

UVA CS 4774: Machine Learning

S4: Lecture 21: Support Vector Machine (nonlinear) Kernel Trick and in Practice

Dr. Yanjun Qi

University of Virginia

Department of Computer Science

Module I

What Left in SVM?

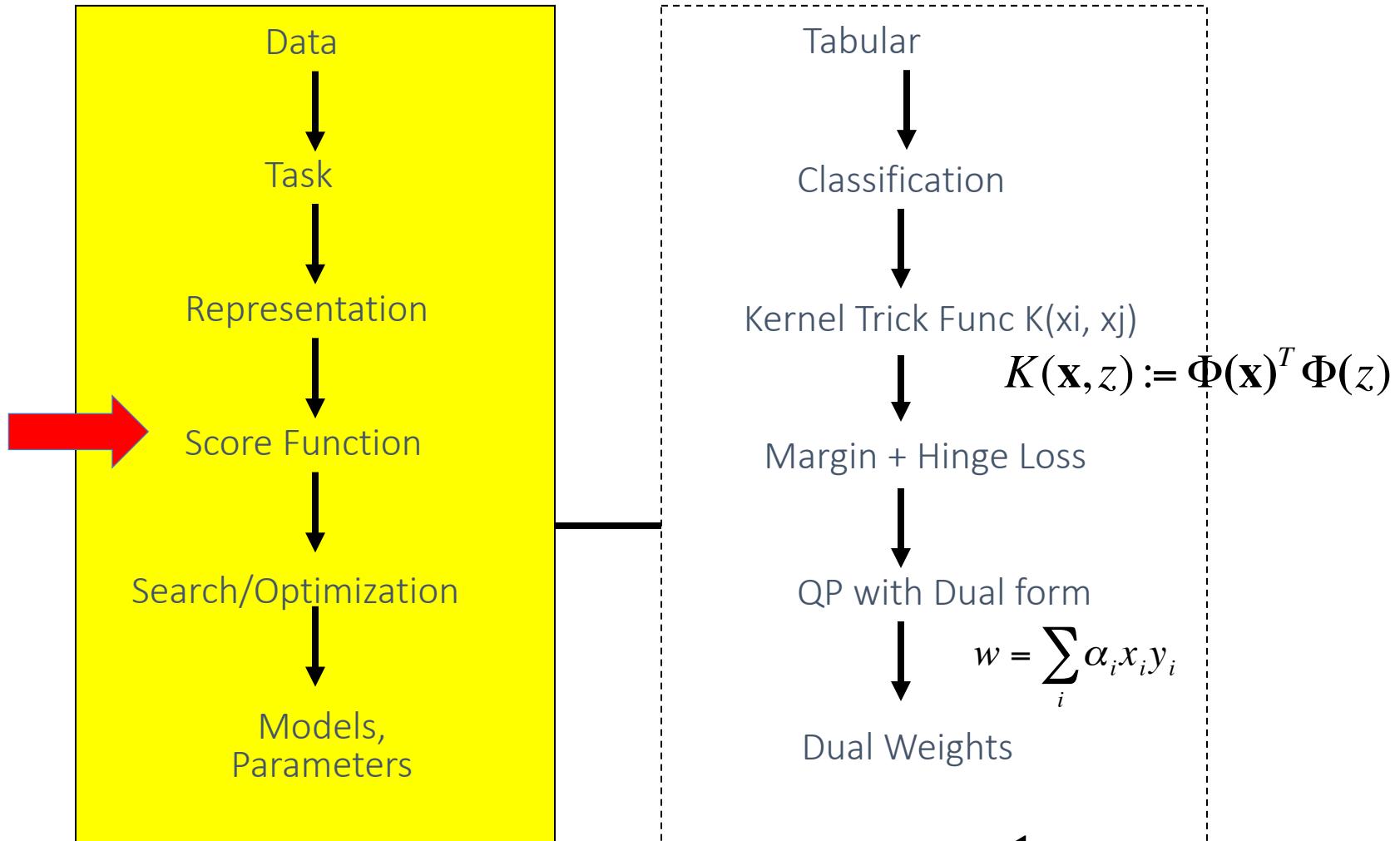
- ❑ Support Vector Machine (SVM)
 - ✓ History of SVM
 - ✓ Large Margin Linear Classifier
 - ✓ Define Margin (M) in terms of model parameter
 - ✓ Optimization to learn model parameters (w, b)
 - ✓ Linearly Non-separable case (soft SVM)
 - ✓ Optimization with dual form
 - ✓ Nonlinear decision boundary
 - ✓ Practical Guide

Today

- ❑ Support Vector Machine (SVM)
 - ✓ History of SVM
 - ✓ Large Margin Linear Classifier
 - ✓ Define Margin (M) in terms of model parameter
 - ✓ Optimization to learn model parameters (w, b)
 - ✓ Non linearly separable case (Extra)
 - ✓ Optimization with dual form (Extra)
 - ✓ Nonlinear decision boundary
 - ✓ Practical Guide



Last: Basic Support Vector Machine

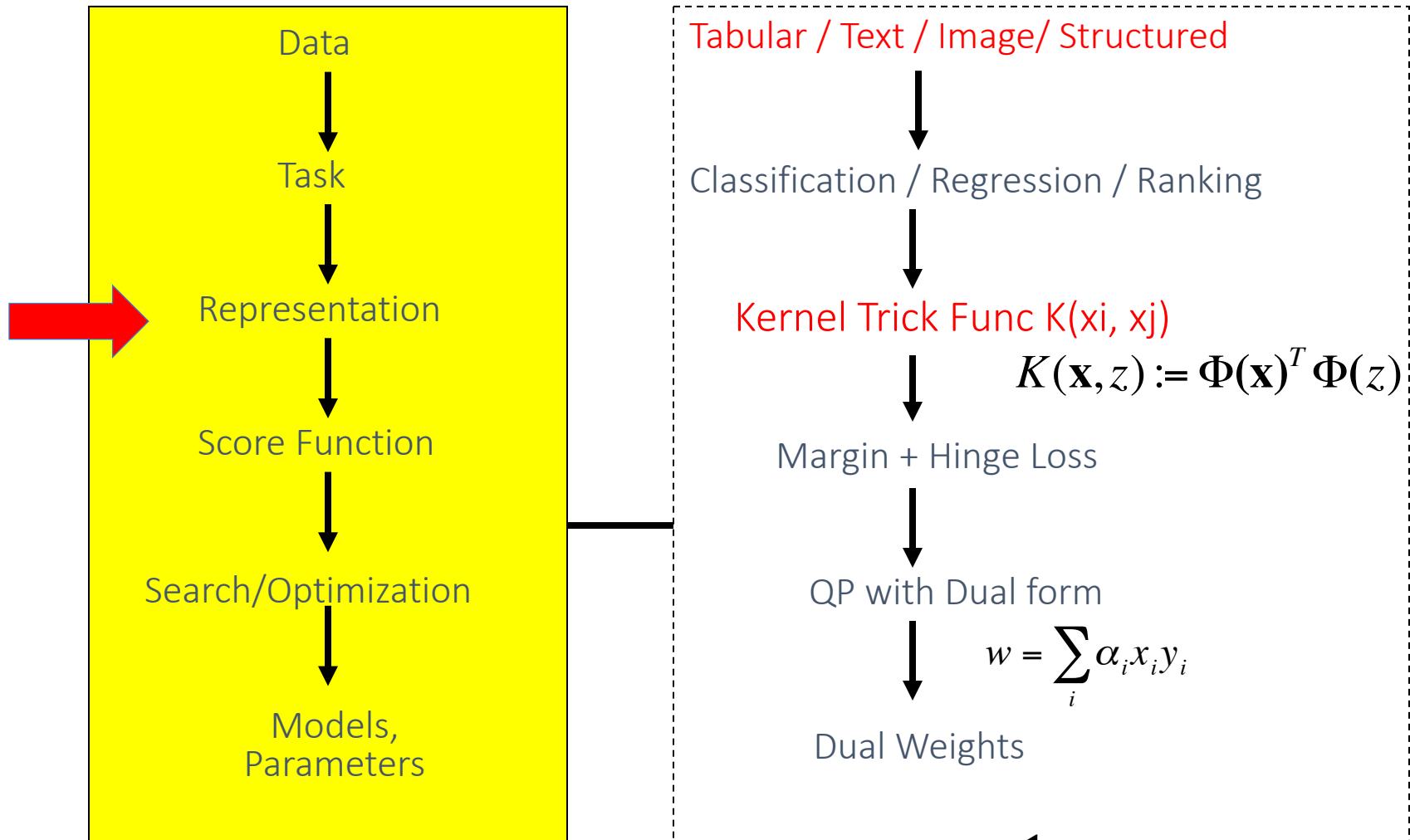


$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{i=1}^n \varepsilon_i$$

subject to $\forall \mathbf{x}_i \in D_{train} : y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$

$$\begin{aligned}
 & \max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\
 & \sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i
 \end{aligned}$$

This: Kernel Support Vector Machine



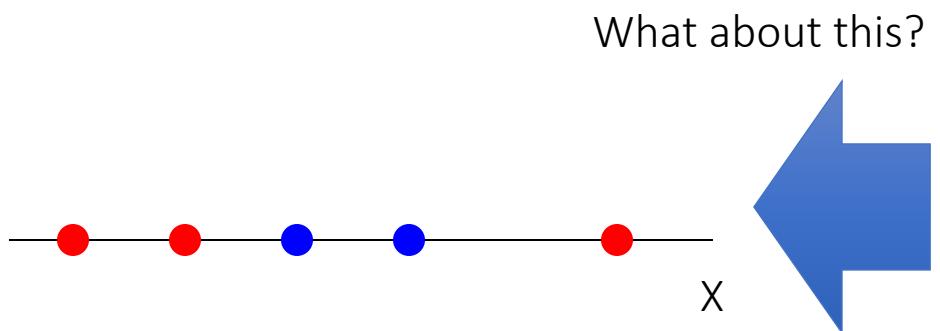
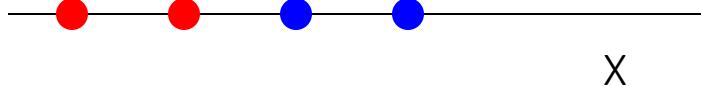
$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{i=1}^n \varepsilon_i$$

subject to $\forall \mathbf{x}_i \in D_{train} : y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$

$$\begin{aligned}
 & \max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\
 & \sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i
 \end{aligned}$$

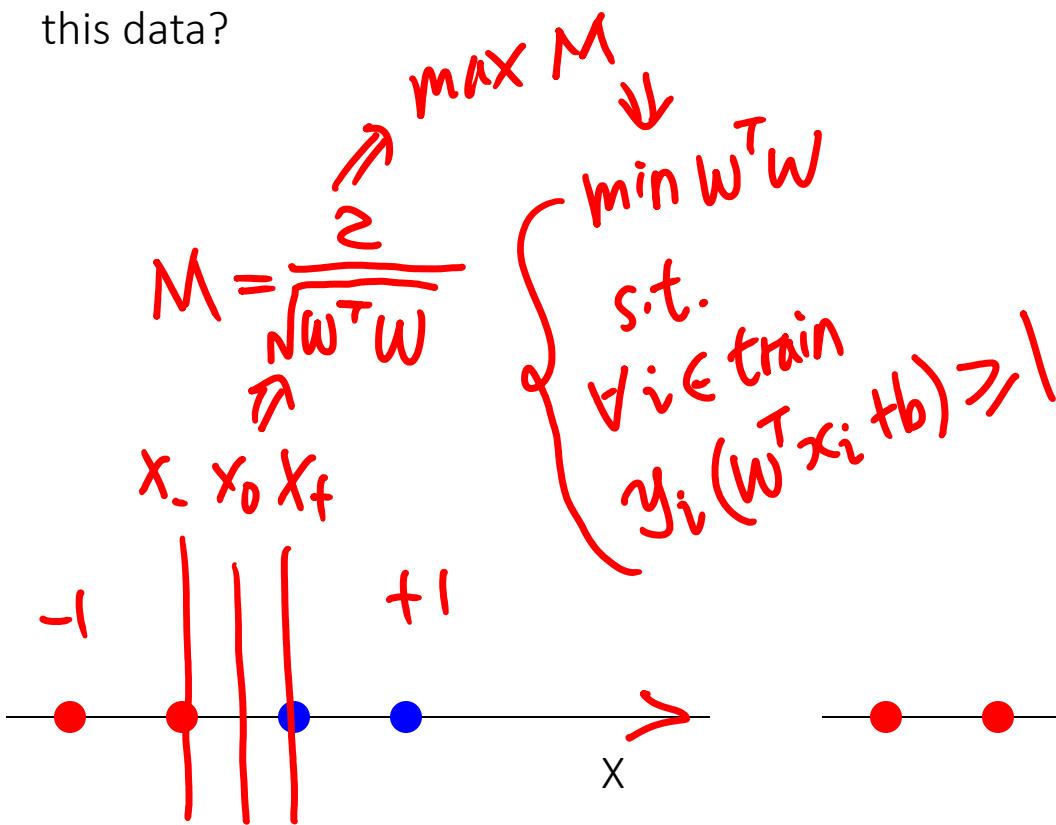
Classifying in 1-d

Can an SVM correctly classify
this data?



Classifying in 1-d

Can an SVM correctly classify this data?



What about this?

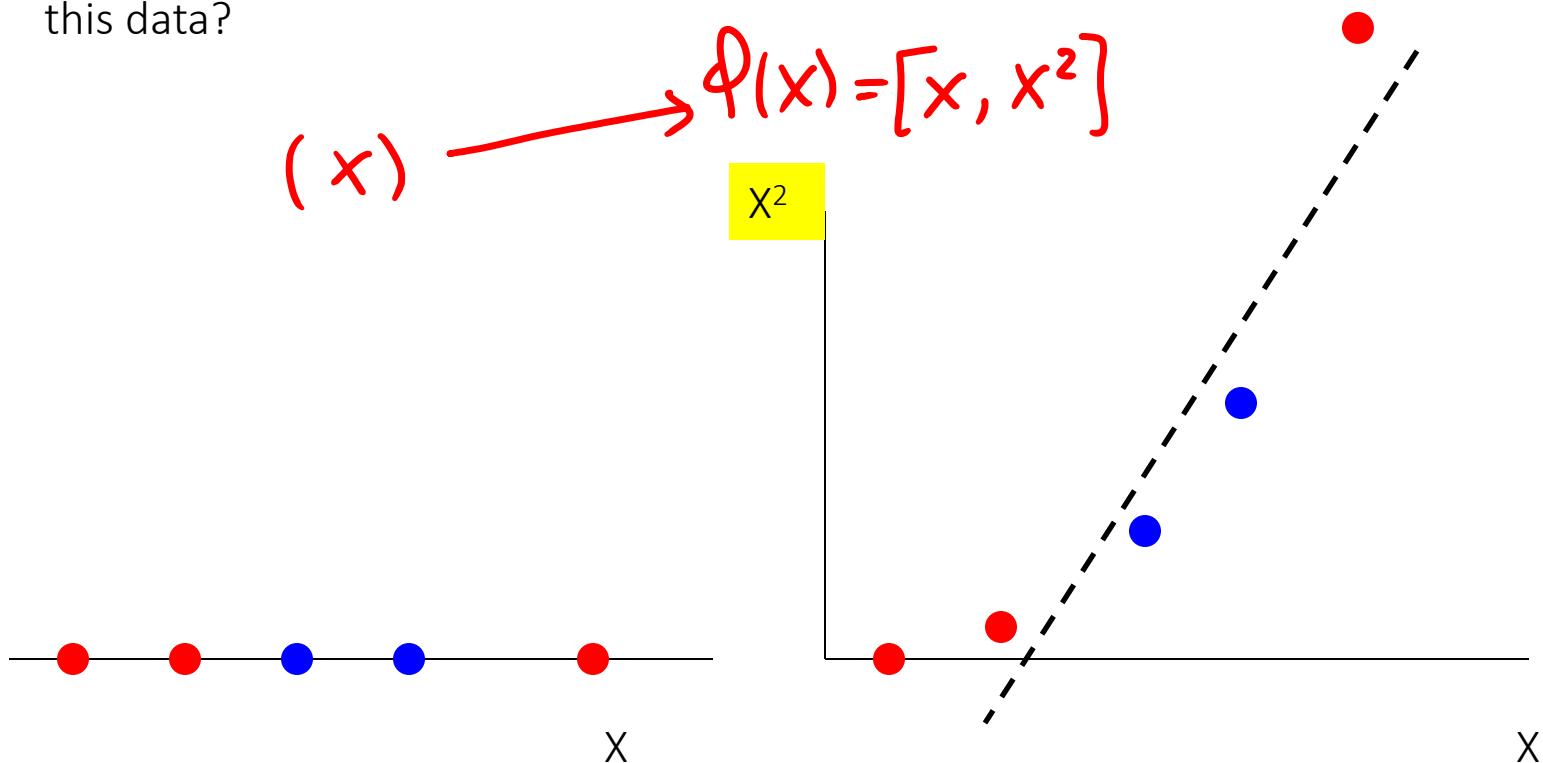


Classifying in 1-d

{
→ Separable
→ nonlinear

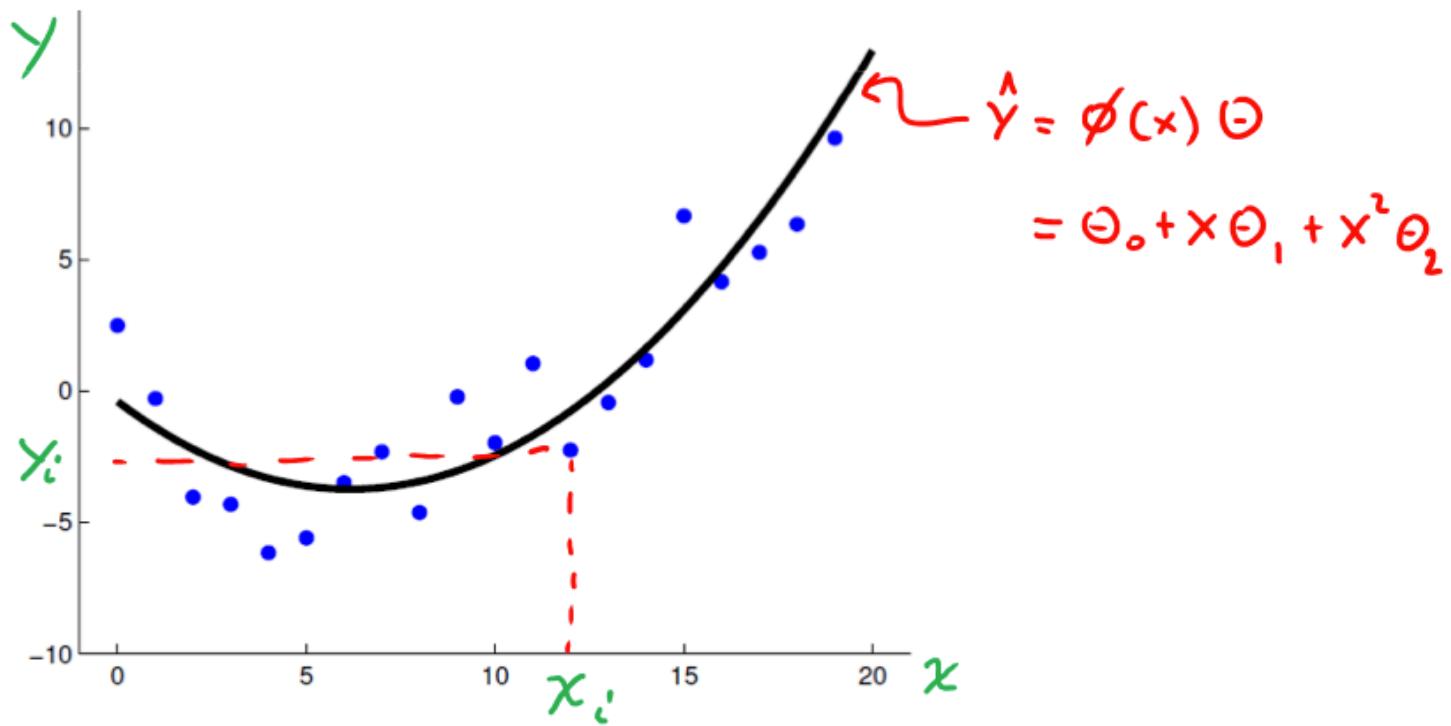
Can an SVM correctly classify
this data?

And now? (extend with polynomial basis)



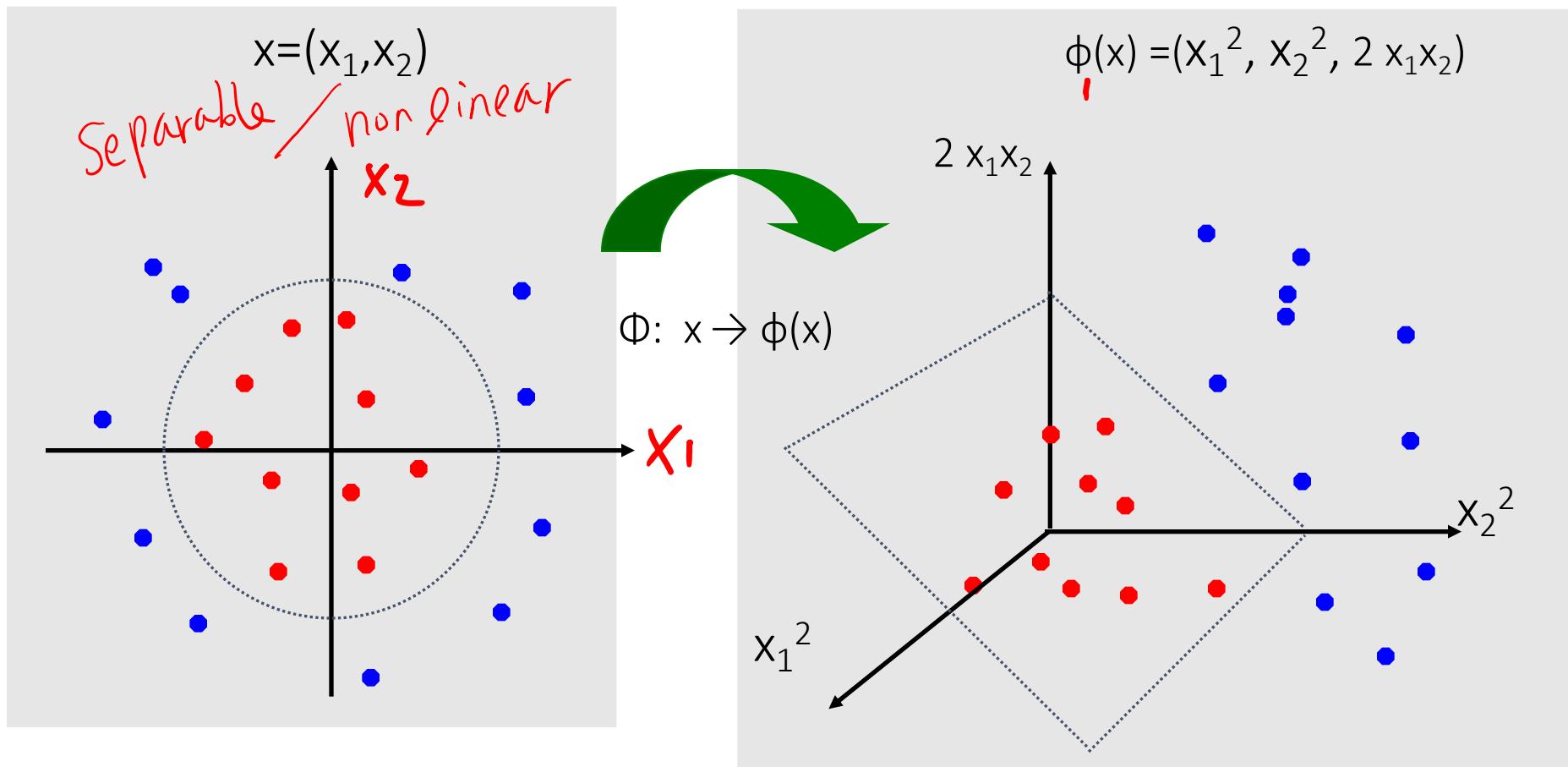
RECAP: Polynomial regression

For example, $\phi(x) = [1, x, x^2]$



Non-linear SVMs: 2D

- The original input space (x) can be mapped to some higher-dimensional feature space ($\phi(x)$) where the training set is separable:

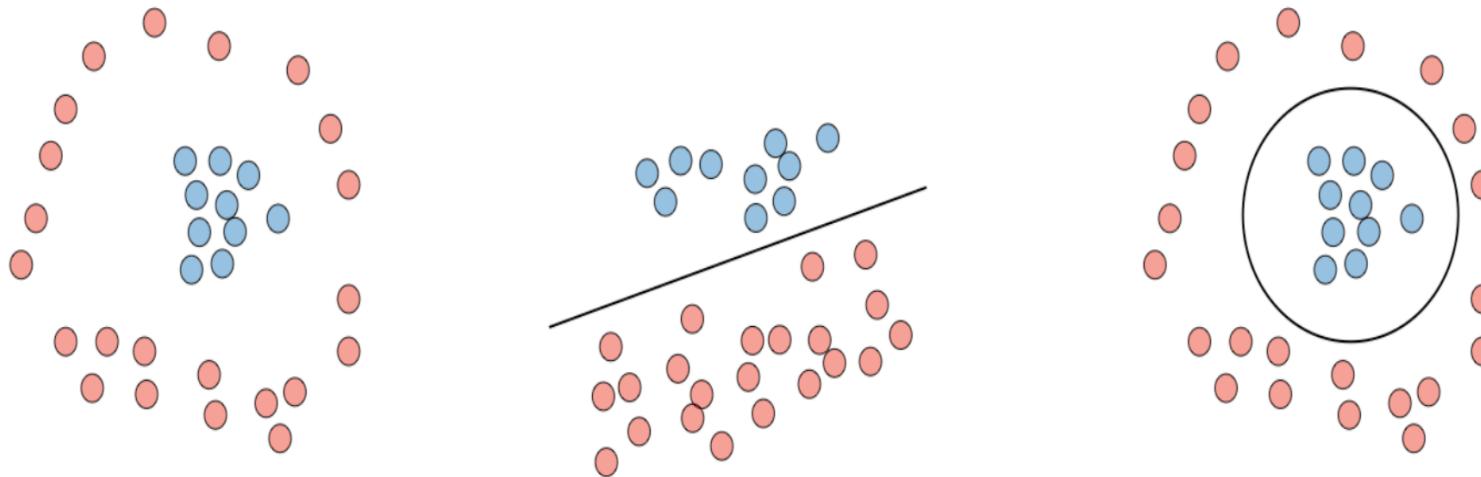


□ Kernel – Given a feature mapping ϕ , we define the kernel K to be defined as:

$$K(x, z) = \phi(x)^T \phi(z)$$

In practice, the kernel K defined by $K(x, z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used.

RBF $\phi_2(x)^T \phi_2(z)$



Non-linear separability \rightarrow Use of a kernel mapping ϕ \rightarrow Decision boundary in the original space

When

~~From~~ we say that we use the "kernel trick" to compute the cost function using the kernel ~~because~~ we actually don't need to know the explicit mapping ϕ , which is often very complicated. Instead, only the values $K(x, z)$ are needed.

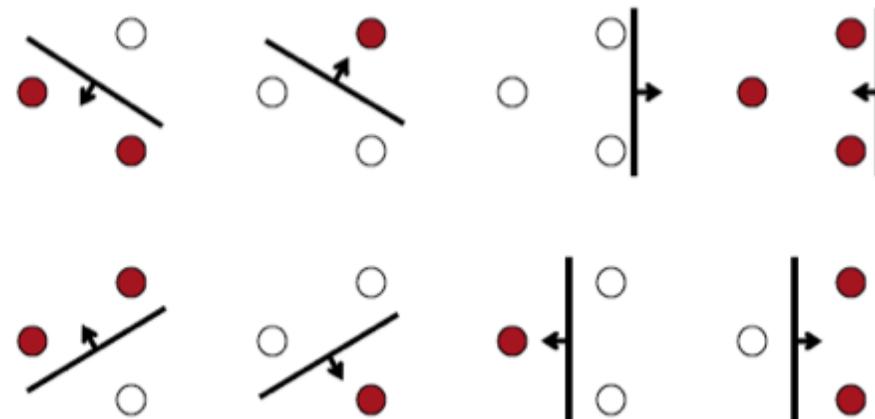
A little bit theory: Vapnik-Chervonenkis (VC) dimension

$$X \rightarrow \varphi(X) \\ \geq N-1$$

If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;

N data points are in general separable in a space of $N-1$ dimensions or more!!!

- VC dimension of the set of oriented lines in R^2 is 3
 - It can be shown that the VC dimension of the family of oriented separating hyperplanes in R^N is at least $N+1$



If data is mapped into sufficiently high dimension, then samples will in general be linearly separable;

N data points are in general separable in a space of N-1 dimensions or more!!!

$$X \rightarrow \mathcal{D}(X)$$

Linearly separated into
two classes $\{+, -\}$



Thank You

Thank you

UVA CS 4774: Machine Learning

S4: Lecture 21: Support Vector Machine (nonlinear) Kernel Trick and in Practice

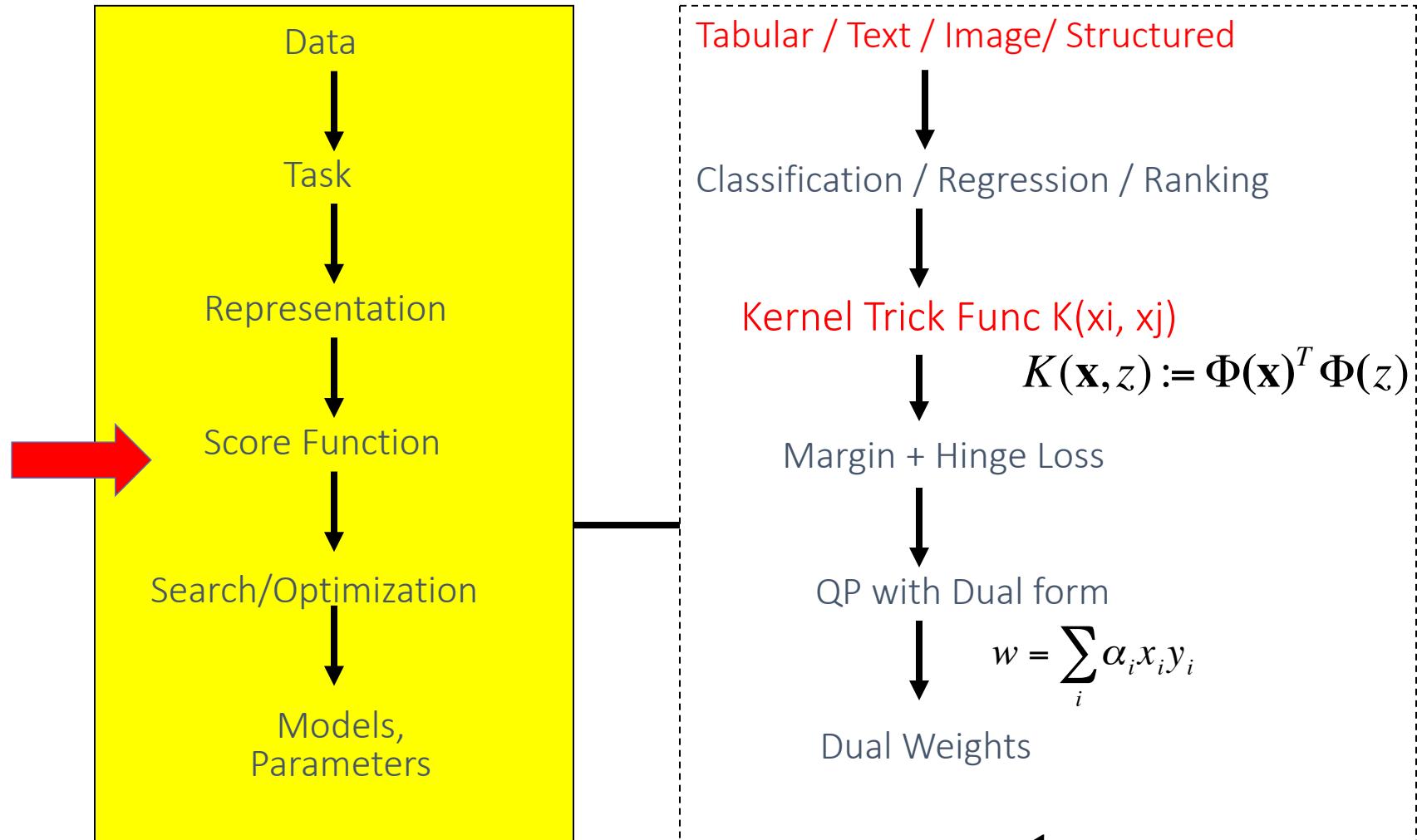
Dr. Yanjun Qi

Module II

University of Virginia

Department of Computer Science

This: Kernel Support Vector Machine



$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{j=1}^n \varepsilon_j$$

subject to $\forall \mathbf{x}_i \in Dtrain : y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \text{16} \quad \forall i$$

Optimization Reformulation (for linearly separable case)

$$x_i \rightarrow \phi(x_i)$$

$$f(x, w, b) = \text{sign}(w^T x + b)$$

1. Correctly classifies all points
2. Maximizes the margin (or equivalently minimizes $w^T w$)

$$\text{Min } (w^T w)/2$$

subject to the following constraints:

For all x in class +1

$$w^T x + b \geq 1$$

For all x in class -1

$$w^T x + b \leq -1$$

}

A total of n constraints if we have n input samples



$$y_i \in \{+1, -1\}$$

Quadratic Objective

$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 = \frac{1}{2} \mathbf{w}^T \mathbf{W}$$

$$\text{subject to } \forall \mathbf{x}_i \in D_{\text{train}} : y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

$$\underbrace{w}_{\|\mathbf{x}\|} \underbrace{\mathbf{XP}}_{\|\mathbf{x}\|} \underbrace{\mathbf{P}x}_{\|\mathbf{x}\|} \underbrace{\mathbf{x}}_{\|\mathbf{x}\|}$$

Quadratic programming i.e.,

- Quadratic objective
- Linear constraints

An alternative representation of the SVM QP

- Instead of encoding the correct classification rule and constraint we will use Lagrange multipliers to encode it as part of the our minimization problem

$$\text{Min } (\mathbf{w}^T \mathbf{w})/2$$

s.t.

$$(\mathbf{w}^T \mathbf{x}_i + b) y_i \geq 1$$

Recall that Lagrange multipliers can be applied to turn the following problem:

$\forall i, \alpha_i \geq 0$ every training

$$L_{\text{primal}}(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i \left(y_i \underbrace{(\mathbf{w}^T \mathbf{x}_i + b)}_{\geq 1} - 1 \right)$$

The Dual Problem (Extra)

$$\max_{\alpha_i \geq 0} \underbrace{\min_{w,b} L(w,b,\alpha)}$$

Dual formulation

- We minimize L with respect to w and b first:

$$\nabla_w L(w,b,\alpha) = w - \sum_{i=1}^{\text{train}} \alpha_i y_i x_i = 0, \quad (*)$$

$$\nabla_b L(w,b,\alpha) = \sum_{i=1}^{\text{train}} \alpha_i y_i = 0, \quad (**)$$

Note that $(*)$ implies:

$$w = \sum_{i=1}^{\text{train}} \alpha_i y_i x_i \quad f(x_t) = \text{Sign}(\hat{w}^T \vec{x}_t + b) \quad (***)$$

- Plus $(***)$ back to L , and using $(**)$, we have:

$$L(w,b,\alpha) = \sum_{i=1} \alpha_i - \frac{1}{2} \sum_{i,j=1} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

SVM goal

Summary: Dual SVM for linearly separable case

Dual formulation

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

$n \times i$

$$\text{Min } (\mathbf{w}^T \mathbf{w})/2$$

subject to the following inequality constraints:

For all \mathbf{x} in class +1

$$\mathbf{w}^T \mathbf{x} + b \geq 1$$

For all \mathbf{x} in class -1

$$\mathbf{w}^T \mathbf{x} + b \leq -1$$

}

A total of n constraints if we have n input samples



.

Easier than original QP, more efficient algorithms exist to find α_i ; e.g. SMO (see extra slides)

Dual SVM for linearly separable case – Training / Testing

Our dual target function:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$\{\alpha_1, \alpha_2, \dots, \alpha_n\}$

Dot product for all training samples

$$\alpha_i \geq 0 \quad \forall i$$

{ most $\alpha_i = 0$
only support vectors $\alpha_i > 0$

Dual SVM for linearly separable case – Training / Testing

Our dual target function:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

Dot product for all training samples

$$\alpha_i \geq 0 \quad \forall i$$

Dot product with ("all" ??) training samples

To evaluate a new sample \mathbf{x}_{ts} we need to compute:

$$\mathbf{w}^T \mathbf{x}_{ts} + b = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b$$

$$K(\mathbf{x}_i, \mathbf{x}_{ts}) \\ = \mathbf{\Phi}(\mathbf{x}_i)^T \mathbf{\Phi}(\mathbf{x}_{ts})$$

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in \text{Support Vectors}} \alpha_i y_i (\mathbf{x}_i^T \mathbf{x}_{ts}) + b \right)$$

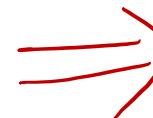
$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i$$

$$\mathbf{x}_i^T \mathbf{x}_j = \mathbf{x}_j^T \mathbf{x}_i$$

nonlinear



$$\max_{\alpha} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i$$

$$\begin{matrix} & 1 & 2 & \dots & j & n \\ \begin{matrix} 1 \\ 2 \\ \vdots \\ n \end{matrix} & \left[\begin{array}{cccccc} & & & & & \\ & & & & & \\ & & & & & \\ & & & & x_i^T x_j & \\ & & & & & \\ & & & & & \\ & & & & & \end{array} \right] & \end{matrix}$$

$n \times h$

$$O(p \cdot n^2 / 2)$$

Training

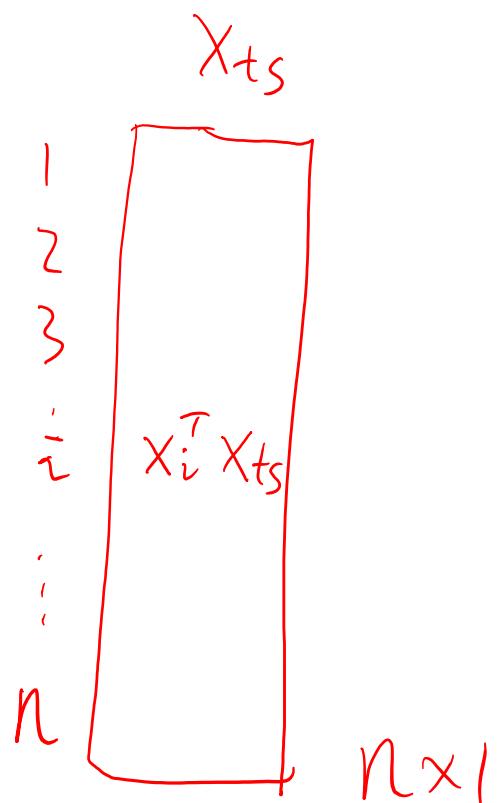
$$\begin{matrix} & 1 & 2 & \dots & j & n \\ \begin{matrix} 1 \\ 2 \\ \vdots \\ n \end{matrix} & \left[\begin{array}{cccccc} & & & & & \\ & & & & & \\ & & & & & \\ & & & & \Phi^T(\mathbf{x}_i) \Phi(\mathbf{x}_j) & \\ & & & & & \\ & & & & & \\ & & & & & \end{array} \right] & \end{matrix}$$

$n \times n$

$$O(p \cdot n^2 / 2)$$

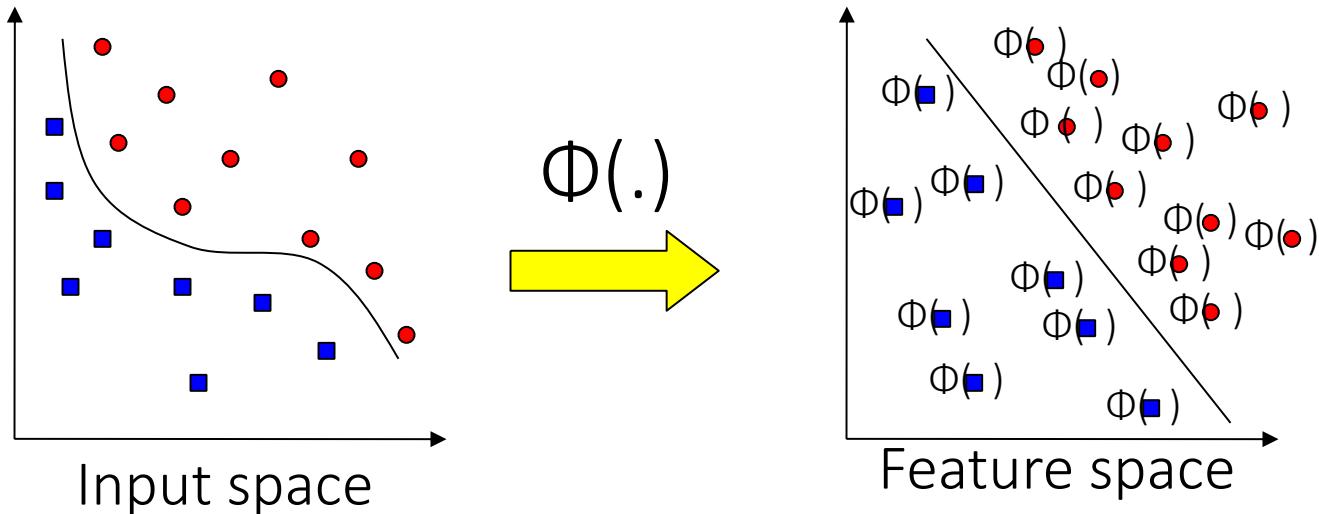
$$\mathbf{w}^T \mathbf{x}_{ts} + b = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b$$

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in SupportVectors} \alpha_i y_i (\mathbf{x}_i^T \mathbf{x}_{ts}) + b \right)$$



$$\Rightarrow \sum_{SV} \alpha_i y_i \underbrace{\Phi(x_i) \Phi(x_{ts})^T}_{+b}$$

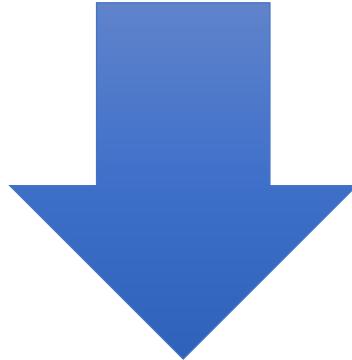
Testing



SVM solves these two issues simultaneously

- “Kernel tricks” for efficient computation
- Dual formulation only assigns parameters to samples, not to features

- SVM solves these two issues simultaneously
 - “Kernel tricks” for efficient computation
 - Dual formulation only assigns parameters to samples, not features



(1). “Kernel tricks” for efficient computation

Never represent features explicitly

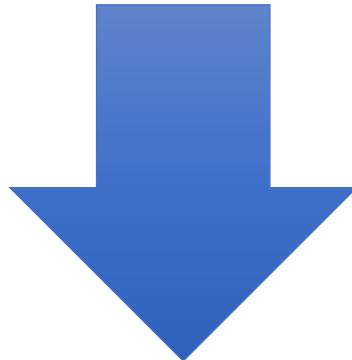
☒ Compute dot products in closed form

Very interesting theory – Reproducing Kernel Hilbert Spaces

☒ Not covered in detail here

$$k(x, \gamma)$$

- SVM solves these two issues simultaneously
 - “Kernel tricks” for efficient computation
 - Dual formulation only assigns parameters to samples, not features



(1). “Kernel tricks” for efficient computation

Never represent features explicitly

- ☒ Compute dot products in closed form
- Very interesting theory – Reproducing Kernel Hilbert Spaces
- ☒ Not covered in detail here

$$k(x, \gamma)$$

$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is called the kernel function.

- Linear kernel (we've seen it)

$$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z}$$

$\left\{ \begin{array}{l} \mathbf{x} \in \mathbb{R}^P \\ \mathbf{z} \in \mathbb{R}^P \end{array} \right.$

- Polynomial kernel (we will see an example)

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^d = \underbrace{\Phi_p(\mathbf{x})^T}_{O(p)} \underbrace{\Phi_p(\mathbf{z})}_{P_d \rightarrow O(p^d)}$$

where $d = 2, 3, \dots$. To get the feature vectors we concatenate all d th order polynomial terms of the components of \mathbf{x} (weighted appropriately)

- Radial basis kernel

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-r \|\mathbf{x} - \mathbf{z}\|^2\right) = \underbrace{\Phi_r(\mathbf{x})^T}_{O(p)} \underbrace{\Phi_r(\mathbf{z})}_{P_d = \infty}$$

In this case., r is hyperpara. The feature space of the RBF kernel has an infinite number of dimensions

Never represent features explicitly

☒ Compute dot products with a closed form

Very interesting theory – Reproducing Kernel Hilbert Spaces

☒ Not covered in detail here

Example: Quadratic kernels

$$K(\mathbf{x}, z) = (1 + \mathbf{x}^T z)^d \quad \rightarrow \quad (1 + \mathbf{x}^T z)^2$$

$$K(\mathbf{x}, z) := \Phi(\mathbf{x})^T \Phi(z)$$

- Consider all quadratic terms for $x_1, x_2 \dots x_p$

$$\max_{\alpha} \sum_i \alpha_i - \sum_{i,j} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0 \quad \forall i$$

$$\Phi(x) = \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \vdots \\ \sqrt{2}x_p \\ x_1^2 \\ \vdots \\ x_p^2 \\ \sqrt{2}x_1 x_2 \\ \vdots \\ \sqrt{2}x_{p-1} x_p \end{bmatrix}$$

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^2, \quad [d=2], \quad [P=2] \quad \begin{cases} \mathbf{x} = (x_1, x_2) \\ \mathbf{z} = (z_1, z_2) \end{cases}$$

$k(x, z) = (1 + x_1 z_1 + x_2 z_2)^2 \Rightarrow O(P)$

$$O(P^2) = \begin{pmatrix} 1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2 \end{pmatrix}^T$$

$$(1, \sqrt{2}z_1, \sqrt{2}z_2, z_1^2, z_2^2, \sqrt{2}z_1 z_2)$$

$$= \underbrace{\Phi(x)^T \Phi(z)}$$

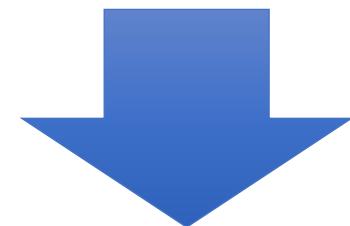
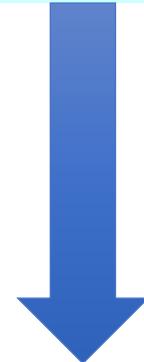
$$\Phi(\mathbf{x})^T \Phi(\mathbf{z})$$

The kernel trick

$O(p^d * n^2)$ operations if using the basis function representations in building a poly-kernel matrix

So, if we define the **kernel function** as follows, there is no need to carry out basis function explicitly

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^d$$



$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

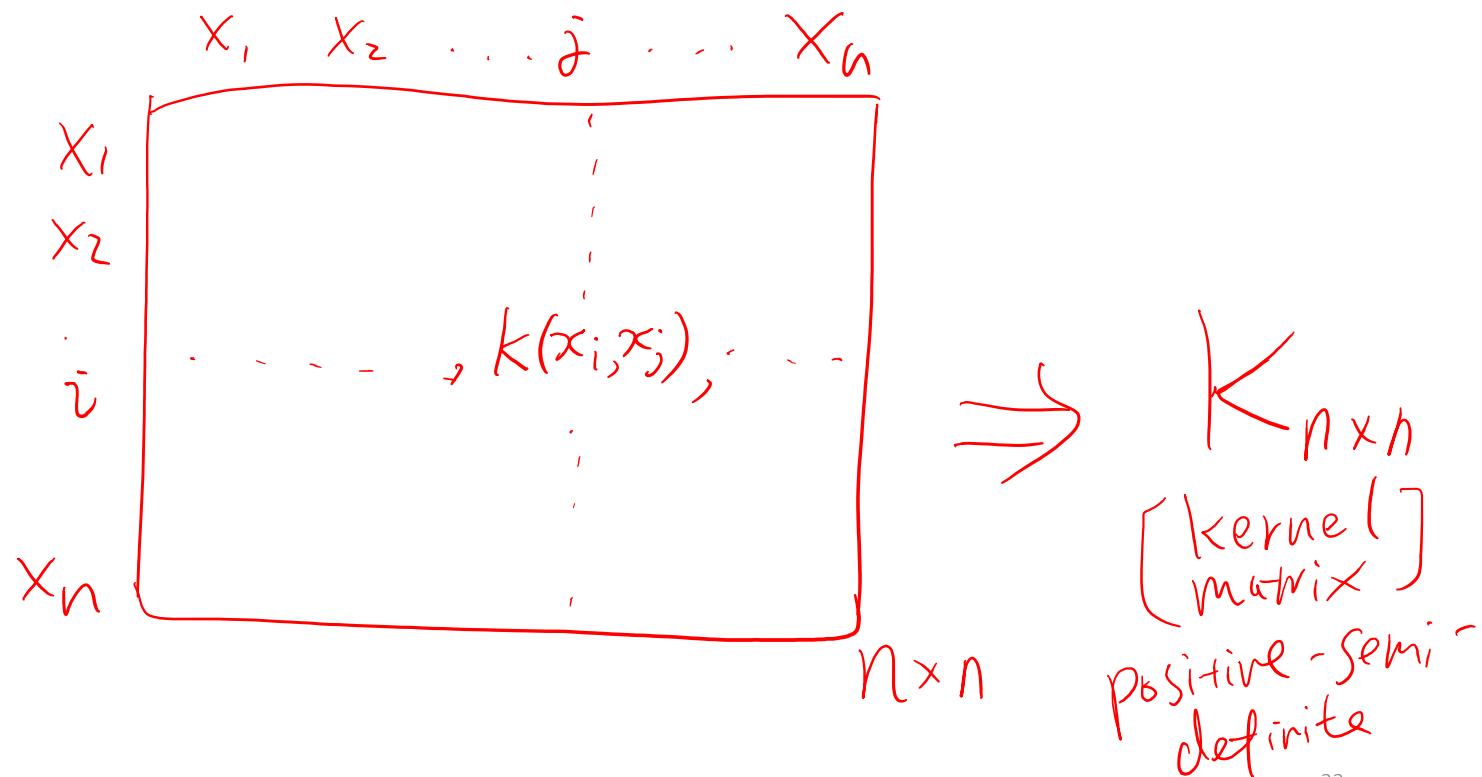
$$C > \alpha_i \geq 0, \forall i \in \text{train}$$

$O(p * n^2)$ operations if building a poly-kernel matrix directly through the $K(\mathbf{x}, \mathbf{z})$ function among n training samples →

This is because $\mathbf{x}^T \mathbf{z}$ gives a scalar, then its power of d only costs constant FLOPS.

Kernel Matrix

- Kernel function creates the kernel matrix, which summarize all the (train) data



Summary: Modification Due to Kernel Trick

- Change all inner products to kernel functions
- For training,

Original
Linear

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

With kernel
function -
nonlinear

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

VC: $\Phi(x)$
large

Summary: Modification Due to Kernel Trick

- For testing, the new data \mathbf{x}_{ts}

Original
Linear

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in \text{supportVectors}} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b \right)$$

With kernel
function -
nonlinear

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in \text{supportVectors}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_{ts}) + b \right)$$

Kernel Trick: Implicit Basis Representation

- For some kernels (e.g. RBF) the implicit transform basis form $\phi(\mathbf{x})$ is infinite-dimensional!
 - But calculations with kernel are done in original space, so computational burden and curse of dimensionality aren't a problem.

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-r\|\mathbf{x} - \mathbf{z}\|^2\right)$$

$O(p * n^2)$ operations in building a RBF-kernel matrix for training

➔ Gaussian RBF Kernel corresponds to an infinite-dimensional vector space.

YouTube video of Caltech: Abu-Mostafa explaining this in more detail
<https://www.youtube.com/watch?v=XUj5JbQihIU&t=25m53s>

Kernel Functions (Extra)

- In practical use of SVM, only the kernel function (and not basis function) is specified
- Kernel function can be thought of as a similarity measure between the input objects
- Not all similarity measure can be used as kernel function, however Mercer's condition states that any positive semi-definite kernel $K(x, y)$, i.e.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

can be expressed as a dot product in a high dimensional space.

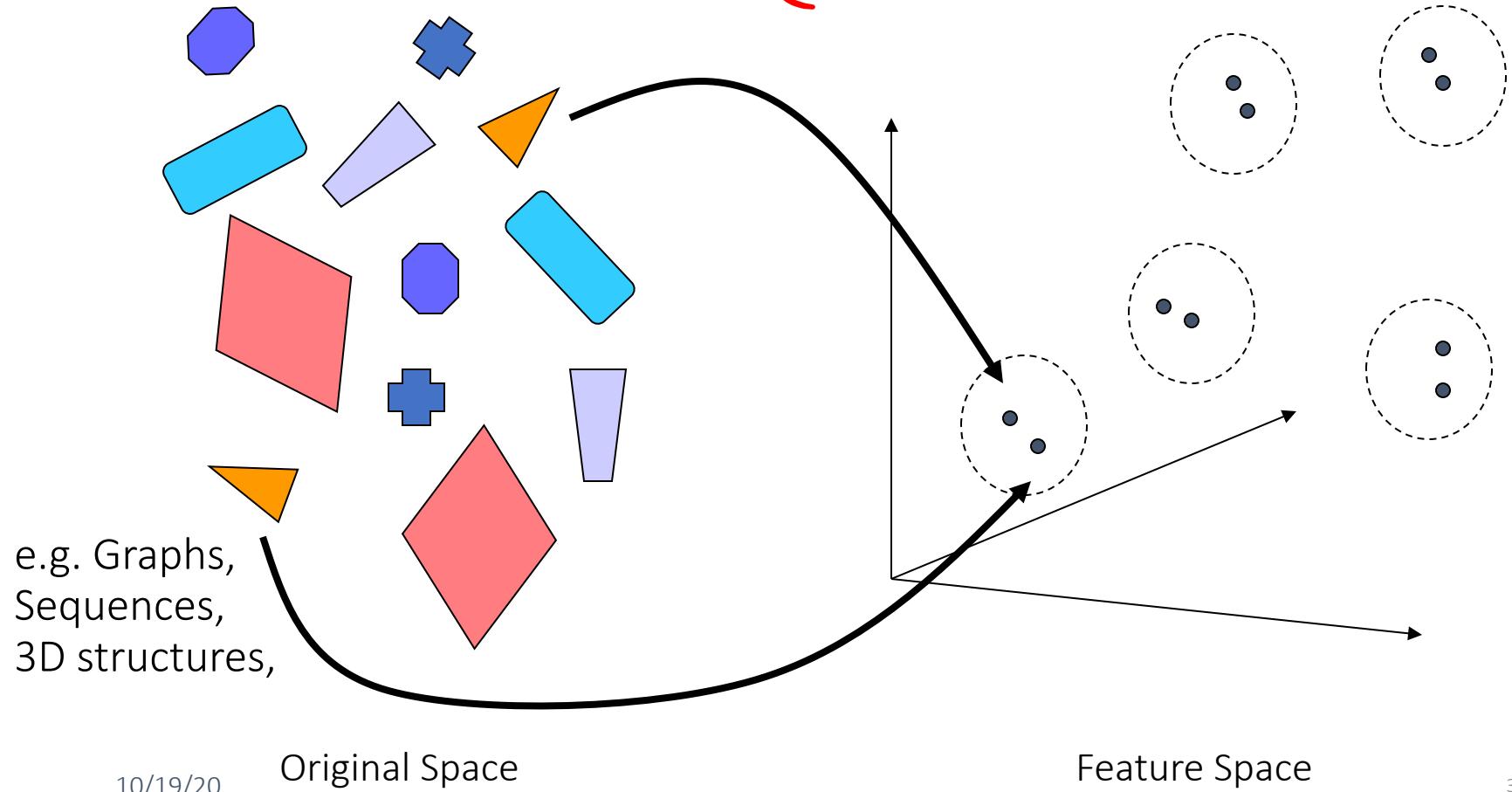
Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarize all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, tree kernel, graph kernel, ...)
 - Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors
- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try for most applications.

Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors

humuice X not available

Vector vs. Relational data



Mercer Kernel vs. Smoothing Kernel (Extra)

- The Kernels used in Support Vector Machines are different from the Kernels used in LocalWeighted /Kernel Regression.
- We can think
 - Support Vector Machines' kernels as **Mercer Kernels**
 - Local Weighted / Kernel Regression's kernels as **Smoothing Kernels**

Why do SVMs work?

- ❑ If we are using **huge** features spaces (e.g., with kernels), how come we are **not overfitting** the data?
 - ✓ Number of parameters remains the same (and most are set to 0)
 - ✓ While we have a lot of inputs, **at the end we only care about the support vectors** and these are usually a small group of samples
 - ✓ The maximizing of the margin acts as a sort of regularization term leading to reduced overfitting

kNN : $\hat{y}_{ts} = \frac{1}{k} \sum_{i \in k \text{ Neighbors of } x_{ts}} y_i$

find k neighbor of x_{ts} $\sim O(n^{\frac{1}{2}})$

SVM : $\hat{y}_{ts} = \sum_{i \in SV} \alpha_i y_i k(\vec{x}_i, \vec{x}_{ts}) + b$

para $\sim O(n)$

Logistic Regression / Linear Classifier
 $\hat{y}_{ts} = \sigma(W^T x_{ts} + b)$

para $\sim O(p)$

Time Cost Comparisons

	$x_i^T x_j$	$\underline{\phi(x_i)}^T \underline{\phi(x_j)}$	$k(x_i, x_j)$	explicit
Training Stage	$O(p \cdot n^2)$	$O(m \cdot n^2)$ polynomial $m \sim p^d$	$O(p \cdot n^2)$	w
Test Stage	$O(p \cdot \#SV)$	$O(m \cdot \#SV)$	$O(p \cdot \#SV)$	$\underline{w}^T \underline{\phi(x_{\text{test}})} + b$ ① ② ③ for some (b) RBF, $m \sim \infty$

Why SVM Works? (Extra)

- Vapnik argues that the fundamental problem is not the number of parameters to be estimated. Rather, the problem is about the flexibility of a classifier
- Vapnik argues that the flexibility of a classifier should not be characterized by the number of parameters, but by the capacity of a classifier
 - This is formalized by the “VC-dimension” of a classifier
- The SVM objective can also be justified by structural risk minimization: the empirical risk (training error), plus a term related to the generalization ability of the classifier, is minimized
- Another view: the SVM loss function is analogous to ridge regression. The term $\frac{1}{2}||w||^2$ “shrinks” the parameters towards zero to avoid overfitting



Thank You

Thank you

UVA CS 4774: Machine Learning

S4: Lecture 21: Support Vector Machine (nonlinear) Kernel Trick and in Practice

Dr. Yanjun Qi

Module III

University of Virginia

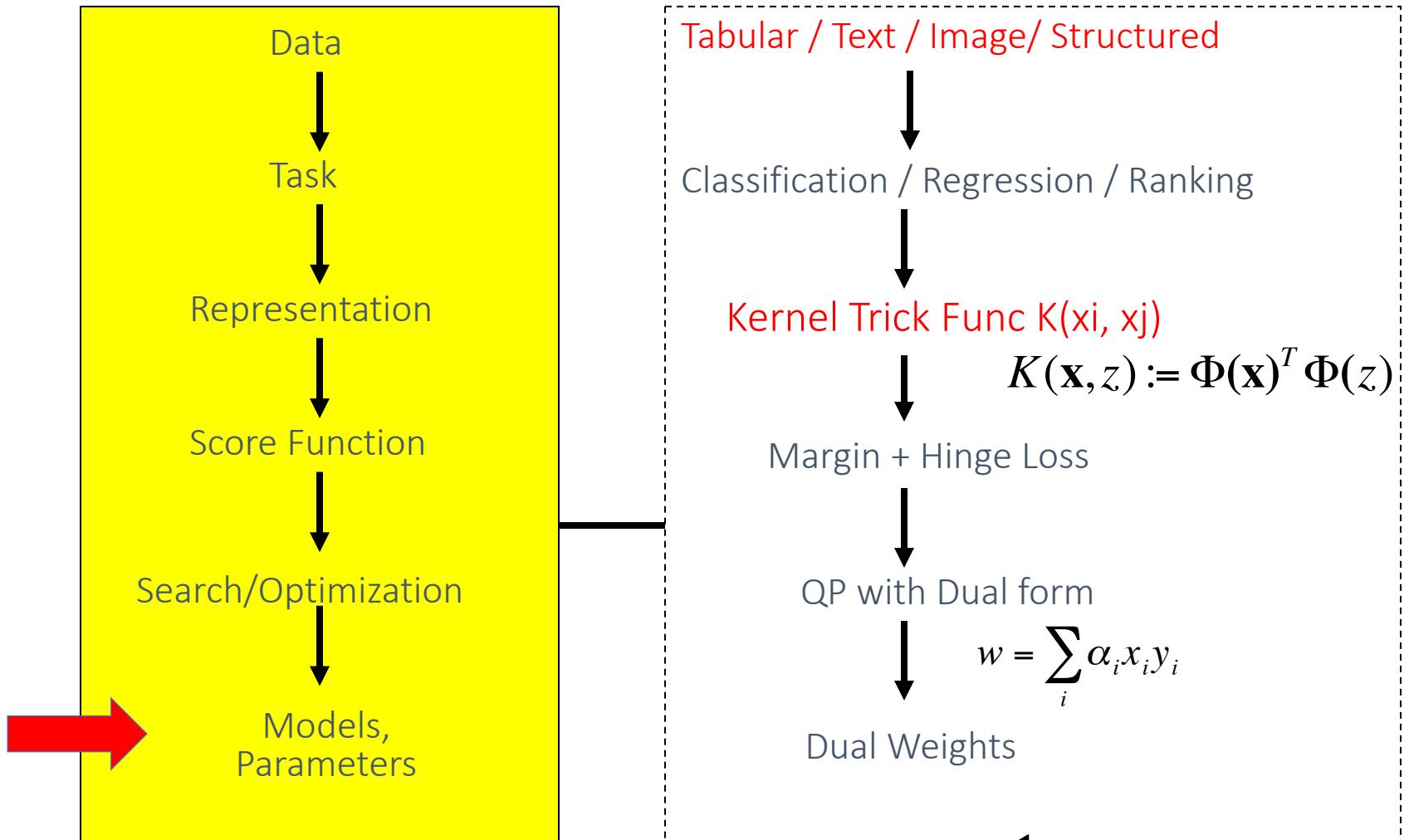
Department of Computer Science

Today

- ❑ Support Vector Machine (SVM)
 - ✓ History of SVM
 - ✓ Large Margin Linear Classifier
 - ✓ Define Margin (M) in terms of model parameter
 - ✓ Optimization to learn model parameters (w, b)
 - ✓ Non linearly separable case
 - ✓ Optimization with dual form
 - ✓ Nonlinear decision boundary
 - ✓ Practical Guide



This: Kernel Support Vector Machine



$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{i=1}^n \varepsilon_i$$

$$\text{subject to } \forall \mathbf{x}_i \in D_{train}: y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$$

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i$$

Software

- A list of SVM implementation can be found at
 - <http://www.kernel-machines.org/software.html>
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Using SVM in HW

- Prepare the feature-data matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of C (*see next 11c slides for details*)
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

Practical Guide to SVM

- From authors of LIBSVM:
 - A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
 - <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>

LIBSVM

- <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

- ✓ Developed by Chih-Jen Lin etc.
- ✓ Tools for Support Vector classification
- ✓ Also support multi-class classification
- ✓ C++/Java/Python/Matlab/Perl wrappers
- ✓ Linux/UNIX/Windows
- ✓ SMO implementation, fast!!!

(a) Data file formats for LIBSVM

- Training.dat

+1 1:0.708333 2:1 3:1 4:-0.320755

-1 1:0.583333 2:-1 4:-0.603774 5:1

+1 1:0.166667 2:1 3:-0.333333 4:-0.433962

-1 1:0.458333 2:1 3:1 4:-0.358491 5:0.374429

...

- Testing.dat

(b) Feature Preprocessing

- (1) Categorical Feature
 - Recommend using m numbers to represent an m -category attribute.
 - Only one of the m numbers is one, and others are zero.
 - For example, a three-category attribute such as {red, green, blue} can be represented as (0,0,1), (0,1,0), and (1,0,0)

Feature Preprocessing

- (2) Scaling before applying SVM is very important
 - to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.
 - to avoid numerical difficulties during the calculation
 - Recommend linearly scaling each attribute to the range [1, +1] or [0, 1].

e.g.

$$\left[\frac{X - X_{\min}}{\max - X_{\min}} \right]$$

① Normalization $\rightarrow \begin{cases} \text{mean } 0 \\ \text{std } 1 \end{cases}$

② Scaling \rightarrow linear $\Rightarrow [ax+b]$

For i -th feature \Rightarrow [Column operation
on $\sum_{n \times p}$]

{ Centering : $X_i - \bar{X}_i \Rightarrow E(X_i) = 0$

Scaling : $aX_i + b \Rightarrow$ e.g. $\frac{X_i - \min(X_i)}{\max(X_i) - \min(X_i)}$

Normalization : $\Rightarrow \begin{cases} E(X_i) = 0 \\ \text{var}(X_i) = 1 \end{cases}$

[good practice : never touch
test samples in
any stage before testing]

Of course we have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from $[-10, +10]$ to $[-1, +1]$. If the first attribute of testing data lies in the range $[-11, +8]$, we must scale the testing data to $[-1.1, +0.8]$. See Appendix B for some real examples.

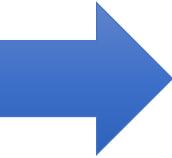
If training and testing sets are separately scaled to $[0, 1]$, the resulting accuracy is lower than 70%.

```
$ ./svm-scale -l 0 svmguide4 > svmguide4.scale
$ ./svm-scale -l 0 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 69.2308% (216/312) (classification)
```

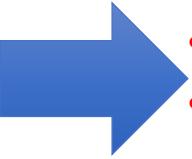
Using the same scaling factors for training and testing sets, we obtain much better accuracy.

```
$ ./svm-scale -l 0 -s range4 svmguide4 > svmguide4.scale
$ ./svm-scale -r range4 svmguide4.t > svmguide4.t.scale
$ python easy.py svmguide4.scale svmguide4.t.scale
Accuracy = 89.4231% (279/312) (classification)
```

Feature Preprocessing

- 
- (3) missing value
 - Very very tricky !
 - **Easy way:** to substitute the missing values by the mean value of the variable
 - A little bit harder way: imputation using nearest neighbors
 - Even more complex: e.g. EM based (beyond the scope)

Feature Preprocessing

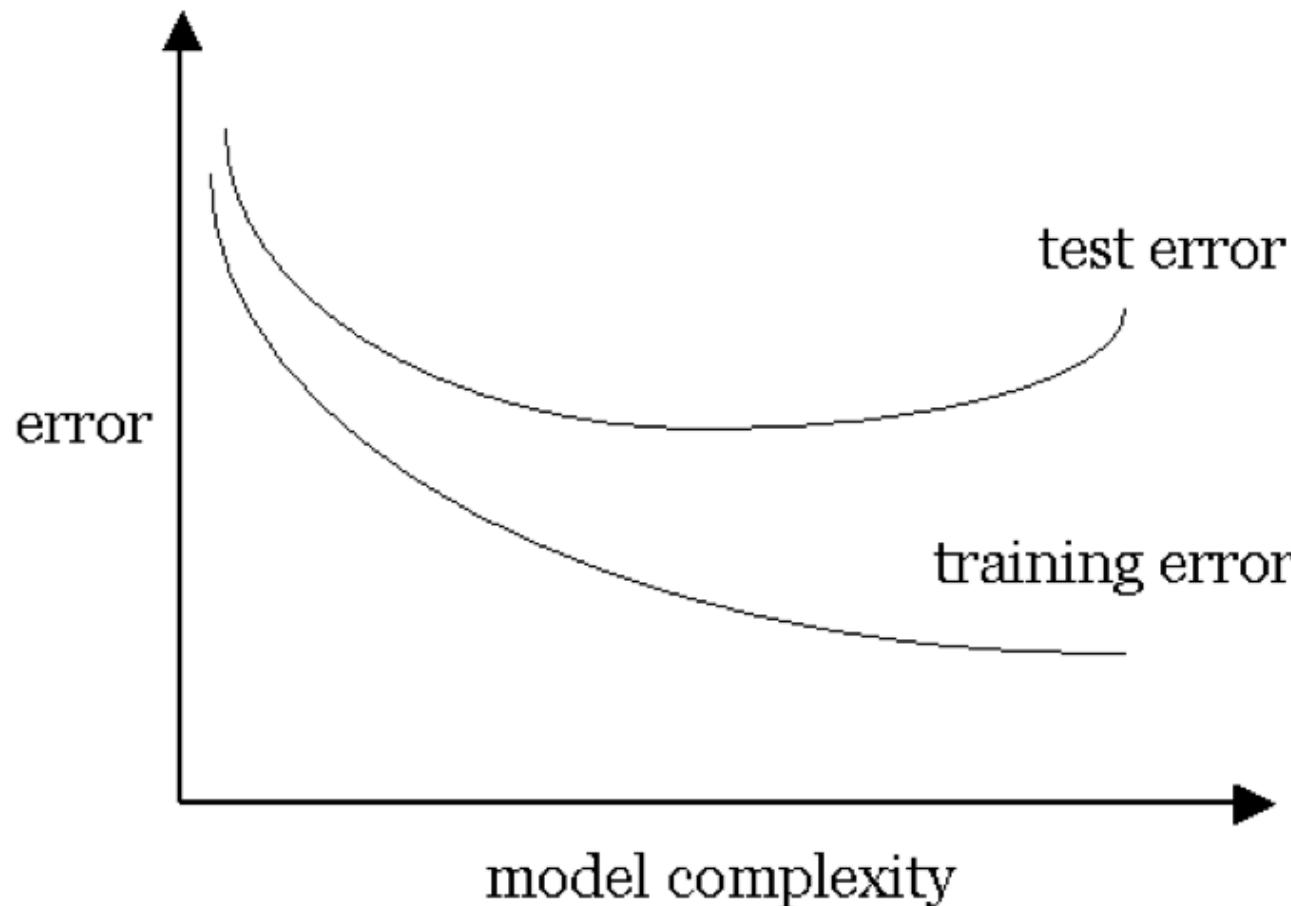
- (4) out of dictionary token issue
 - For discrete feature variable, very trick to handle
 - **Easy way:** to substitute the values by the most likely value (in train) of the variable
 - **Easy way:** to substitute the values by a random value (in train) of the variable
 - More solutions later in the NaiveBayes slides!
- 

Practical Guide to SVM

- From authors of **as LIBSVM**:
 - A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
 - <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf>

(c) Model Selection

Our goal: find the model M which minimizes the test error:

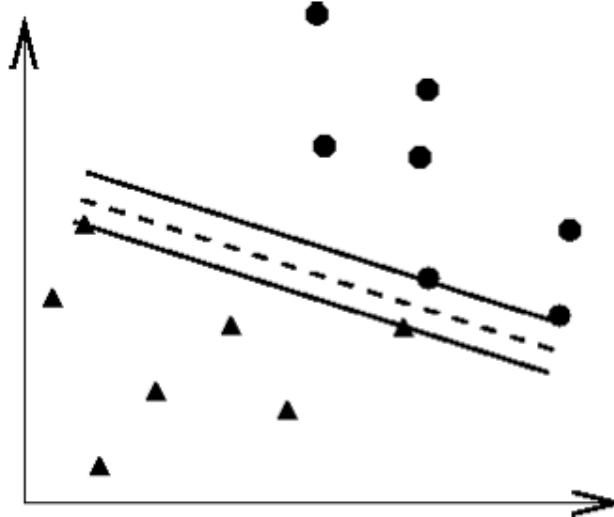


Model Selection, find right C

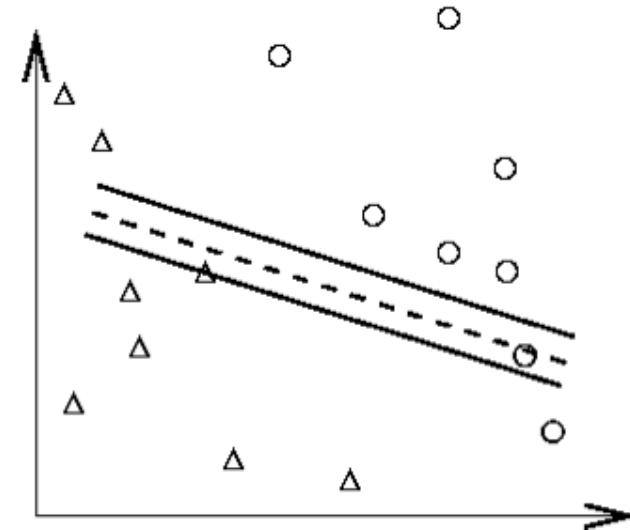
large C

Select the
right
penalty
parameter
 C

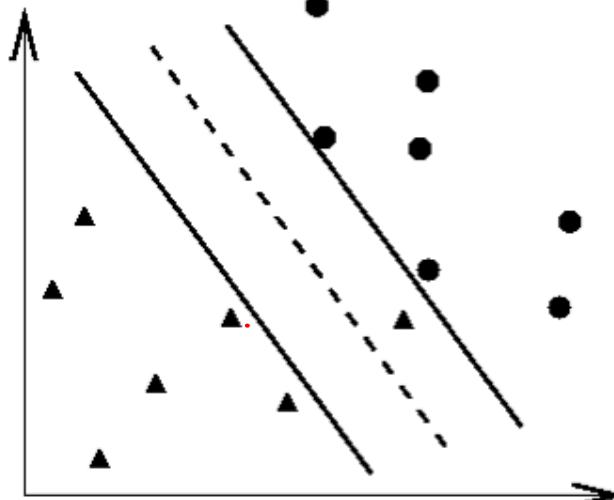
small C



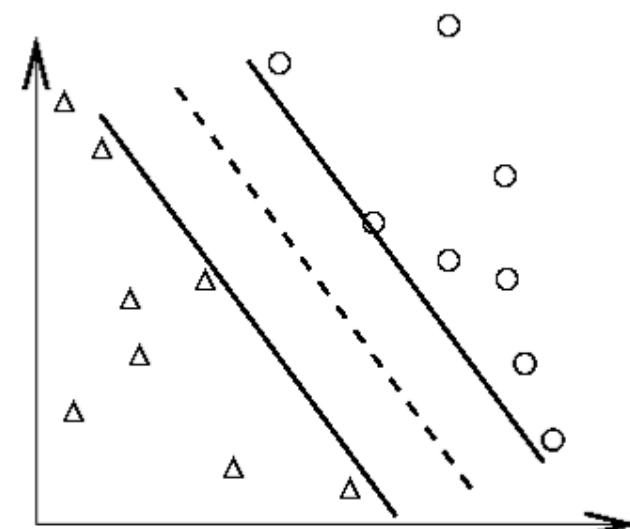
(a) Training data and an overfitting classifier



(b) Applying an overfitting classifier on testing data



(c) Training data and a better classifier



(d) Applying a better classifier on testing data

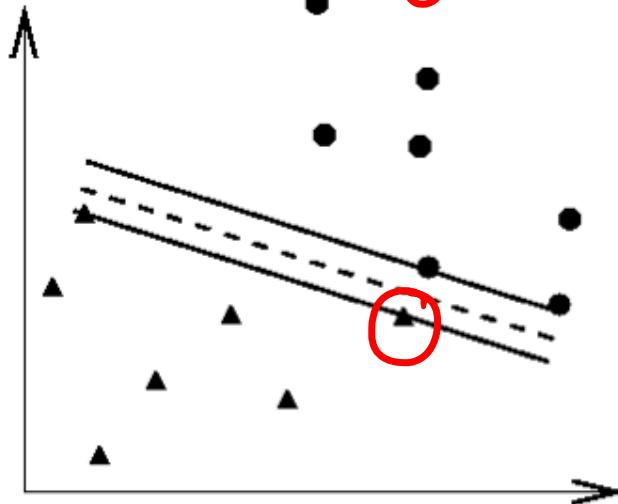
Model Selection, find right C

large C

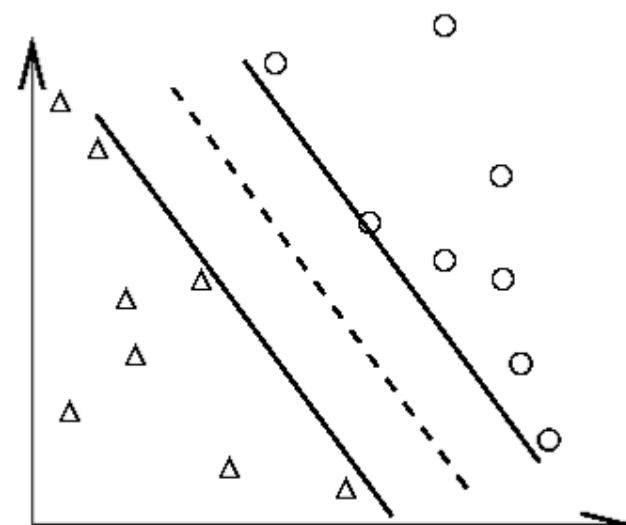
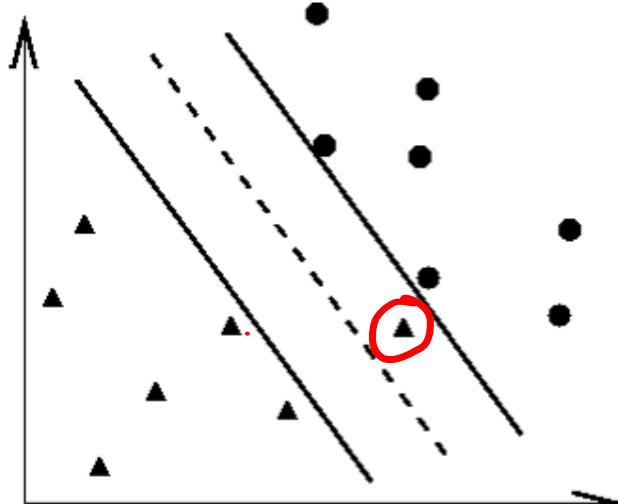
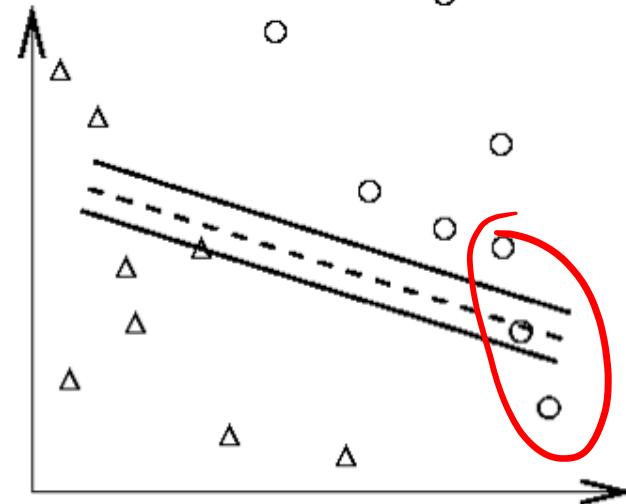
Select the
right
penalty
parameter
 C

small C

Training



Test



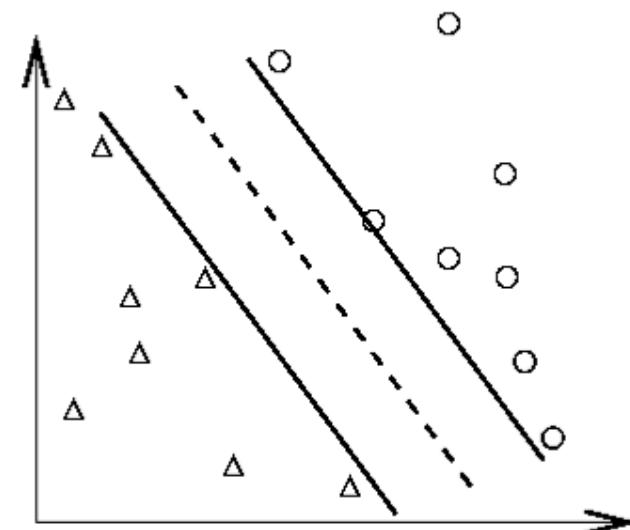
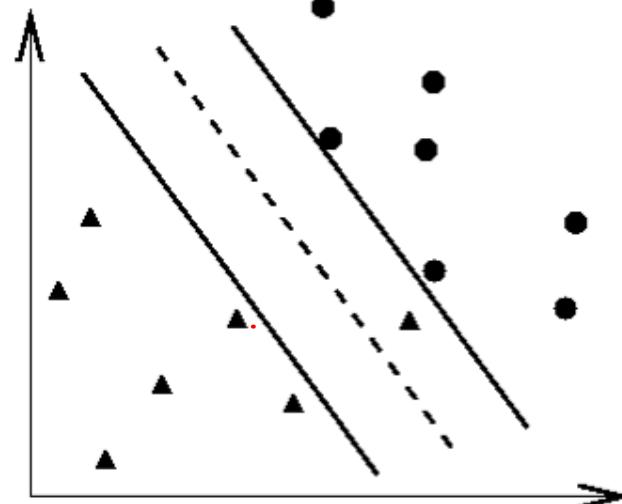
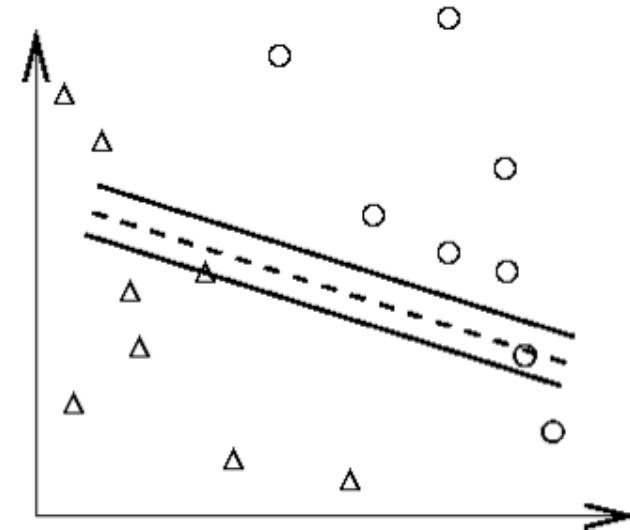
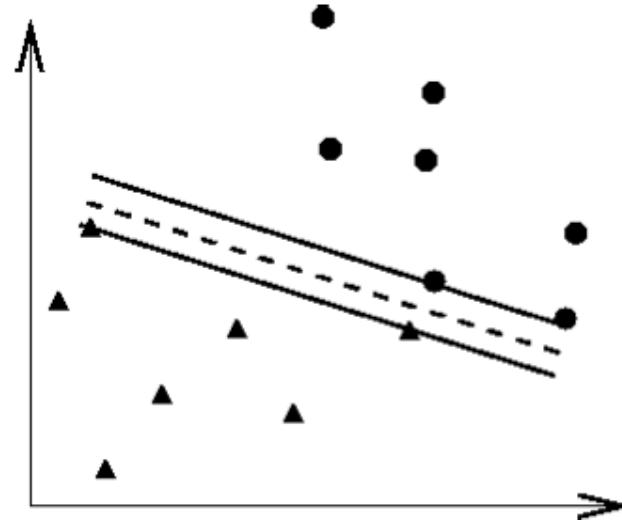
Model Selection, find right C

large C

A large value of C means that misclassifications are bad - resulting in smaller margins and less training error (but more expected true error).

A small C results in more training error, hopefully better true error.

small C



(c) Model Selection

- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.

two parameters for an RBF kernel: C and γ

- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d$, $\gamma > 0$.

Three parameters for a polynomial kernel

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarize all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, tree kernel, graph kernel, ...)
 - Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors
- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try for most applications.

(d) Pipeline Procedures

- (I) train / test
- (II) k-folds cross validation
- (III) k-CV on train to choose hyperparameter / then test

Evaluation Choice-III:

Many beginners use the following procedure now:

- Transform data to the format of an SVM package
- Randomly try a few kernels and parameters
- Test

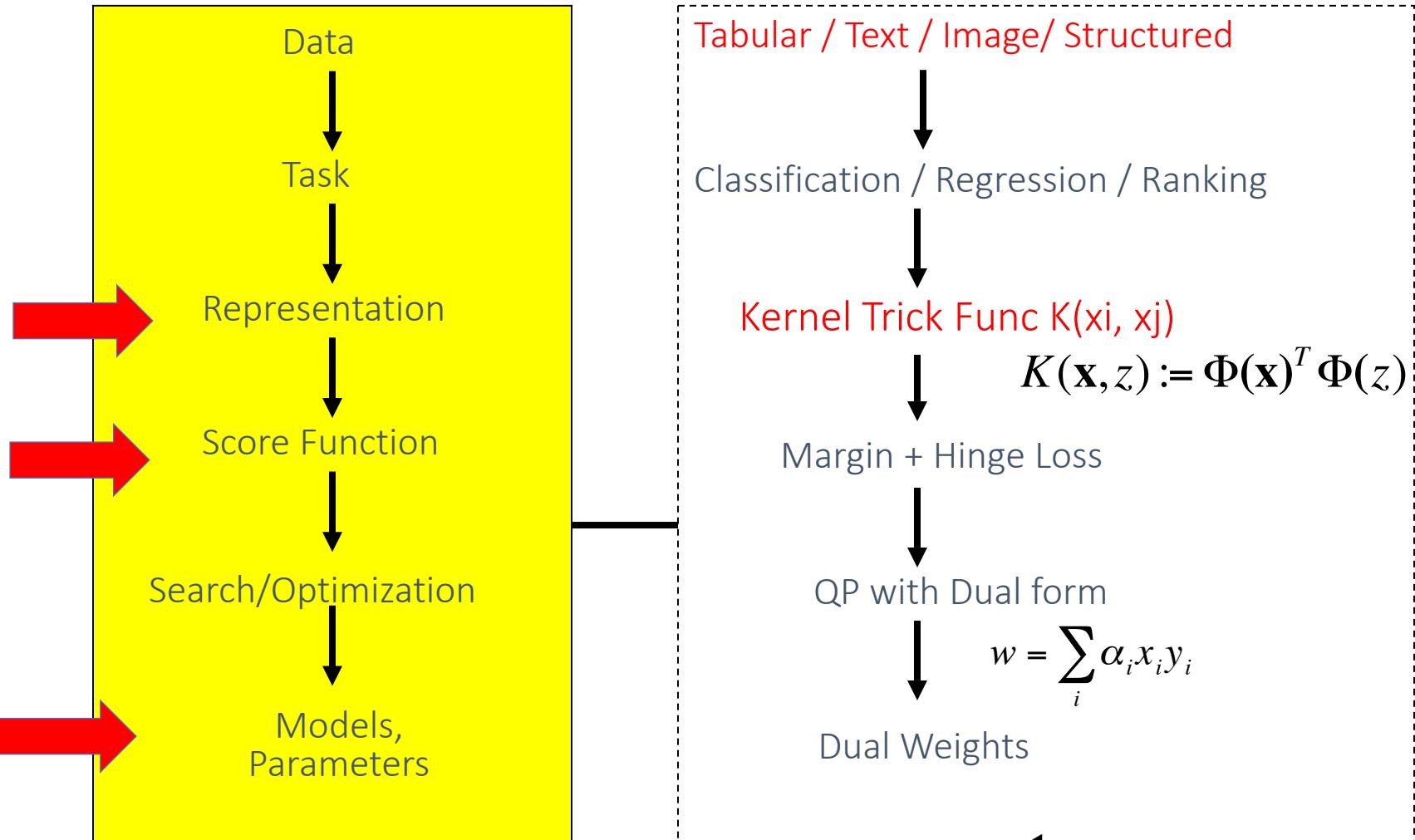
We propose that beginners try the following procedure first:

- Transform data to the format of an SVM package
- Conduct simple scaling on the data
- Consider the RBF kernel $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2}$
- Use cross-validation to find the best parameter C and γ
- Use the best parameter C and γ to train the whole training set⁵
- Test



We use option III for HW

Today : Kernel Support Vector Machine



$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{i=1}^n \varepsilon_i$$

$$\text{subject to } \forall \mathbf{x}_i \in D_{train} : y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$$

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

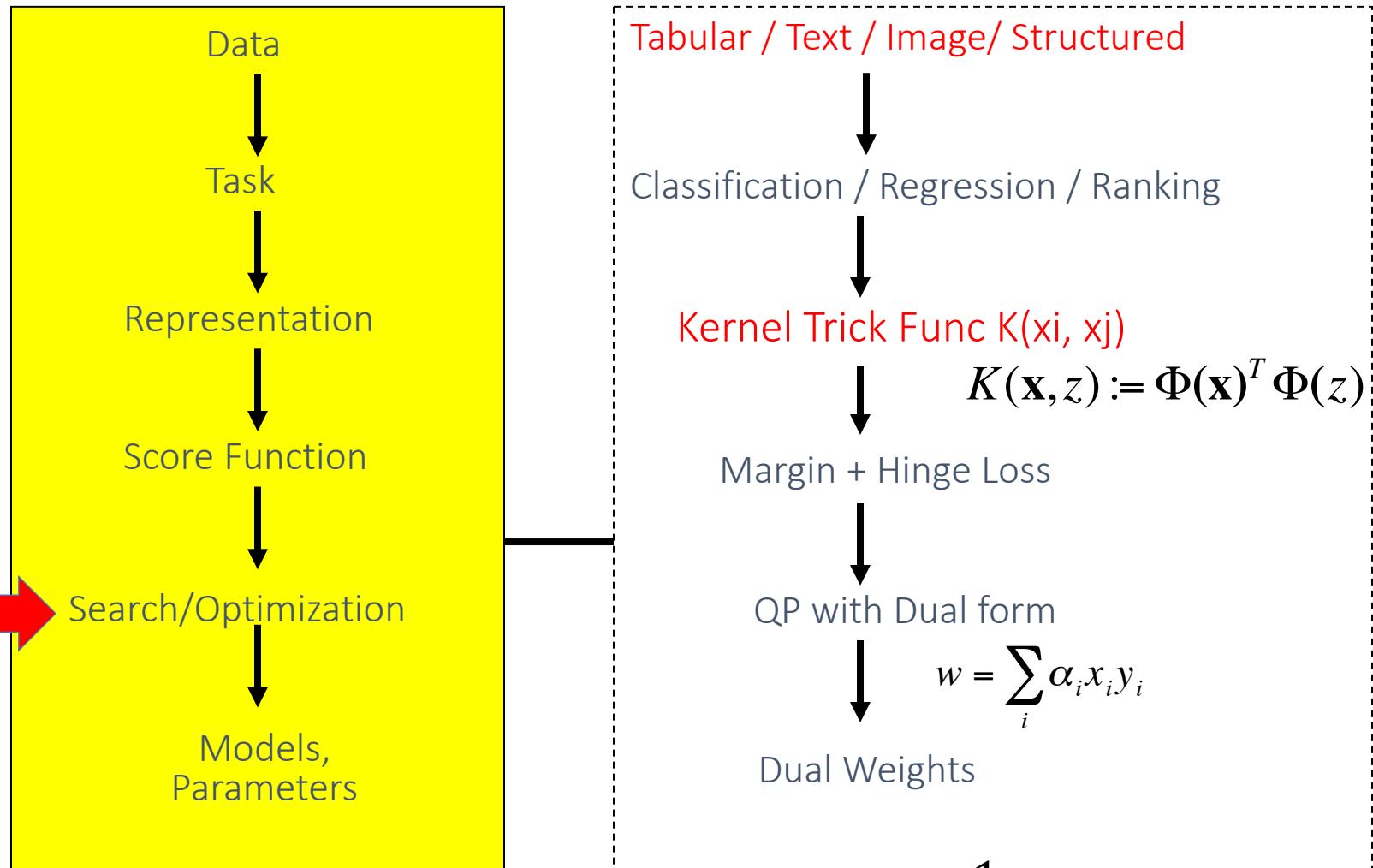
$$\sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i$$

Today: Nonlinear SVM & Practical Guide

❑ Support Vector Machine (SVM)

- ✓ History of SVM
- ✓ Large Margin Linear Classifier
- ✓ Define Margin (M) in terms of model parameter
- ✓ Optimization to learn model parameters (w, b)
- ✓ Non linearly separable case
- ✓ Optimization with dual form
- ✓ Nonlinear decision boundary
- ✓ Practical Guide
 - ✓ File format / LIBSVM
 - ✓ Feature preprocessing
 - ✓ Model selection
 - ✓ Pipeline procedure

This: Kernel Support Vector Machine



$$\underset{\mathbf{w}, b}{\operatorname{argmin}} \sum_{i=1}^p w_i^2 + C \sum_{i=1}^n \varepsilon_i$$

subject to $\forall \mathbf{x}_i \in D_{train} : y_i (\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 - \varepsilon_i$

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0, \quad \alpha_i \geq 0 \quad \forall i$$



Thank You

Thank you

References

- Big thanks to Prof. Ziv Bar-Joseph and Prof. Eric Xing @ CMU for allowing me to reuse some of his slides
- Elements of Statistical Learning, by Hastie, Tibshirani and Friedman
- Prof. Andrew Moore @ CMU's slides
- Tutorial slides from Dr. Tie-Yan Liu, MSR Asi
- A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, 2003-2010
- Tutorial slides from Stanford “Convex Optimization I — Boyd & Vandenberghe

Extra on Kernel SVM

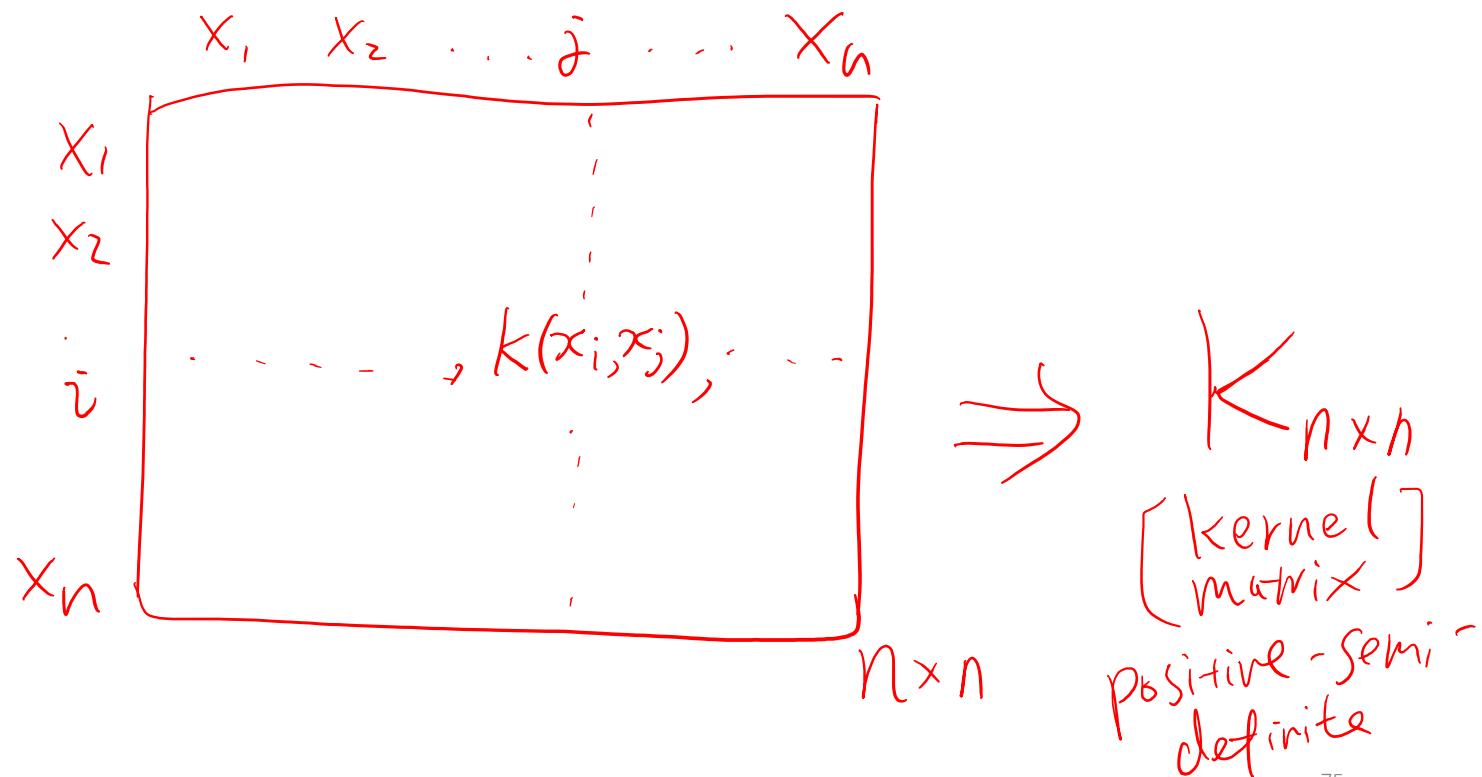
Why do SVMs work?

$x \rightarrow \phi(x)$ e.g. RBF

- If we are using huge features spaces (e.g., with kernels), how come we are not overfitting the data?
 - ✓ Number of parameters remains the same (and most are set to 0) $O(n)$, α_i $i=1, \dots, n$
 - ✓ While we have a lot of inputs, at the end we only care about the support vectors and these are usually a small group of samples
 - ✓ The maximizing of the margin acts as a sort of regularization term leading to reduced overfitting

Kernel Matrix

- Kernel function creates the kernel matrix, which summarize all the (train) data



Summary: Modification Due to Kernel Trick

- Change all inner products to kernel functions
- For training,

Original
Linear

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

With kernel
function -
nonlinear

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C > \alpha_i \geq 0, \forall i \in train$$

VC: $\Phi(x)$
large

Summary: Modification Due to Kernel Function

- For testing, the new data \mathbf{x}_{ts}

Original
Linear

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in \text{supportVectors}} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_{ts} + b \right)$$

With kernel
function -
nonlinear

$$\hat{y}_{ts} = \text{sign} \left(\sum_{i \in \text{supportVectors}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_{ts}) + b \right)$$

Kernel Trick: Implicit Basis Representation

- For some kernels (e.g. RBF) the implicit transform basis form $\phi(x)$ is infinite-dimensional!
 - But calculations with kernel are done in original space, so computational burden and curse of dimensionality aren't a problem.

$O(P)$

$$K(x, z) = \exp\left(-r \|x - z\|^2\right)$$

$O(p * n^2)$ operations in building
a RBF-kernel matrix for training

➔ Gaussian RBF Kernel corresponds to an infinite-dimensional vector space.

YouTube video of Caltech: Abu-Mostafa
explaining this in more

detail <https://www.youtube.com/watch?v=XUj5JbQihlU&t=25m53s>

Time Cost Comparisons

	$x_i^T x_j$	$\underline{\phi(x_i)}^T \underline{\phi(x_j)}$	$k(x_i, x_j)$	explicit
Training Stage	$O(p \cdot n^2)$	$O(m \cdot n^2)$ polynomial $m \sim p^d$	$O(p \cdot n^2)$	w
Test Stage	$O(p \cdot \#SV)$	$O(m \cdot \#SV)$	$O(p \cdot \#SV)$	$\underline{w}^T \underline{\phi(x_{\text{test}})} + b$ (a) $O(p^d)$ (b) for some RBF, $m \sim \infty$

Kernel Functions (Extra)

- In practical use of SVM, only the kernel function (and not basis function) is specified
- Kernel function can be thought of as a similarity measure between the input objects
- Not all similarity measure can be used as kernel function, however Mercer's condition states that **any positive semi-definite** kernel $K(x, y)$, i.e.

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

can be expressed as a dot product in a high dimensional space.

Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarize all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, tree kernel, graph kernel, ...)
 - Kernel trick has helped Non-traditional data like strings and trees able to be used as input to SVM, instead of feature vectors
- In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try for most applications.

Mercer Kernel vs. Smoothing Kernel (Extra)

- The Kernels used in Support Vector Machines are different from the Kernels used in LocalWeighted /Kernel Regression.
- We can think
 - Support Vector Machines' kernels as **Mercer Kernels**
 - Local Weighted / Kernel Regression's kernels as **Smoothing Kernels**

Why SVM Works? (Extra)

- Vapnik argues that the fundamental problem is not the number of parameters to be estimated. Rather, the problem is about the flexibility of a classifier
- Vapnik argues that the flexibility of a classifier should not be characterized by the number of parameters, but by the capacity of a classifier
 - This is formalized by the “VC-dimension” of a classifier
- The SVM objective can also be justified by structural risk minimization: the empirical risk (training error), plus a term related to the generalization ability of the classifier, is minimized
- Another view: the SVM loss function is analogous to ridge regression. The term $\frac{1}{2} \|w\|^2$ “shrinks” the parameters towards zero to avoid overfitting