[SWCON253] Machine Learning – Lec.15c

Support Vector Machine

Fall 2025

김 휘 용

hykim.v@khu.ac.kr

경희대학교
KYUNG HEE UNIVERSITY

Contents

References

- *기계 학습* by 오일석, *패턴 인식* by 오일석

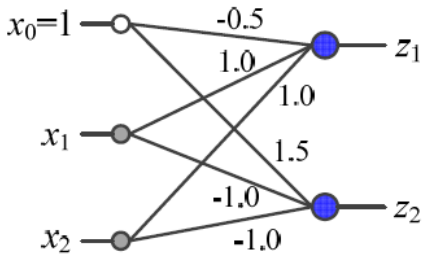
4. Nonlinear SVM

- ✓ Feature Space Conversion
- ✓ Kernel Trick & Kernel Function
- ✓ Nonlinear SVM (Kernel SVM)

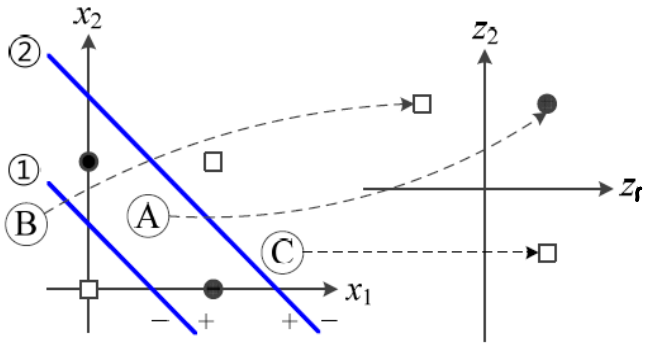
Feature Space Conversion (공간변환)

공간변환을 이용한 비선형 분류 문제 해결

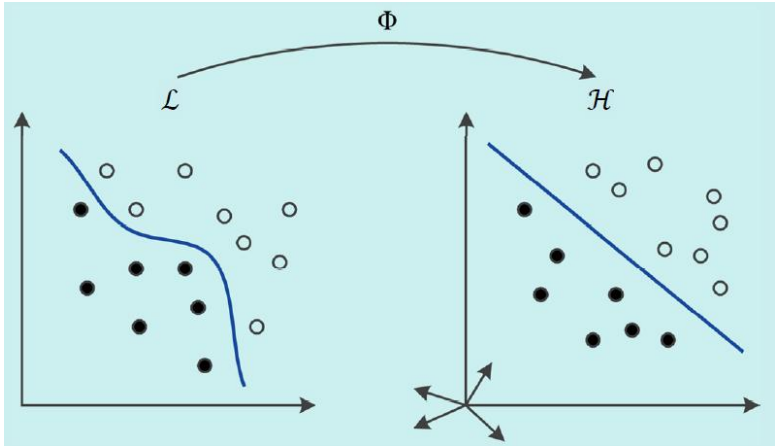
- **특징공간 변환**은 기계 학습의 핵심 연산
  - 원래 특징 공간을 목적 달성에 더 유리한 새로운 공간으로 변환하는 작업
  - 앞에서 공부한 사례: MLP에서 은닉층을 이용한 특징 공간 변환을 통해 XOR 문제 해결



(a) 두 퍼셉트론을 병렬로 결합

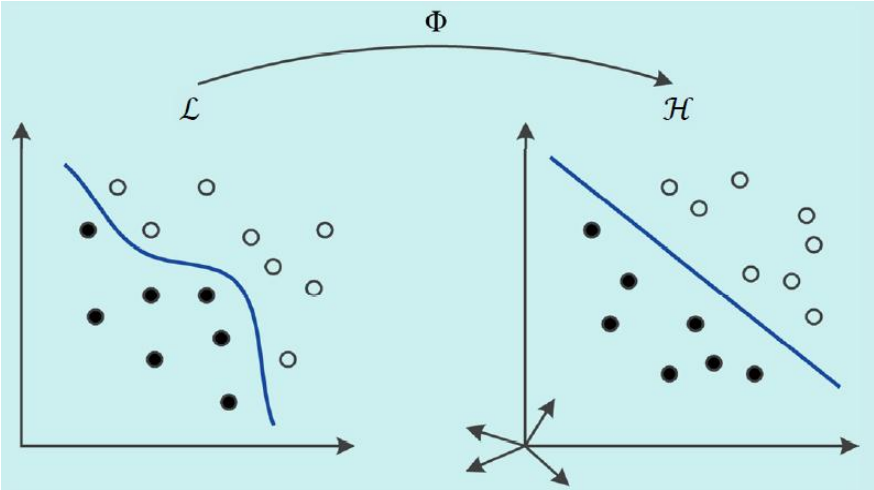


(b) 원래 특징 공간 x를 새로운 특징 공간 z로 변환



공간변환을 이용한 비선형 분류 문제 해결 (cont'd)

- 특징공간 변환에 기반한 비선형 분류문제 해결 방안
  - 선형 분리 불가능한 입력 데이터를 다른 공간으로 매핑하여 선형 분리 가능하도록 만들자.



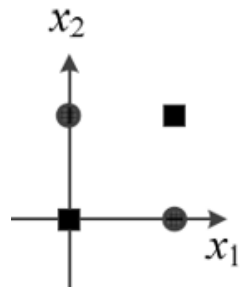
$$\mathbf{\Phi}(\mathbf{x}) = \mathbf{\Phi}((x_1, x_2, \cdots, x_d)^{\mathrm{T}}) = \left(\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \cdots, \phi_q(\mathbf{x})\right)^{\mathrm{T}}$$

  - 매핑 함수를 어떻게 찾을 것인가?
    1) Perceptron (step activation)을 이용한 2차원 to 2차원 매핑 (d=q=2)
    2) Gaussian function (RBF)을 이용한 2차원 to 2차원 매핑 (d=q=2)
    3) 2원에서 3차원(고차원) 공간으로의 매핑 (d=2, q=3)

Examples

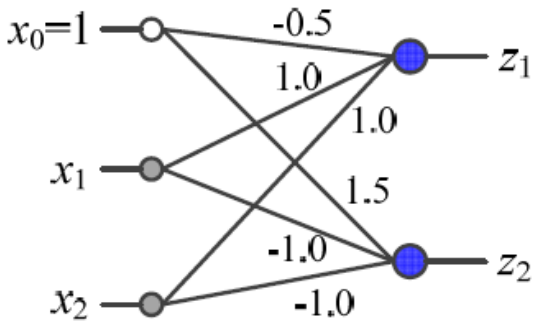
Training Set (XOR 문제)

$$\mathbf{x}_1 = \begin{pmatrix}0\\0\end{pmatrix}, y_1 = -1, \mathbf{x}_2 = \begin{pmatrix}1\\0\end{pmatrix}, y_2 = 1, \mathbf{x}_3 = \begin{pmatrix}0\\1\end{pmatrix}, y_3 = 1, \mathbf{x}_4 = \begin{pmatrix}1\\1\end{pmatrix}, y_4 = -1$$

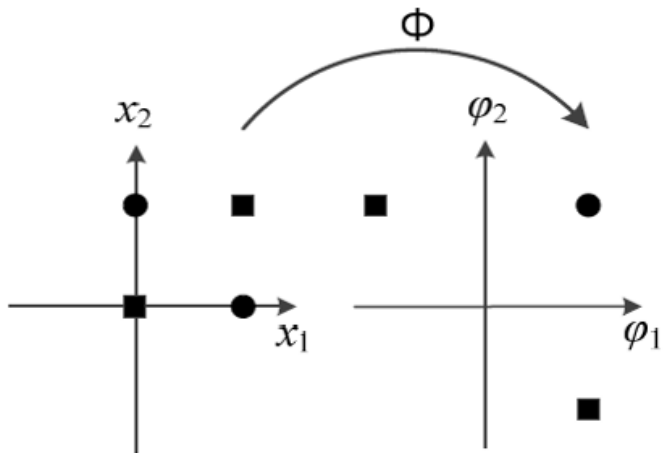


1) Perceptron (step activation)을 이용한 2차원 to 2차원 매핑 (*d*=*q*=2)

- Perceptron with step activation :



$$\phi_i(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$$



$$\begin{aligned}\mathbf{\Phi}(\mathbf{x}) &= \left(\phi_1(\mathbf{x}), \phi_2(\mathbf{x})\right)^{\mathrm{T}} \\ &= \left(\text{sign}(x_1 + x_2 - 0.5), \text{sign}(-x_1 - x_2 + 1.5)\right)^{\mathrm{T}}\end{aligned}$$

Examples (cont'd)

2) Gaussian function (RBF)을 이용한 2차원 to 2차원 매핑 (d=q=2)

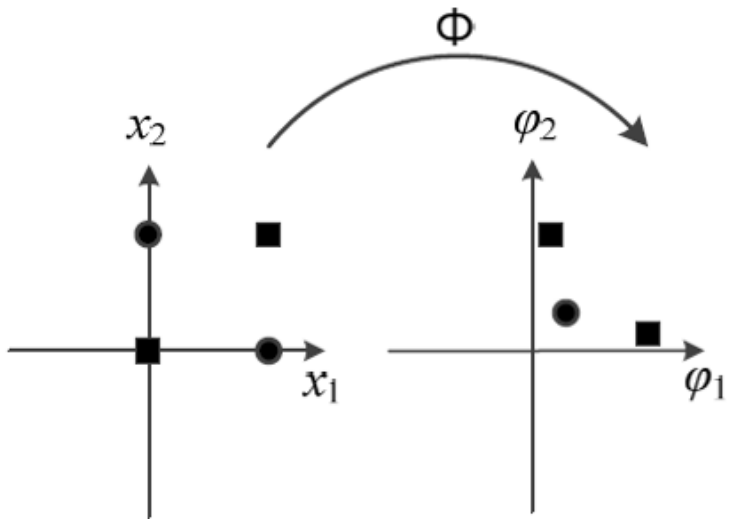
- ***RBF (Radial Basis Function)*** : a real valued function φ whose value *depends only on the distance* between the input $\mathbf{x}$ and some fixed point $\mathbf{c}$ so that $\phi(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{c}\|)$.

- ***Gaussian function*** (one of an RBF): $\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}\|_2^2}{2\sigma^2}\right)$

$$\mathbf{\Phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}))^T$$

$$= \left(\exp\left(-\left\|\mathbf{x} - \begin{pmatrix}1\\1\end{pmatrix}\right\|_2^2\right), \exp\left(-\left\|\mathbf{x} - \begin{pmatrix}0\\0\end{pmatrix}\right\|_2^2\right)\right)^T$$

c_1 c_2

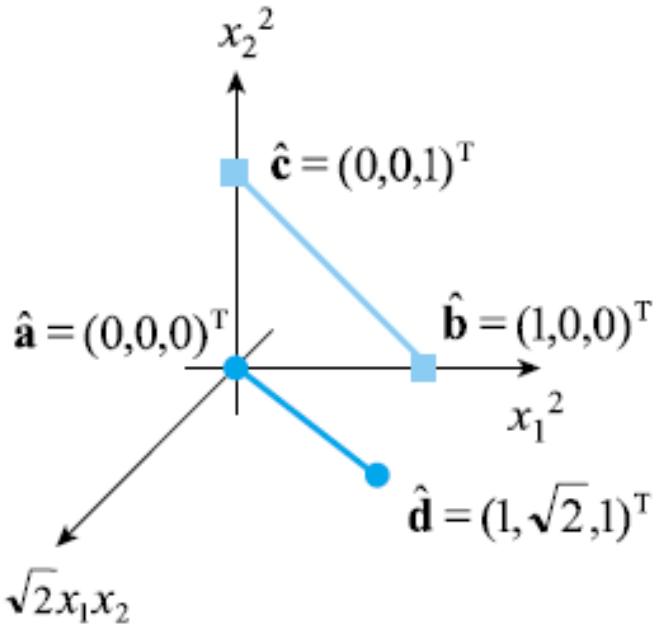


Examples (cont'd)

3) 2원에서 3차원(고차원) 공간으로의 매핑 (d=2, q=3)

$$\Phi(\mathbf{x}) = \left(\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \phi_3(\mathbf{x})\right)^{\mathrm{T}} = \left(x_1^2, \sqrt{2}\,x_1 x_2, x_2^2\right)^{\mathrm{T}}$$

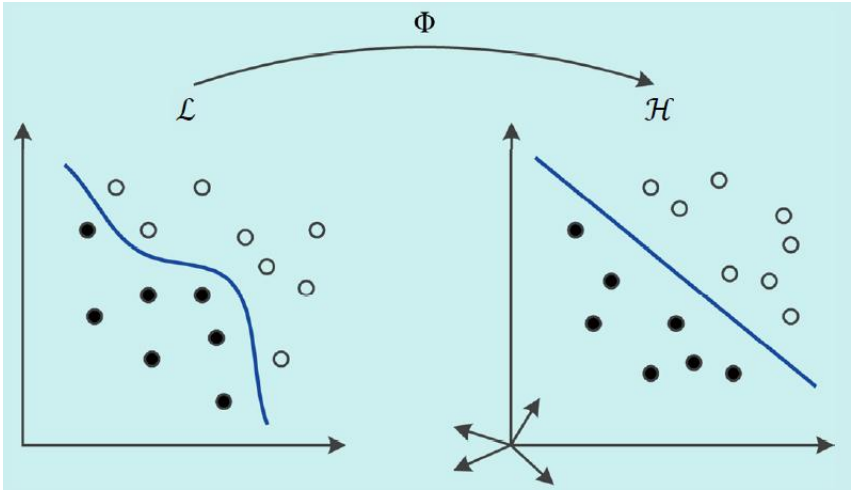
$$\mathbf{a} = (0,0)^{\mathrm{T}} \rightarrow \hat{\mathbf{a}} = (0,0,0)^{\mathrm{T}}$$
$$\mathbf{b} = (1,0)^{\mathrm{T}} \rightarrow \hat{\mathbf{b}} = (1,0,0)^{\mathrm{T}}$$
$$\mathbf{c} = (0,1)^{\mathrm{T}} \rightarrow \hat{\mathbf{c}} = (0,0,1)^{\mathrm{T}}$$
$$\mathbf{d} = (1,1)^{\mathrm{T}} \rightarrow \hat{\mathbf{d}} = (1,\sqrt{2},1)^{\mathrm{T}}$$



Summary & Issues

공간변환 + 선형SVM

- 변환된 공간에서의 특징들이 선형분리가 용이하도록 **공간변환 함수** Φ를 결정 한다.
- 데이터셋의 모든 입력 샘플에 대해 **공간변환**을 적용한다: $\{\mathbf{x}_i\}_{i=1}^n \rightarrow \{\Phi(\mathbf{x}_i)\}_{i=1}^n$
- 공간변환 된 Feature들($\{\Phi(\mathbf{x}_i)\}$)을 입력으로 하여 **선형 SVM**을 적용한다.



Issue

- 각 샘들을 모두 공간변환하기 위한 **복잡도** 증가 → ***Kernel Trick***을 사용하면 공간변환 하지 않아도 됨
- 적절한 **공간변환 함수**를 어떻게 **선택**할 것인가? → ***Kernel Trick***을 사용하면 *Kernel Function*만 결정하면 됨

Kernel Trick

Kernel Trick

- ***Kernel Trick*** **(Kernel Substitution)**
  - **공간 매핑($\Phi : \mathcal{L} \rightarrow \mathcal{H}$)을 모든 데이터셋에 적용하게 되면 데이터셋이 커질수록 계산 복잡도가 증가**
  - **커널 트릭은 어떤 수식이 벡터 내적을 포함할 때 그 내적을 커널 함수로 대치하여 계산하는 기법(트릭)**
    - $\mathbf{\Phi(x)}$로 변환한 $\mathcal{H}$ 공간에서 내적 연산을 원래 특징 공간 $\mathcal{L}$에서 커널함수 계산으로 대치
    - 제약 사항: $\mathcal{H}$ 공간에서의 연산이 내적으로 표현되어야 함 → 쌍대성duality을 이용하여 내적 표현 유도

    ★ 공간 매핑과 마찬가지로 선형 분리 불가능한 데이터를 다른 공간으로 매핑하여 선형 분리 가능하게 만들 수 있음

    ★ 모든 데이터셋에 공간 변환을 적용할 필요 없음 → 복잡도 감소

  - **커널 트릭은 메모리 기반 방법:**

    ★ 학습이 끝난 후에 훈련집합 전체 또는 일부를 메모리에 저장하고 있다가 예측에 사용

  - **적용 예:**

    ★ SVM, Fisher LD의 커널 LD로의 확장, PCA를 커널 PCA로 확장 등

Kernel Function (커널함수)

Kernel Function (*Kernel*)

원래 특징 공간 $\mathcal{L}$에 정의된 두 특징 벡터 $\mathbf{x}$와 $\mathbf{z}$에 대해 $K(\mathbf{x},\mathbf{z}) = \boldsymbol{\Phi}(\mathbf{x}) \cdot \boldsymbol{\Phi}(\mathbf{z})$인 변환함수 $\boldsymbol{\Phi}$가 존재하면 $K(\mathbf{x},\mathbf{z})$를 커널함수라 부른다.

- $L = X^d$: 원래 특징 공간
- $H = R^q$: 변환된 특징 공간
- Φ: $L \rightarrow H$ (공간 매핑)
- K: $L \times L \rightarrow R$ (커널 함수)

Kernels as *Similarity Functions*

- The dot product can be interpreted as a measure of how similar two points are.
- We now use the same intuition for a kernel:
  the kernel is a measure of how "similar" two points in the feature space are.

Kernel Function (cont'd)

- **Example:** $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$ *is a kernel function?*
  - *d=2, q=3* 인 경우의 증명:

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{z}) &= (\mathbf{x} \cdot \mathbf{z})^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= \left(x_1^2, x_2^2, \sqrt{2} x_1 x_2\right) \cdot \left(z_1^2, z_2^2, \sqrt{2} z_1 z_2\right) \\
&= \boldsymbol{\Phi}(\mathbf{x}) \cdot \boldsymbol{\Phi}(\mathbf{z})
\end{aligned}
$$

Kernel Function (cont'd)

- 널리 쓰이는 커널 함수
  - **Polynomial** Kernel : $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^p$
  - **Gaussian** (RBF) Kernel : $K(\mathbf{x}, \mathbf{z}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{z}\|_2^2}{2\sigma^2}\right)$
  - **Hyperbolic Tangent** Kernel : $K(\mathbf{x}, \mathbf{z}) = \tanh(\alpha\mathbf{x} \cdot \mathbf{z} + \beta)$
  - 1~2개의 하이퍼 매개변수를 가짐
    - ★ Polynomial: p
    - ★ Gaussian: σ
    - ★ Hyperbolic Tangent: α, β

Kernel Function (cont'd)

- **Kernel Function은 어떻게 선택해야 하나?**
  - In general, it is really difficult to prove that a certain function *K* is indeed a kernel.
  - In practice, it usually does not work to come up with a nice *similarity function* and "hope" that it is a kernel.

- **There are *some simple rules* that can help to transform and combine elementary kernels:**

Assume that $k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ are kernel functions. Then:

- $\tilde{k} = \alpha \cdot k_1$ for some constant $\alpha > 0$ is a kernel.
- $\tilde{k} = k_1 + k_2$ is a kernel
- $\tilde{k} = k_1 \cdot k_2$ is a kernel
- The pointwise limit of a sequence of kernels is a kernel.
- For any function $f : \mathcal{X} \to \mathbb{R}$, the expression $\tilde{k}(x, y) := f(x)k(x, y)f(y)$ defines a kernel.

  In particular, $\tilde{k}(x, y) = f(x)f(y)$ is a kernel.

Reference: "Statistical Machine Learning" lecture by Ulrike von Luxburg @Tubingen Univ.

Nonlinear SVM (Kernel SVM)

Nonlinear SVM (*Kernel SVM*)

Wolfe Dual (Soft-Margin Linear SVM)

$$\text{max.}: \quad \tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$$

$$\text{s.t.}: \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \le \alpha_i \le C, \qquad i = 1, 2, \cdots, n$$

① 이렇게 공간변환 후 Linear SVM을 적용하자. ⇒ 공간변환 방법

② 그런데 만약 $x_i^T x_j$? ⟶ $\phi(x_i)^T \phi(x_j) = K(x_i, x_j)$

⇒ 따라서 $x_i^T x_j$를 $K(x_i, x_j)$로 대체한 후 Linear SVM을 적용하면 ① 과 동일한 효과를 볼 수 있다! ⇒ "Kernel Trick"

Feature Conversion

$$\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^{\mathrm{T}} \Phi(\mathbf{x}_j)$$

Kernel Trick

$$\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

- 커널 함수에 대응하는 매핑 함수는 몰라도 된다. 단지 존재한다는 사실만 알면 된다.
  (왜? 실재 계산은 벡터의 내적을 커널 함수로 대치하여 하면 되기 때문!)

5. SVM Training & Inference

- ✓ SVM Learning
- ✓ SVM Prediction
- ✓ Open Source SVMs

SVM Learning

Nonlinear SVM Learning Problem (Finding α_i's)

Maximize: $\tilde{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$

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

- 목적함수 $\tilde{L}(\boldsymbol{\alpha})$는 1차 항이 n개, 2차 항이 n^2개인 아주 복잡한 식
- 등식조건 1개와 부등식 조건 n개 포함

- Linear SVM은 $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$ 인 경우로 볼 수 있음

- ***n*이 작을 때:** 해석적으로(analytically) 해를 구한다. (앞 강의 example들 참조)
- ***n*이 클 때:** 수치 최적화로(numerical optimization) 해를 구한다. (α_i 초기화 → iterative update)

SVM Learning (cont'd)

SVM 학습 전략

- 원래 문제를 다룰 수 있을 정도의 **작은 문제로 분해**해 보자.
- 먼저, n개의 Lagrange Multiplier α_i를 ***active set*** **Y**와 ***inactive set*** **Z**로 나눈다.
- Z에 속한 α_i는 상수로 간주하고, Y에 대해 최적화를 수행한다.
- 이 과정을 전체 최적화 조건을 만족할 때까지 반복한다.

알고리즘 11-1 SVM 학습

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbb{Y} = \{y_1, y_2, \cdots, y_n\}$
선형과 비선형 선택, 비선형을 선택한 경우 커널함수
출력: 라그랑주 승수 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)^T$

```
1  α1, α2, ···, αn을 초기화한다.
2  repeat
3      q개 라그랑주 승수를 선택하여 집합 Y에 넣고, 나머지는 Z에 넣는다.
4      Z에 있는 승수는 상수, Y에 있는 승수는 변수로 취급한다.
5      if(선형) [문제 11-5]를 분해한다.
6      else [문제 11-10]을 분해한다.
7      분해된 문제를 풀어 Y에 있는 승수를 새로운 값으로 변경한다.
8  until (멈춤 조건)
```

SVM Learning (cont'd)

- 대표적 SVM 최적화 알고리즘
  - **SMO** (Sequential Minimal Optimization)
    - ★ q=2를 사용
    - ★ 두 개의 Lagrange Multiplier만 구하면 됨 (해석적으로 풀 수 있음)
  - **Cutting-Plane** Algorithm [Joachims 06]
    - ★ KDD 국제학회 최고 논문상
    - ★ SMO보다 빠름
  - SVM 구현은 여전히 중요한 연구주제

SVM Prediction

SVM Prediction

- SVM 학습이 끝난 후, 새로운 샘플에 대한 label을 예측할 때

Linear SVM Prediction

- **Weight** 계산:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \, y_i \mathbf{x}_i = \sum_{\alpha_k \neq 0} \alpha_k \, y_k \mathbf{x}_k$$

- **Bias** 계산:
  Support vector들($\alpha_i > 0$)에 대해, $b = y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$

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

$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b = y_i$

★ 방법1) 아무 support vector 하나에 대해 계산한 결과를 사용

★ 방법2) 수치적 정확도를 위해 모든 support vector들에 대해 계산한 결과의 평균을 사용

- **Output** (prediction) 계산:

$$d(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \gtrless 0 \quad \begin{cases} out = 1 \\ out = -1 \end{cases}$$

★ 새로운 샘플 $\mathbf{x}$에 대한 output 계산을 위해서는 $\mathbf{w}$ 와 b를 저장해 두고 있어야 함

SVM Prediction (cont'd)

Nonlinear SVM (*Kernel* SVM) Prediction

- **Weight** 계산:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \sum_{\alpha_k \neq 0} \alpha_k y_k \mathbf{x}_k$$

- **Bias** 계산:
  Support vector들($\alpha_i > 0$)에 대해,

$$b = y_i - \mathbf{w}^T \mathbf{x}_i$$

$$b_K = y_i - \sum_{k=1}^{n} \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_i)$$

$$\mathbf{w}^T \mathbf{x}_i = \langle \mathbf{w}, \mathbf{x}_i \rangle = \left\langle \sum_{k=1}^{n} \alpha_k y_k \mathbf{x}_k, \mathbf{x}_i \right\rangle = \sum_{k=1}^{n} \alpha_k y_k \langle \mathbf{x}_k, \mathbf{x}_i \rangle$$

★ 방법1) 아무 support vector 하나에 대해 계산한 결과를 사용

★ 방법2) 수치적 정확도를 위해 모든 support vector들에 대해 계산한 결과의 평균을 사용

- **Output** (prediction) 계산:

$$d(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

$$d_K(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \gtrless 0 \begin{cases} out = 1 \\ out = -1 \end{cases}$$

★ 새로운 샘플 $\mathbf{x}$에 대한 output 계산을 위해서는 훈련집합의 일부(support vector $(\mathbf{x}_i, y_i, \alpha_i)$들)와 b를 저장해 두고 있어야 함

⇒ Kernel SVM은 메모리기반 방법!

Open Source SVMs

- **SVMLight library**
  - http://svmlight.joachim.org
  - Osuna Algorithm 사용 [Joachims 1999]

- **LIBSVM library ← *Recommended***
  - http://www.csie.ntu.edu.tw/~cjlin/libsvm
  - 개선된 SMO Algorithm 사용 [Fan 2005]

- **Others**
  - SVMJS (데모)
  - R, Matlab, SAS, Python 언어
  - OpenCV
  - Scikit Learn

Why Are SVMs So Successful?

SVMs	Neural Networks
Convex optimization problem, easy to implement	Training a NN is a *non-convex* problem
Very few variables to tune that can be done by cross validation - C and a kernel parameter (e.g., σ in the Gaussian kernel)	*Lots of parameters to tune* - how many neurons - how many layers, etc.
So simple to run Many efficient algorithms are available	So many tricks are involved Needs a large amount of experience
The kernel framework boosts the potential of the SVM to the non-linear regime, but does *not lead to excessive overfitting*	By now, DNNs are very successful for highly structured data (speech, text, images) but: - they requires *high computational power* - It is needed to prevent *excessive overfitting* - techniques are needed to train *small dataset*
Statistical learning *theory* shows many nice guarantees about the SVM (consistency, etc.)	From theory point of view, *not understood very well* why they actually work

Q & A

Q. Multi-class SVM ?

Q. SVM Regression ?

Kernel Function (참고)

- Formal Definition

Kernel function — definition

Let $\mathcal{X}$ be any space. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a kernel function if for all $n \geq 1$, $x_1, x_2, ..., x_n \in \mathcal{X}$ and $c_1, ..., c_n \in \mathbb{R}$ we have

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0.$$

Given a set of points $x_1, ..., x_n \in \mathcal{X}$, we define the corresponding kernel matrix as the matrix K with entries $k_{ij} = k(x_i, x_j)$.

The condition above is equivalent to saying that $c'Kc \geq 0$ for all $c \in \mathbb{R}^n$.

Reference: "Statistical Machine Learning" lecture by Ulrike von Luxburg @Tubingen Univ.

Kernel Function (참고)

- **Theorem:** ***Kernel Implies Embedding***

A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if and only if there exists a Hilbert space $\mathcal{H}$ and a map $\Phi : \mathcal{X} \to \mathcal{H}$ such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$.

If you have never heard of Hilbert spaces, just think of the space $\mathbb{R}^d$. The crucial properties are:

- $\mathcal{H}$ is a vector space with a scalar product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$
- Space is complete (all Cauchy sequences converge)
- Scalar product gives rise to a norm: $\|x\|_{\mathcal{H}} := \langle x, x \rangle_{\mathcal{H}}$

Reference: "Statistical Machine Learning" lecture by Ulrike von Luxburg @Tubingen Univ.

Cf.) Wikipedia:

- A **Hilbert space** is a vector space equipped with an inner product which defines a distance function for which it is a complete metric space.
- Hilbert spaces allow generalizing the methods of linear algebra and calculus from finite-dimensional Euclidean vector spaces to spaces that may be infinite-dimensional.
- Hilbert spaces arise naturally and frequently in mathematics and physics, typically as function spaces.