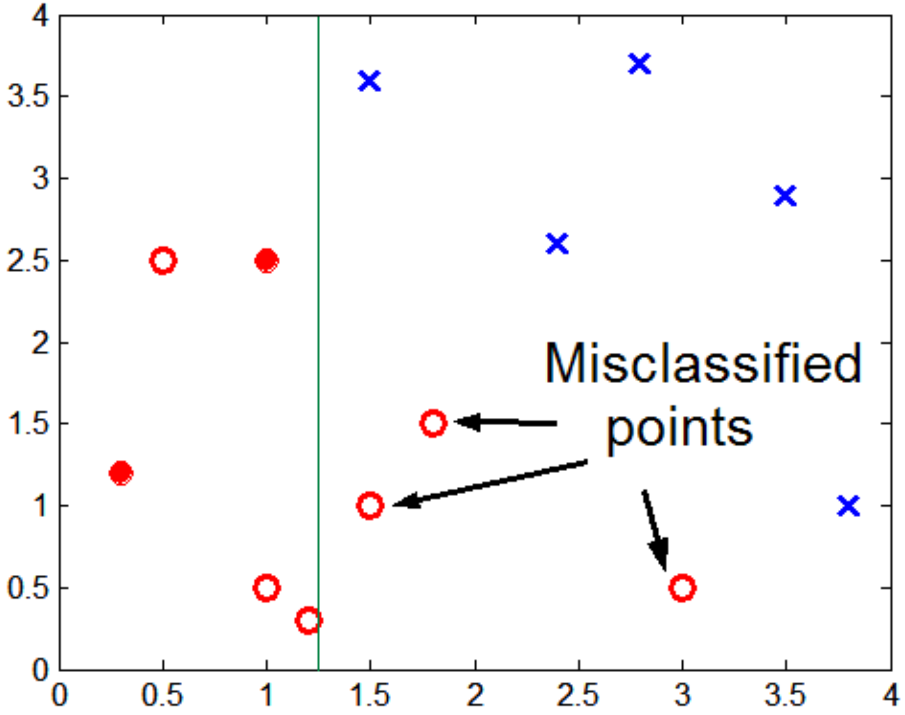
CSE 5243 INTRO. TO DATA MINING

Classification (Basic Concepts & Advanced Methods)

Yu Su, CSE@The Ohio State University

Slides adapted from UIUC CS412 by Prof. Jiawei Han and OSU CSE5243 by Prof. Huan Sun

Overfitting due to Insufficient Examples



Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Classification: Basic Concepts

- □ Classification: Basic Concepts
- □ Decision Tree Induction
- □ Model Evaluation and Selection
- □ Practical Issues of Classification
- □ Bayes Classification Methods
- □ Techniques to Improve Classification Accuracy: Ensemble Methods

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X})=\frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}=P(\mathbf{X}|H)\times P(H)/P(\mathbf{X})$$

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., *posterior probability*): the probability that the hypothesis holds given the observed data sample **X**

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed

Bayes' Theorem: Basics

- Bayes' Theorem:
  - Let **X** be a data sample ("*evidence*"): class label is unknown
  - Let H be a *hypothesis* that X belongs to class C
  - Classification is to determine P(H|**X**), (i.e., ***posterior probability***): the probability that the hypothesis holds given the observed data sample **X**

$$P(H\,|\,\mathbf{X}) = \frac{P(\mathbf{X}\,|\,H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}\,|\,H) \times P(H) / P(\mathbf{X})$$

  - P(H) (*prior probability*): the initial probability
    - E.g., **X** will buy computer, regardless of age, income, …
  - P(**X**): probability that sample data is observed
  - P(**X**|H) (likelihood): the probability of observing the sample **X**, given that the hypothesis holds
    - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Prediction Based on Bayes' Theorem

- □ Given training data **X**, ***posterior probability** of a hypothesis* H, P(H|**X**), follows the Bayes' theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- □ Informally, this can be viewed as

  posterior = likelihood x prior/evidence

- □ Predicts **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- □ Practical difficulty: It requires initial knowledge of many probabilities, involving significant computational cost

Classification Is to Derive the Maximum A Posteriori

- □ Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-dimensional attribute vector $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
- □ Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- □ Classification is to derive the maximum a posteriori, i.e., the maximal $P(C_i|\mathbf{X})$

Classification Is to Derive the Maximum Posteriori

- □ Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-dimensional attribute vector **X** = $(x_1, x_2, \ldots, x_n)$
- □ Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- □ Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- □ This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- □ Since P(X) is constant for all classes, only

$$\boxed{P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)}$$

needs to be maximized

Naïve Bayes Classifier (why Naïve? :-)

□ A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

□ This greatly reduces the computation cost: Only counts the per-class distributions

Naïve Bayes Classifier

- □ A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- □ This greatly reduces the computation cost: Only counts the per-class distributions

- □ If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i, D}|$ (# of tuples in C_i)

Naïve Bayes Classifier

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with sample mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayes Classifier

- A simplifying assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

and $P(x_k|C_i)$ is

$$P(x_k | C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Here, mean μ and standard deviation σ are estimated based on the values of attribute A_k for training tuples of class $C_{i.}$

Naïve Bayes Classifier: Training Dataset

Class:

C1:buys_computer = ‘yes’

C2:buys_computer = ‘no’

Data to be classified:

X = (age <=30, Income = medium,

Student = yes, Credit_rating = Fair)

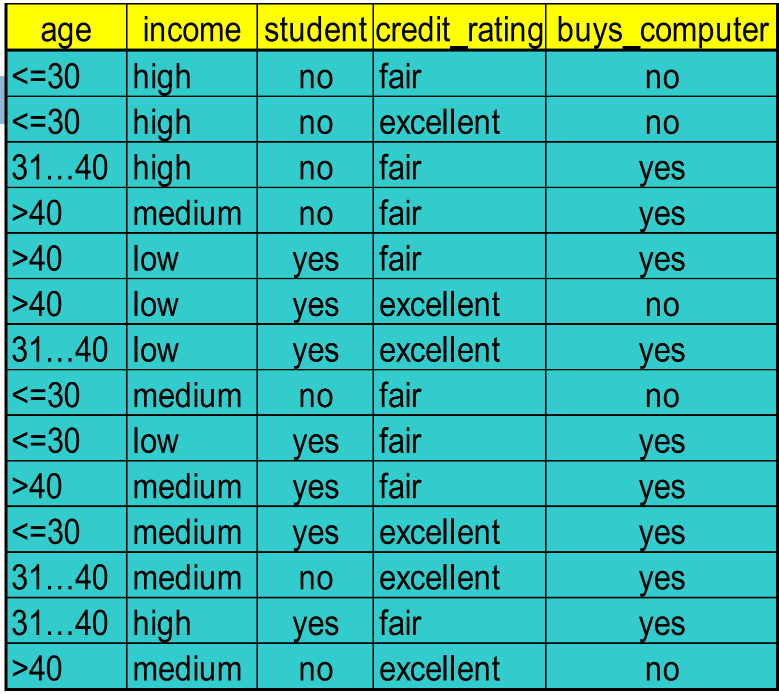
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

- Prior probability $P(C_i)$:

  P(buys_computer = “yes”) = 9/14 = 0.643

  P(buys_computer = “no”) = 5/14= 0.357



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

- □ $P(C_i)$: P(buys_computer = “yes”) = 9/14 = 0.643

  P(buys_computer = “no”) = 5/14= 0.357

- □ Compute $P(X|C_i)$ for each class, where,

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

According to “the naïve assumption”, first get:

P(age = “<=30”|buys_computer = “yes”) = 2/9 = 0.222

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

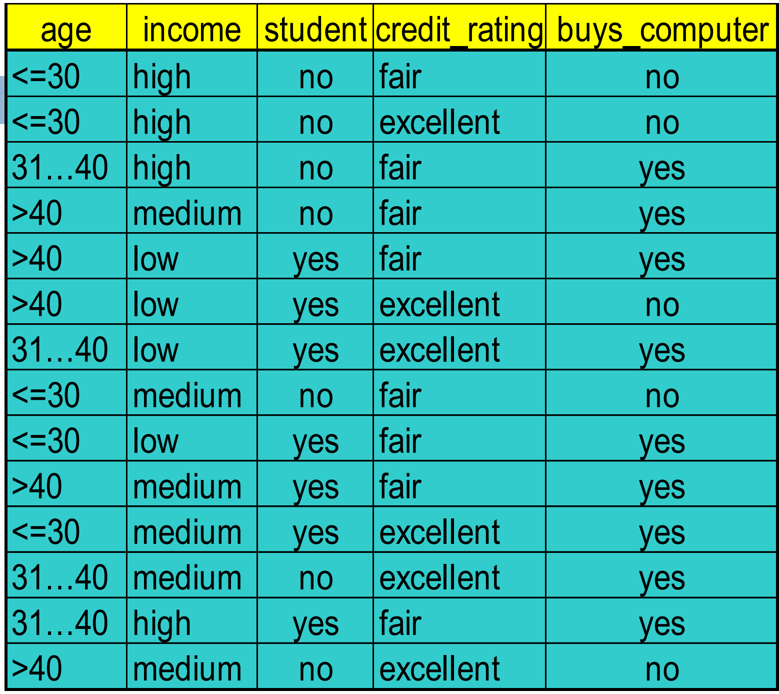
- □ $P(C_i)$: P(buys_computer = “yes”) = 9/14 = 0.643
  P(buys_computer = “no”) = 5/14= 0.357

- □ Compute $P(X|C_i)$ for each class, where,

X = (age <=30, Income = medium, Student = yes, Credit_rating = Fair)

According to “the naïve assumption”, first get:

P(age = “<=30”|buys_computer = “yes”) = 2/9 = 0.222
P(age = “<= 30”|buys_computer = “no”) = 3/5 = 0.6
P(income = “medium” | buys_computer = “yes”) = 4/9 = 0.444
P(income = “medium” | buys_computer = “no”) = 2/5 = 0.4
P(student = “yes” | buys_computer = “yes) = 6/9 = 0.667
P(student = “yes” | buys_computer = “no”) = 1/5 = 0.2
P(credit_rating = “fair” | buys_computer = “yes”) = 6/9 = 0.667
P(credit_rating = “fair” | buys_computer = “no”) = 2/5 = 0.4



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- □ P(C_i): P(buys_computer = “yes”) = 9/14 = 0.643

  P(buys_computer = “no”) = 5/14= 0.357

- □ Compute P(Xi|C_i) for each class

  P(age = “<=30”|buys_computer = “yes”) = 2/9 = 0.222

  P(age = “<= 30”|buys_computer = “no”) = 3/5 = 0.6

  P(income = “medium” | buys_computer = “yes”) = 4/9 = 0.444

  P(income = “medium” | buys_computer = “no”) = 2/5 = 0.4

  P(student = “yes” | buys_computer = “yes) = 6/9 = 0.667

  P(student = “yes” | buys_computer = “no”) = 1/5 = 0.2

  P(credit_rating = “fair” | buys_computer = “yes”) = 6/9 = 0.667

  P(credit_rating = “fair” | buys_computer = “no”) = 2/5 = 0.4

- □ **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

P(X|C_i) : P(X|buys_computer = “yes”) = P(age = “<=30”|buys_computer = “yes”) x P(income = “medium” | buys_computer = “yes”) x P(student = “yes” | buys_computer = “yes) x P(credit_rating = “fair” | buys_computer = “yes”) = 0.044

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

□ P(C_i): P(buys_computer = "yes") = 9/14 = 0.643
P(buys_computer = "no") = 5/14= 0.357

□ Compute P(Xi|C_i) for each class
P(age = "<=30"|buys_computer = "yes") = 2/9 = 0.222
P(age = "<= 30"|buys_computer = "no") = 3/5 = 0.6
P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444
P(income = "medium" | buys_computer = "no") = 2/5 = 0.4
P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667
P(student = "yes" | buys_computer = "no") = 1/5 = 0.2
P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667
P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

□ **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

P(X|C_i) : P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044
P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

P(X|C_i)*P(C_i) : P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028
P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Take into account the prior probabilities

Naïve Bayes Classifier: An Example

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

- $P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643

  P(buys_computer = "no") = 5/14= 0.357

- Compute $P(X_i|C_i)$ for each class

  P(age = "<=30"|buys_computer = "yes") = 2/9 = 0.222

  P(age = "<= 30"|buys_computer = "no") = 3/5 = 0.6

  P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

  P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

  P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

  P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

  P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

  P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = 0.028

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Since Red > Blue here, X belongs to class ("buys_computer = yes")

Avoiding the Zero-Probability Problem

- Naïve Bayesian prediction requires each conditional prob. be **non-zero**. Otherwise, the predicted prob. will be zero

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i)$$

- Ex. Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)

- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*

    Prob(income = low) = 1/1003

    Prob(income = medium) = 991/1003

    Prob(income = high) = 11/1003
  - *Assumption*: dataset is large enough such that adding 1 would only make a negligible difference in the estimated probability values
  - The "corrected" prob. estimates are close to their "uncorrected" counterparts

Naïve Bayes Classifier

- □ If A_k is continuous-valued, $P(x_k|C_i)$ is usually computed based on Gaussian distribution with a mean μ and standard deviation σ

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (1)$$

and $P(x_k|C_i)$ is

$$P(x_k | C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Here, mean μ and standard deviation σ are estimated based on the values of attribute A_k for training tuples of class $C_{i.}$

Ex. Let X = (35, \$40K), where A1 and A2 are the attribute *age* and *income, class label is buys_computer.*

To calculate P(age = 35 | buys_computer = yes)

1. Estimate the mean and standard deviation of the age attribute for customers in D who buy a computer. Let us say μ = 38 and σ =12.

2. calculate the probability with equation (1).

Naïve Bayes Classifier: Comments

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases

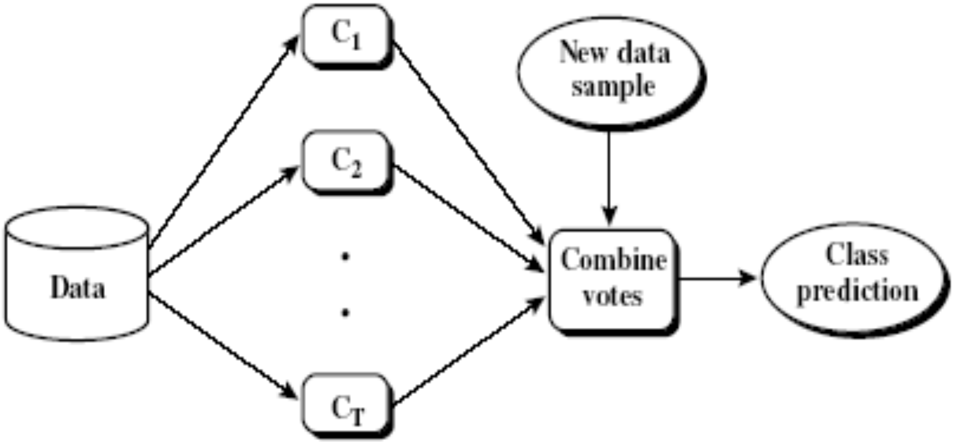
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy
  - Practically, dependencies exist among variables
    - E.g., hospitals: patients: Profile: age, family history, etc.
      Symptoms: fever, cough etc., Disease: lung cancer, diabetes, etc.
    - Dependencies among these cannot be modeled by Naïve Bayes Classifier

- How to deal with these dependencies? Bayesian Belief Networks (Chapter 9 in Han et al.)

Classification: Basic Concepts

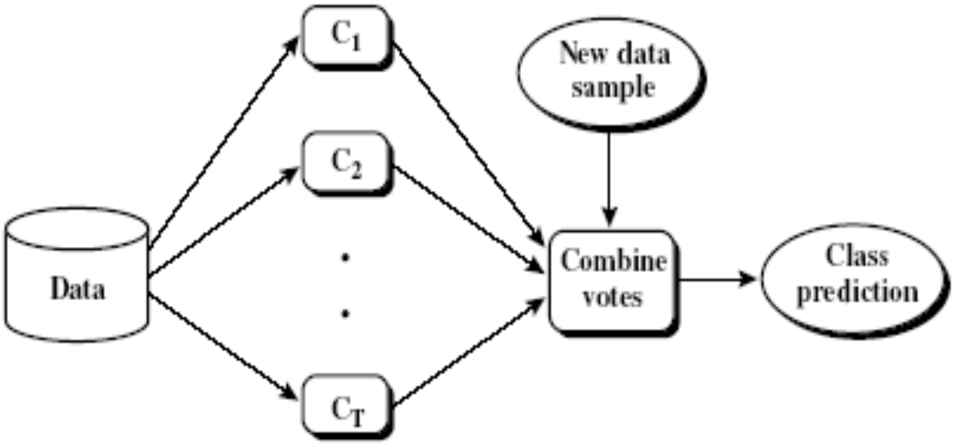
- Classification: Basic Concepts
- Decision Tree Induction
- Model Evaluation and Selection
- Practical Issues of Classification
- Bayes Classification Methods
- Techniques to Improve Classification Accuracy: Ensemble Methods

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*

Ensemble Methods: Increasing the Accuracy



- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*

- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Random forests: Imagine that each of the classifiers in the ensemble is a decision tree classifier so that the collection of classifiers is a "forest"

Bagging: Boostrap Aggregation

- Analogy: Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Regression: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy: Proved improved accuracy in prediction
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost (Freund and Schapire, 1997)

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, o.w. it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is

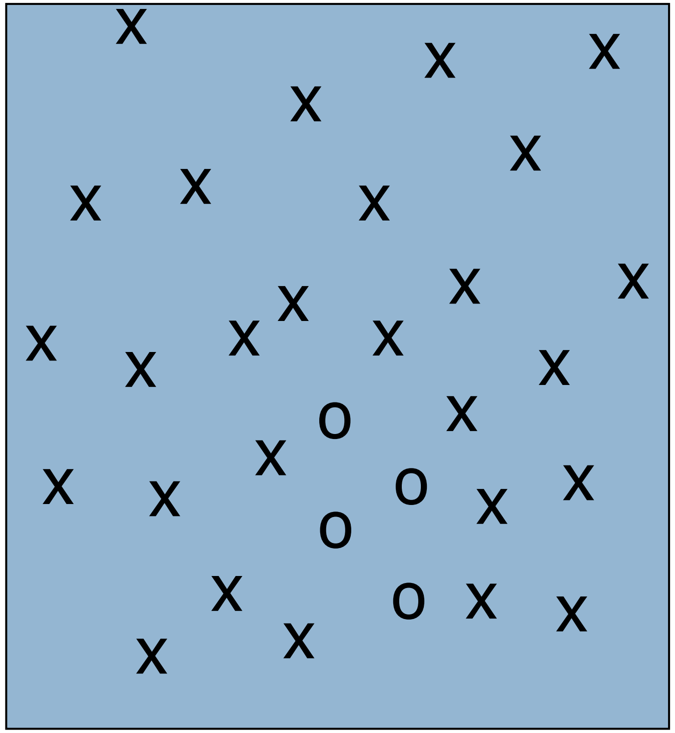
$$\log \frac{1 - error(M_i)}{error(M_i)}$$

Random Forest (Breiman 2001)

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a random selection of attributes at each node to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*): Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but more robust to errors and outliers
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

Classification of Class-Imbalanced Data Sets

- Class-imbalance problem: Rare positive example but numerous negative ones, e.g., medical diagnosis, fraud, oil-spill, fault, etc.
- Traditional methods assume a balanced distribution of classes and equal error costs: not suitable for class-imbalanced data



- Typical methods in two-class classification:
  - **Oversampling**: re-sampling of data from positive class
  - **Under-sampling**: randomly eliminate tuples from negative class
  - **Threshold-moving**: move the decision threshold, t, so that the rare class tuples are easier to classify, and hence, less chance of costly false negative errors
  - **Ensemble techniques**: Ensemble multiple classifiers

- Still difficult for class imbalance problem on multiclass tasks

Classification: Advanced Methods

- Lazy Learners and K-Nearest Neighbors
- Neural Networks
- Support Vector Machines
- Additional Topics: Semi-Supervised Methods, Active Learning, etc.
- Summary

Lazy vs. Eager Learning

- Lazy vs. eager learning
  - **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 previously 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.

  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

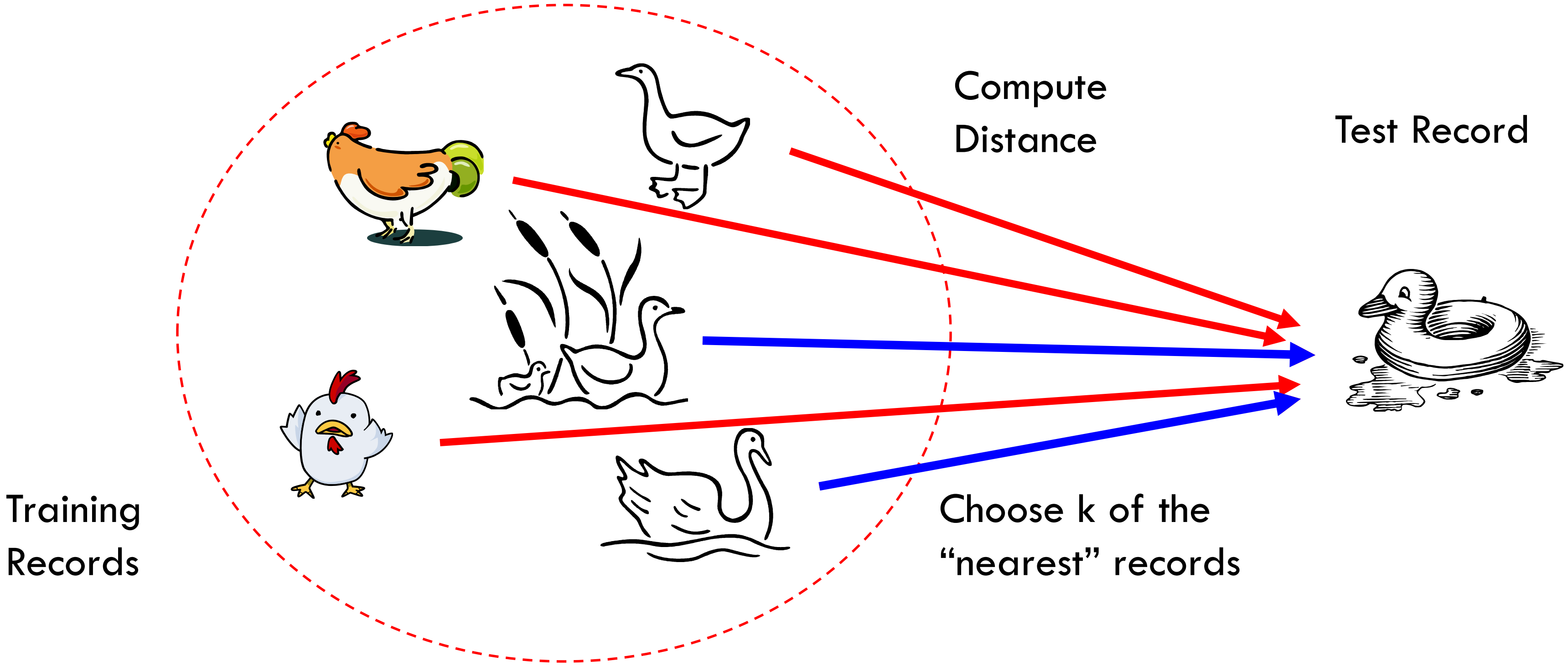
k-Nearest Neighbor (k-NN):

- Training method:
  - Save the training examples
- At prediction time:
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those y_i's.

- O(q) for each tuple to be classified. (Here q is the size of the training set.)

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it’s probably a duck



Compute Distance

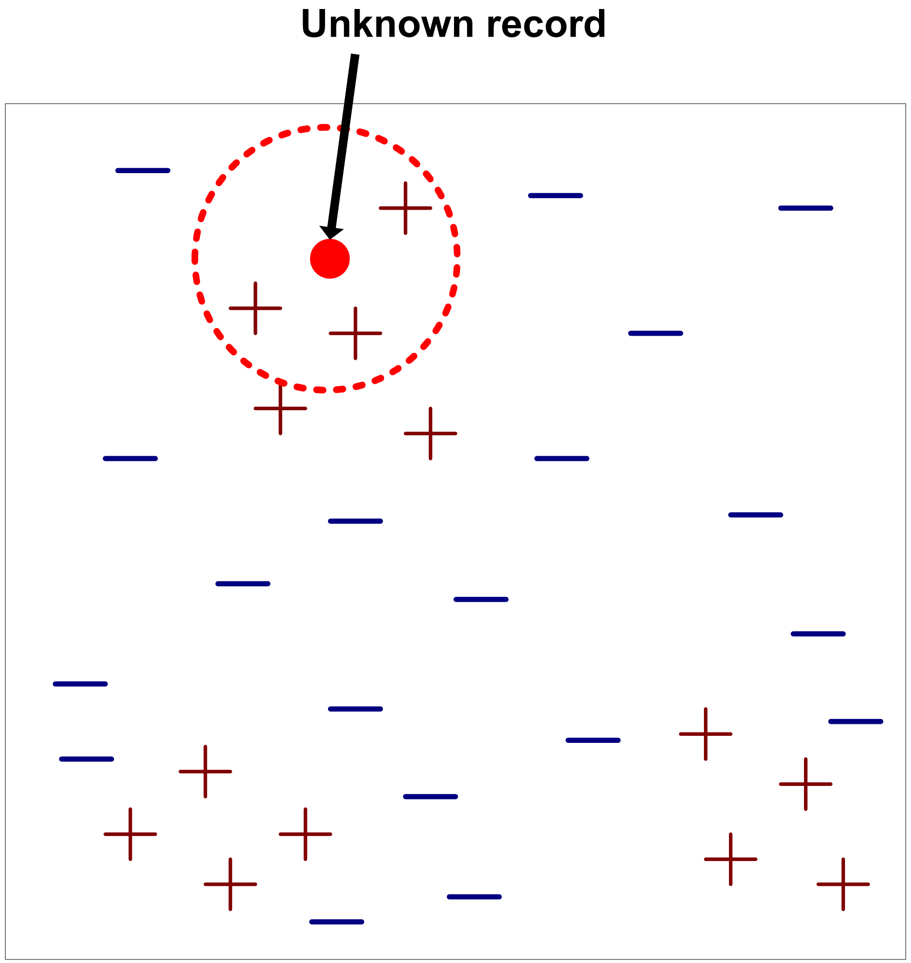
Test Record

Training Records

Choose k of the “nearest” records

Slides adapted from SlideShare

Nearest-Neighbor Classifiers

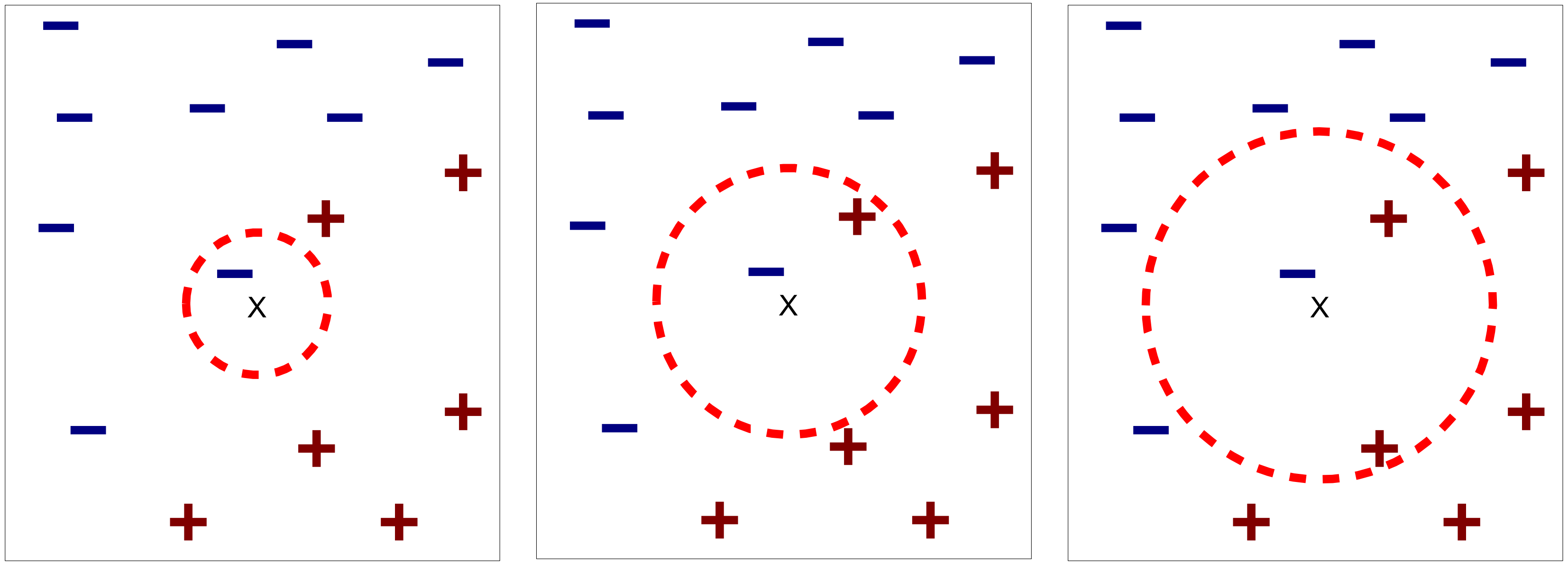


- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

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

Slides adapted from SlideShare

Definition of Nearest Neighbor



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

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Slides adapted from SlideShare

Example

- Data.
  - Two attributes: acid durability and strength
  - Label: a special paper tissue is good or not
  - X1 = Acid Durability, X2= Strength, Y =classification

    D1=(7, 7, Bad), D2= (7, 4, Bad), D3= (3, 4, Good), D4=(1, 4, Good)
- Query instance: X1 = 3, and X2 = 7. Let us set K = 3.

- Distance between query-instance and all training examples.

  D1's Squared Distance to query instance (3, 7): (7-3)^2 + (7-7)^2 = 16

  D2's: 25, D3's: 9, D4's: 13

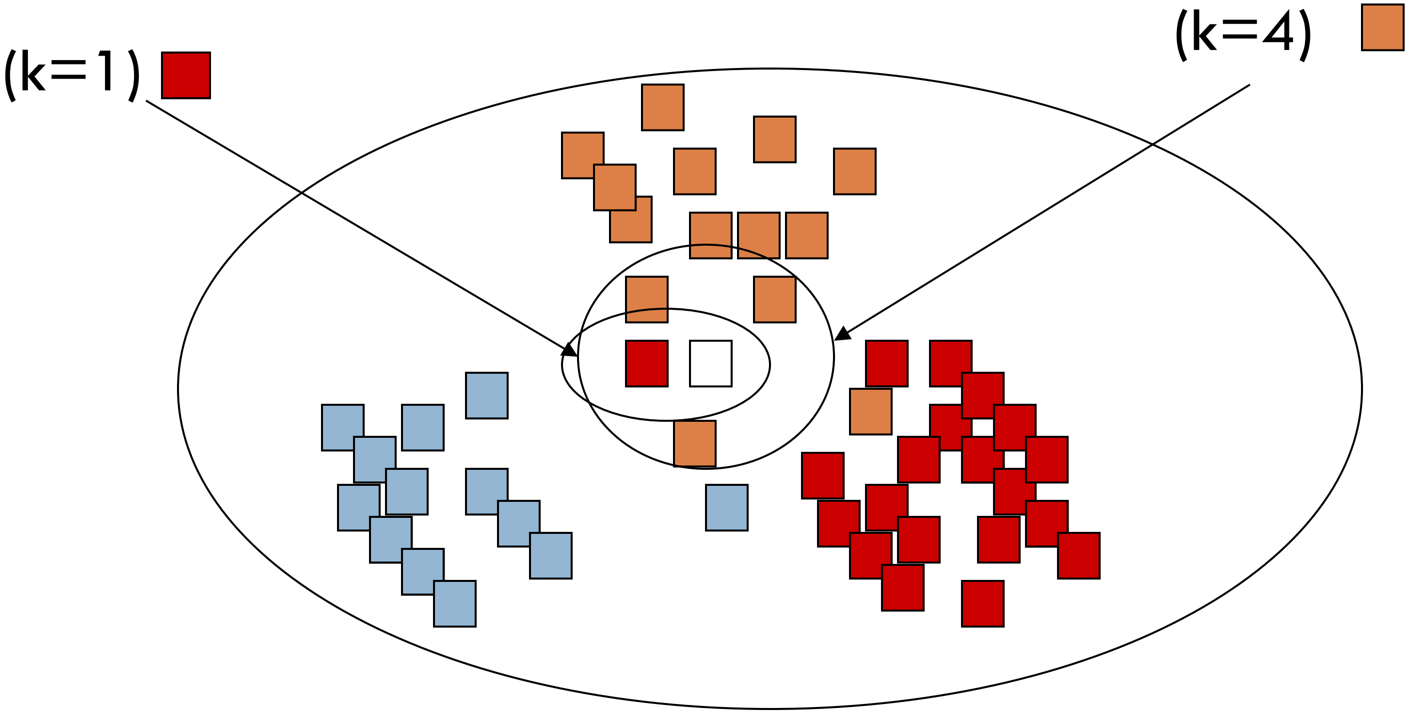
- Gather the category Y of the 3 nearest neighbors: Bad, Good, Good

- Majority voting for the predicted label: Good

http://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

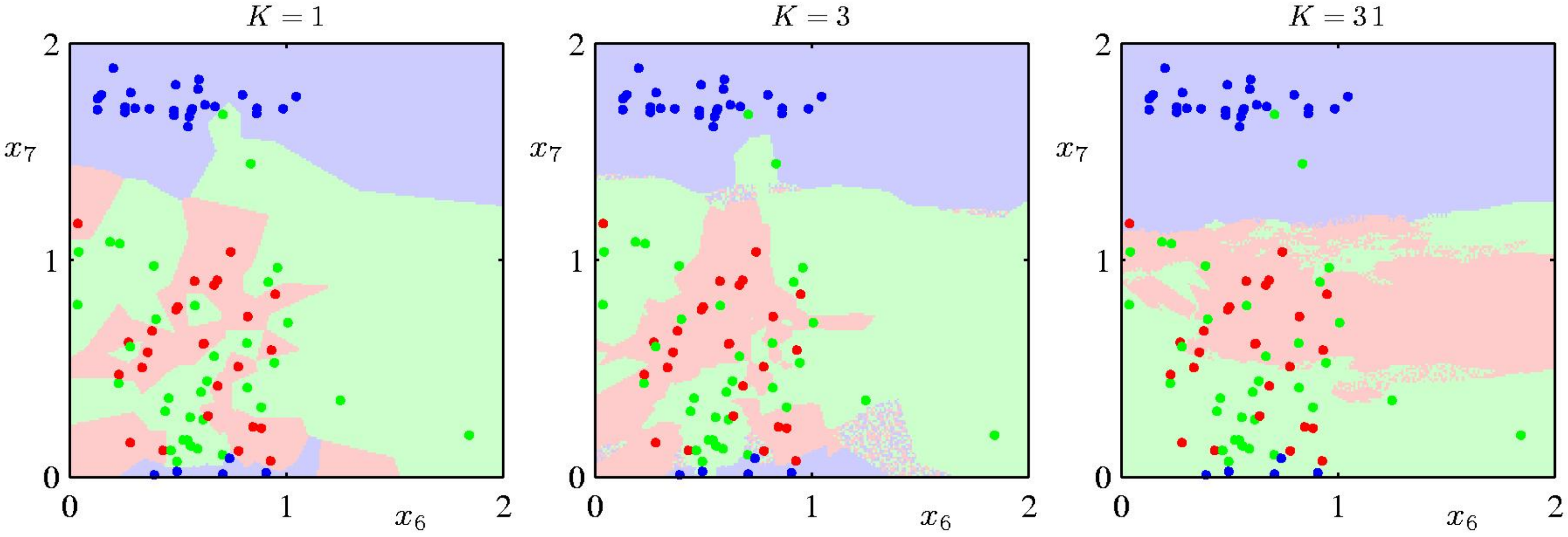
K-Nearest-Neighbour (*k*-NN) Classifier

How many neighbors should we count ?



Slides adapted from k-NN lectures given by Prof. Rong Jin at MSU

K-Nearest-Neighbour (*k*-NN) Classifier



- K acts as a smoother

Slides adapted from k-NN lectures given by Prof. Rong Jin at MSU

How to Choose K: Cross Validation

- Divide training examples into two sets
  - A training set (80%) and a validation set (20%)
- Predict the class labels for validation set by using the examples in training set
- Choose the number of neighbors *k* that maximizes the classification accuracy

Slides adapted from k-NN lectures given by Prof. Rong Jin at MSU

Discussion on the *k*-NN Algorithm

- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values 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: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

Classification: Advanced Methods

□ Lazy Learners and K-Nearest Neighbors

□ Neural Networks

□ Support Vector Machines

□ Bayesian Belief Networks

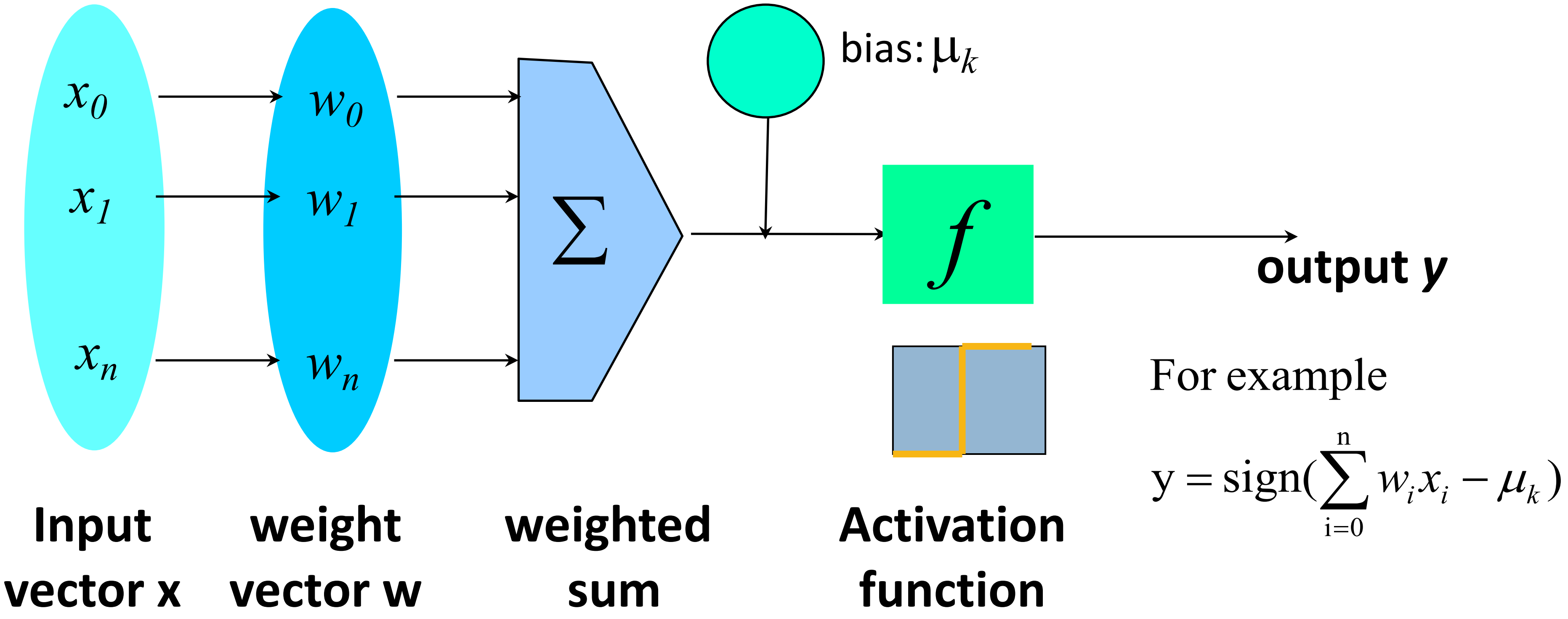
□ Additional Topics: Semi-Supervised Methods, Active Learning, etc.

□ Summary

Neural Network for Classification

- 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
- Backpropagation: A **neural network** learning algorithm

Neuron: A Hidden/Output Layer Unit

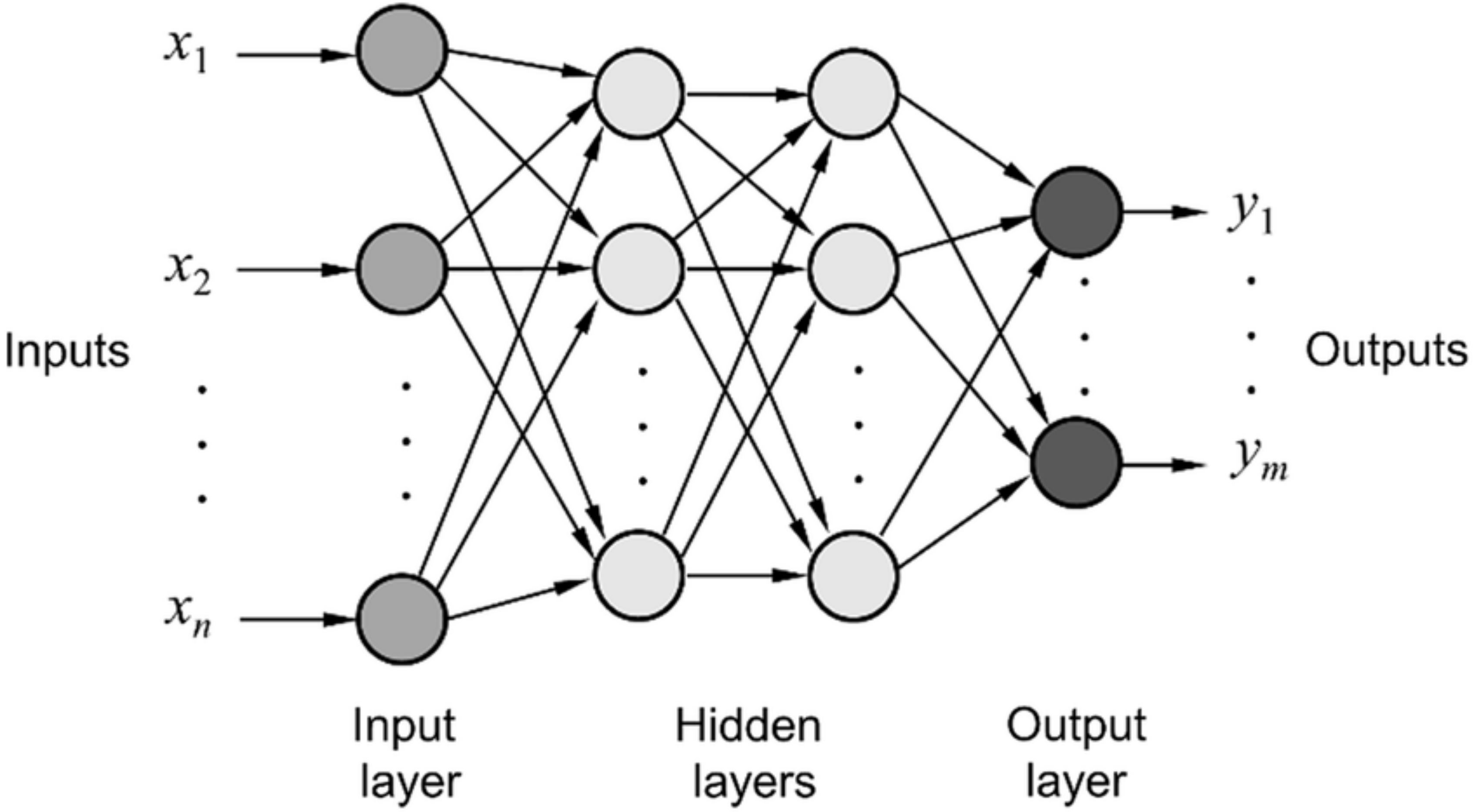


For example

$$y = \mathrm{sign}\left(\sum_{i=0}^{n} w_i x_i - \mu_k\right)$$

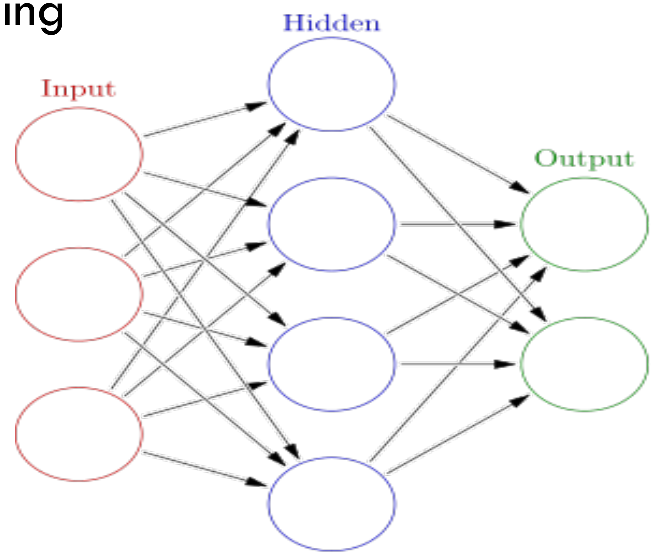
- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated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

A Multi-Layer Feed-Forward Neural Network



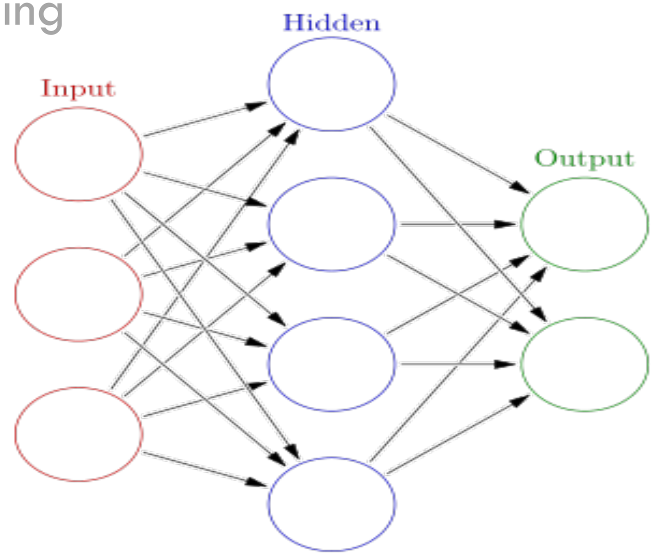
How a Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**

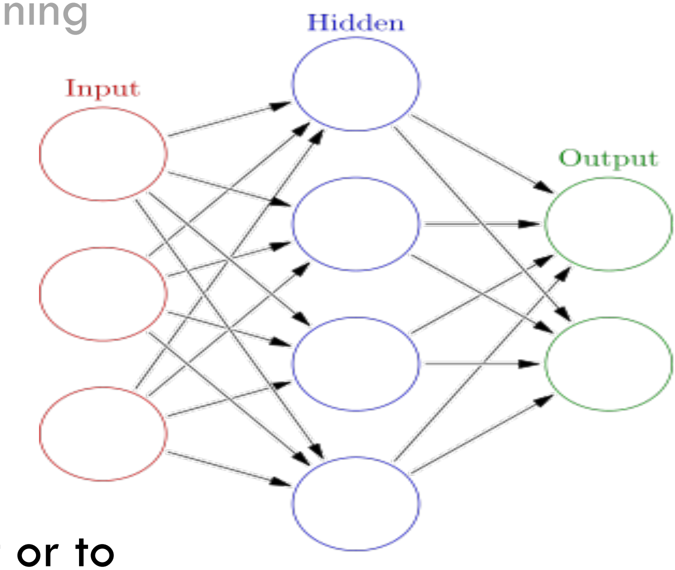


How a Multi-Layer Neural Network Works

- □ The **inputs** to the network correspond to the attributes measured for each training tuple
- □ Inputs are fed simultaneously into the units making up the **input layer**
- □ They are then weighted and fed simultaneously to a **hidden layer**
- □ The number of hidden layers is arbitrary
- □ The weighted outputs of the last hidden layer are input to units making up the **output layer,** which emits the network's prediction



How a Multi-Layer Neural Network Works



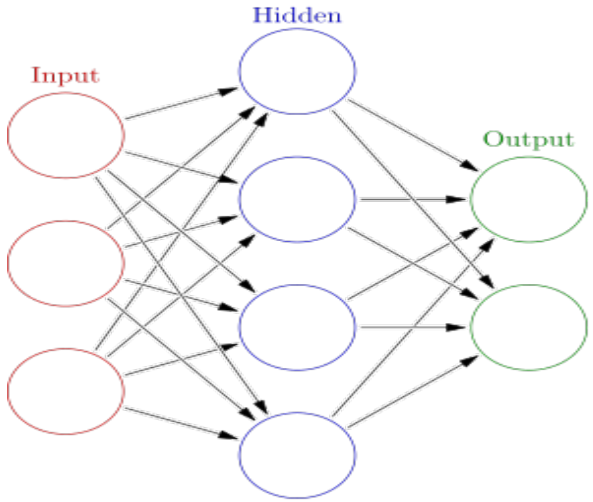
- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**
  - Given enough hidden units and enough training samples, they can closely approximate any function

Defining a 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]
- One **input** unit per domain value, 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*

Back Propagation

- **Back propagation**: Reset weights on the "front" neural units and this is sometimes done in combination with training where the correct result is known
- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: 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)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)



From Neural Networks to Deep Learning

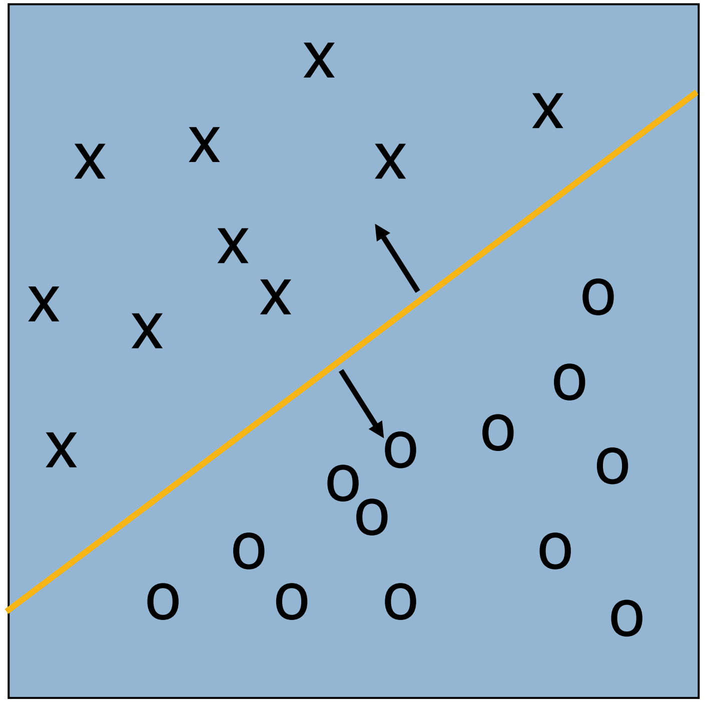
- Train networks with many layers (vs. shallow nets with just a couple of layers)
- Multiple layers work to build an improved feature space
  - First layer learns 1st order features (e.g., edges, …)
  - 2nd layer learns higher order features (combinations of first layer features, combinations of edges, etc.)
  - In current models, layers often learn in an unsupervised mode and discover general features of the input space—serving multiple tasks related to the unsupervised instances (image recognition, etc.)
  - Then final layer features are fed into supervised layer(s)
    - And entire network is often subsequently tuned using supervised training of the entire net, using the initial weightings learned in the unsupervised phase
  - Could also do fully supervised versions (back-propagation)

Classification: Advanced Methods

- Lazy Learners and K-Nearest Neighbors
- Neural Networks
- Support Vector Machines
- Bayesian Belief Networks
- Additional Topics: Semi-Supervised Methods, Active Learning, etc.
- 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_1 : # of word "homepage"
    - x_2 : # 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



Discriminative Classifiers

- Advantages
  - Prediction accuracy is generally high
    - As compared to Bayesian methods
  - Robust, works when training examples contain errors
  - Fast evaluation of the learned target function
    - Bayesian networks are normally slow

- Criticism
  - Long training time
  - Difficult to understand the learned function (weights)
    - Bayesian networks can be used easily for pattern discovery
  - Not easy to incorporate domain knowledge
    - Easy in the form of priors on the data or distributions

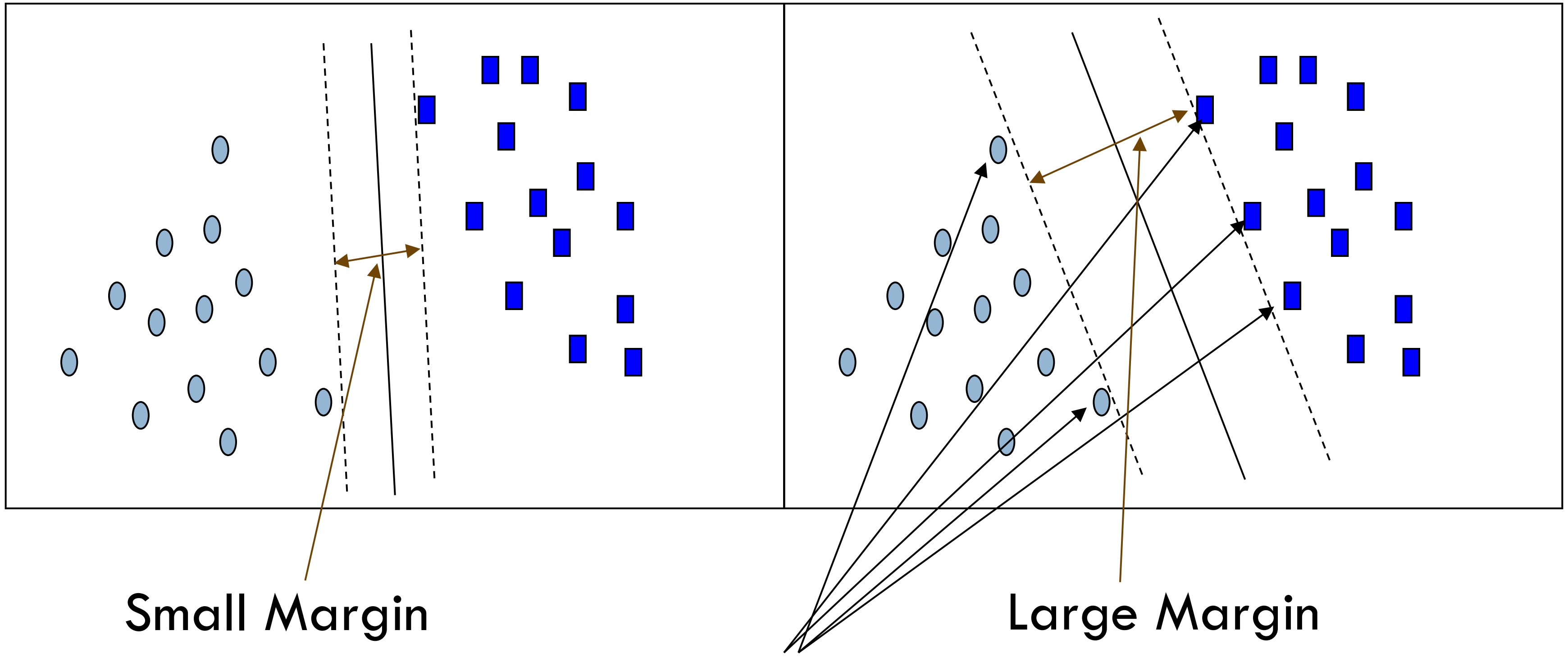
SVM—Support Vector Machines

- A relatively new (compared to decision tree or naïve bayes) classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** (“essential” training tuples) and **margins** (defined by the support vectors)

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

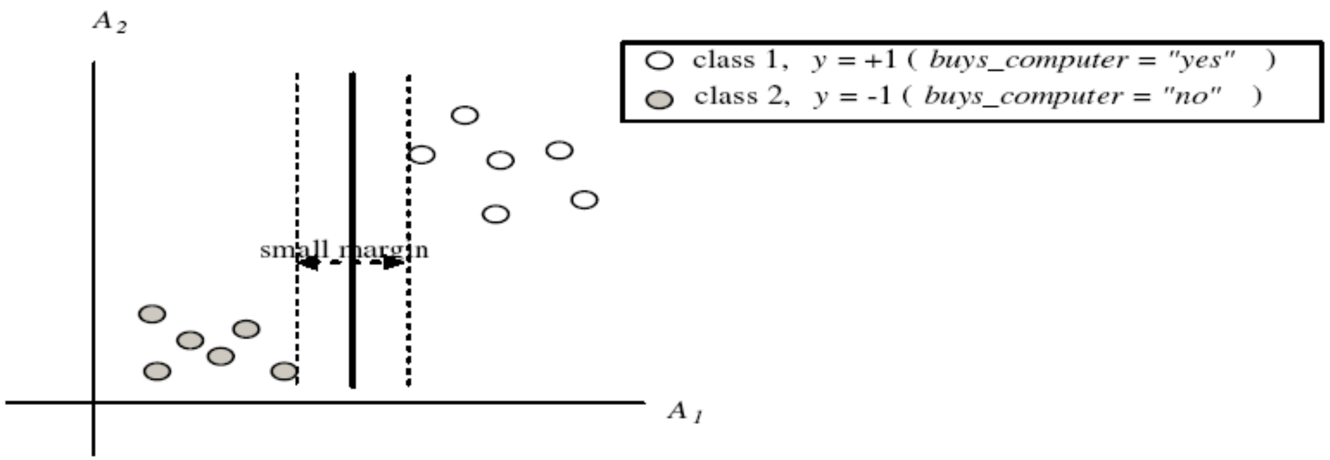


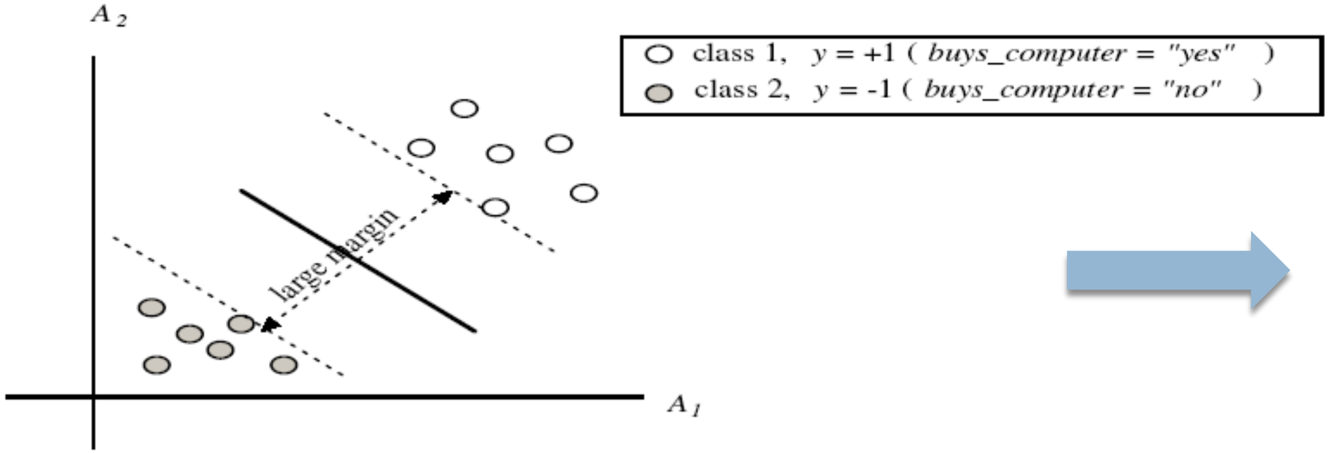
Small Margin

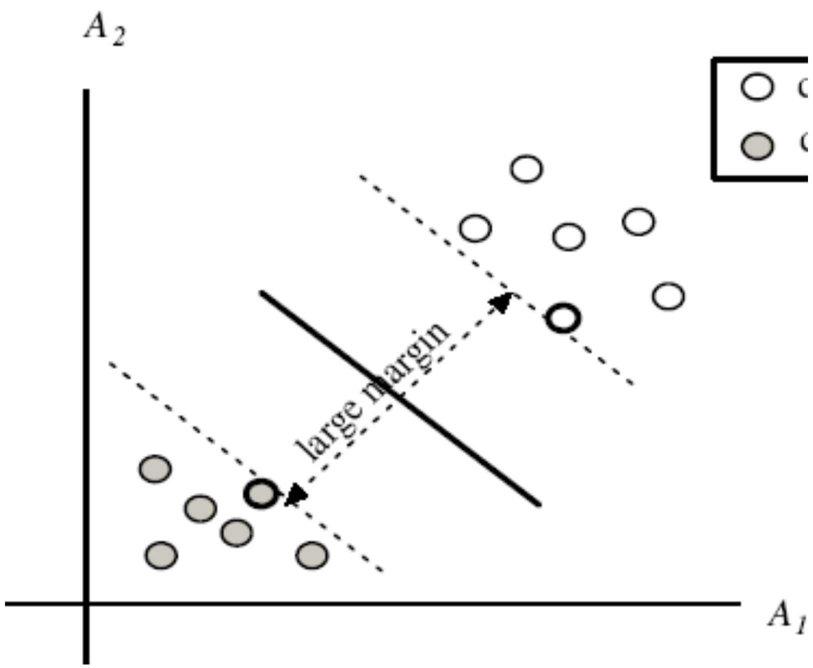
Large Margin

Support Vectors

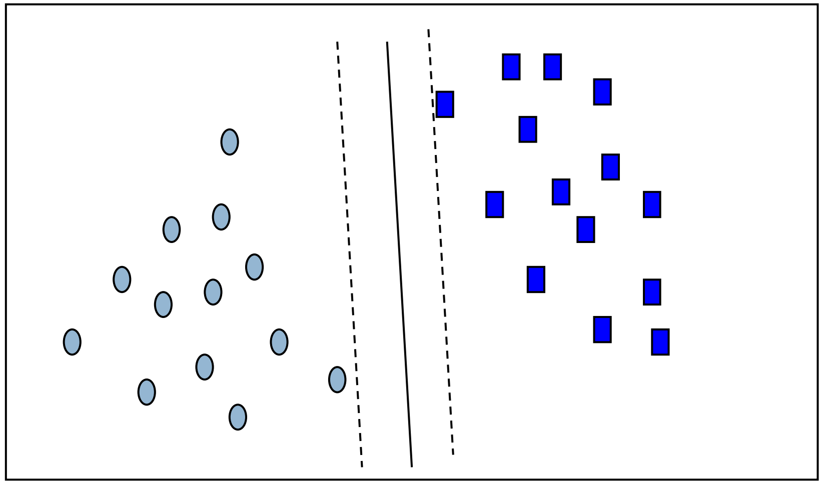
SVM—Margins and Support Vectors

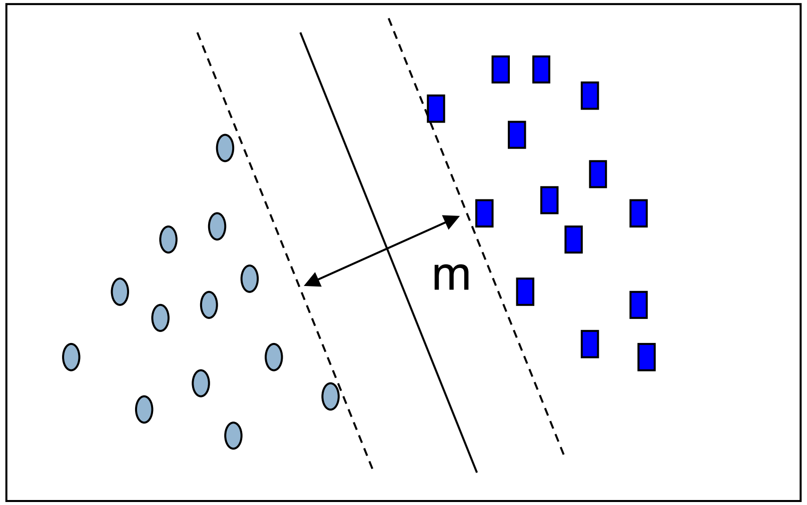

A 2
class 1, y = +1 (buys_computer = "yes")
class 2, y = -1 (buys_computer = "no")
small margin
A 1


A 2
class 1, y = +1 (buys_computer = "yes")
class 2, y = -1 (buys_computer = "no")
large margin
A 1



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)

SVM—Linearly Separable

- A separating hyperplane can be written as

  $$\mathbf{W} \bullet \mathbf{X} + b = 0$$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)

- For 2-D it can be written as: $w_0 + w_1\, x_1 + w_2\, x_2 = 0$
- The hyperplane defining the sides of the margin:

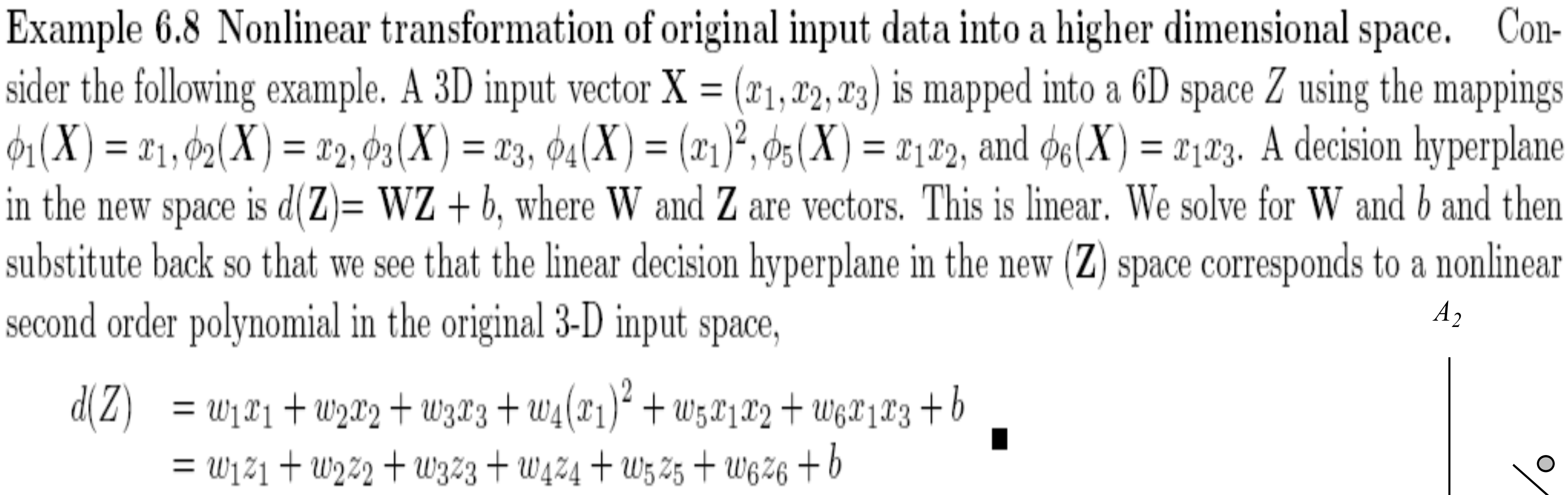
  H_1: $w_0 + w_1\, x_1 + w_2\, x_2 \geq 1$ for $y_i = +1$, and

  H_2: $w_0 + w_1\, x_1 + w_2\, x_2 \leq -1$ for $y_i = -1$

- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem:
  - Quadratic objective function and linear constraints → *Quadratic Programming (QP)* → Lagrangian multipliers

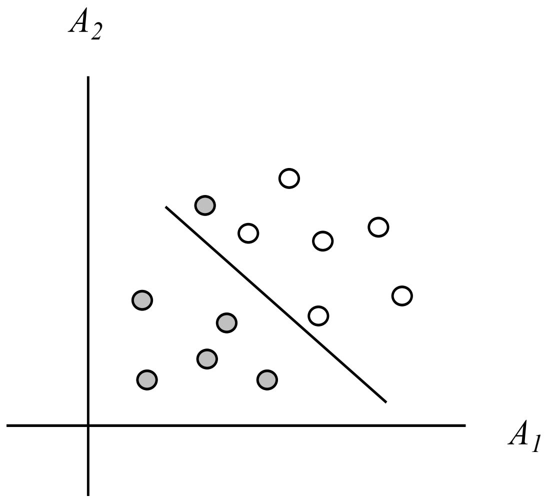
SVM—Linearly Inseparable

- Transform the original input data into a higher dimensional space



Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3$, $\phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z})= \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new $(\mathbf{Z})$ space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned} \qquad \blacksquare$$



- Search for a linear separating hyperplane in the new space

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

Kernel Functions for Nonlinear Classification

- Instead of computing the dot product on the transformed data, it is mathematically equivalent to applying a kernel function K($\mathbf{X}_i$, $\mathbf{X}_j$) to the original data, i.e.,
  - K($\mathbf{X}_i$, $\mathbf{X}_j$) = Φ($\mathbf{X}_i$) Φ($\mathbf{X}_j$)
- Typical Kernel Functions

$$\textbf{Polynomial kernel of degree } h: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = (\boldsymbol{X_i} \cdot \boldsymbol{X_j} + 1)^h$$

$$\textbf{Gaussian radial basis function kernel}: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = e^{-\|\boldsymbol{X_i} - \boldsymbol{X_j}\|^2 / 2\sigma^2}$$

$$\textbf{Sigmoid kernel}: \quad K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \tanh(\kappa \boldsymbol{X_i} \cdot \boldsymbol{X_j} - \delta)$$

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

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