UNIT II

Road Map

- **Classification using association rules**
- Naïve Bayesian classification
- Naïve Bayes for text classification
- Support vector machines
- Summary

Three approaches

- Three main approaches of using association rules for classification.
  - Using class association rules to build classifiers
  - Using class association rules as attributes/features
  - Using normal association rules for classification

Using Class Association Rules

- **Classification:** mine a small set of rules existing in the data to form a classifier or predictor.
  - It has a target attribute: Class attribute
- **Association rules:** have no fixed target, but we can fix a target.
- Class association rules (CAR): has a target class attribute. E.g.,

  Own_house = true $\rightarrow$ Class =Yes [sup=6/15, conf=6/6]
  - CARs can obviously be used for classification.

Decision tree vs. CARs

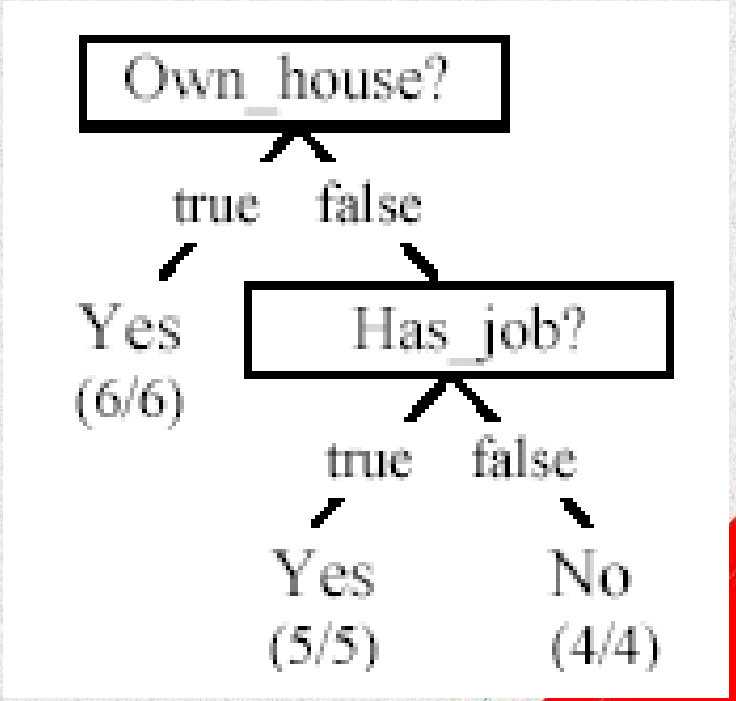
- **The decision tree below generates the following 3 rules**.

Own_house = true $\rightarrow$ Class =Yes [sup=6/15, conf=6/6]

Own_house = false, Has_job = true $\rightarrow$ Class=Yes [sup=5/15, conf=5/5]

Own_house = false, Has_job = false $\rightarrow$ Class=No [sup=4/15, conf=4/4]

- But there are many other rules that are not found by the decision tree



There are many more rules

Age = young, Has_job = true → Class=Yes [sup=2/15, conf=2/2]
Age = young, Has_job = false → Class=No [sup=3/15, conf=3/3]
Credit_Rating = fair → Class=No [sup=4/15, conf=4/4]
Credit_Rating = good → Class=Yes [sup=5/15, conf=5/6]

and many more, if we use minsup = 2/15 = 13.3% and minconf = 80%. ■

- CAR mining finds all of them.
- In many cases, rules not in the decision tree (or a rule list) may perform classification better.
- Such rules may also be actionable in practice

ID	Age	Has_Job	Own_House	Credit_Rating	Class
1	young	false	false	fair	No
2	young	false	false	excellent	No
3	young	true	false	good	Yes
4	young	true	true	good	Yes
5	young	false	false	fair	No
6	middle	false	false	fair	No
7	middle	false	false	good	No
8	middle	true	true	good	Yes
9	middle	false	true	excellent	Yes
10	middle	false	true	excellent	Yes
11	old	false	true	excellent	Yes
12	old	false	true	good	Yes
13	old	true	false	good	Yes
14	old	true	false	excellent	Yes
15	old	false	false	fair	No

Decision tree vs. CARs (cont ...)

- Association mining require discrete attributes. Decision tree learning uses both discrete and continuous attributes.
  - CAR mining requires continuous attributes discretized. There are several such algorithms.
- Decision tree is not constrained by minsup or minconf, and thus is able to find rules with very low support. Of course, such rules may be pruned due to the possible overfitting.

Considerations in CAR mining

- Multiple minimum class supports
  - Deal with imbalanced class distribution, e.g., some class is rare, 98% negative and 2% positive.
  - We can set the minsup(positive) = 0.2% and minsup(negative) = 2%.
  - If we are not interested in classification of negative class, we may not want to generate rules for negative class. We can set minsup(negative)=100% or more.
- Rule pruning may be performed.

Building classifiers

- There are many ways to build classifiers using CARs. Several existing systems available.
- Strongest rules: After CARs are mined, do nothing.
  - For each test case, we simply choose the most confident rule that covers the test case to classify it. Microsoft SQL Server has a similar method.
  - Or, using a combination of rules.
- Selecting a subset of Rules
  - used in the CBA system.
  - similar to sequential covering.

CBA: Rules are sorted first

Definition: Given two rules, r_i and r_j, $r_i \succ r_j$ (also called *ri* precedes r_j or r_i has a higher precedence than r_j) if

- the confidence of r_i is greater than that of r_j, or
- their confidences are the same, but the support of r_i is greater than that of r_j, or
- both the confidences and supports of r_i and r_j are the same, but r_i is generated earlier than r_j.

A CBA classifier *L* is of the form:

$L = \langle r_1, r_2, \ldots, r_k, \textit{default-class} \rangle$

Classifier building using CARs

```
Algorithm CBA(S, D)
1  S = sort(S);                    // sorting is done according to the precedence ≻
2  RuleList = ∅;                   // the rule list classifier
3  for each rule r ∈ S in sequence do
4      if D ≠ ∅ AND r classifies at least one example in D correctly then
5          delete from D all training examples covered by r;
6          add r at the end of RuleList
7      end
8  end
9  add the majority class as the default class at the end of RuleList
```

- This algorithm is very inefficient
- CBA has a very efficient algorithm (quite sophisticated) that scans the data at most two times.

Using rules as features

- Most classification methods do not fully explore multi-attribute correlations, e.g., naïve Bayesian, decision trees, rules induction, etc.
- This method creates extra attributes to augment the original data by
  - Using the conditional parts of rules
  - Each rule forms an new attribute
  - If a data record satisfies the condition of a rule, the attribute value is 1, and 0 otherwise
- One can also use only rules as attributes
  - Throw away the original data

Using normal association rules for classification

- A widely used approach
- Main approach: strongest rules
- Main application
  - Recommendation systems in e-commerce Web site (e.g., amazon.com).
  - Each rule consequent is the recommended item.
- Major advantage: any item can be predicted.
- Main issue:
  - Coverage: rare item rules are not found using classic algo.
  - Multiple min supports and support difference constraint help a great deal.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- **Naïve Bayesian classification**
- Naïve Bayes for text classification
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Bayesian classification

- Probabilistic view: Supervised learning can naturally be studied from a probabilistic point of view.
- Let A_1 through A_k be attributes with discrete values. The class is C.
- Given a test example *d* with observed attribute values a_1 through a_k.
- Classification is basically to compute the following posteriori probability. The prediction is the class c_j such that

is maximal

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

Apply Bayes' Rule

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|})}$$

$$= \frac{\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_j)\Pr(C = c_j)}{\sum_{r=1}^{|C|} \Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_r)\Pr(C = c_r)}$$

- $\Pr(C=c_j)$ is the class *prior* probability: easy to estimate from the training data.

Computing probabilities

- The denominator $P(A_1=a_1,...,A_k=a_k)$ is irrelevant for decision making since it is the same for every class.
- We only need $P(A_1=a_1,...,A_k=a_k \mid C=c_i)$, which can be written as

  $Pr(A_1=a_1 \mid A_2=a_2,...,A_k=a_k, C=c_j)^* Pr(A_2=a_2,...,A_k=a_k \mid C=c_j)$
- Recursively, the second factor above can be written in the same way, and so on.
- Now an assumption is needed.

Conditional independence assumption

- All attributes are conditionally independent given the class $C = c_j$.
- Formally, we assume,

  $\Pr(A_1=a_1 \mid A_2=a_2, \ldots, A_{|A|}=a_{|A|}, C=c_j) = \Pr(A_1=a_1 \mid C=c_j)$

  and so on for A_2 through $A_{|A|}$. I.e.,

$$\Pr(A_1 = a_1, \ldots, A_{|A|} = a_{|A|} \mid C = c_i) = \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

Final naïve Bayesian classifier

$$\Pr(C = c_j \mid A_1 = a_1, \ldots, A_{|A|} = a_{|A|})$$

$$= \frac{\Pr(C = c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)}{\sum_{r=1}^{|C|} \Pr(C = c_r) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_r)}$$

- We are done!
- How do we estimate $P(A_i = a_i \mid C = c_j)$? Easy!.

Classify a test instance

- If we only need a decision on the most probable class for the test instance, we only need the numerator as its denominator is the same for every class.
- Thus, given a test example, we compute the following to decide the most probable class for the test instance

$$c = \arg\max_{c_j} \Pr(c_j) \prod_{i=1}^{|A|} \Pr(A_i = a_i \mid C = c_j)$$

An example

- Compute all probabilities required for classification

A	B	C
m	b	t
m	s	t
g	q	t
h	s	t
g	q	t
g	q	f
g	s	f
h	b	f
h	q	f
m	b	f

$Pr(C = t) = 1/2$, $Pr(C = f) = 1/2$

$Pr(A=m \mid C=t) = 2/5$
$Pr(A=m \mid C=f) = 1/5$
$Pr(B=b \mid C=t) = 1/5$
$Pr(B=b \mid C=f) = 2/5$

$Pr(A=g \mid C=t) = 2/5$
$Pr(A=g \mid C=f) = 2/5$
$Pr(B=s \mid C=t) = 2/5$
$Pr(B=s \mid C=f) = 1/5$

$Pr(A=h \mid C=t) = 1/5$
$Pr(A=h \mid C=n) = 2/5$
$Pr(B=q \mid C=t) = 2/5$
$Pr(B=q \mid C=f) = 2/5$

Now we have a test example:

$A = m \quad B = q \quad C = ?$

An Example (cont ...)

- For C = t, we have

$$\Pr(C = t)\prod_{j=1}^{2}\Pr(A_j = a_j \mid C = t) = \frac{1}{2}\times\frac{2}{5}\times\frac{2}{5} = \frac{2}{25}$$

- For class C = f, we have

$$\Pr(C = f)\prod_{j=1}^{2}\Pr(A_j = a_j \mid C = f) = \frac{1}{2}\times\frac{1}{5}\times\frac{2}{5} = \frac{1}{25}$$

- C = t is more probable. t is the final class.

Additional issues

- **Numeric attributes**: Naïve Bayesian learning assumes that all attributes are categorical. Numeric attributes need to be discretized.
- **Zero counts**: An particular attribute value never occurs together with a class in the training set. We need smoothing.

- **Missing values**: Ignored

$$\Pr(A_i = a_i \mid C = c_j) = \frac{n_{ij} + \lambda}{n_j + \lambda n_i}$$

On naïve Bayesian classifier

- Advantages:
  - Easy to implement
  - Very efficient
  - Good results obtained in many applications
- Disadvantages
  - Assumption: class conditional independence, therefore loss of accuracy when the assumption is seriously violated (those highly correlated data sets)

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- **Naïve Bayes for text classification**
- Support vector machines
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Text classification/categorization

- Due to the rapid growth of online documents in organizations and on the Web, automated document classification has become an important problem.
- Techniques discussed previously can be applied to text classification, but they are not as effective as the next three methods.
- We first study a naïve Bayesian method specifically formulated for texts, which makes use of some text specific features.
- However, the ideas are similar to the preceding method.

Probabilistic framework

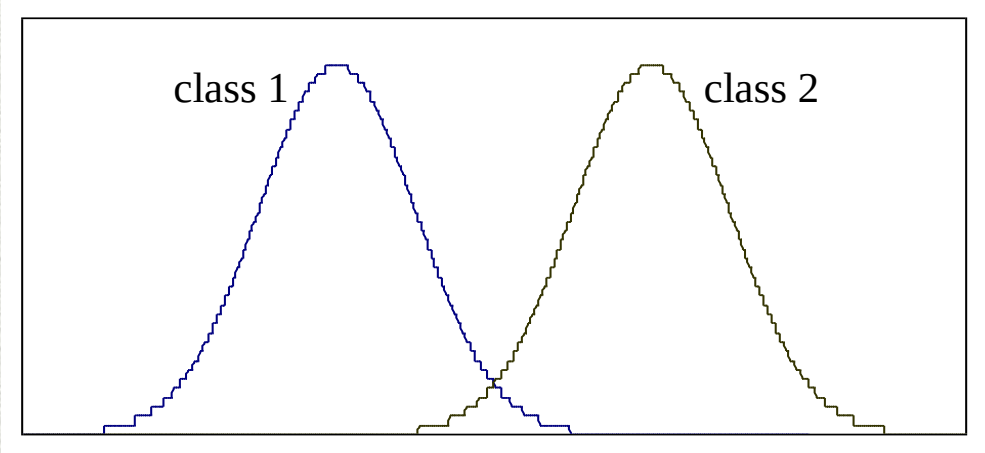
- Generative model: Each document is generated by a parametric distribution governed by a set of hidden parameters.
- The generative model makes two assumptions
  - The data (or the text documents) are generated by a mixture model,
  - There is one-to-one correspondence between mixture components and document classes.

Mixture model

- A **mixture model** models the data with a number of statistical distributions.
  - Intuitively, each distribution corresponds to a data cluster and the parameters of the distribution provide a description of the corresponding cluster.
- Each distribution in a mixture model is also called a **mixture component**.
- The distribution/component can be of any kind

An example

- The figure shows a plot of the **probability density function** of a 1-dimensional data set (with two classes) generated by
  - a mixture of two Gaussian distributions,
  - one per class, whose parameters (denoted by θ_i) are the mean (μ_i) and the standard deviation (σ_i), i.e., $\theta_i = (\mu_i, \sigma_i)$.



Mixture model (cont …)

- Let the number of mixture components (or distributions) in a mixture model be *K*.
- Let the *j*th distribution have the parameters θ_j.
- Let Θ be the set of parameters of all components, $\Theta = \{\varphi_1, \varphi_2, \ldots, \varphi_K, \theta_1, \theta_2, \ldots, \theta_K\}$, where φ_j is the *mixture weight* (or *mixture probability*) of the mixture component *j* and θ_j is the parameters of component *j*.
- How does the model generate documents?

Document generation

- Due to one-to-one correspondence, each class corresponds to a mixture component. The mixture weights are *class prior probabilities*, i.e., $\varphi_j = \Pr(c_j|\Theta)$.
- The mixture model generates each document d_i by:
  - first selecting a mixture component (or class) according to class prior probabilities (i.e., mixture weights), $\varphi_j = \Pr(c_j|\Theta)$.
  - then having this selected mixture component (c_j) generate a document d_i according to its parameters, with distribution $\Pr(d_i|c_j; \Theta)$ or more precisely $\Pr(d_i|c_j; \theta_j)$.

$$\Pr(d_i \mid \Theta) = \sum_{j=1}^{|C|} \Pr(c_j \mid \Theta) \Pr(d_i \mid c_j; \Theta) \qquad (23)$$

Model text documents

- The naïve Bayesian classification treats each document as a “bag of words”. The generative model makes the following further assumptions:
  - Words of a document are generated independently of context given the class label. The familiar naïve Bayes assumption used before.
  - The probability of a word is independent of its position in the document. The document length is chosen independent of its class.

Multinomial distribution

- With the assumptions, each document can be regarded as generated by a multinomial distribution.
- In other words, each document is drawn from a multinomial distribution of words with as many independent trials as the length of the document.
- The words are from a given vocabulary $V = \{w_1, w_2, \ldots, w_{|V|}\}$.

Use probability function of multinomial distribution

$$\Pr(d_i \mid c_j;\Theta) = \Pr(|d_i|)\,|d_i|!\prod_{t=1}^{|V|}\frac{\Pr(w_t \mid c_j;\Theta)^{N_{ti}}}{N_{ti}!} \quad (24)$$

where N_{ti} is the number of times that word w_t occurs in document d_i and

$$\sum_{t=1}^{|V|} N_{it} = |d_i| \qquad \sum_{t=1}^{|V|} \Pr(w_t \mid c_j;\Theta) = 1. \quad (25)$$

Parameter estimation

- The parameters are estimated based on empirical counts.

$$\Pr(w_t \mid c_j; \hat{\Theta}) = \frac{\sum_{i=1}^{|D|} N_{ti} \Pr(c_j \mid d_i)}{\sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si} \Pr(c_j \mid d_i)}. \quad (26)$$

- In order to handle 0 counts for infrequent occurring words that do not appear in the training set, but may appear in the test set, we need to smooth the probability. *Lidstone* smoothing, $0 \leq \lambda \leq 1$

$$\Pr(w_t \mid c_j; \hat{\Theta}) = \frac{\lambda + \sum_{i=1}^{|D|} N_{ti} \Pr(c_j \mid d_i)}{\lambda |V| + \sum_{s=1}^{|V|} \sum_{i=1}^{|D|} N_{si} \Pr(c_j \mid d_i)}. \quad (27)$$

Parameter estimation (cont …)

- Class prior probabilities, which are mixture weights φ_j, can be easily estimated using training data

$$\Pr(c_j \mid \hat{\Theta}) = \frac{\sum_{i=1}^{|D|} \Pr(c_j \mid d_i)}{|D|} \qquad (28)$$

Classification

- Given a test document d_i, from Eq. (23) (27) and (28)

$$\Pr(c_j \mid d_i; \hat{\Theta}) = \frac{\Pr(c_j \mid \hat{\Theta}) \Pr(d_i \mid c_j; \hat{\Theta})}{\Pr(d_i \mid \hat{\Theta})}$$

$$= \frac{\Pr(c_j \mid \hat{\Theta}) \prod_{k=1}^{|d_i|} \Pr(w_{d_i,k} \mid c_j; \hat{\Theta})}{\sum_{r=1}^{|C|} \Pr(c_r \mid \hat{\Theta}) \prod_{k=1}^{|d_i|} \Pr(w_{d_i,k} \mid c_r; \hat{\Theta})}$$

where $w_{d_i,k}$ is the word in position k of document d_i. If the final classifier is to classify each document into a single class, then the class with the highest posterior probability is selected:

$$\arg\max_{c_j \in C} \Pr(c_j \mid d_i; \hat{\Theta}) \qquad (30)$$

Discussions

- Most assumptions made by naïve Bayesian learning are violated to some degree in practice.
- Despite such violations, researchers have shown that naïve Bayesian learning produces very accurate models.
  - The main problem is the mixture model assumption. When this assumption is seriously violated, the classification performance can be poor.
- Naïve Bayesian learning is extremely efficient.

Road Map

- Basic concepts
- Decision tree induction
- Evaluation of classifiers
- Rule induction
- Classification using association rules
- Naïve Bayesian classification
- Naïve Bayes for text classification
- **Support vector machines**
- K-nearest neighbor
- Ensemble methods: Bagging and Boosting
- Summary

Introduction

- Support vector machines were invented by V. Vapnik and his co-workers in 1970s in Russia and became known to the West in 1992.
- SVMs are linear classifiers that find a hyperplane to separate two class of data, positive and negative.
- Kernel functions are used for nonlinear separation.
- SVM not only has a rigorous theoretical foundation, but also performs classification more accurately than most other methods in applications, especially for high dimensional data.
- It is perhaps the best classifier for text classification.

Basic concepts

- Let the set of training examples D be

  $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$,

  where $\mathbf{x}_i = (x_1, x_2, \ldots, x_n)$ is an **input vector** in a real-valued space $X \subseteq R^n$ and y_i is its **class label** (output value), $y_i \in \{1, -1\}$.

  1: positive class and -1: negative class.

- SVM finds a linear function of the form (**w**: weight vector)

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

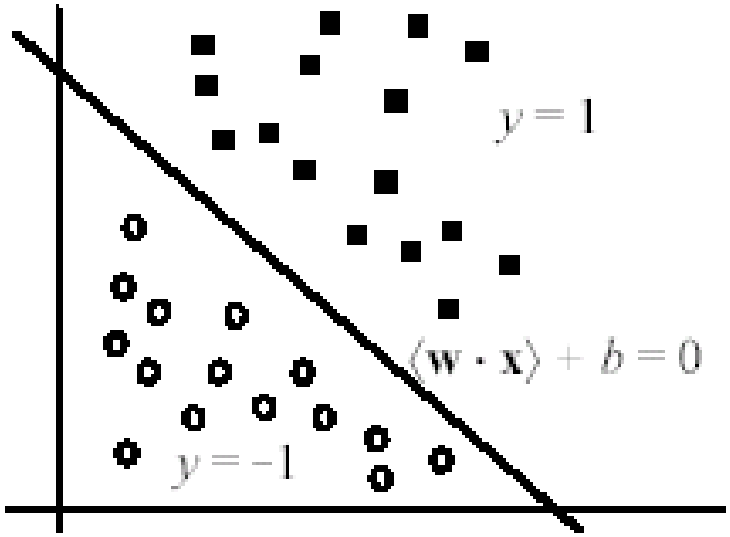
$$y_i = \begin{cases} 1 & if \ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 0 \\ -1 & if \ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b < 0 \end{cases}$$

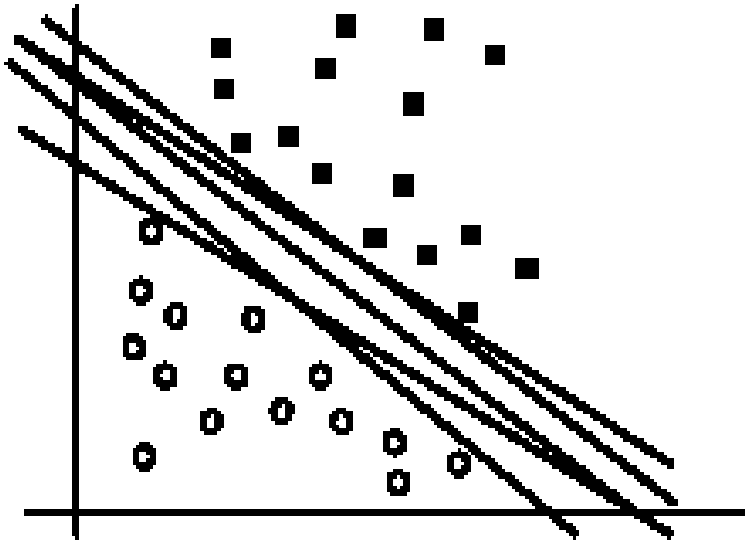
The hyperplane

- The hyperplane that separates positive and negative training data is

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$$

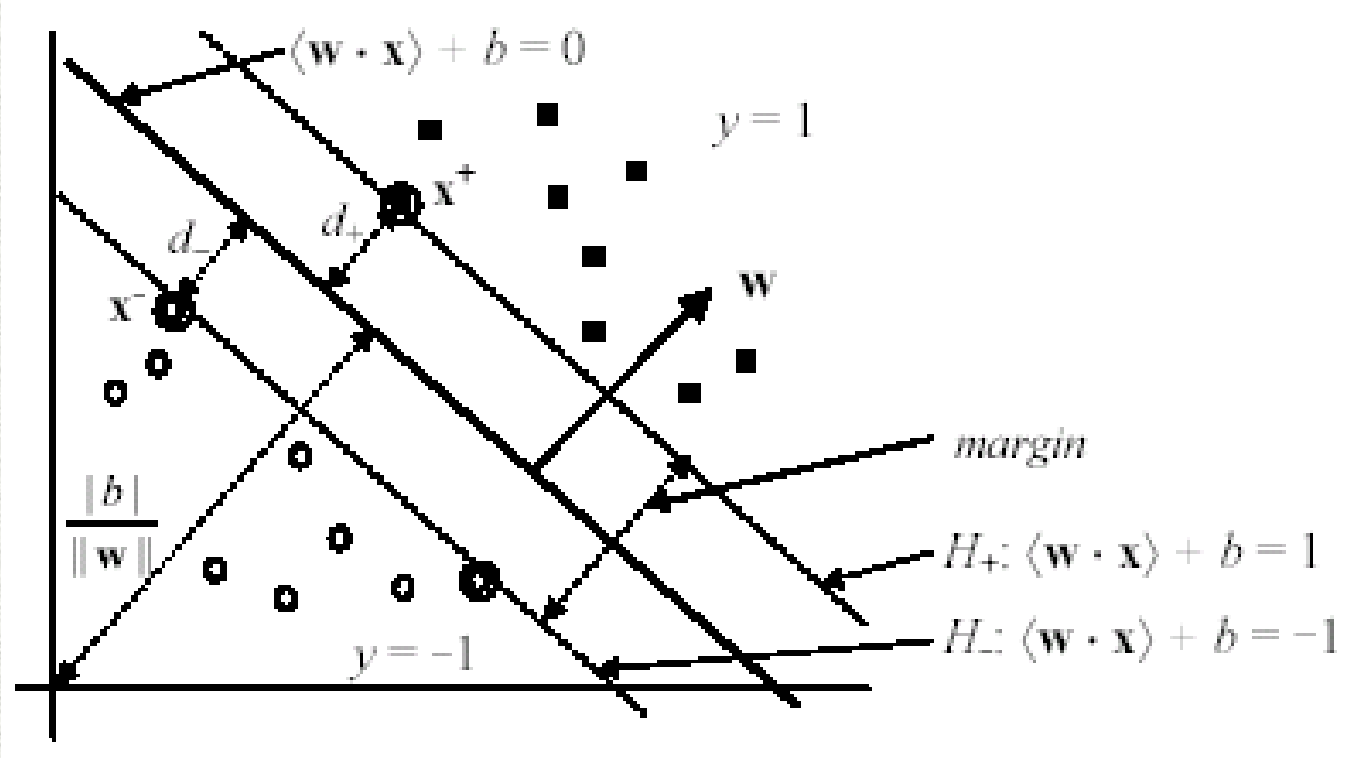
- It is also called the decision boundary (surface).
- So many possible hyperplanes, which one to choose?





Maximal margin hyperplane

- SVM looks for the separating hyperplane with the largest margin.
- Machine learning theory says this hyperplane minimizes the error bound



Linear SVM: separable case

- Assume the data are linearly separable.
- Consider a positive data point ($\mathbf{x}^+$, 1) and a negative ($\mathbf{x}^-$, -1) that are closest to the hyperplane

  $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$.
- We define two parallel hyperplanes, H_+ and H_-, that pass through $\mathbf{x}^+$ and $\mathbf{x}^-$ respectively. H_+ and H_- are also parallel to $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$.

$$H_+: \quad \langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b = 1$$

$$H_-: \quad \langle \mathbf{w} \cdot \mathbf{x}^- \rangle + b = -1$$

$$\text{such that} \quad \begin{cases} \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1 & \text{if } y_i = 1 \\ \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 & \text{if } y_i = -1, \end{cases}$$

Compute the margin

- Now let us compute the distance between the two margin hyperplanes H_+ and H_-. Their distance is the **margin** (d_+ + d_- in the figure).
- Recall from vector space in algebra that the (perpendicular) **distance** from a point $\mathbf{x}_i$ to the hyperplane $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ is:

$$\frac{|\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b|}{\|\mathbf{w}\|} \tag{36}$$

where $||\mathbf{w}||$ is the norm of $\mathbf{w}$,

$$\|\mathbf{w}\| = \sqrt{<\mathbf{w} \cdot \mathbf{w}>} = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2} \tag{37}$$

Compute the margin (cont …)

- Let us compute d_+.
- Instead of computing the distance from $\mathbf{x}^+$ to the separating hyperplane $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$, we pick up any point $\mathbf{x}_s$ on $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$ and compute the distance from $\mathbf{x}_s$ to $\langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b = 1$ by applying the distance Eq. (36) and noticing $\langle \mathbf{w} \cdot \mathbf{x}_s \rangle + b = 0$,

$$d_+ = \frac{|\langle \mathbf{w} \cdot \mathbf{x_s} \rangle + b - 1|}{\| \mathbf{w} \|} = \frac{1}{\| \mathbf{w} \|} \tag{38}$$

$$margin = d_+ + d_- = \frac{2}{\| \mathbf{w} \|} \tag{39}$$

A optimization problem!

Definition (Linear SVM: separable case): Given a set of linearly separable training examples,

$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$

Learning is to solve the following constrained minimization problem,

$$\text{Minimize} : \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} \qquad (40)$$

$$\text{Subject to} : y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, \ldots, r$$

summarizes

$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1, \quad i = 1, 2, \ldots, r$

$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1$ for $y_i = 1$

$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1$ for $y_i = -1$.

Solve the constrained minimization

- Standard Lagrangian method

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{r} \alpha_i [y_i (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \qquad (41)$$

where $\alpha_i \geq 0$ are the **Lagrange multipliers**.

- Optimization theory says that an optimal solution to (41) must satisfy certain conditions, called **Kuhn-Tucker conditions**, which are necessary (but not sufficient)
- Kuhn-Tucker conditions play a central role in constrained optimization.

Kuhn-Tucker conditions

$$\frac{\partial L_P}{\partial w_j} = w_j - \sum_{i=1}^{r} y_i \alpha_i \mathbf{x}_i = 0, \quad j = 1, 2, \ldots, m \tag{48}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{r} y_i \alpha_i = 0 \tag{49}$$

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 \geq 0, \quad i = 1, 2, \ldots, r \tag{50}$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r \tag{51}$$

$$\alpha_i (y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1) = 0, \quad i = 1, 2, \ldots, r \tag{52}$$

- Eq. (50) is the original set of constraints.
- The complementarity condition (52) shows that only those data points on the margin hyperplanes (i.e., H_+ and H_-) can have $\alpha_i > 0$ since for them $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 = 0$.
- These points are called the **support vectors**, All the other parameters $\alpha_i = 0$.

Solve the problem

- In general, Kuhn-Tucker conditions are necessary for an optimal solution, but not sufficient.
- However, for our minimization problem with a convex objective function and linear constraints, the Kuhn-Tucker conditions are both **necessary** and **sufficient** for an optimal solution.
- Solving the optimization problem is still a difficult task due to the inequality constraints.
- However, the Lagrangian treatment of the convex optimization problem leads to an alternative **dual** formulation of the problem, which is easier to solve than the original problem (called the **primal**).

Dual formulation

- From primal to a dual: Setting to zero the partial derivatives of the Lagrangian (41) with respect to the **primal variables** (i.e., **w** and *b*), and substituting the resulting relations back into the Lagrangian.
  - I.e., substitute (48) and (49), into the original Lagrangian (41) to eliminate the primal variables

$$L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,\, j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle, \qquad (55)$$

Dual optimization problem

$$\text{Maximize: } L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} y_i y_i \alpha_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \tag{56}$$

$$\text{Subject to: } \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r.$$

- This dual formulation is called the **Wolfe dual**.
- For the convex objective function and linear constraints of the primal, it has the property that the maximum of L_D occurs at the same values of **w**, b and α_i, as the minimum of L_P (the primal).
- Solving (56) requires numerical techniques and clever strategies, which are beyond our scope.

The final decision boundary

- After solving (56), we obtain the values for α_i, which are used to compute the weight vector **w** and the bias *b* using Equations (48) and (52) respectively.
- **The decision boundary**

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i \in sv} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b = 0 \qquad (57)$$

- **Testing**: Use (57). Given a test instance **z**,

$$sign(\langle \mathbf{w} \cdot \mathbf{z} \rangle + b) = sign\left(\sum_{i \in sv} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{z} \rangle + b \right) \qquad (58)$$

- If (58) returns 1, then the test instance **z** is classified as positive; otherwise, it is classified as negative.

Linear SVM: Non-separable case

- Linear separable case is the ideal situation.
- Real-life data may have noise or errors.
  - Class label incorrect or randomness in the application domain.
- Recall in the separable case, the problem was

$$\text{Minimize} : \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2}$$

$$\text{Subject to} : y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, \ldots, r$$

- With noisy data, the constraints may not be satisfied. Then, no solution!

Relax the constraints

- To allow errors in data, we relax the margin constraints by introducing **slack** variables, ξ_i (≥ 0) as follows:

  $\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq 1 - \xi_i$ for $y_i = 1$

  $\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 + \xi_i$ for $y_i = -1$.

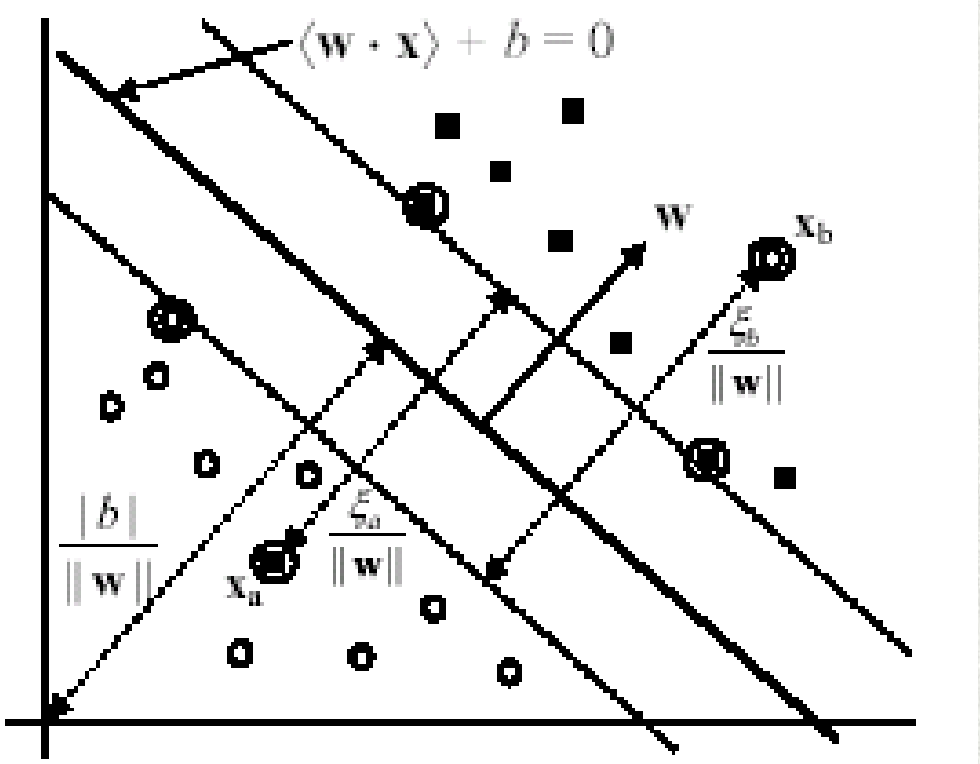
- The new constraints:

  Subject to: $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$, $i = 1, \ldots, r$,

  $\xi_i \geq 0$, $i = 1, 2, \ldots, r$.

Geometric interpretation

- Two error data points $\mathbf{x}_a$ and $\mathbf{x}_b$ (circled) in wrong regions



Penalize errors in objective function

- We need to penalize the errors in the objective function.
- A natural way of doing it is to assign an extra cost for errors to change the objective function to

- $k = 1$ is commonly used, which has the advantage that neither ξ_i nor its Lagrangian multipliers appear in the dual formulation.

$$\text{Minimize} : \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\left(\sum_{i=1}^{r} \xi_i\right)^k \qquad (60)$$

New optimization problem

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \qquad (61)$$

$$\text{Subject to}: y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, \ldots, r$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, r$$

- This formulation is called the **soft-margin SVM**. The primal Lagrangian is

(62)

where $\alpha_i, \mu_i \geq 0$ are the **Lagrange multipliers**

$$L_P = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C\sum_{i=1}^{r} \xi_i - \sum_{i=1}^{r} \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i] - \sum_{i=1}^{r} \mu_i \xi_i$$

Kuhn-Tucker conditions

$$\frac{\partial L_P}{\partial w_j} = w_j - \sum_{i=1}^{r} y_i \alpha_i \mathbf{x}_i = 0, \quad j = 1, 2, \ldots, m \tag{63}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{r} y_i \alpha_i = 0 \tag{64}$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0, \quad i = 1, 2, \ldots, r \tag{65}$$

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i \geq 0, \quad i = 1, 2, \ldots, r \tag{66}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, r \tag{67}$$

$$\alpha_i \geq 0, \quad i = 1, 2, \ldots, r \tag{68}$$

$$\mu_i \geq 0, \quad i = 1, 2, \ldots, r \tag{69}$$

$$\alpha_i(y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 + \xi_i) = 0, \quad i = 1, 2, \ldots, r \tag{70}$$

$$\mu_i \xi_i = 0, \quad i = 1, 2, \ldots, r \tag{71}$$

From primal to dual

- As the linear separable case, we transform the primal to a dual by setting to zero the partial derivatives of the Lagrangian (62) with respect to the **primal variables** (i.e., **w**, b and ξ_i), and substituting the resulting relations back into the Lagrangian.
- Ie.., we substitute Equations (63), (64) and (65) into the primal Lagrangian (62).
- From Equation (65), $C - \alpha_i - \mu_i = 0$, we can deduce that $\alpha_i \leq C$ because $\mu_i \geq 0$.

Dual

- The dual of (61) is

$$\text{Maximize: } L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle. \tag{72}$$

$$\text{Subject to: } \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, r.$$

- Interestingly, ξ_i and its Lagrange multipliers μ_i are not in the dual. The objective function is identical to that for the separable case.
- The only difference is the constraint $\alpha_i \le C$.

Find primal variable values

- The dual problem (72) can be solved numerically.
- The resulting α_i values are then used to compute **w** and *b*. **w** is computed using Equation (63) and *b* is computed using the Kuhn-Tucker complementarity conditions (70) and (71).
- Since no values for ξ_i, we need to get around it.
  - From Equations (65), (70) and (71), we observe that if $0 < \alpha_i < C$ then both $\xi_i = 0$ and $y_i\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - 1 + \xi_i = 0$. Thus, we can use any training data point for which $0 < \alpha_i < C$ and Equation (69) (with $\xi_i = 0$) to compute *b.*

$$b = \frac{1}{y_i} - \sum_{i=1}^{r} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle = 0. \qquad (73)$$

(65), (70) and (71) in fact tell us more

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 \text{ and } \xi_i = 0 \\ 0 < \alpha_i < C &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1 \text{ and } \xi_i = 0 \\ \alpha_i = C &\Rightarrow y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \leq 1 \text{ and } \xi_i \geq 0 \end{aligned} \quad (74)$$

- (74) shows a very important property of SVM.
  - The solution is **sparse** in α_i. Many training data points are outside the margin area and their α_i's in the solution are 0.
  - Only those data points that are on the margin (i.e., $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) = 1$, which are support vectors in the separable case), inside the margin (i.e., $\alpha_i = C$ and $y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) < 1$), or errors are non-zero.
  - Without this sparsity property, SVM would not be practical for large data sets.

The final decision boundary

- The final decision boundary is (we note that many α_i's are 0)

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \sum_{i=1}^{r} y_i \alpha_i \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b = 0 \qquad (75)$$

- The decision rule for classification (testing) is the same as the separable case, i.e.,

  sign($\langle \mathbf{w} \cdot \mathbf{x} \rangle + b$).

- Finally, we also need to determine the parameter *C* in the objective function. It is normally chosen through the use of a validation set or cross-validation.

How to deal with nonlinear separation?

- The SVM formulations require linear separation.
- Real-life data sets may need nonlinear separation.
- To deal with nonlinear separation, the same formulation and techniques as for the linear case are still used.
- We only transform the input data into another space (usually of a much higher dimension) so that
  - a linear decision boundary can separate positive and negative examples in the transformed space,
- The transformed space is called the **feature space**. The original data space is called the **input space**.

Space transformation

- The basic idea is to map the data in the input space *X* to a feature space *F* via a nonlinear mapping ϕ,

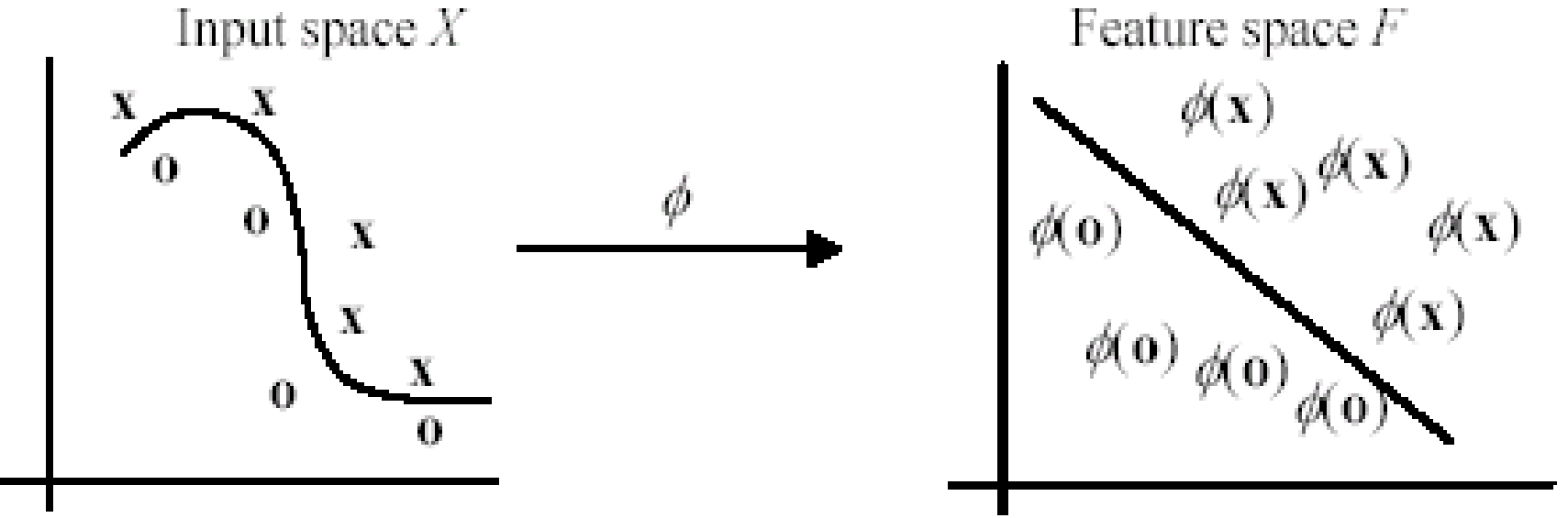
$$\phi : X \rightarrow F \tag{76}$$

$$\mathbf{x} \mapsto \phi(\mathbf{x})$$

- After the mapping, the original training data set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$ becomes:

$$\{(\phi(\mathbf{x}_1), y_1), (\phi(\mathbf{x}_2), y_2), \ldots, (\phi(\mathbf{x}_r), y_r)\} \tag{77}$$

Geometric interpretation



- In this example, the transformed space is also 2-D. But usually, the number of dimensions in the feature space is much higher than that in the input space

Optimization problem in (61) becomes

With the transformation, the optimization problem in (61) becomes

$$\text{Minimize: } \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \tag{78}$$

$$\text{Subject to: } y_i(\langle \mathbf{w} \cdot \phi(\mathbf{x}_i) \rangle + b) \geq 1 - \xi_i, \quad i = 1, 2, ..., r$$

$$\xi_i \geq 0, \quad i = 1, 2, ..., r$$

The dual is

$$\text{Maximize: } L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{r} y_i y_j \alpha_i \alpha_j \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle. \tag{79}$$

$$\text{Subject to: } \sum_{i=1}^{r} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., r.$$

The final decision rule for classification (testing) is

$$\sum_{i=1}^{r} y_i \alpha_i \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) \rangle + b \tag{80}$$

An example space transformation

- Suppose our input space is 2-dimensional, and we choose the following transformation (mapping) from 2-D to 3-D:

$$(x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- The training example ((2, 3), -1) in the input space is transformed to the following in the feature space:

  ((4, 9, 8.5), -1)

Problem with explicit transformation

- The potential problem with this explicit data transformation and then applying the linear SVM is that it may suffer from the curse of dimensionality.
- The number of dimensions in the feature space can be huge with some useful transformations even with reasonable numbers of attributes in the input space.
- This makes it computationally infeasible to handle.
- Fortunately, explicit transformation is not needed.

Kernel functions

- We notice that in the dual formulation both
  - the construction of the optimal hyperplane (79) in *F* and
  - the evaluation of the corresponding decision function (80)

  only require dot products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ and never the mapped vector $\phi(\mathbf{x})$ in its explicit form. This is a crucial point.
- Thus, if we have a way to compute the dot product $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ using the input vectors **x** and **z** directly,
  - no need to know the feature vector $\phi(\mathbf{x})$ or even ϕ itself.
- In SVM, this is done through the use of **kernel functions**, denoted by *K*,

$$K(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle \tag{82}$$

An example kernel function

- Polynomial kernel

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d \tag{83}$$

- Let us compute the kernel with degree $d = 2$ in a 2-dimensional space: $\mathbf{x} = (x_1, x_2)$ and $\mathbf{z} = (z_1, z_2)$.

$$\begin{aligned}
\langle \mathbf{x} \cdot \mathbf{z} \rangle^2 &= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \cdot (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle \\
&= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle,
\end{aligned} \tag{84}$$

- This shows that the kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^2$ is a dot product in a transformed feature space

Kernel trick

- The derivation in (84) is only for illustration purposes.
- We do not need to find the mapping function.
- We can simply apply the kernel function directly by
  - replace all the dot products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ in (79) and (80) with the kernel function $K(\mathbf{x}, \mathbf{z})$ (e.g., the polynomial kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^d$ in (83)).
- This strategy is called the **kernel trick**.

Is it a kernel function?

- The question is: how do we know whether a function is a kernel without performing the derivation such as that in (84)? I.e,
  - How do we know that a kernel function is indeed a dot product in some feature space?
- This question is answered by a theorem called the Mercer's theorem, which we will not discuss here.

Commonly used kernels

- It is clear that the idea of kernel generalizes the dot product in the input space. This dot product is also a kernel with the feature map being the identity

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle. \tag{85}$$

Commonly used kernels include

$$\text{Polynomial:} \quad K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x} \cdot \mathbf{z} \rangle + \theta)^d \tag{86}$$

$$\text{Gaussian RBF:} \quad K(\mathbf{x}, \mathbf{z}) = e^{-\|\mathbf{x}-\mathbf{z}\|^2 / 2\sigma} \tag{87}$$

$$\text{Sigmoidal:} \quad K(\mathbf{x}, \mathbf{z}) = \tanh(k\langle \mathbf{x} \cdot \mathbf{z} \rangle - \delta) \tag{88}$$

where $\theta \in R$, $d \in N$, $\sigma > 0$, and $k, \delta \in R$.

Some other issues in SVM

- SVM works only in a real-valued space. For a categorical attribute, we need to convert its categorical values to numeric values.
- SVM does only two-class classification. For multi-class problems, some strategies can be applied, e.g., one-against-rest, and error-correcting output coding.
- The hyperplane produced by SVM is hard to understand by human users. The matter is made worse by kernels. Thus, SVM is commonly used in applications that do not required human understanding.

Summary

- Applications of supervised learning are in almost any field or domain.
- We studied 8 classification techniques.
- There are still many other methods, e.g.,
  - Bayesian networks
  - Neural networks
  - Genetic algorithms
  - Fuzzy classification

  This large number of methods also show the importance of classification and its wide applicability.
- It remains to be an active research area.