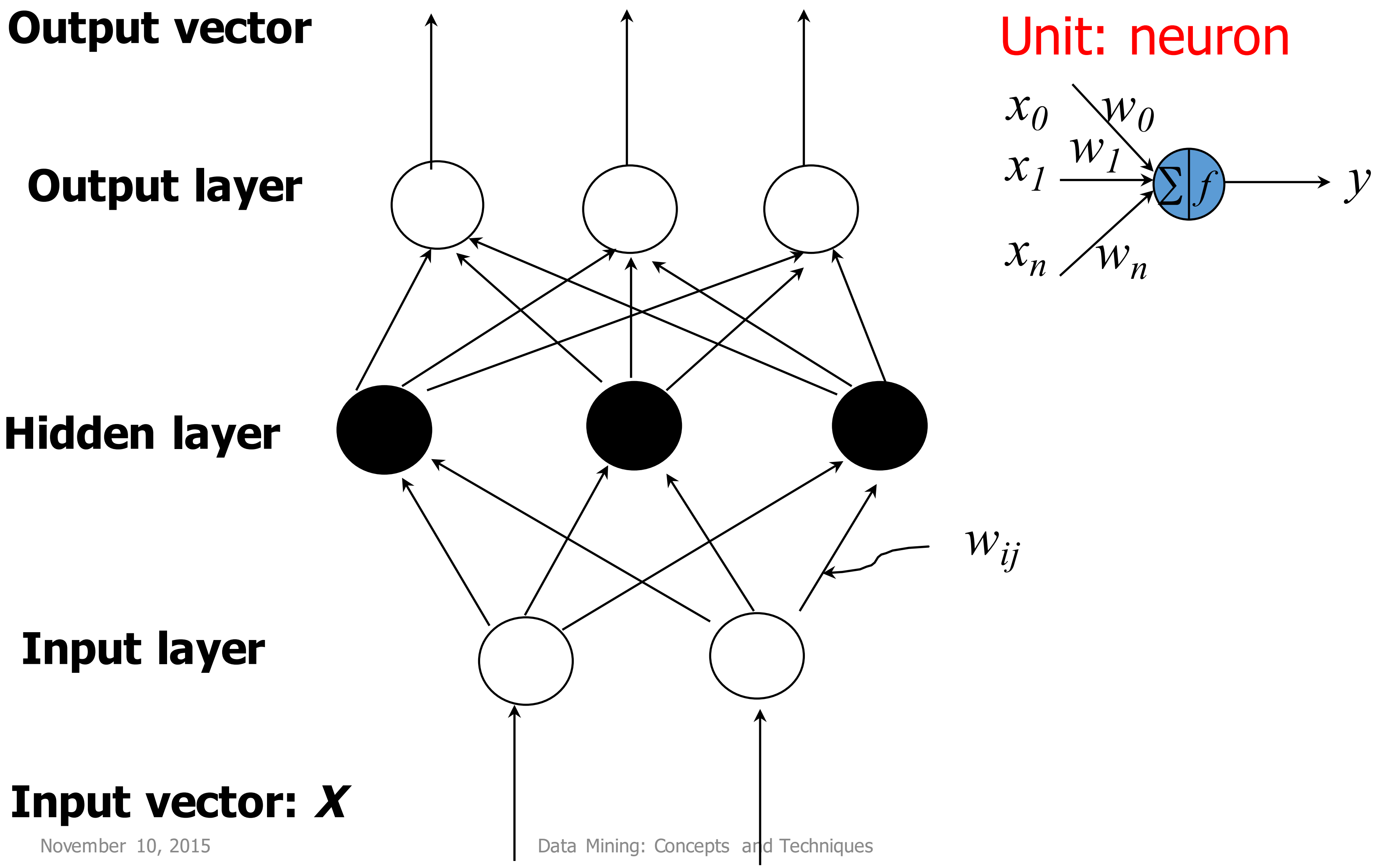
Each nueron is perceptron

Neuron: A Hidden/Output Layer Unit



- An n-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

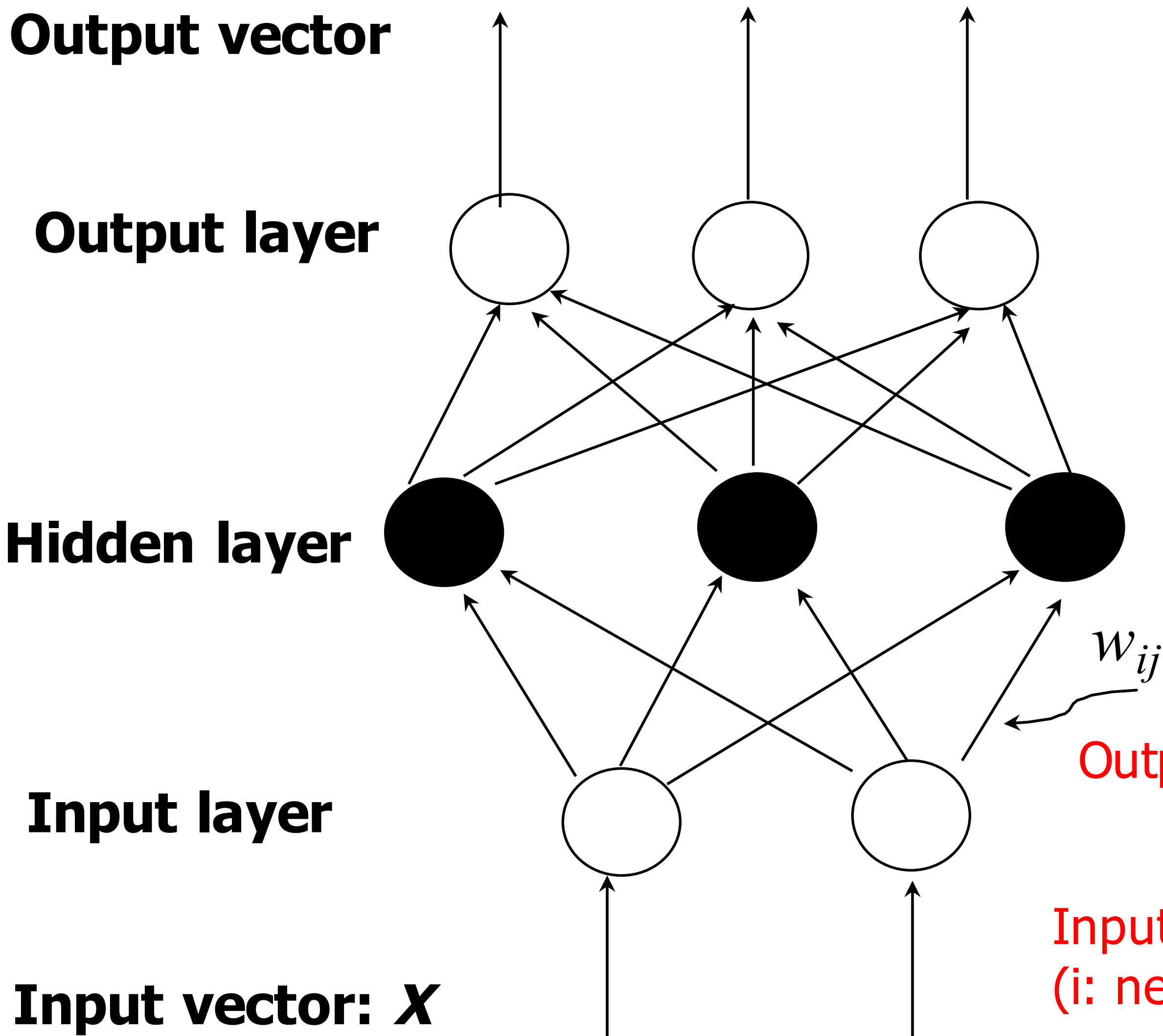
Building Network with Neurons



How A Multi-Layer Neural Network Works

- Input to the network: attributes measured for each training tuple
- Inputs are fed simultaneously into the units in the **input layer**
- Outputs from input layer are weighted and fed to neurons in the **hidden layer**
  - Though # of hidden layers can be arbitrary; usually only one
- The weighted outputs of the hidden layer are fed to neurons in **output layer**, which emits the network's prediction
  - E.g, $f(x)=1/(1+\exp(-x))$ as activation function for classification
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- Given enough hidden units and enough training samples, they can approximate any function to any precision

How A Multi-Layer Neural Network Works

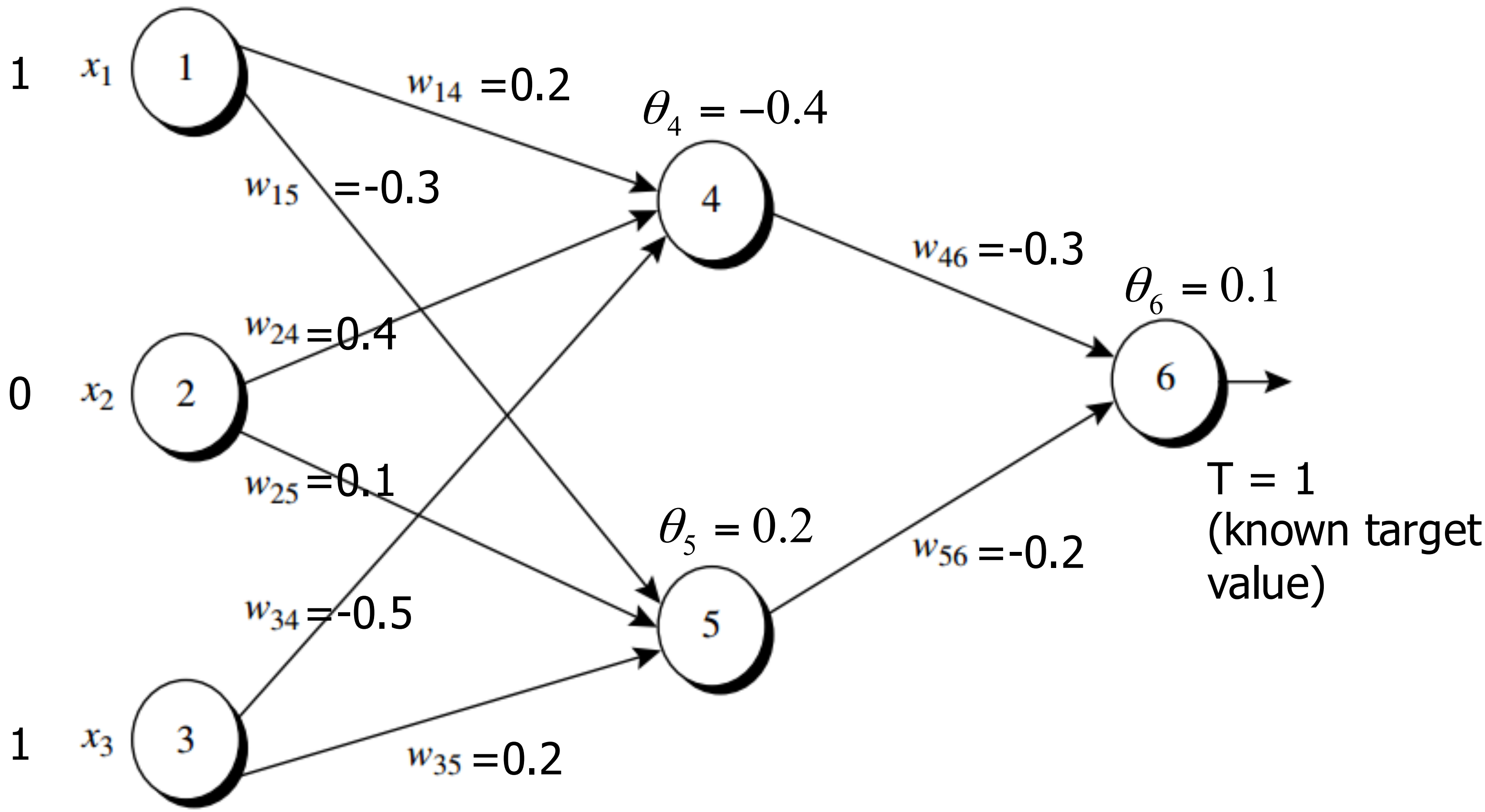


Output of unit j $$O_j = \frac{1}{1+e^{-I_j}}$$

Input of unit j $$I_j = \sum_i w_{ij} O_i + \theta_j$$

(i: neighbor of j in previous layer)

Example: feed forward



Example: feed forward

Net Input and Output Calculations

Unit, j	*Net Input*, I_j	*Output*, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Defining Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- **Input**: one input unit per attribute, each initialized to 0
- **Output**: if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a different network topology or a different set of initial weights

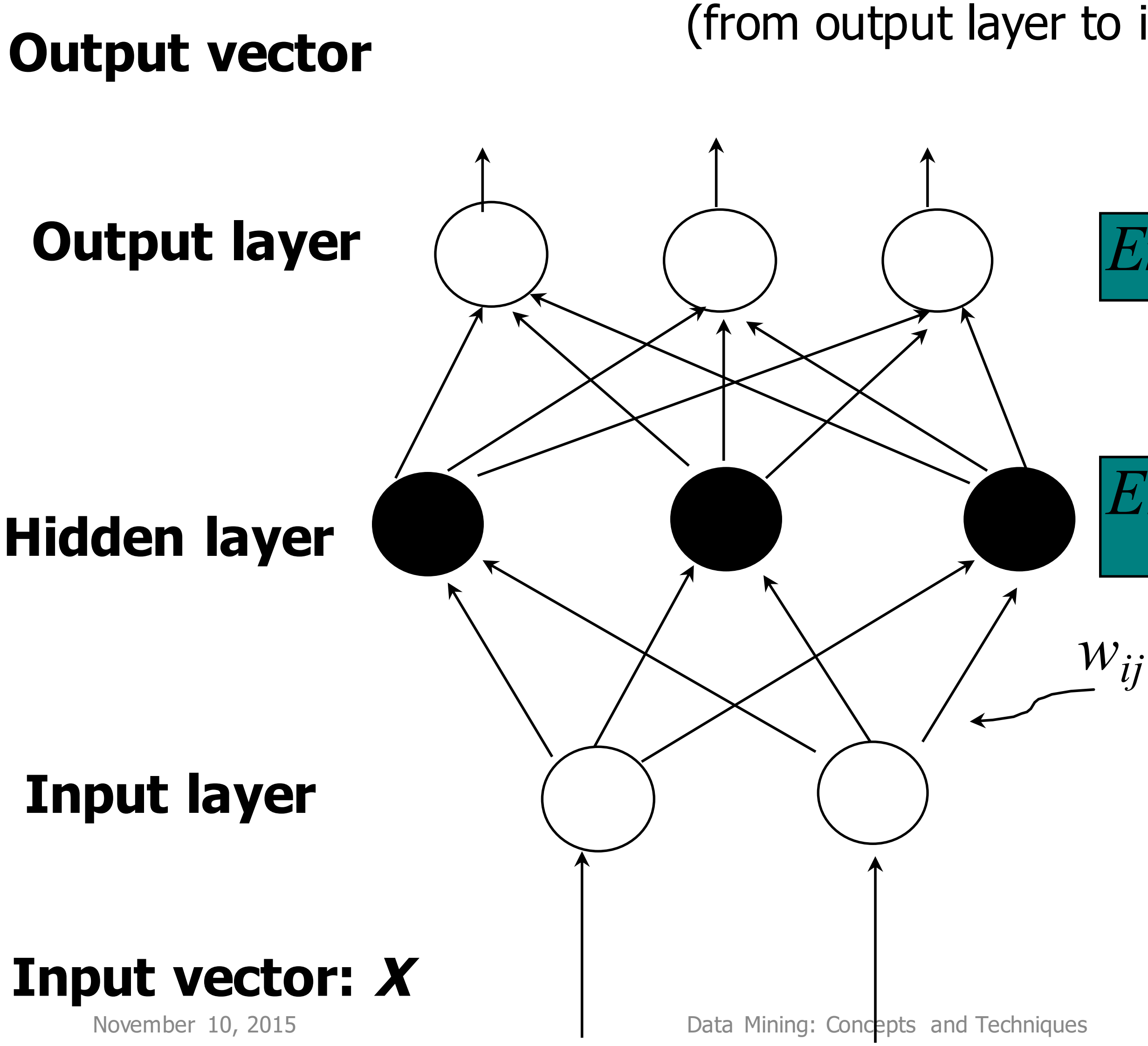
Learn the weight

Network Training: Backpropagation

- Iteratively process each training tuple & compare the network's prediction with the actual target value
- For each training tuple, the weights are modified to **minimize a loss function, e.g, the mean squared error** between the network's prediction and the actual target value
- Weight modifications are made "**backwards**": from the output layer, through each hidden layer, down to the first hidden layer, hence "**backpropagation**"
- Steps
  - **Initialize** weights to small random numbers, associated with biases
  - Propagate the inputs **forward** (by applying activation function)
  - Updating weights and biases **backward**
  - **Terminating** condition (when error is very small, etc.)

Network Training: Backpropagation

Err_j and update on *w_ij* are computed **backwards** (from output layer to input layer)



Error on **output layer unit** j
(*T_j* : known target value)

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

Error on **hidden layer unit** j
(*k*: neighbor of *j* in next layer)

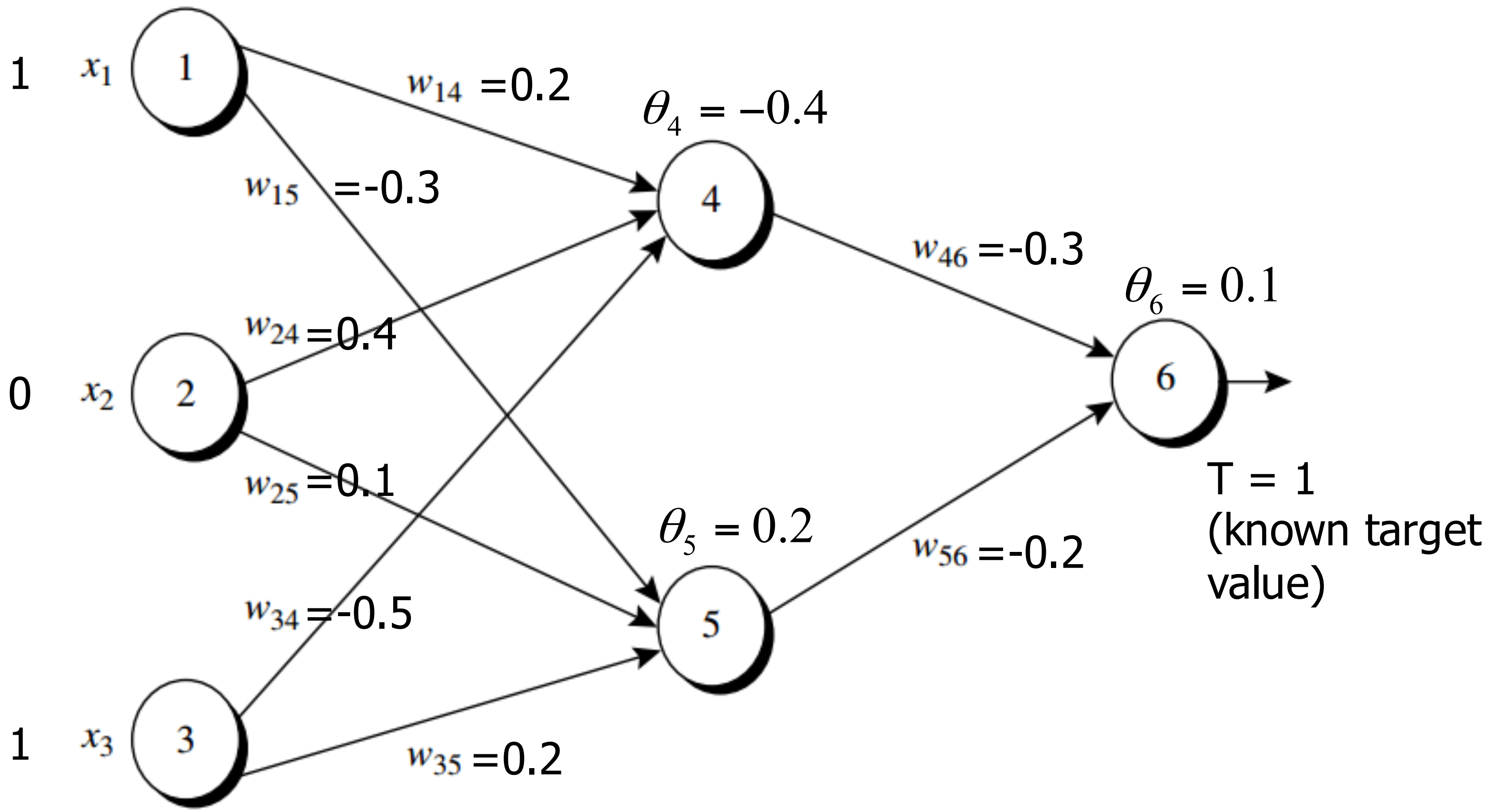
$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

update rule for (all) weights
(*l*: learning rate)

$$\theta_j = \theta_j + (l)Err_j$$

$$w_{ij} = w_{ij} + (l)Err_j O_i$$

Example: backpropagation



Example: backpropagation

Calculation of the Error at Each Node

Unit, j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Example: backpropagation

Calculations for Weight and Bias Updating

Weight or Bias	*New Value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Neural Network as a Classifier

- **Efficiency** of backpropagation: Each epoch (one pass through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights.
- **Weakness**
  - Long training time
  - Often need to determine some parameters empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- **Strength**
  - High tolerance to noisy data
  - Ability to classify untrained patterns (good generalization?)
  - Well-suited for continuous-valued inputs *and outputs*
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel

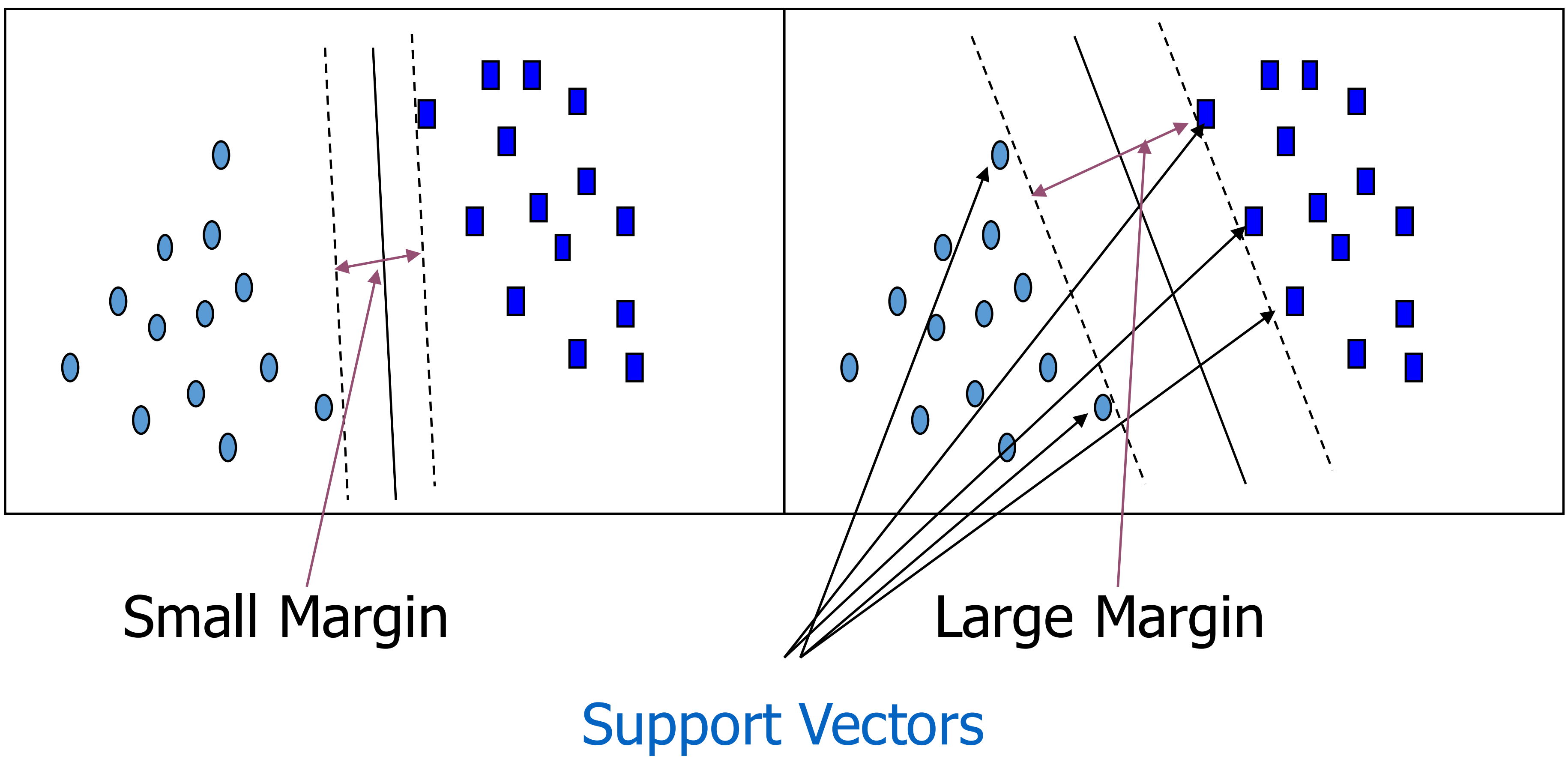
Chapter 9. Classification: Advanced Methods

- ~~Bayesian Belief Networks~~
- Perceptron, Backpropagation (Neural Network)
- Support Vector Machine
- ~~Classification by Using Frequent Patterns~~
- Lazy Learners (or Learning from Your Neighbors)
- Other Classification Methods
- Additional Topics Regarding Classification
- Summary

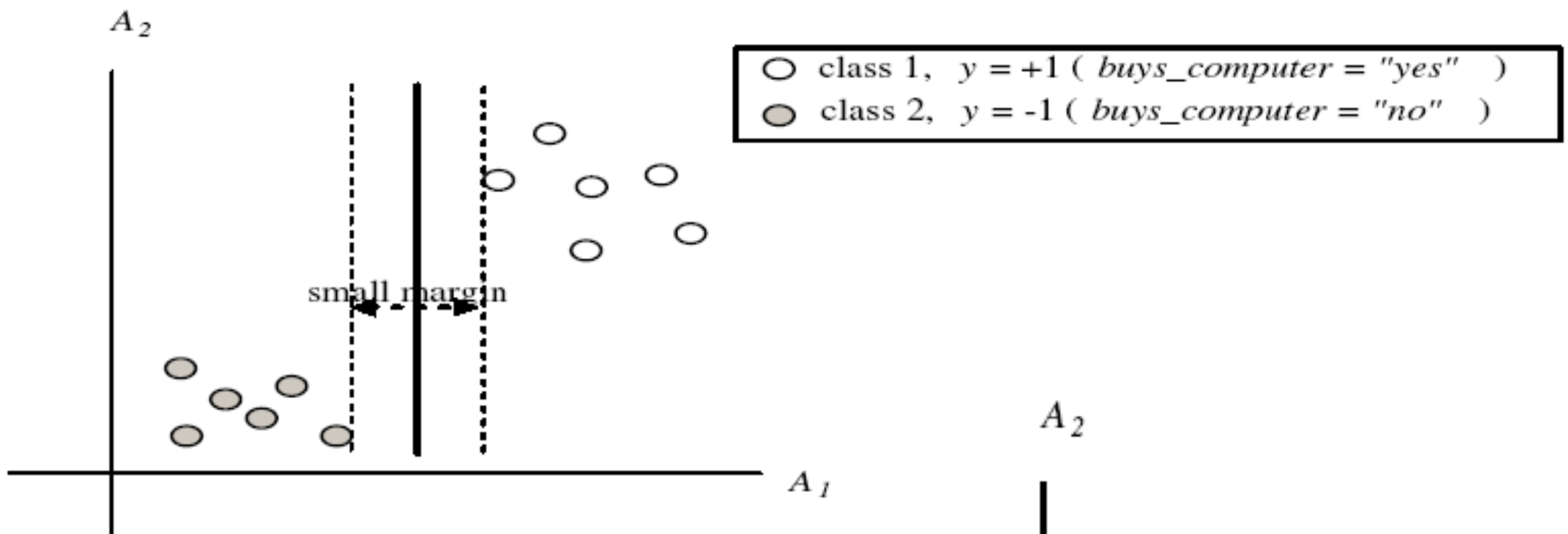
SVM—History and Applications

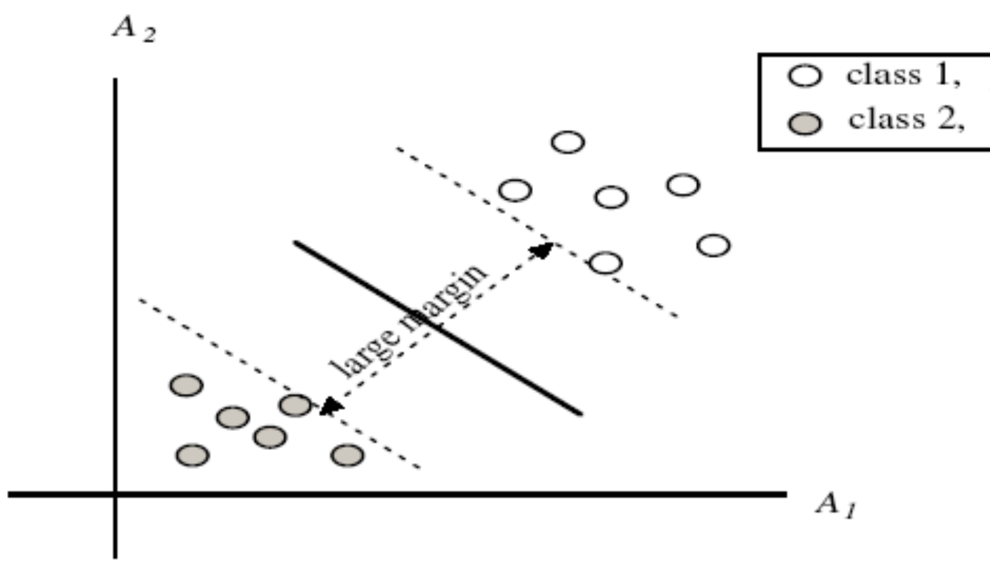
- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used for: classification and numeric prediction
- Applications:
  - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

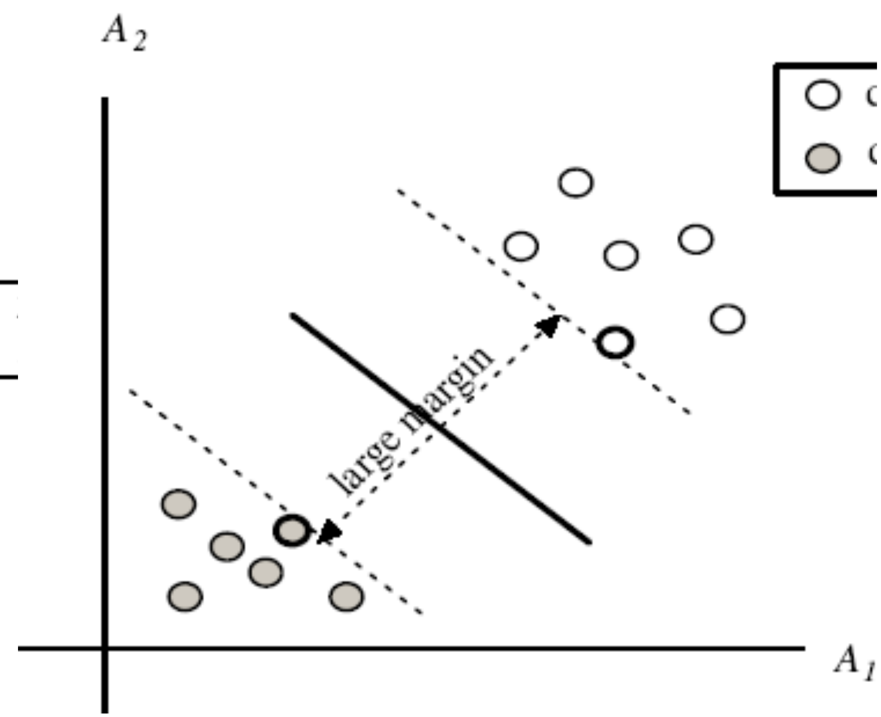
SVM—General Philosophy



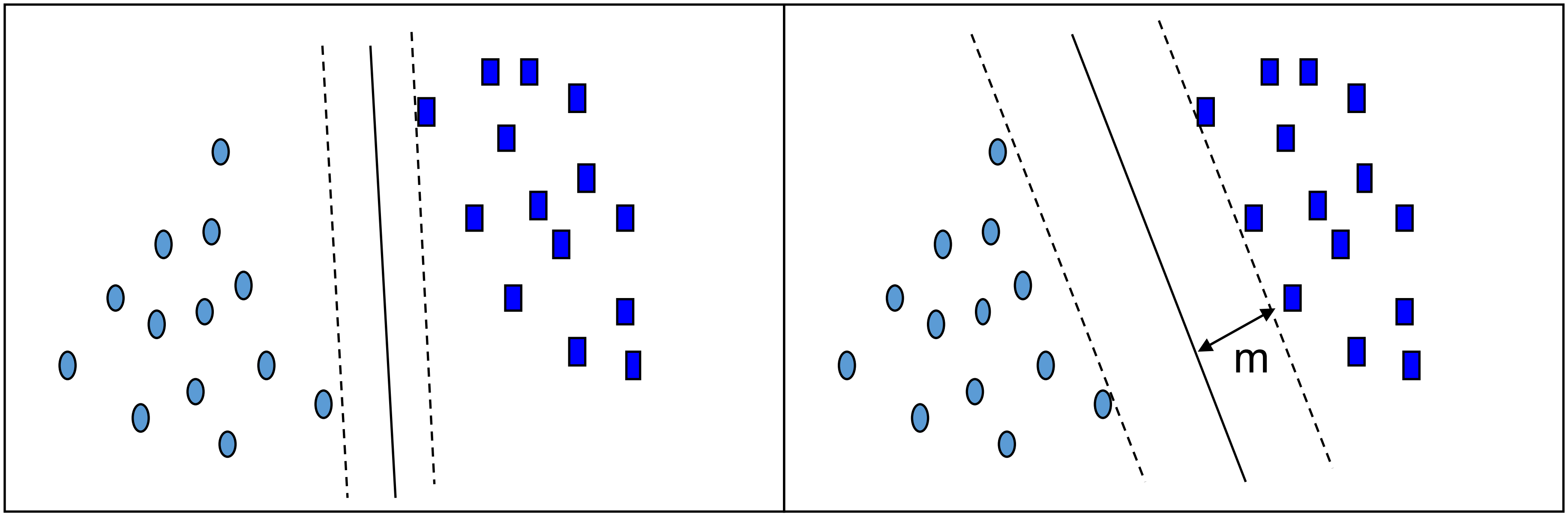
SVM—Margins and Support Vectors







SVM—When Data Is Linearly Separable



Let data D be ($\mathbf{X}_1$, y_1), …, ($\mathbf{X}_{|D|}$, $y_{|D|}$), where $\mathbf{X}_i$ is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

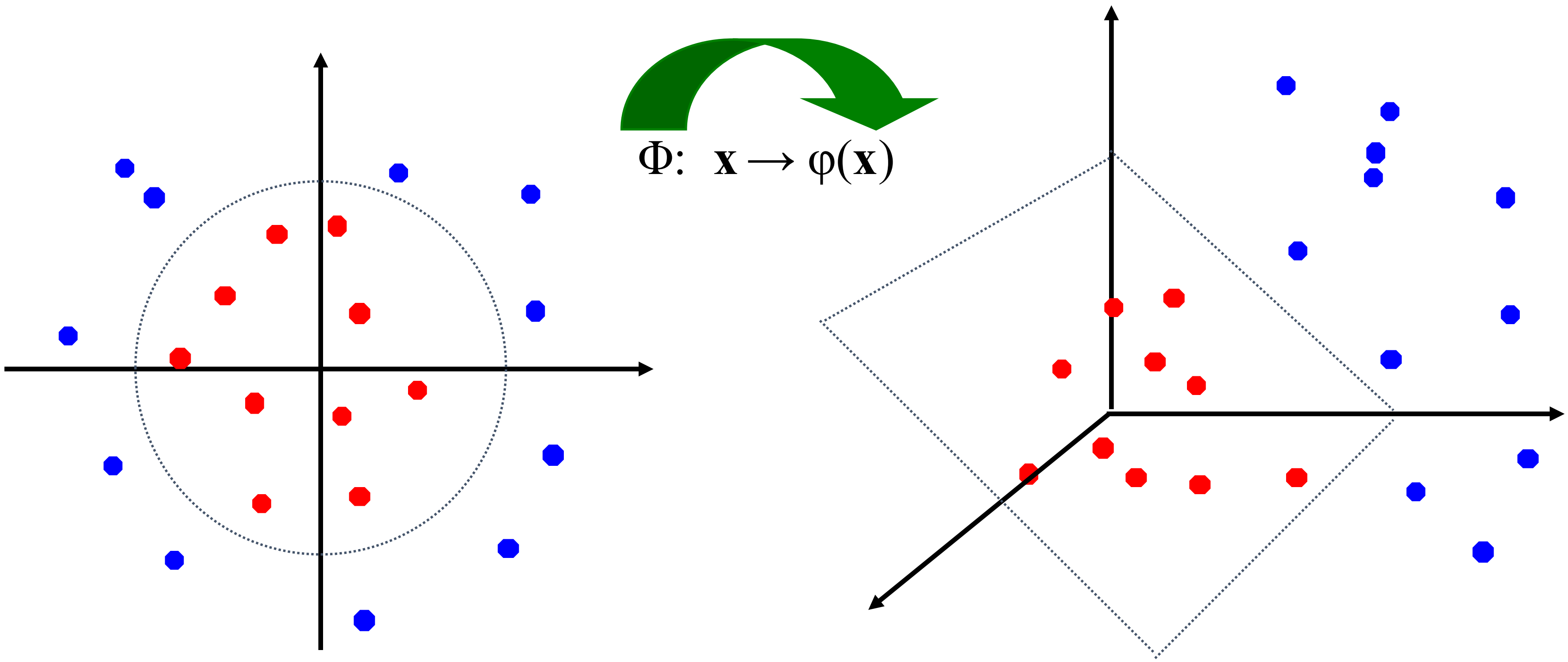
SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane** (MMH)

Why Is SVM Effective on High Dimensional Data?

- The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The **support vectors** are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



SVM Related Links

- SVM Website: http://www.kernel-machines.org/
- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - **SVM-light**: simpler but performance is not better than LIBSVM, support only binary classification and only in C
  - **SVM-torch**: another recent implementation also written in C

Chapter 9. Classification: Advanced Methods

- ~~Bayesian Belief Networks~~
- Perceptron, Backpropagation (Neural Network)
- Support Vector Machine
- ~~Classification by Using Frequent Patterns~~
- Lazy Learners (or Learning from Your Neighbors)
- ~~Other Classification Methods~~
- Additional Topics Regarding Classification
- Summary

Lazy vs. Eager Learning

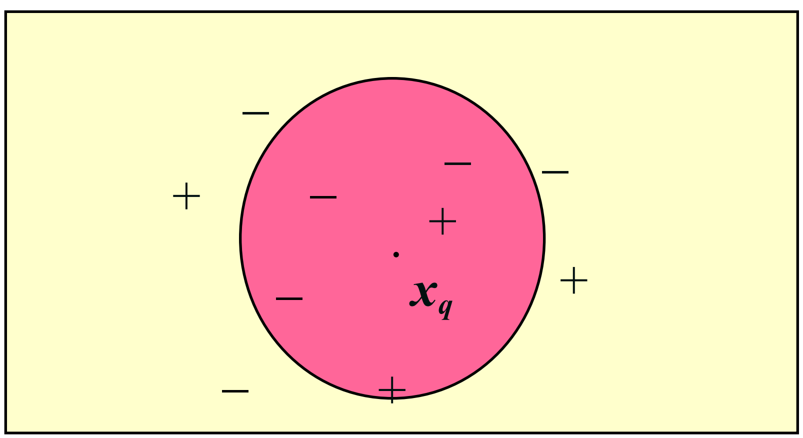
- <u>Lazy vs. eager learning</u>
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

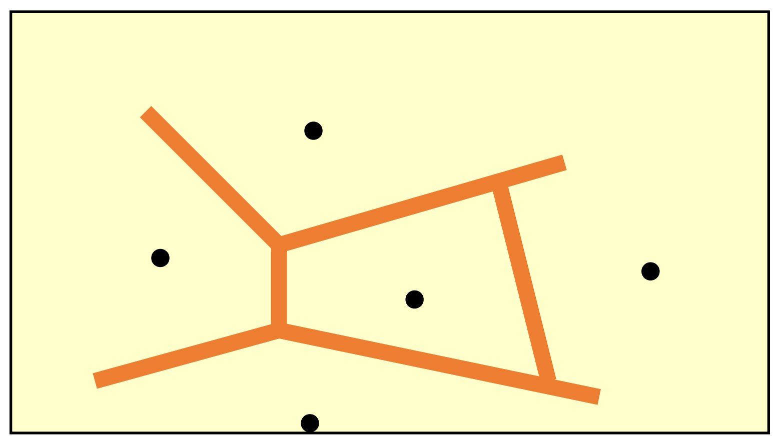
Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

The *k*-Nearest Neighbor Algorithm

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X_1}$, $\mathbf{X_2}$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the **most common value** among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





Discussion on the *k*-NN Algorithm

- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the **mean value** of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - **Weight** the contribution of each of the *k* neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by **averaging** *k*-nearest neighbors
- Curse of dimensionality: in high-dimensional space, nearest neighbors becomes far away– less representative of x_q

Chapter 9. Classification: Advanced Methods

- ~~Bayesian Belief Networks~~
- Perceptron, Backpropagation (Neural Network)
- Support Vector Machine
- ~~Classification by Using Frequent Patterns~~
- Lazy Learners (or Learning from Your Neighbors)
- ~~Other Classification Methods~~
- Additional Topics Regarding Classification
- Summary

Multiclass Classification

- Classification involving more than two classes (i.e., > 2 Classes)
- Method 1. **One-vs.-all** (OVA): Learn a classifier one at a time
  - Given m classes, train m classifiers: one for each class
  - Classifier j: treat tuples in class j as *positive* & all others as *negative*
  - To classify a tuple **X**, the set of classifiers vote as an ensemble
- Method 2. **All-vs.-all** (AVA): Learn a classifier for each pair of classes
  - Given m classes, construct m(m-1)/2 binary classifiers
  - A classifier is trained using tuples of the two classes
  - To classify a tuple **X**, each classifier votes. X is assigned to the class with maximal vote
- Comparison
  - All-vs.-all tends to be superior to one-vs.-all
  - Problem: Binary classifier is sensitive to errors, and errors affect vote count

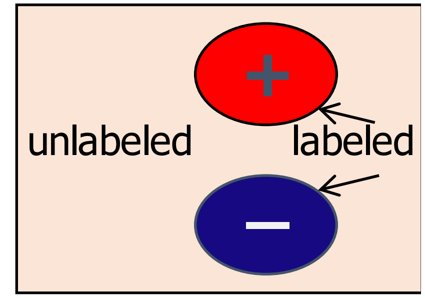
Error-Correcting Codes for Multiclass Classification

- Originally designed to correct errors during data transmission for communication tasks by exploring data redundancy
- Example
  - A 7-bit codeword associated with classes 1-4

Class	Error-Corr. Codeword						
C_1	1	1	1	1	1	1	1
C_2	0	0	0	0	1	1	1
C_3	0	0	1	1	0	0	1
C_4	0	1	0	1	0	1	0

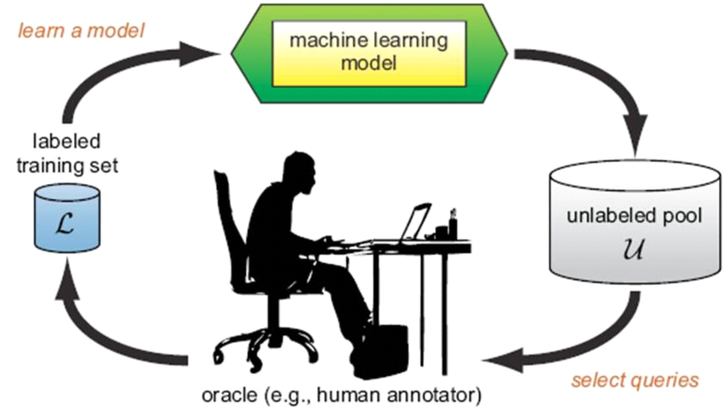
  - Given a unknown tuple **X**, the 7-trained classifiers output: 0001010
  - Hamming distance: # of different bits between two codewords
  - $H(\mathbf{X}, C_1) = 5$, by checking # of bits between [1111111] & [0001010]
  - $H(\mathbf{X}, C_2) = 3$, $H(\mathbf{X}, C_3) = 3$, $H(\mathbf{X}, C_4) = 1$, thus C_4 as the label for **X**
- Error-correcting codes can correct up to $(h - 1)/2$ 1-bit error, where h is the minimum Hamming distance between any two codewords
- If we use 1-bit per class, it is equiv. to one-vs.-all approach, the code are insufficient to self-correct
- When selecting error-correcting codes, there should be good row-wise and col.-wise separation between the codewords

Semi-Supervised Classification



- Semi-supervised: Uses labeled and unlabeled data to build a classifier
- Self-training:
  - Build a classifier using the labeled data
  - Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
  - Repeat the above process
  - Adv: easy to understand; disadv: may reinforce errors
- Co-training: Use two or more classifiers to teach each other
  - Each learner uses a mutually independent set of features of each tuple to train a good classifier, say f_1
  - Then f_1 and f_2 are used to predict the class label for unlabeled data X
  - Teach each other: The tuple having the most confident prediction from f_1 is added to the set of labeled data for f_2, & vice versa
- Other methods, e.g., joint probability distribution of features and labels

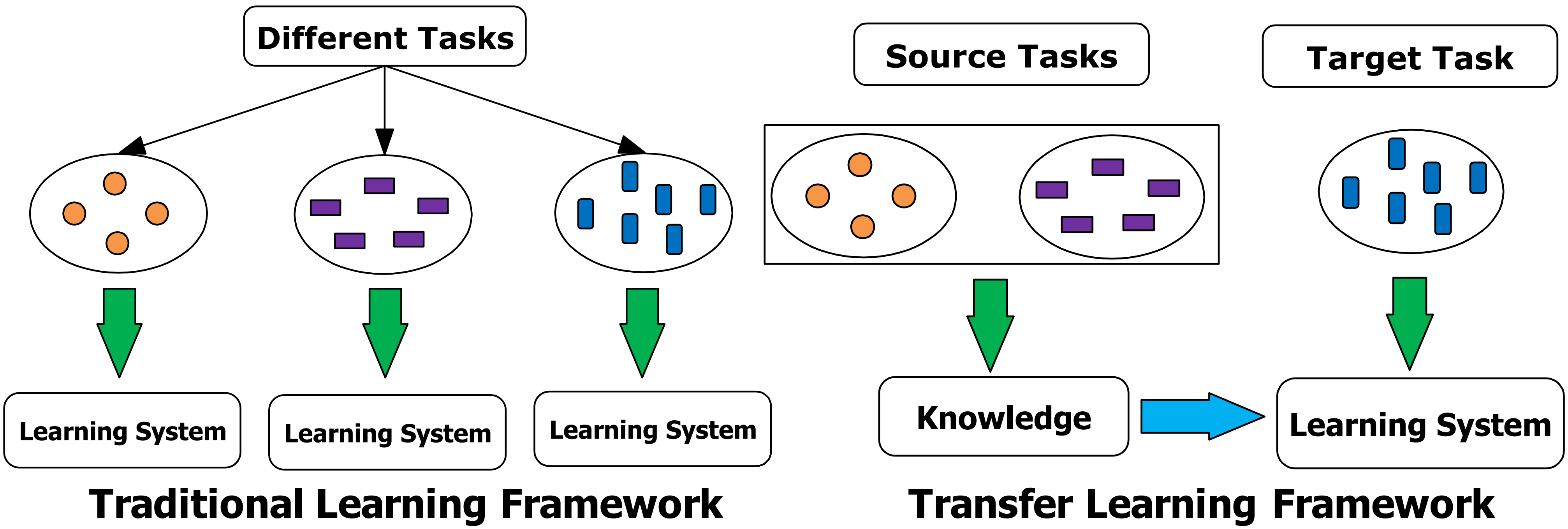
Active Learning



- Class labels are expensive to obtain
- Active learner: query human (oracle) for labels
- Pool-based approach: Uses a pool of unlabeled data
  - L: a small subset of D is labeled, U: a pool of unlabeled data in D
  - Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)
  - The newly labeled samples are added to L, and learn a model
  - Goal: Achieve high accuracy using as few labeled data as possible
- Evaluated using *learning curves*: Accuracy as a function of the number of instances queried (# of tuples to be queried should be small)
- Research issue: How to choose the data tuples to be queried?
  - Uncertainty sampling: choose the least certain ones
  - Reduce *version space*, the subset of hypotheses consistent w. the training data
  - Reduce expected entropy over U: Find the greatest reduction in the total number of incorrect predictions

Transfer Learning: Conceptual Framework

- Transfer learning: Extract knowledge from one or more source tasks and apply the knowledge to a target task
- Traditional learning: Build a new classifier for each new task
- Transfer learning: Build new classifier by applying existing knowledge learned from source tasks



Transfer Learning: Methods and Applications

- Applications: Especially useful when data is outdated or distribution changes, e.g., Web document classification, e-mail spam filtering
- *Instance-based transfer learning*: Reweight some of the data from source tasks and use it to learn the target task
- TrAdaBoost (Transfer AdaBoost)
  - Assume source and target data each described by the same set of attributes (features) & class labels, but rather diff. distributions
  - Require only labeling a small amount of target data
  - Use source data in training: When a source tuple is misclassified, reduce the weight of such tupels so that they will have less effect on the subsequent classifier
- Research issues
  - Negative transfer: When it performs worse than no transfer at all
  - Heterogeneous transfer learning: Transfer knowledge from different feature space or multiple source domains
  - Large-scale transfer learning

Chapter 9. Classification: Advanced Methods

- Bayesian Belief Networks
- Classification by Backpropagation
- Support Vector Machines
- Classification by Using Frequent Patterns
- Lazy Learners (or Learning from Your Neighbors)
- Other Classification Methods
- Additional Topics Regarding Classification
- Summary

Summary

- Effective and advanced classification methods
  - Bayesian belief network (probabilistic networks)
  - Perceptron and neural networks
  - Support Vector Machine (SVM)
  - Pattern-based classification
  - Other classification methods: lazy learners (KNN, case-based reasoning), genetic algorithms, rough set and fuzzy set approaches
- Additional Topics on Classification
  - Multiclass classification
  - Semi-supervised classification
  - Active learning
  - Transfer learning