Machine Learning and Data Mining
(IT4242E)

Quang Nhat NGUYEN

quang.nguyennhat@hust.edu.vn

Hanoi University of Science and Technology
School of Information and Communication Technology

Academic year 2021-2022

The course's content:

- Introduction
- Performance evaluation of the ML/DM system
- Regression problem
- **Classification problem**
  - **Support vector machine**
- Clustering problem
- Association rule mining problem

Support vector machines – Introduction (1)

- Support vector machines (SVMs) were first introduced by V. Vapnik and his colleagues in 1970s in Russia, and became well-known in 1990s
- SVM is a **linear classifier** that finds a hyperplane to separate **two classes** of data, e.g., positive and negative
- **Kernel functions** (i.e., transformation functions) are used for non-linear separation
- SVM has a rigorous theoretical foundation
- A good candidate for those classification problems with high dimensional input spaces
- Known as one of the best classifiers for text classification

Support vector machines – Introduction (2)

- *In this lecture, a vector is denoted by a bold symbol!*
- Representation of the set of r training examples

  $\{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_r}, y_r)\}$,

  - $\mathbf{x_i}$ is an **input <u>vector</u>** in a space $X \subseteq R^n$
  - y_i is the **class label** (i.e., output value), $y_i \in \{1,-1\}$
  - y_i=1: *positive* class; y_i=-1: *negative* class
- For an instance $\mathbf{x_i}$:

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

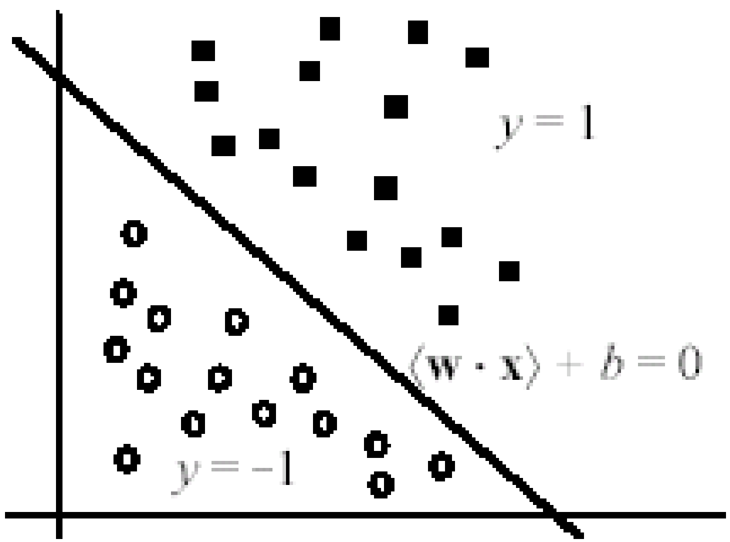
- SVM finds a linear separation function

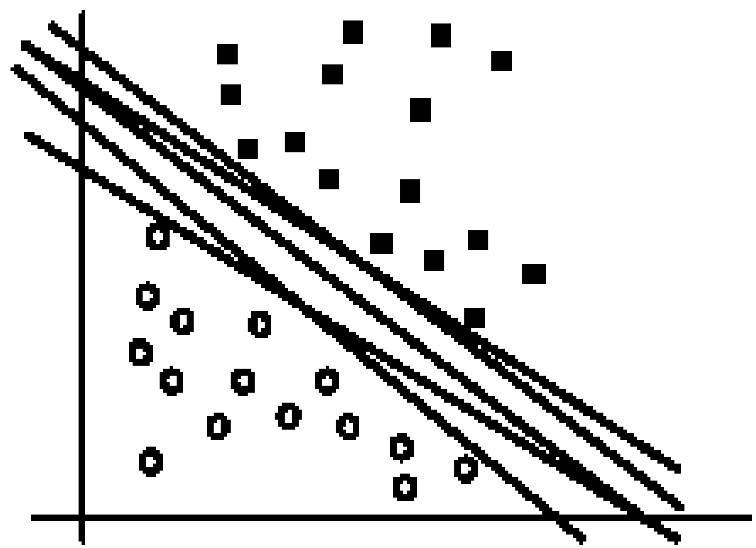
$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \qquad \text{[Eq.2]}$$

  - **w** is a weights vector; b is a real value

The separation hyperplane

- The hyperplane that separates positive and negative training examples: $\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$
- It is also called the decision boundary (surface)
- Many possible such hyperplanes. Which one?

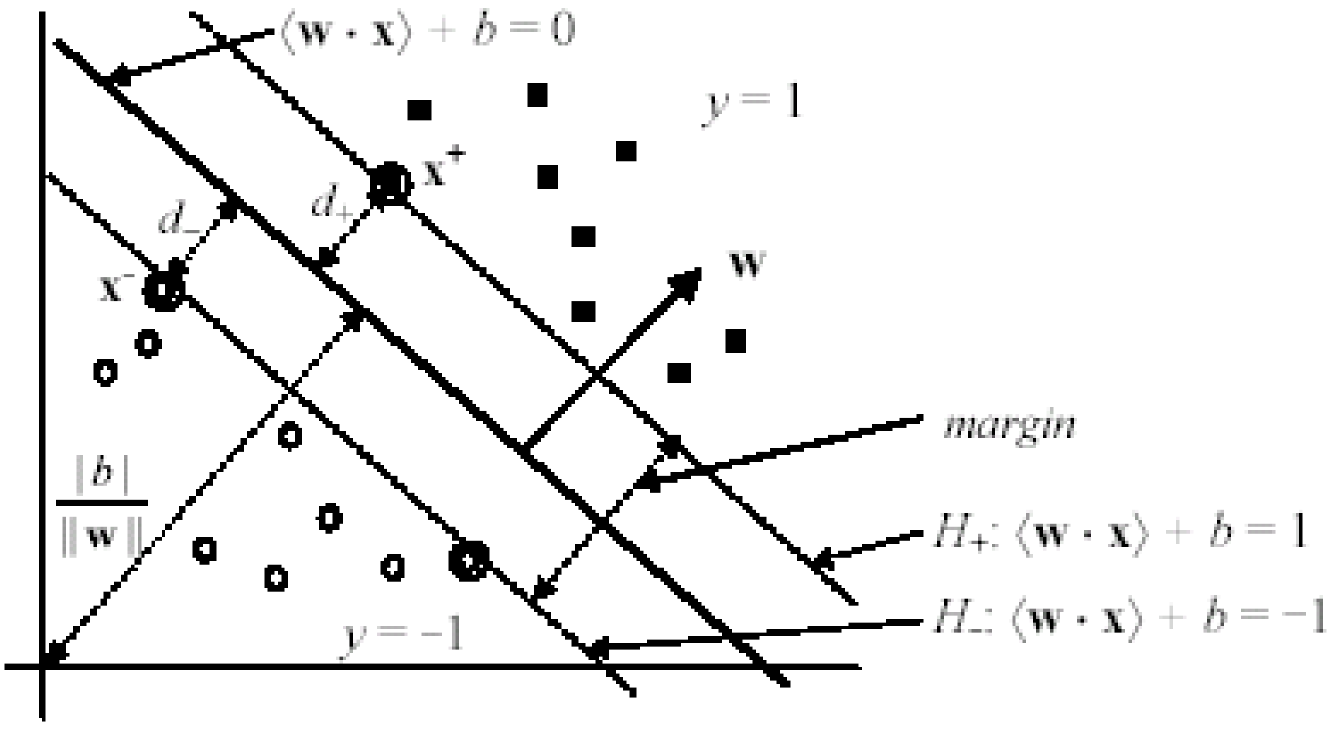




[Liu, 2006]

Maximal-margin hyperplane

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



[Liu, 2006]

Linear SVM – The separable case

- Assume the data (i.e., the training instances) are linearly separable
- Consider a positive instance ($\mathbf{x}^+$,1) and a negative ($\mathbf{x}^-$,-1) that are *closest* to the **separating** hyperplane H_0 ($\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0$)
- We define two parallel **margin** hyperplanes
  - H_+ passes through $\mathbf{x}^+$, and is parallel to H_0
  - H_- passes through $\mathbf{x}^-$, and is parallel to H_0

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

$$H_-: \ \langle \mathbf{w} \cdot \mathbf{x}^- \rangle + b = -1 \qquad \text{[Eq.3]}$$

such that:
$$\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b \geq 1, \quad \text{if } y_i = 1$$
$$\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b \leq -1, \quad \text{if } y_i = -1$$

Margin computation (1)

- The **margin** is the distance between these two (margin) hyperplanes H_+ and H_-. In the previous figure:
  - d_+ is the distance between H_+ and H_0
  - d_- is the distance between H_- and H_0
  - $(d_+ + d_-)$ is the margin
- 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:

$$\frac{|\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b|}{\| \mathbf{w} \|} \qquad \text{[Eq.4]}$$

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

$$\| \mathbf{w} \| = \sqrt{< \mathbf{w} \cdot \mathbf{w} >} = \sqrt{w_1^{\ 2} + w_2^{\ 2} + ... + w_n^{\ 2}} \qquad \text{[Eq.5]}$$

Margin computation (2)

- Compute d_+ – the distance from $\mathbf{x}^+$ to $(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0)$
  - By applying [Eq.3-4]:

$$d_+ = \frac{|\langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b|}{\|\mathbf{w}\|} = \frac{|1|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \qquad \text{[Eq.6]}$$

- Compute d_- – the distance from $\mathbf{x}^-$ to $(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = 0)$
  - By applying [Eq.3-4]:

$$d_- = \frac{|\langle \mathbf{w} \cdot \mathbf{x}^- \rangle + b|}{\|\mathbf{w}\|} = \frac{|-1|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \qquad \text{[Eq.7]}$$

- Compute the margin

$$margin = d_+ + d_- = \frac{2}{\|\mathbf{w}\|} \qquad \text{[Eq.8]}$$

SVM learning – Margin maximization

Definition (**Linear SVM – The separable case**)

- Given a set of r linearly separable training examples

  $D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_r}, y_r)\}$

- SVM learns a classifier that maximizes the margin
- Is equivalent to solve the following **quadratic optimization problem**
  - Find ***w*** and *b* such that: $margin = \frac{2}{\|\mathbf{w}\|}$ is maximized
  - Subject to:

$$\begin{cases} \langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b \geq 1, \text{if } \mathrm{y_i} = 1 \\ \langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b \leq -1, \text{if } \mathrm{y_i} = \text{-}1 \end{cases}; \text{ for every training example } \mathbf{x_i} (i = 1..r)$$

Margin max. – An optimization problem

- SVM learning is equivalent to solve the following **constrained minimization problem**

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} \qquad \text{[Eq.9]}$$

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

- …is equivalent to

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} \qquad \text{[Eq.10]}$$

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

Recall from Constrained optimization theory

- The minimization problem with an equality constraint:

  Minimize $f(\mathbf{x})$, subject to $g(\mathbf{x})=0$

- The necessary condition for $\mathbf{x_0}$ to be a solution:

$$\begin{cases} \dfrac{\partial}{\partial \mathbf{x}} \left(f(\mathbf{x}) + \alpha g(\mathbf{x}) \right) \Big|_{\mathbf{x}=\mathbf{x_0}} = 0 \\ g(\mathbf{x}) = 0 \end{cases}$$

where α is a Lagrange multiplier

- In case of multiple equality constraints $g_i(\mathbf{x})=0$ (i=1..r), we need a Lagrange multiplier for each constraint:

$$\begin{cases} \dfrac{\partial}{\partial \mathbf{x}} \left(f(\mathbf{x}) + \sum_{i=1}^{r} \alpha_i g_i(\mathbf{x}) \right) \Bigg|_{\mathbf{x}=\mathbf{x_0}} = 0 \\ g_i(\mathbf{x}) = 0 \end{cases}$$

where α_i is a Lagrange multiplier

Recall from Constrained optimization theory

- The minimization problem with multiple inequality constraints:

  Minimize $f(\mathbf{x})$, subject to $g_i(\mathbf{x}) \leq 0$

- A similar condition for a solution $\mathbf{x_0}$, except that the Lagrange multipliers $\boldsymbol{\alpha_i}$ must be positive

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}} \left(f(\mathbf{x}) + \sum_{i=1}^{r} \alpha_i g_i(\mathbf{x}) \right) \Bigg|_{\mathbf{x}=\mathbf{x_0}} = 0 \\ g_i(\mathbf{x}) \leq 0 \end{cases} \quad \text{where } \alpha_i \geq 0$$

- The function

$$L = f(\mathbf{x}) + \sum_{i=1}^{r} \alpha_i g_i(\mathbf{x})$$

is called the Lagrangian

Solving the constrained minimization prob.

- The Lagrangian formulation

$$L_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{r} \alpha_i [y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1] \quad \text{[Eq.11]}$$

where α_i ($\geq$0) are the **Lagrange multipliers**

- Optimization theory says that an optimal solution to [Eq.11] must satisfy certain conditions, called **Karush-Kuhn-Tucker conditions**, which are necessary (but not sufficient)
- Karush-Kuhn-Tucker conditions play a central role in both the theory and practice of constrained optimization

Karush-Kuhn-Tucker conditions

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{r} \alpha_i y_i \mathbf{x_i} = 0 \qquad \text{[Eq.12]}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{r} \alpha_i y_i = 0 \qquad \text{[Eq.13]}$$

$$y_i(\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b) - 1 \geq 0, \text{ for every training example } \mathbf{x_i} \ (i = 1..r) \qquad \text{[Eq.14]}$$

$$\alpha_i \geq 0 \qquad \text{[Eq.15]}$$

$$\alpha_i(y_i(\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b) - 1) = 0 \qquad \text{[Eq.16]}$$

- [Eq.14] is the original set of constraints
- The *complementarity* condition in [Eq.16] shows that only those instances (i.e., data points) on the margin hyperplanes (i.e., H_+ and H_-) can have α_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**!
- For the other instances (i.e., non-support vectors), α_i=0

Solving the constrained minimization prob.

- In general, the Karush-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 Karush-Kuhn-Tucker conditions are *both necessary and sufficient* for an optimal solution
- However, solving the optimization problem is still a difficult task due to the inequality constraints!
- The Lagrangian treatment of the convex optimization problem leads to an alternative **dual** formulation of the problem

  $\rightarrow$ Easier to solve than the original problem – the **primal**

Dual formulation

- To derive a **dual** formulation from the **primal** problem:
  - →Setting to zero the partial derivatives of the Lagrangian formulation in [Eq.11] w.r.t. the **primal variables** (i.e., **w** and *b*)
  - →Then, substituting the resulting relations back into the Lagrangian
  - i.e., substitute [Eq.12-13] into the original Lagrangian formulation in [Eq.11] to eliminate the primal variables
- The dual formulation L_D

$$L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \qquad \text{[Eq.17]}$$

- Both L_P (primal) and L_D (dual) are Lagrangian formulations
  - Arise from the same objective function, but with different constraints
  - The solution is found by minimizing L_P or by maximizing L_D

Dual optimization prolem

$$\text{Maximize}: L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j \langle \mathbf{x_i} \cdot \mathbf{x_j} \rangle \quad \text{[Eq.18]}$$

$$\text{subject to}: \begin{cases} \sum_{i=1}^{r} \alpha_i y_i = 0 \\ \alpha_i \geq 0, \forall i = 1..r \end{cases}$$

- 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)
- By solving [Eq.18], we obtain the Lagrange multipliers α_i (which are then used to compute **w** and b)
- Solving [Eq.18] requires *numerical methods* (for solving linearly constrained convex quadratic optimization problems)
  $\rightarrow$ Out of the scope of this lecture!

Find the solutions for ***w**** and *b**

- Let's call `SV` the set of the support vectors
  - `SV` is a subset of the set of `r` training examples
  - $\rightarrow \alpha_i > 0$ for a support vector $\mathbf{x_i}$
  - $\rightarrow \alpha_i = 0$ for a non-support vector $\mathbf{x_i}$
- Using [Eq.12], we can compute the solution **w***

$$\mathbf{w^*} = \sum_{i=1}^{r} \alpha_i y_i \mathbf{x_i} = \sum_{\mathbf{x_i} \in SV} \alpha_i y_i \mathbf{x_i};\ \text{because } \forall \mathbf{x_i} \notin SV: \alpha_i = 0$$

- Using [Eq.16] and a support vector $\mathbf{x_k}$, we have
  - $\alpha_k(y_k(<\mathbf{w^*} \cdot \mathbf{x_k}>+b^*)-1)=0$
  - Recall that $\alpha_k > 0$ for a support vector $\mathbf{x_k}$
  - Hence, $[y_k(<\mathbf{w^*} \cdot \mathbf{x_k}>+b^*)-1]=0$
  - So, we have the solution $b^* = y_k - <\mathbf{w^*} \cdot \mathbf{x_k}>$

The final decision boundary

- **The decision boundary**

$$f(\mathbf{x}) = \langle \mathbf{w}^* \cdot \mathbf{x} \rangle + b^* = \sum_{\mathbf{x_i} \in SV} \alpha_i y_i \langle \mathbf{x_i} \cdot \mathbf{x} \rangle + b^* = 0 \qquad \text{[Eq.19]}$$

- Given a test instance **z**, compute the value:

$$sign(\langle \mathbf{w}^* \cdot \mathbf{z} \rangle + b^*) = sign\left(\sum_{\mathbf{x_i} \in SV} \alpha_i y_i \langle \mathbf{x_i} \cdot \mathbf{z} \rangle + b^* \right) \qquad \text{[Eq.20]}$$

→ If [Eq.20] returns 1, then the test instance **z** is classified as positive; otherwise, it is classified as negative

- The classification depends:
  - <u>Only on the support vectors</u>
  - The inner (dot)-product of two vectors (i.e., not the vectors themselves!)

Linear SVM: Non-separable case (1)

- How SVM works, if instances are not separable?
  - The separable case is the ideal situation
  - The dataset may contain noise or error (e.g., some instances have incorrect class labels)
- Recall from the separable case, the problem is:

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

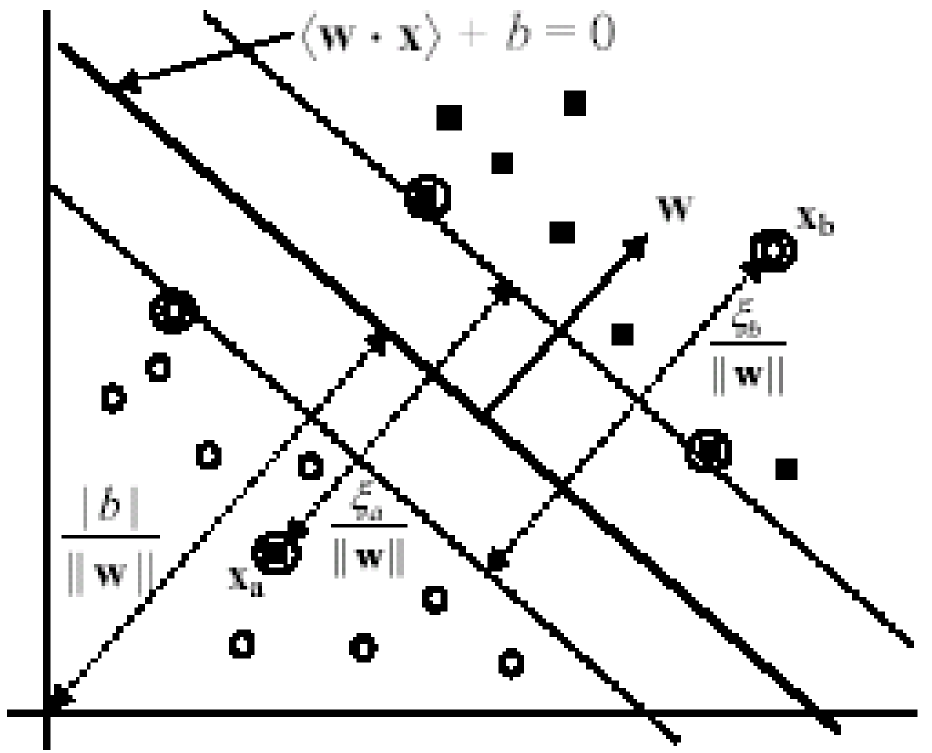
$$\text{Subject to}: y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1, \quad i = 1, 2, ..., r$$

- With noisy data, the constraints may not be satisfied

  →No (**w*** and b*) solution!

Linear SVM: Non-separable case (2)

The two (error) instances $\mathbf{x}_a$ and $\mathbf{x}_b$ are in wrong regions



[Liu, 2006]

Relax the constraints

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

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

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

- For an error to occur: $\xi_i > 1$
- $(\Sigma_i \xi_i)$ is the upper bound on the number of training errors
- The new constraints for the non-separable case

$$y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad \forall i = 1..r$$

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

Penalize errors in the objective function

- We need to penalize the errors in the objective function
- By assigning an extra cost for errors, and incorporate this cost in the (new) objective function

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

  - Where C (>0) is a parameter that decides the ***penalty degree*** assigned to errors

  → A larger C assigns a higher penalty to errors

- k =1 is commonly used, because of the advantage that neither ξ_i nor their Lagrange multipliers appear in the dual formulation

New optimization problem

$$\text{Minimize}: \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \qquad \text{[Eq.21]}$$

$$\text{Subject to}: \begin{cases} y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, & \forall i = 1..r \\ \xi_i \geq 0, & \forall i = 1..r \end{cases}$$

- This new formulation is called the **soft-margin SVM**
- The (new) primal Lagrangian formulation

[Eq.22]

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

where α_i (≥0) and μ_i (≥0) are the **Lagrange multipliers**

Karush-Kuhn-Tucker conditions (1)

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{r} \alpha_i y_i \mathbf{x_i} = 0$$ [Eq.23]

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{r} \alpha_i y_i = 0$$ [Eq.24]

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0, \quad \forall i = 1..r$$ [Eq.25]

Karush-Kuhn-Tucker conditions (2)

$$y_i(\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b) - 1 + \xi_i \geq 0, \quad \forall i = 1..r$$ [Eq.26]

$$\xi_i \geq 0$$ [Eq.27]

$$\alpha_i \geq 0$$ [Eq.28]

$$\mu_i \geq 0$$ [Eq.29]

$$\alpha_i(y_i(\langle \mathbf{w} \cdot \mathbf{x_i} \rangle + b) - 1 + \xi_i) = 0$$ [Eq.30]

$$\mu_i \xi_i = 0$$ [Eq.31]

Transform from primal to dual

- As for the separable case, we transform the primal formulation to a dual
  - Setting to zero the partial derivatives of the Lagrangian ([Eq.22]) with respect to the **primal variables** (i.e., **w**, b and ξ_i)
  - Substituting the resulting relations back into the primal Lagrangian

→ i.e., substitute [Eq.23-25] into the primal Lagrangian in [Eq.22]

- From [Eq.25], we have: $C - \alpha_i - \mu_i = 0$,
  - and because $\mu_i \geq 0$,
  - we can deduce that $\alpha_i \leq C$

The dual problem

$$\text{Maximize}: \quad L_D(\boldsymbol{\alpha}) = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j \langle \mathbf{x_i} \cdot \mathbf{x_j} \rangle$$

$$\text{subject to}: \quad \begin{cases} \sum_{i=1}^{r} \alpha_i y_i = 0 \\ 0 \le \alpha_i \le C, \quad \forall i = 1..r \end{cases}$$

[Eq.32]

- Interestingly, ξ_i and theirs Lagrange multipliers μ_i do not appear in the dual formulation
  - $\rightarrow$The objective function is identical to that for the separable case!
- The only difference is the new constraints: $\alpha_i \le C$

Find the solutions for the primal variables

- The dual problem (in [Eq.32]) can be solved using *numerical methods*
- The resulting α_i values (i.e., the solution) are then used to compute **w*** and b^*
  - **w*** is computed using [Eq.23]
  - b^* is computed using the Karush-Kuhn-Tucker complementarity conditions in [Eq.30-31] …but, there is a problem here: ξ_i unknown!
- To compute b*
  - From [Eq.25&31], we deduce that ξ_i=0 if α_i<C
  - Hence, we can use a training instance $\boldsymbol{x_k}$ for which (0<α_k<C) and [Eq.30] (with ξ_k =0) to compute b^*
  - The computation is similar as for the separable case!

Important observations

- Using the equations [Eq.25-31], we can deduce that:

$$\begin{array}{llll} \text{if} & \alpha_{\mathrm{i}}=0 & \text{then} & y_i(\langle \mathbf{w}\cdot\mathbf{x_i}\rangle+b)\ge 1, \quad \text{and } \xi_{\mathrm{i}}=0 \\ \text{if} & 0<\alpha_{\mathrm{i}}<C & \text{then} & y_i(\langle \mathbf{w}\cdot\mathbf{x_i}\rangle+b)=1, \quad \text{and } \xi_{\mathrm{i}}=0 \\ \text{if} & \alpha_{\mathrm{i}}=C & \text{then} & y_i(\langle \mathbf{w}\cdot\mathbf{x_i}\rangle+b)<1, \quad \text{and } \xi_{\mathrm{i}}>0 \end{array} \qquad \text{[Eq.33]}$$

- [Eq.33] shows a very important property of SVM
  - The solution is **sparse** in α_i
    - Many training instances are <u>outside</u> the margin area, and their α_i are 0
    - For those instances that are <u>on</u> the margin (i.e., $y_i(\langle\mathbf{w}\cdot\mathbf{x}_i\rangle+b)=1$, which are the **support vectors**), their α_i are non-zero ($0<\alpha_i<C$)
    - Those instances that are <u>inside</u> the margin (i.e., $y_i(\langle\mathbf{w}\cdot\mathbf{x}_i\rangle+b)<1$) are error (noisy) instances, and their α_i are non-zero ($\alpha_i=C$)
  - Without this sparsity property, SVM would not be practical for large datasets

The decision boundary

- The decision boundary is the following hyperplane

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

  → Note that for many training instance $\boldsymbol{x_i}$, their α_i are zero! (i.e., the sparsity property of SVM)

- For a test instance ***z***, it is classified by

$$sign(\langle \mathbf{w}^* \cdot \mathbf{z} \rangle + b^*)$$

- We also need to determine the value of the parameter C (introduced in the objective function)

  → Often determined using a validation set

Linear SVMs – Summary

- The classifier is a separating hyperplane
- The separating hyperplane is defined by the set of **support vectors**
- Only for the support vectors, their Lagrange multipliers are non-zero
  - For the other training instances (non-support vectors), their Lagrange multipliers are zero
- The identification of the support vectors (i.e., from the training instances) requires solving quadratic optimization problems
- Both in the dual formulation and in the solution (i.e., the separating hyperplane), the training instances appear only inside the inner (dot)-products

Non-linear SVMs

- Recall that: the SVM formulations require linear separation
- However, in practical application problems the datasets may be non-linearly separable
- Non-linear SVM learning consists of the two main steps
  - First, to **transform the input data space into another space** (usually of a much higher dimension) so that
    - → the transformed data space is linearly separable (i.e., there exists a linear decision boundary that can separate positive and negative examples in the transformed space)
  - Second, to apply the same formulations and techniques as for the linear case
- The original data space is called the **input space**
- The transformed space is called the **feature space**

Space transformation (1)

- The basic idea is to map (i.e., transform) the data in the input space (i.e., original) X to a feature space (i.e., transformed) F by a non-linear mapping ϕ

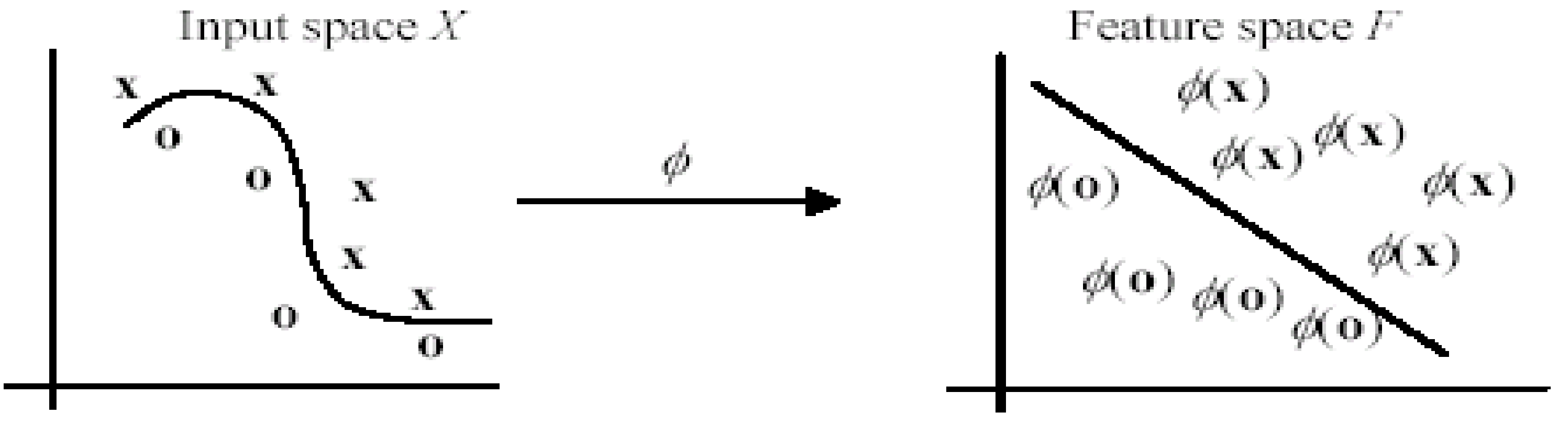
$$\phi : X \to F$$

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

- In the feature space, the set of the original training instances $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_r, y_r)\}$ is represented as:

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

Space transformation (2)



- In this example, the transformed space is also 2-D
- But usually, the dimensionality of the feature space is much larger than that of the input space

[Liu, 2006]

Non-linear SVM – Optimization problem

- After the space transformation, the optimization problem is:

$$\text{Minimize:} \quad L_P = \frac{\langle \mathbf{w} \cdot \mathbf{w} \rangle}{2} + C\sum_{i=1}^{r} \xi_i \qquad \text{[Eq.34]}$$

$$\text{subject to:} \quad \begin{cases} y_i\left(\langle \mathbf{w} \cdot \phi(\mathbf{x_i}) \rangle + b\right) \geq 1 - \xi_i, \quad \forall i = 1..r \\ \xi_i \geq 0, \quad \forall i = 1..r \end{cases}$$

- The dual problem is:

$$\text{Maximize:} \quad L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j}) \rangle \qquad \text{[Eq.35]}$$

$$\text{subject to:} \quad \begin{cases} \sum_{i=1}^{r} \alpha_i y_i = 0 \\ 0 \leq \alpha_i \leq C, \quad \forall i = 1..r \end{cases}$$

- The decision boundary (i.e., for the classification) is the separating hyperplane:

$$f(\mathbf{z}) = \langle \mathbf{w}^* \cdot \phi(\mathbf{z}) \rangle + b^* = \sum_{i=1}^{r} \alpha_i y_i \langle \phi(\mathbf{x_i}) \cdot \phi(\mathbf{z}) \rangle + b^*) = 0 \qquad \text{[Eq.36]}$$

Space transformation – Example

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

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

- Let's consider the training instance (**x**=(2, 3), y=-1) in the input space (i.e., 2-D)
- In the feature space (i.e., 3-D), the training instance is represented as:

(ϕ(**x**)=(4, 9, 8.49), y=-1)

Problem with explicit transformation

- The problem with the explicit space transformation is that it may suffer from the curse of dimensionality
- Even with a reasonable (not large) number of dimensions of the input space, some useful transformation may result in a feature space with a huge number of dimensions
  - → "useful" here means that the transformation results in a feature space that is linearly separable
- This problem makes the explicit space transformation computationally infeasible to handle
- Fortunately, the explicit transformation is not needed…

Kernel functions

- Note that in the dual formulation ([Eq.35]) and in the decision boundary ([Eq.36])
  - The explicit mapped vectors $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$ are not required
  - Only the inner (dot)-products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ is needed

  $\rightarrow$ ***The reason why an explicit transformation is not needed*!**
- If we can *compute the inner (dot)-product* $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ *directly from the input vectors **x** and **z***, then we don't need to know
  - the feature vectors $\phi(\mathbf{x})$ and $\phi(\mathbf{z})$, and
  - even the mapping (transformation) function ϕ
- In SVM, this goal is achieved through the use of **kernel functions**, denoted by *K*

$$K(\mathbf{x},\mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle \qquad \text{[Eq.37]}$$

Kernel function – Example

- Polynomial kernel

  $K(\mathbf{x},\mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d$ [Eq.38]

- Let's compute the kernel with the degree d = 2, for the two 2-D vectors: $\mathbf{x}=(x_1,x_2)$ and $\mathbf{z}=(z_1,z_2)$

$$\langle \mathbf{x} \cdot \mathbf{z} \rangle^2 = (x_1 z_1 + x_2 z_2)^2$$

$$= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2$$

$$= \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2) \cdot (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle$$

$$= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$$

- The above computation shows that the kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^2$ is an inner (dot)-product in a transformed (3-D) feature space

Kernel trick

- The computation in the previous example is just for illustration purpose
- In practice, we do not need to find the mapping function ϕ
- Because we can apply the kernel function *directly*
  - →By replacing all the inner (dot)-products $\langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle$ in [Eq.35-36] with a selected kernel function $K(\mathbf{x},\mathbf{z})$ (e.g., the polynomial kernel $\langle \mathbf{x} \cdot \mathbf{z} \rangle^d$ given in [Eq.38])
- This strategy is called the **kernel trick**!

Kernel function – How to know?

- How can we know whether a function is a kernel or not – without performing the derivation such as that illustrated in the previous example?
  - →How can we know if a (kernel) function is indeed an inner (dot)-product in some feature space?
- This question is answered by a theorem called the **Mercer's theorem**
  - → Out of the scope of this lecture!

Commonly used kernels

- Polynomial

$$K(\mathbf{x},\mathbf{z}) = \left(\langle \mathbf{x}\cdot\mathbf{z}\rangle + \theta\right)^d; \quad \text{where } \theta \in R, d \in N$$

- Gaussian RBF

$$K(\mathbf{x},\mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma}}; \quad \text{where } \sigma > 0$$

- Sigmoidal

$$K(\mathbf{x},\mathbf{z}) = \tanh(\beta\langle \mathbf{x}\cdot\mathbf{z}\rangle - \lambda) = \frac{1}{1+e^{-(\beta\langle \mathbf{x}\cdot\mathbf{z}\rangle-\lambda)}}; \quad \text{where } \beta,\lambda \in R$$

SVM classification – Issues

- SVM works only with a real-valued input space
  - → For nominal attributes, we need to convert their nominal values to numeric ones
- SVM does only two-class classification
  - → For a multi-class classification problem, we need to convert it into a number of two-class classification problems, and then solve each individually
  - → E.g., the "one-against-rest" method
- The separating hyperplane produced by SVM is hard to understand by human
  - This (hard-to-understand) problem is even much worse if kernel functions are used
  - SVM is usually used in those application problems in that human understanding of the system behavior is not highly required

SVM libraries

- **SVM light** (https://www.cs.cornell.edu/people/tj/svm_light/)
  - C++, Python, Java, Matlab (and others)
- **LIBSVM** (https://www.csie.ntu.edu.tw/~cjlin/libsvm/)
  - C++, Java, Python, Matlab (and others)
- **scikit-learn** (https://scikit-learn.org/stable/modules/svm.html)
  - Python
- **WEKA** (https://www.cs.waikato.ac.nz/ml/weka/)
  - Java

References

- B. Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer, 2006.
- C. J. C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*. Data Mining and Knowledge Discovery, 2(2): 121-167, 1998.