

<https://bit.ly/2I3JKMY>

KYOTO UNIVERSITY

# Statistical Learning Theory - Classification -

Hisashi Kashima

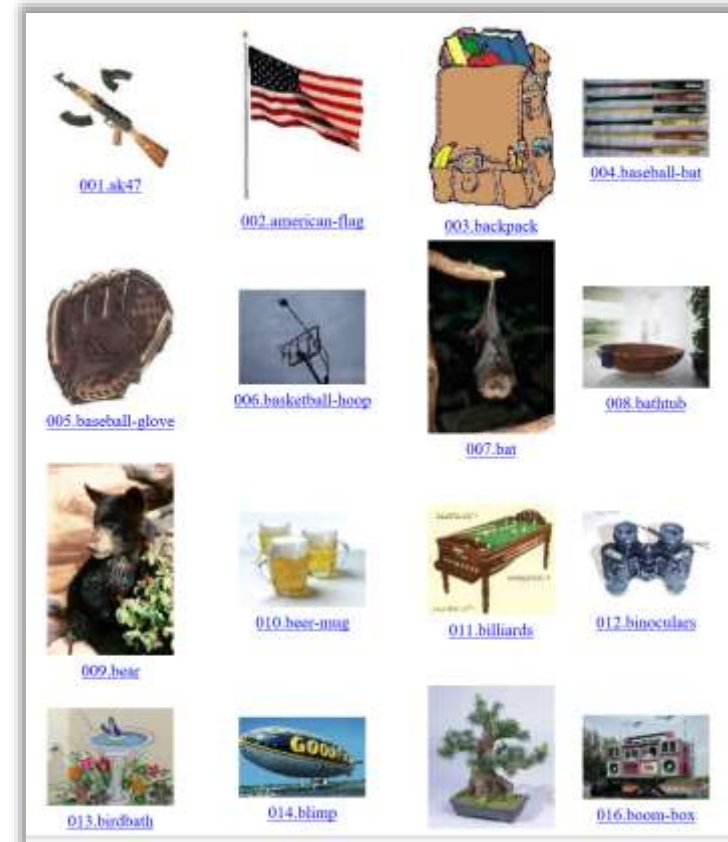
DEPARTMENT OF INTELLIGENCE SCIENCE  
AND TECHNOLOGY

# Classification

# Classification:

## Supervised learning for predicting discrete variable

- Goal: Obtain a function  $f: \mathcal{X} \rightarrow \mathcal{Y}$  ( $\mathcal{Y}$ : discrete domain)
  - E.g.  $x \in \mathcal{X}$  is an image and  $y \in \mathcal{Y}$  is the type of object appearing in the image
  - Two-class classification:  $\mathcal{Y} = \{+1, -1\}$
- Training dataset:  
 $N$  pairs of an input and an output  
 $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$



[http://www.vision.caltech.edu/Image\\_Datasets/Caltech256/](http://www.vision.caltech.edu/Image_Datasets/Caltech256/)

# Some applications of classification:

## From binary to multi-class classification

---

- Binary (two-class) classification:
  - Purchase prediction: Predict if a customer  $\mathbf{x}$  will buy a particular product (+1) or not (-1)
  - Credit risk prediction: Predict if a obligor  $\mathbf{x}$  will pay back a debt (+1) or not (-1)
- Multi-class classification:
  - Text classification: Categorize a document  $\mathbf{x}$  into one of several categories, e.g., {politics, economy, sports, ...}
  - Image classification: Categorize the object in an image  $\mathbf{x}$  into one of several object names, e.g., {AK5, American flag, backpack, ...}
  - Action recognition: Recognize the action type ({running, walking, sitting, ...}) that a person is taking from sensor data  $\mathbf{x}$

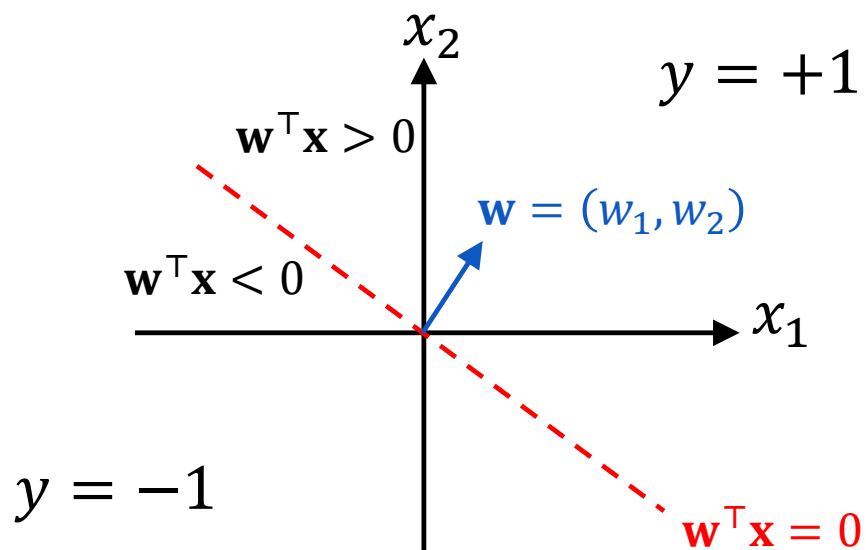
# Model for classification:

## Linear classifier

- Linear classification: Linear regression model

$$y = \text{sign}(\mathbf{w}^\top \mathbf{x}) = \text{sign}(w_1 x_1 + w_2 x_2 + \cdots + w_D x_D)$$

- $|\mathbf{w}^\top \mathbf{x}|$  indicates the intensity of belief
- $\mathbf{w}^\top \mathbf{x} = 0$  gives a separating hyperplane
- $\mathbf{w}$ : normal vector perpendicular to the separating hyperplane



# Learning framework:

## Loss minimization and statistical estimation

---

### ■ Two learning frameworks

1. Loss minimization:  $L(\mathbf{w}) = \sum_{i=1}^N \ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w})$

- Loss function  $\ell^{(i)}$ : directly handles utility of predictions
- Regularization term  $R(\mathbf{w})$

2. Statistical estimation (likelihood maximization):

$$L(\mathbf{w}) = \prod_{i=1}^N f(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w})$$

- Probabilistic model: Noise assumptions are clear
- Prior distribution  $P(\mathbf{w})$

– They are often equivalent :  $\begin{cases} \text{Loss} = \text{Probabilistic model} \\ \text{Regularization} = \text{Prior} \end{cases}$

# Classification problem in loss minimization framework:

## Minimize loss function + regularization term

- Minimization problem:  $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} L(\mathbf{w}) + R(\mathbf{w})$ 
  - Loss function  $L(\mathbf{w})$  : Fitness to training data
  - Regularization term  $R(\mathbf{w})$  : Penalty on the model complexity to avoid overfitting to training data (usually norm of  $\mathbf{w}$ )
- Loss function should reflect the number of misclassifications on training data
  - Zero-one loss:

$$\ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w}) = \begin{cases} 0 & \left( y^{(i)} = \operatorname{sign}(\mathbf{w}^\top \mathbf{x}^{(i)}) \right) \\ 1 & \left( y^{(i)} \neq \operatorname{sign}(\mathbf{w}^\top \mathbf{x}^{(i)}) \right) \end{cases}$$

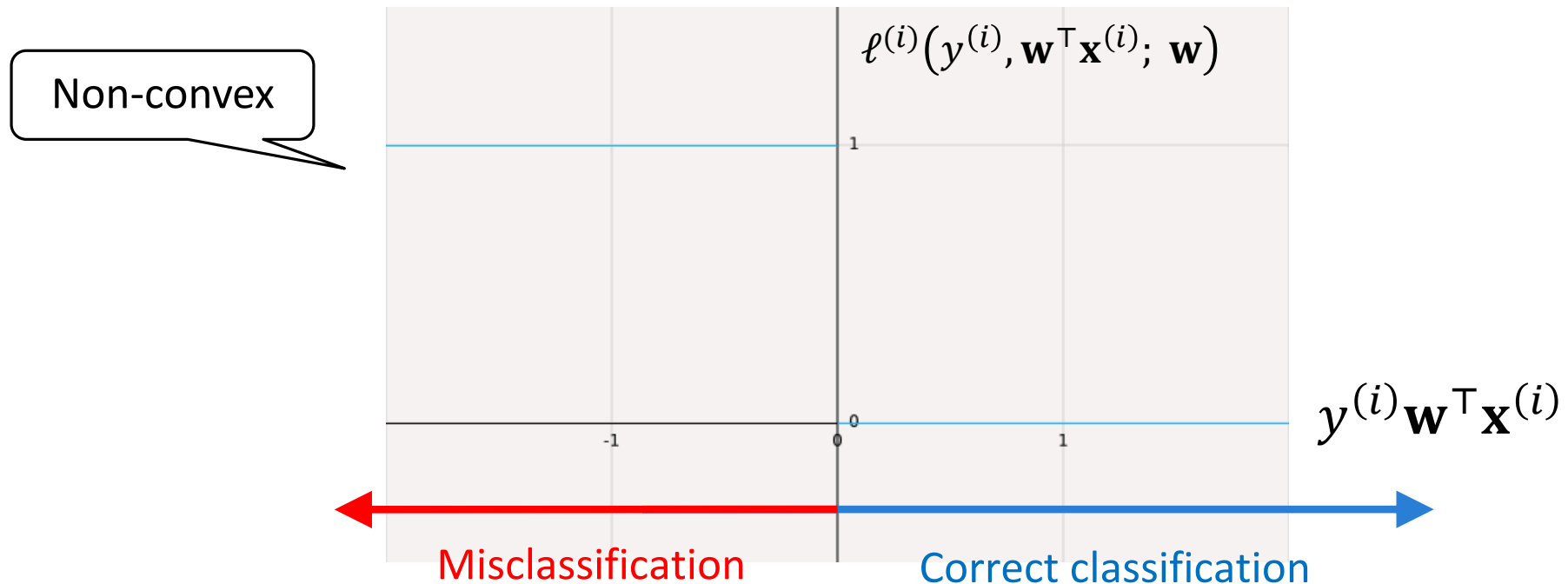
Correct classification

Incorrect classification

# Zero-one loss:

Number of misclassification is hard to minimize

- Zero-one loss:  $\ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w}) = \begin{cases} 0 & (y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} > 0) \\ 1 & (y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0) \end{cases}$
- Non-convex function is hard to optimize directly

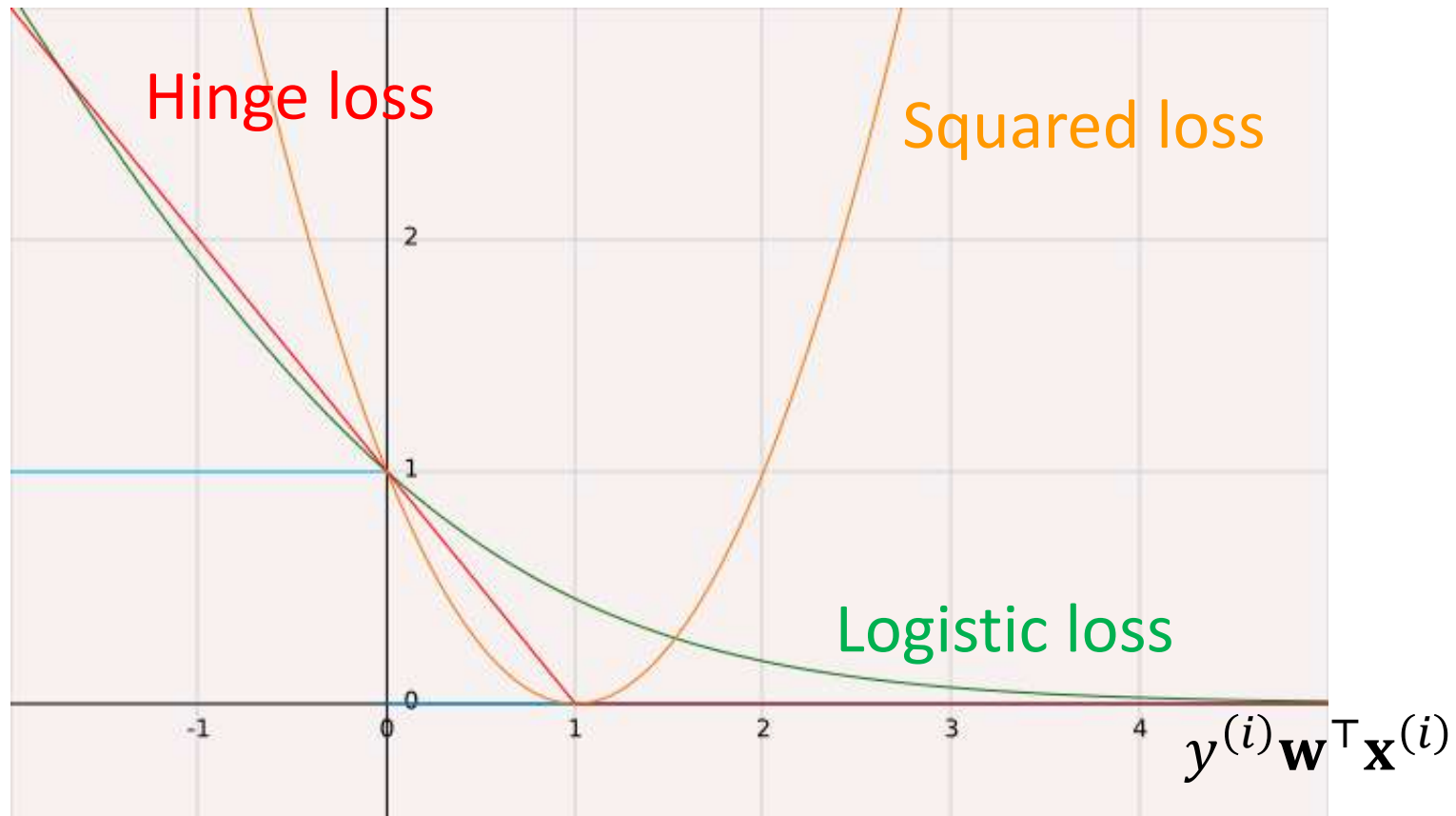




# Convex surrogates of zero-one loss:

## Different functions lead to different learning machines

- Convex surrogates: Upper bounds of zero-one loss
  - Hinge loss = SVM, Logistic loss = logistic regression, ...



# Logistic regression

# Logistic regression:

## Minimization of logistic loss is a convex optimization

- Logistic loss:

$$\ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w}) = \frac{1}{\ln 2} \ln(1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}))$$

- (Regularized) Logistic regression:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^N \ln(1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})) + \lambda \|\mathbf{w}\|_2^2$$

Convex



# Statistical interpretation:

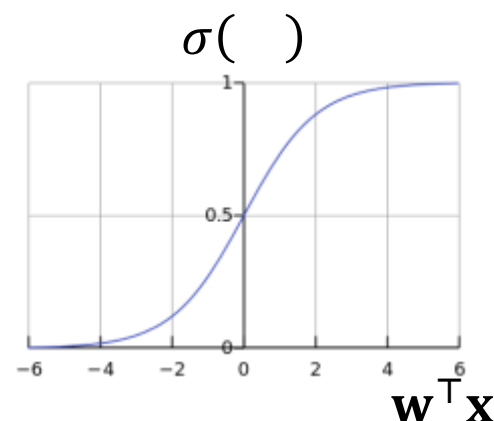
## Logistic loss min. as MLE of logistic regression model

- Minimization of logistic loss is equivalent to maximum likelihood estimation of logistic regression model

- Logistic regression model (conditional probability):

$$f(y = 1|\mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}$$

— $\sigma$ : Logistic function ( $\sigma: \mathbb{R} \rightarrow (0,1)$ )



- Log likelihood:

$$L(\mathbf{w}) = \sum_{i=1}^N \log f(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = - \sum_{i=1}^N \log(1 + \exp(-y^{(i)}\mathbf{w}^\top \mathbf{x}))$$

$$\left( = \sum_{i=1}^N \delta(y^{(i)} = 1) \log \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} + \delta(y^{(i)} = -1) \log \left( 1 - \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} \right) \right)$$

# Parameter estimation of logistic regression :

## Numerical nonlinear optimization

---

- Objective function of (regularized) logistic regression:

$$L(\mathbf{w}) = \sum_{i=1}^N \ln(1 + \exp(-y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)})) + \lambda \|\mathbf{w}\|_2^2$$

- Minimization of logistic loss / MLE of logistic regression model has no closed form solution
- Numerical nonlinear optimization methods are used
  - Iterate parameter updates:  $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} + \mathbf{d}$



## Parameter update :

Find the best update minimizing the objective function

- By update  $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} + \mathbf{d}$ , the objective function will be:

$$L_{\mathbf{w}}(\mathbf{d}) = \sum_{i=1}^N \ln(1 + \exp(-y^{(i)}(\mathbf{w} + \mathbf{d})^\top \mathbf{x}^{(i)})) + \lambda \|\mathbf{w} + \mathbf{d}\|_2^2$$

- Find  $\mathbf{d}^*$  that minimizes  $L_{\mathbf{w}}(\mathbf{d})$ :  
 $-\mathbf{d}^* = \operatorname{argmin}_{\mathbf{d}} L_{\mathbf{w}}(\mathbf{d})$

# Finding the best parameter update :

## Approximate the objective with Taylor expansion

- Taylor expansion:

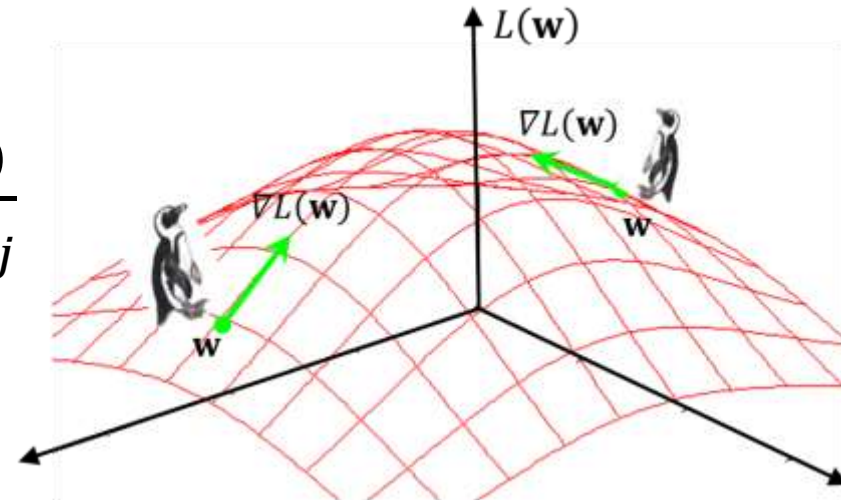
3rd-order term

$$L_{\mathbf{w}}(\mathbf{d}) = L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2} \mathbf{d}^\top \mathbf{H}(\mathbf{w}) \mathbf{d} + O(\mathbf{d}^3)$$

- Gradient vector:  $\nabla L(\mathbf{w}) = \left( \frac{\partial L(\mathbf{w})}{\partial w_1}, \frac{\partial L(\mathbf{w})}{\partial w_2}, \dots, \frac{\partial L(\mathbf{w})}{\partial w_D} \right)^\top$

- Steepest direction

- Hessian matrix:  $[H(\mathbf{w})]_{i,j} = \frac{\partial^2 L(\mathbf{w})}{\partial w_i \partial w_j}$



# Newton update :

## Minimizes the second order approximation

- Approximated Taylor expansion (neglecting the 3<sup>rd</sup> order term):

$$L_{\mathbf{w}}(\mathbf{d}) \approx L(\mathbf{w}) + \mathbf{d}^\top \nabla L(\mathbf{w}) + \frac{1}{2} \mathbf{d}^\top \mathbf{H}(\mathbf{w}) \mathbf{d} - \cancel{O(\mathbf{d}^3)}$$

- Derivative w.r.t.  $\mathbf{d}$ :  $\frac{\partial L_{\mathbf{w}}(\mathbf{d})}{\partial \mathbf{d}} \approx \nabla L(\mathbf{w}) + \mathbf{H}(\mathbf{w}) \mathbf{d}$

- Setting it to be  $\mathbf{0}$ ,  $\mathbf{d} = -\mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$

- Newton update formula:

$$\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$$





## Modified Newton update:

### Second order approximation + linear search

---

- The correctness of the update  $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$  depends on the second-order approximation:

$$L_{\mathbf{w}}(\mathbf{d}) \approx L(\mathbf{w}) + \mathbf{d}^{\top} \nabla L(\mathbf{w}) + \frac{1}{2} \mathbf{d}^{\top} \mathbf{H}(\mathbf{w}) \mathbf{d}$$

—This is not actually true for most cases

- Use only the direction of  $\mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$  and update with  $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$
- Learning rate  $\eta > 0$  is determined by linear search:  
$$\eta^* = \operatorname{argmax}_{\eta} L(\mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w}))$$

# Steepest gradient descent:

## Simple update without computing inverse Hessian

- Computing the inverse of Hessian matrix is costly

- Newton update:  $\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \mathbf{H}(\mathbf{w})^{-1} \nabla L(\mathbf{w})$

- Steepest gradient descent:

- Replacing  $\mathbf{H}(\mathbf{w})^{-1}$  with  $\mathbf{I}$  will give

$$\mathbf{w}^{\text{NEW}} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w})$$

Gradient of  
objective function

- $\nabla L(\mathbf{w})$  is the steepest direction
  - Learning rate  $\eta$  is determined by line search



## (Supplement) :

### Computing the gradient of logistic regression

- $L(\mathbf{w}) = \sum_{i=1}^N \ln(1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}))$

- $$\begin{aligned} \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} &= \sum_{i=1}^N \frac{1}{1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})} \frac{\partial (1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}))}{\partial \mathbf{w}} \\ &= - \sum_{i=1}^N \frac{1}{1 + \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})} \exp(-y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) y^{(i)} \mathbf{x}^{(i)} \\ &= - \sum_{i=1}^N (1 - f(y^{(i)} | \mathbf{x}^{(i)}, \mathbf{w})) y^{(i)} \mathbf{x}^{(i)} \end{aligned}$$

Can be easily computed with the current prediction probabilities

## Mini batch:

### Efficient training using data subsets

---

- Objective function for  $N$  instances:

$$L(\mathbf{w}) = \sum_{i=1}^N \ell(\mathbf{w}^\top \mathbf{x}^{(i)}) + \lambda R(\mathbf{w})$$

- Its derivative  $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^N \frac{\partial \ell(\mathbf{w}^\top \mathbf{x}^{(i)})}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}}$  needs  $O(N)$  computation

- Approximate this with only one instance:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx N \frac{\partial \ell(\mathbf{w}^\top \mathbf{x}^{(j)})}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}} \quad (\text{Stochastic approximation})$$

- Also we can do this with  $1 < M < N$  instances:

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} \approx \frac{N}{M} \sum_{j \in \text{MiniBatch}} \frac{\partial \ell(\mathbf{w}^\top \mathbf{x}^{(j)})}{\partial \mathbf{w}} + \lambda \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}} \quad (\text{Mini batch})$$

# Support Vector Machine and Kernel Methods

# Support vector machine:

## One of the most successful learning methods

---

- One of the most important achievements in machine learning
  - Proposed in 1990s by Cortes & Vapnik
  - Suitable for small to middle sized data
- A learning algorithm of linear classifiers
  - Based on “margin maximization” principle
  - Understood as hinge loss + L2-regularization
- Kernel methods: Capable of non-linear classification through kernel functions
  - SVM is one of the kernel methods

# Loss function of support vector machine:

## Hinge loss

- In SVM, we use hinge loss as a convex upper bound of 0-1 loss

$$\ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w}) = \max\{1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}, 0\}$$

- Squared hinge loss  $\max\{(1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)})^2, 0\}$  is also sometimes used



# Two formulations of SVM training:

## Soft-margin SVM and hard margin SVM

---

- When we use L2 regularization, we have “soft-margin” SVM:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^N \max\{1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}, 0\} + \lambda \|\mathbf{w}\|_2^2$$

—This is a convex optimization problem ☺

- With constraint on the loss, we have “hard-margin” SVM:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|_2^2 \text{ s.t. } \sum_{i=1}^N \max\{\mathbf{1} - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}, 0\} = 0$$

—Equivalently, the constraint is written as

$$\mathbf{1} - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0 \text{ (for all } i = 1, 2, \dots, N)$$

—The originally proposed SVM formulation was in this form



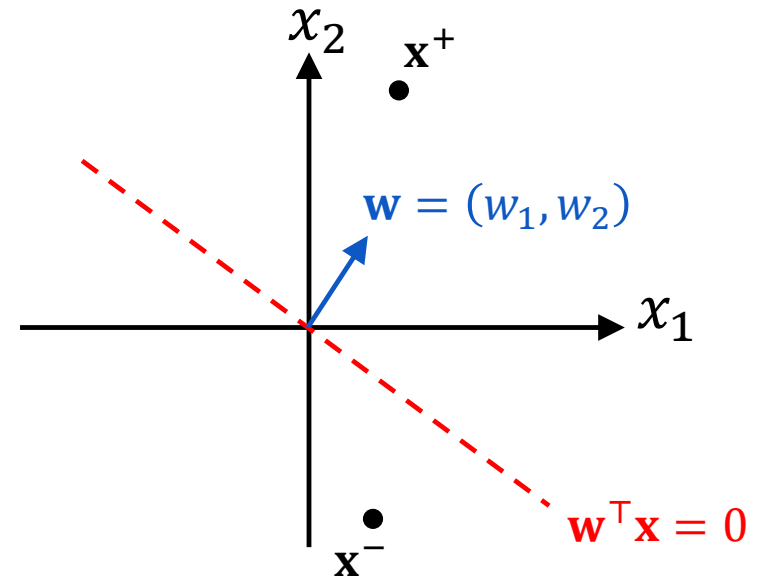
# Geometric interpretation:

## Hard-margin SVM maximizes the margin

- $\min \frac{1}{2} \| \mathbf{w} \|_2^2 \leftrightarrow \max \frac{1}{\| \mathbf{w} \|_2}$  ( $\frac{1}{\| \mathbf{w} \|_2}$  is called *margin*)
- $\frac{\mathbf{w}^T (\mathbf{x}^+ - \mathbf{x}^-)}{\| \mathbf{w} \|_2}$ : Sum of distances between separating hyperplane and a positive instance  $\mathbf{x}^+$  and a negative instance  $\mathbf{x}^-$
- Since  $1 - y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} \leq 0 \quad \forall i$ ,

$\frac{\mathbf{w}^T (\mathbf{x}^+ - \mathbf{x}^-)}{\| \mathbf{w} \|_2}$  is lower bounded  $\frac{2}{\| \mathbf{w} \|_2}$

They can be taken as the closest instance to the separating hyperplane



# Solution of hard-margin SVM (Step I): Introducing Lagrange multipliers

- $\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|_2^2 \text{ s.t. } 1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0 \ (i = 1, 2, \dots, N)$

- Lagrange multipliers  $\{\alpha_i\}_i$  :

$$\min_{\mathbf{w}} \max_{\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \left( \frac{1}{2} \|\mathbf{w}\|_2^2 + \sum_{i=1}^N \alpha_i (1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) \right)$$

- If  $1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} > 0$  for some  $i$ , we have  $\alpha_i = \infty$

- The objective function becomes  $\infty$ , that cannot be optimal

- If  $1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq 0$  for some  $i$ , we have either

- $\alpha_i = 0$  or  $(1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) = 0$ , i.e. objective function remains the same as the original one  $(\frac{1}{2} \|\mathbf{w}\|_2^2)$

# Solution of hard-margin SVM (Step II):

## Dual formulation as a quadratic programming problem

- By changing the order of min and max:

$$\min_{\mathbf{w}} \max_{\alpha=(\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \left( \frac{\|\mathbf{w}\|_2^2}{2} + \sum_{i=1}^N \alpha_i (1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) \right)$$

$\Downarrow$

$$\max_{\alpha=(\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \min_{\mathbf{w}} \left( \frac{\|\mathbf{w}\|_2^2}{2} + \sum_{i=1}^N \alpha_i (1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}) \right)$$

- Solving min gives  $\mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$ , which finally results in

$$\max_{\alpha=(\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$$

## Support vectors:

SVM model depends only on support vectors

- The dual problem:

$$\max_{\alpha=(\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$$

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

- Support vectors: the set of  $i$  such that  $\alpha_i > 0$ 
  - For such  $i$ ,  $1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} = 0$  holds
  - They are the closest instance to the separating hyperplane
- Non-support vectors ( $\alpha_i = 0$ ) do not appear in the model:
$$\mathbf{w}^\top \mathbf{x} = \sum_{j=1}^N \alpha_j y^{(j)} \mathbf{x}^{(j)\top} \mathbf{x}$$

# Solution of soft-margin SVM:

## Additional constraints

- Equivalent formulation of soft-margin SVM:

$$\begin{aligned} \min_{\mathbf{w}} \quad & \|\mathbf{w}\|_2^2 + C \sum_{i=1}^N e_i \\ \text{s. t.} \quad & 1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)} \leq e_i \\ & (i = 1, 2, \dots, N) \end{aligned}$$

Hinge loss  
(Slack variable)

- Similar dual problem with **additional constraints**:

$$\max_{\alpha = (\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$$

$$0 \leq \alpha_i \leq C \quad (i = 1, 2, \dots, N)$$

# Kernel methods:

## Data access through kernel function

- The dual form objective function and the classifier access to data always through inner products  $\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$ 
  - The inner product  $\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$  is considered as similarity
- Can we use some similarity function  $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$  instead of  $\mathbf{x}^{(i)\top} \mathbf{x}^{(j)}$ ? – Yes (under certain conditions)

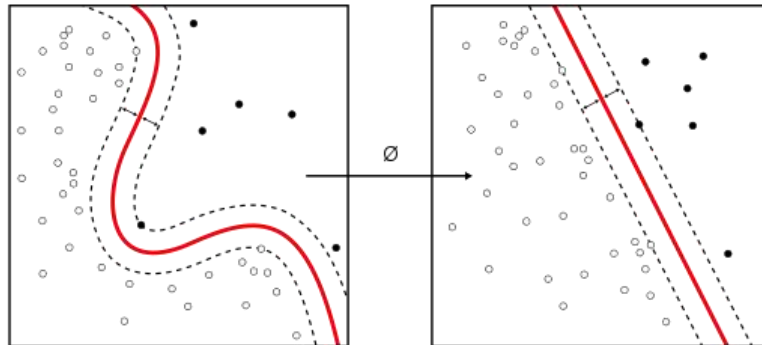
$$\max_{\alpha=(\alpha_1, \alpha_2, \dots, \alpha_N) \geq 0} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_i^N \sum_j^N \alpha_i \alpha_j y^{(i)} y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

– Model :  $\mathbf{w}^\top \mathbf{x} = \sum_{j=1}^N \alpha_j y^{(j)} K(\mathbf{x}^{(j)}, \mathbf{x})$

# Kernel functions:

## Introducing non-linearity in linear models

- Consider a (nonlinear) mapping  $\phi: \mathbb{R}^D \rightarrow \mathbb{R}^{D'}$ 
  - $D$ -dimensional space to  $D' (\gg D)$ -dimensional space
  - Vector  $\mathbf{x}$  is mapped to a high-dimensional vector  $\phi(\mathbf{x})$
- Define kernel  $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^\top \phi(\mathbf{x}^{(j)})$
- SVM is a linear classifier in the  $D'$ -dimensional space, while is a non-linear classifier in the original space



# Advantage of kernel methods:

## Computational efficiency in terms of input dimensions

- Advantage of using kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \boldsymbol{\phi}(\mathbf{x}^{(i)})^\top \boldsymbol{\phi}(\mathbf{x}^{(j)})$$

- Even if  $\boldsymbol{\phi}$  is high-dimensional (possibly infinite dimensional), as far as its inner product  $\boldsymbol{\phi}(\mathbf{x}^{(i)})^\top \boldsymbol{\phi}(\mathbf{x}^{(j)})$  is given as an efficiently computable function, the dimension of  $\boldsymbol{\phi}$  does not matter

- Problem size:

$$D(\text{number of dimensions}) \rightarrow N(\text{number of data})$$

- Advantageous when  $\boldsymbol{\phi}$  is especially high or infinite dimensional



## Example of kernel functions:

### Polynomial kernel can consider high-order cross terms

- Combinatorial features: Not only the original features  $x_1, x_2, \dots, x_D$ , use their combinations (i.e. products)

- Exponential number of dimensions wrt  $d$

- Polynomial kernel:  $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left( \mathbf{x}^{(i)\top} \mathbf{x}^{(j)} + c \right)^d$

- E.g.  $c = 0, d = 2$ , two dimensional case

$$\mathbf{x}^{(i)} = \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \end{pmatrix}$$

$$\begin{aligned} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) &= \left( x_1^{(i)} x_1^{(j)} + x_2^{(i)} x_2^{(j)} \right)^2 \\ &= \left( x_1^{(i)2}, x_2^{(i)2}, \sqrt{2} x_1^{(i)} x_2^{(i)} \right) \left( x_1^{(j)2}, x_2^{(j)2}, \sqrt{2} x_1^{(j)} x_2^{(j)} \right) \end{aligned}$$

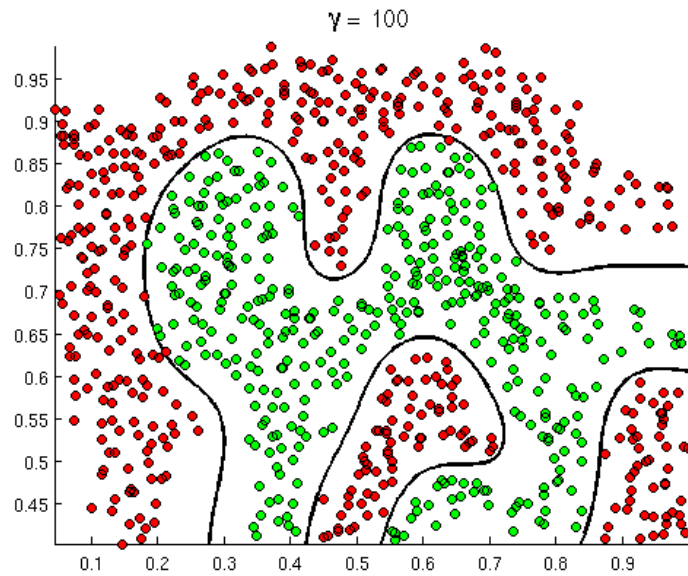
- Note that it can be computed in  $O(D)$

# Example of kernel functions:

## Gaussian kernel with infinite feature space

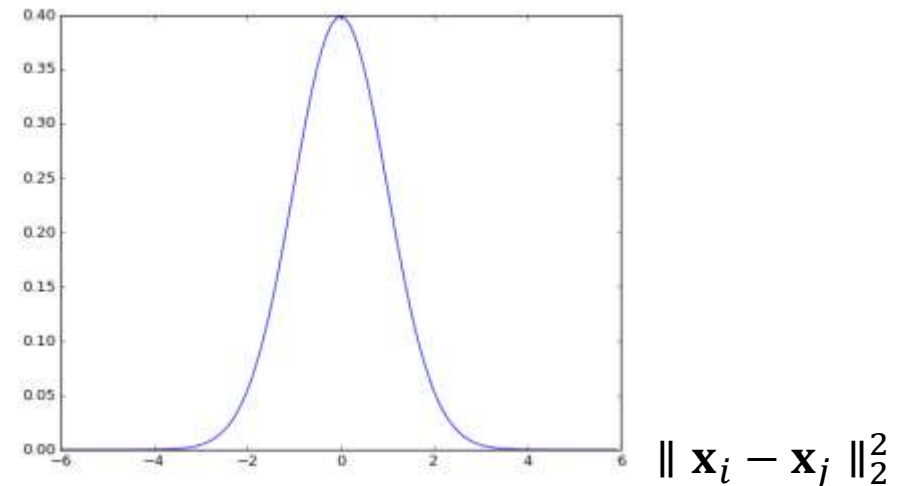
- Gaussian kernel (RBF kernel):  $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{\sigma}\right)$ 
  - Can be interpreted as an inner product in an infinite-dimensional space

Discrimination surface with Gaussian kernel



<http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html>

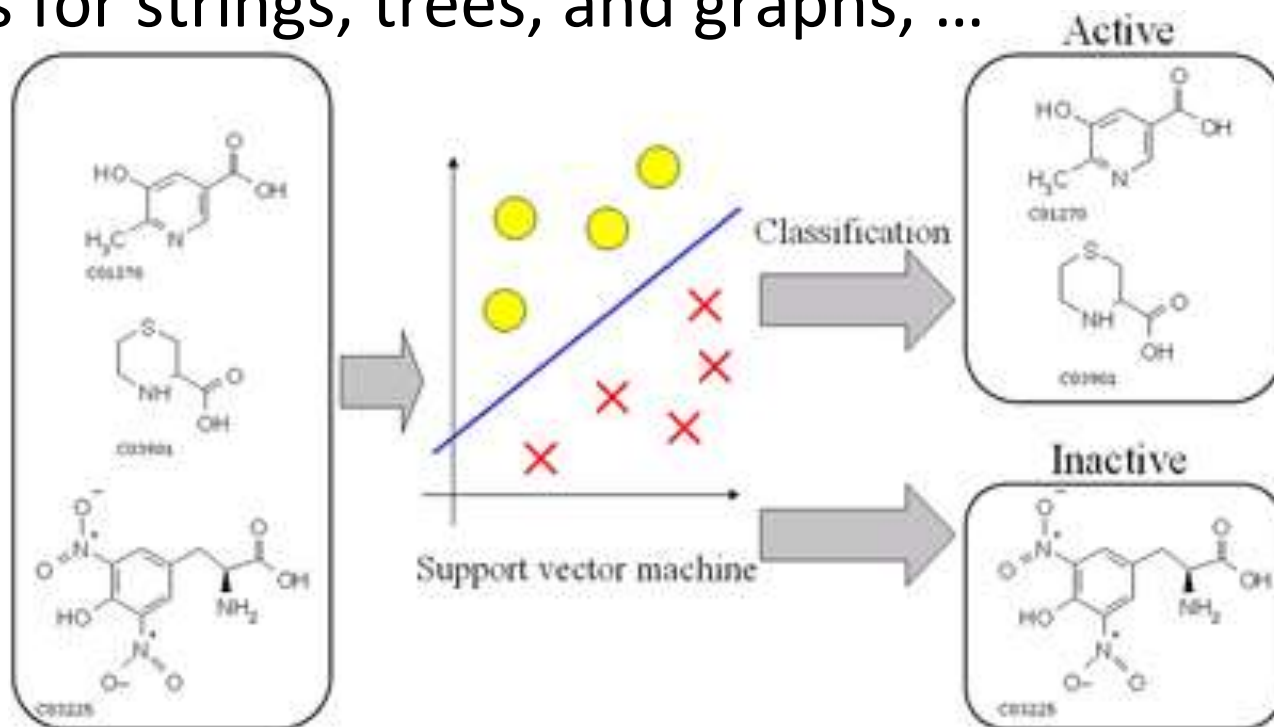
Gaussian kernel (RBF kernel)



# Kernel methods for non-vectorial data:

## Kernels for sequences, trees, and graphs

- Kernel methods can handle any kinds of objects (even non-vectorial objects) as long as efficiently computable kernel function is available
  - Kernels for strings, trees, and graphs, ...



[http://www.bic.kyoto-u.ac.jp/coe/img/akutsu\\_fig\\_e\\_02.gif](http://www.bic.kyoto-u.ac.jp/coe/img/akutsu_fig_e_02.gif)

## Representer theorem:

### Theoretical underpinning of kernel methods

---

- Can we use some similarity function as a kernel function?
  - Yes (under certain conditions)
- Kernel methods rely on the fact that the optimal parameter is represented as a linear combination of input vectors:

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

- Gives the dual form  $\mathbf{w}^\top \mathbf{x} = \sum_{j=1}^N \alpha_j y^{(j)} \mathbf{x}^{(j)\top} \mathbf{x}$
- Representer theorem:  
This is guaranteed under L2-regularization

# (Simple) proof of representer theorem:

Obj. func. depends only on the linear combination

- Assumption: Loss  $\ell^{(i)}$  for  $i$ -th data depends only on  $\mathbf{w}^\top \mathbf{x}^{(i)}$ 
  - Objective function:  $L(\mathbf{w}) = \sum_{i=1}^N \ell^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)}) + \lambda \|\mathbf{w}\|_2^2$
- Divide the optimal parameter  $\mathbf{w}^*$  into two parts  $\mathbf{w} + \mathbf{w}^\perp$ :
  - $\mathbf{w}$ : Linear combination of input data  $\{\mathbf{x}^{(i)}\}_i$
  - $\mathbf{w}^\perp$ : Other parts (orthogonal to all input data)
- $L(\mathbf{w}^*)$  depends only on  $\mathbf{w}$ :  $\sum_{i=1}^N \ell^{(i)}(\mathbf{w}^{*\top} \mathbf{x}^{(i)}) + \lambda \|\mathbf{w}^*\|_2^2$ 
$$= \sum_{i=1}^N \ell^{(i)} \left( \mathbf{w}^\top \mathbf{x}^{(i)} + \underbrace{\mathbf{w}^\perp{}^\top \mathbf{x}^{(i)}}_{=0} \right) + \lambda (\underbrace{\|\mathbf{w}\|_2^2}_{=0} + \underbrace{2\mathbf{w}^\top \mathbf{w}^\perp}_{=0} + \underbrace{\|\mathbf{w}^\perp\|_2^2}_{\text{Minimized to } =0})$$

## Primal objective function:

Kernel representation is also available in the primal form

- Primal objective function of SVM:

$$L(\mathbf{w}) = \sum_{i=1}^N \max\{1 - y^{(i)} \mathbf{w}^\top \mathbf{x}^{(i)}, 0\} + \lambda \|\mathbf{w}\|_2^2$$

- Primal objective function using kernel:

$$\begin{aligned} L(\boldsymbol{\alpha}) &= \sum_{i=1}^N \max\left\{1 - y^{(i)} \sum_{j=1}^N \alpha_j y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}), 0\right\} \\ &\quad + \lambda \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \end{aligned}$$

Using

$$\mathbf{w} = \sum_{i=1}^N \alpha_i y^{(i)} \mathbf{x}^{(i)}$$

# Support vector regression:

## Use $\epsilon$ -insensitive loss instead of hinge loss

- Instead of the hinge loss, use  $\epsilon$ -insensitive loss:

$$\ell^{(i)}(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}; \mathbf{w}) = \max\{|y_i - \mathbf{w}^\top \mathbf{x}^{(i)}| - \epsilon, 0\}$$

- Incurs no loss if the difference between the prediction and the target  $|y_i - \mathbf{w}^\top \mathbf{x}^{(i)}|$  is less than  $\epsilon$

