Support Vector Machines in the Primal

Mingmin Chi

Fudan University, Shanghai, China

Outline

Objective Function

- Recall that the primal objective function for the soft-margin SVMs with constraints is

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i, \text{ s.t. } \quad \forall_{i=1}^{n} : y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i, \xi_i \geq 0$$

Objective Function

- Recall that the primal objective function for the soft-margin SVMs with constraints is

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i, \text{ s.t. } \forall_{i=1}^{n} : y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i, \xi_i \geq 0$$

- By the algebra operation, the constraints can be integrated in the objective function such that the objective function without constraints is:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} V\left(y_i, \mathbf{w}^\top\mathbf{x}_i + b\right)$$

Hinge Loss Function

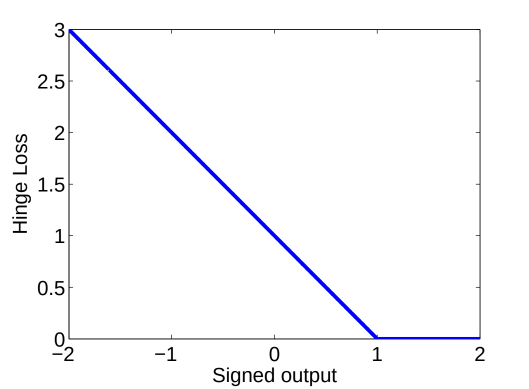
Losses for training samples

$V\left(y_i, \mathbf{w}^\top \mathbf{x}_i + b\right)$ is the loss for the training patterns $\mathbf{x}_i \in \mathbf{X}_l$, defined by $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})$.
This is so-called *hinge loss*

Hinge Loss Function

Losses for training samples

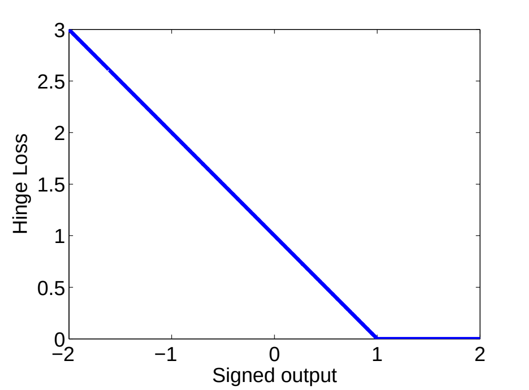
$V\left(y_i, \mathbf{w}^\top \mathbf{x}_i + b\right)$ is the loss for the training patterns $\mathbf{x}_i \in \mathbf{X}_l$, defined by $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})$. This is so-called *hinge loss*



Hinge Loss Function

Losses for training samples

$V\left(y_i, \mathbf{w}^\top \mathbf{x}_i + b\right)$ is the loss for the training patterns $\mathbf{x}_i \in \mathbf{X}_l$, defined by $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})$. This is so-called *hinge loss*



The objective function of SVMs is convex $\Rightarrow$ no local minima

Support Vectors

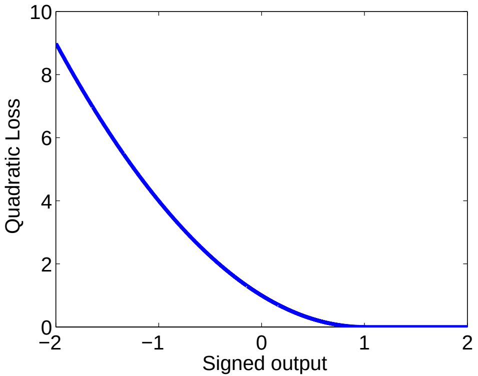
- Only the training sample with $V(y_i, f(\mathbf{x}_i)) \neq 0$ makes a contribution on the loss function,

Support Vectors

- Only the training sample with $V(y_i, f(\mathbf{x}_i)) \neq 0$ makes a contribution on the loss function, i.e., $y_i f(\mathbf{x}_i) < 1$
- The samples with the nonzero losses are support vectors

Quadratic Loss Function

We can approximate the hinge loss by a quadratic form



Quadratic Loss Function (Contd)

- To do so, we can replace $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})$ by $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})^2$

Quadratic Loss Function (Contd)

- To do so, we can replace $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})$ by $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})^2$
- We use the L2-norm loss in the primal objective function as follows:
  - with constraints

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i^2, \text{ s.t. } \quad \forall_{i=1}^{n} : y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i, \xi_i \geq 0$$

  - without constraints:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} V\left(y_i, \mathbf{w}^\top\mathbf{x}_i + b\right)$$

where $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})^2$

Loss Functions

In general, we can write down the common used L1/L2-norm loss as follows:

- with constraints

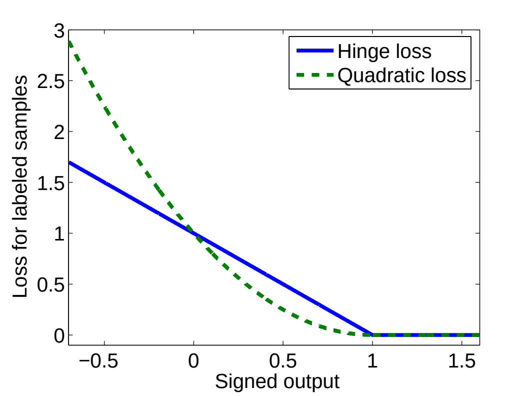
$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i^p, \text{ s.t. } \quad \forall_{i=1}^{n} : y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq 1 - \xi_i, \xi_i \geq 0$$

- without constraints:

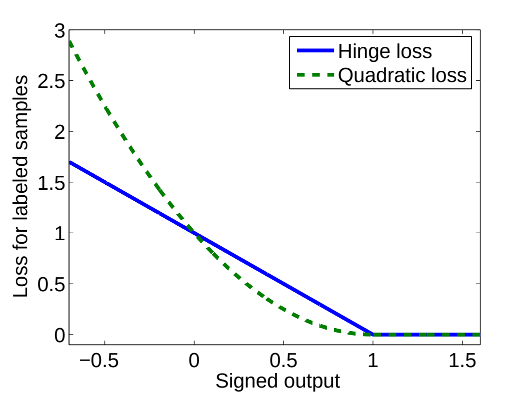
$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} V\left(y_i, \mathbf{w}^\top\mathbf{x}_i + b\right)$$

where $V(y, \mathbf{t}) = \max(0, 1 - y\mathbf{t})^p$

Loss Functions (Contd)

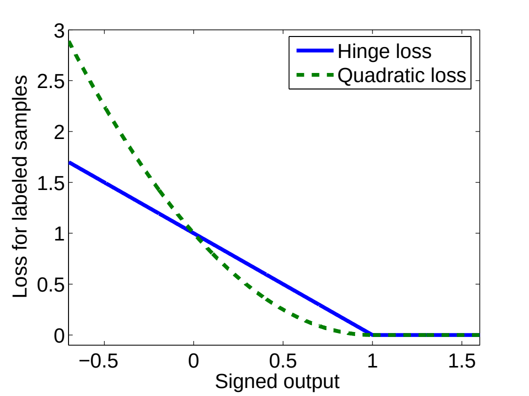


Loss Functions (Contd)



$p = 1$: Hinge loss

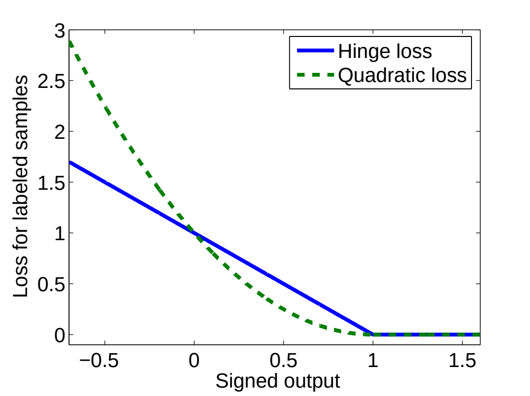
Loss Functions (Contd)



$p = 1$: Hinge loss

- Non-differentiable
- Used in dual SVMs in the last lecture

Loss Functions (Contd)

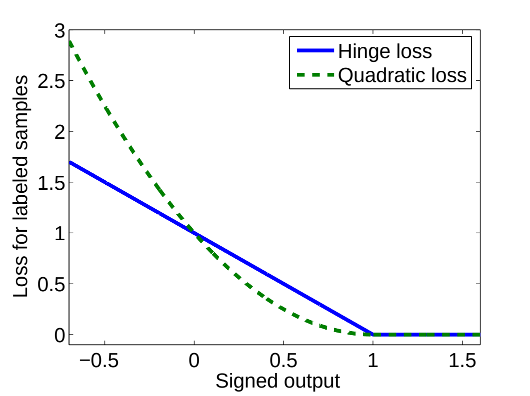


$p = 1$: Hinge loss

- Non-differentiable
- Used in dual SVMs in the last lecture

$p = 2$: Quadratic loss

Loss Functions (Contd)



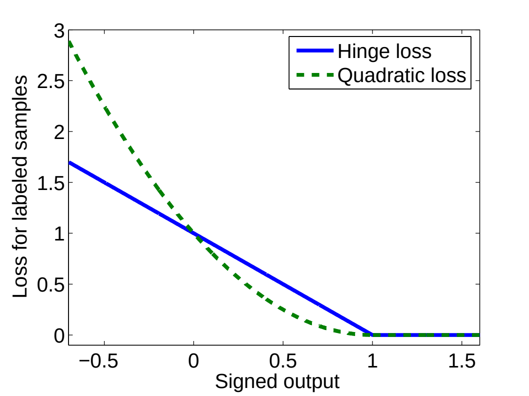
$p = 1$: Hinge loss

- Non-differentiable
- Used in dual SVMs in the last lecture

$p = 2$: Quadratic loss

- Differentiable
- Used in primal SVMs by gradient descent

Loss Functions (Contd)



$p = 1$: Hinge loss

- Non-differentiable
- Used in dual SVMs in the last lecture

$p = 2$: Quadratic loss

- Differentiable
- Used in primal SVMs by gradient descent

In general, the $V(\cdot, \cdot)$ could be any loss function but usually is chosen to be convex for the ease of optimization problems

Finite Newton Method

Implementation

- Mangasarian finite newton method (Mangasarian, 2002) does iterations of the form

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \delta_k \mathbf{p}_k$$
$$\text{and } \mathbf{p}_k = -H_k^{-1} \nabla_k$$

  The step size δ_k is chosen to satisfy an Armijo condition that ensures convergence
- Modified Newton method (Keerthi and DeCoste, 2005)

Empirical Risk Functionals

Recall that the empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V\left(y_i, f(\mathbf{x}_i)\right)$$

Empirical Risk Functionals

Recall that the empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n} \sum_{i=1}^{n} V\left(y_i, f(\mathbf{x}_i)\right)$$

- for the classical regularization Networks

$$V\left(y_i, f(\mathbf{x}_i)\right) = \left(y_i - f(\mathbf{x}_i)\right)^2$$

Empirical Risk Functionals

Recall that the empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V\left(y_i, f(\mathbf{x}_i)\right)$$

- for the classical regularization Networks

$$V\left(y_i, f(\mathbf{x}_i)\right) = \left(y_i - f(\mathbf{x}_i)\right)^2$$

- for the support vector classification:

$$V\left(y_i, f(\mathbf{x}_i)\right) = |1 - y_i f(\mathbf{x}_i)|_+, \text{ where } |t|_+ = \begin{cases} t, & t \geq 0 \\ 0, & t < 1 \end{cases}$$

Empirical Risk Functionals (Contd)

Empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V\left(y_i, f(\mathbf{x}_i)\right)$$

Empirical Risk Functionals (Contd)

Empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i))$$

- for the support vector classification:

$$V(y_i, f(\mathbf{x}_i)) = |1 - y_i f(\mathbf{x}_i)|_+, \text{ where } |t|_+ = \begin{cases} t, & t \geq 0 \\ 0, & t < 1 \end{cases}$$

Empirical Risk Functionals (Contd)

Empirical risk functional is defined as

$$R_{\text{emp}}[f] = \frac{1}{n}\sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i))$$

- for the support vector classification:

$$V(y_i, f(\mathbf{x}_i)) = |1 - y_i f(\mathbf{x}_i)|_+, \text{ where } |t|_+ = \begin{cases} t, & t \geq 0 \\ 0, & t < 1 \end{cases}$$

- for the support vector regression:

$$V(y_i, f(\mathbf{x}_i)) = |y_i - f(\mathbf{x}_i)|_\epsilon, \text{ where } |t|_\epsilon = \begin{cases} 0, & |t| \leq \epsilon \\ |t| - \epsilon, & |t| > \epsilon \end{cases}$$

Regularized Risk Functionals

- The problem of approximating a function from sparse data is ill-posed and a classical way to solve it is regularization theory

Regularized Risk Functionals

- The problem of approximating a function from sparse data is ill-posed and a classical way to solve it is regularization theory
- With small-sized training dataset problem, we would like to formulate the classification problem as a variational problem of finding the function f that minimizes the functional

$$\min_{f \in \mathcal{H}} R_{\text{reg}}[f] = R_{\text{emp}}[f] + \lambda \Omega[f]$$

where

1. $\Omega[f]$ is a smooth function and usually chosen to be convex, e.g., $\|f\|_{\mathcal{H}}^2$
2. λ is the regularization parameter

The objective function of SVMs

SVMs are the typical application of the regularized risk functional:

$$\begin{aligned}\min_{f \in \mathcal{H}} R_{\text{reg}}[f] &= R_{\text{emp}}[f] + \lambda \Omega[f] \\ &= \frac{1}{n} \sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_{\mathcal{H}}^2\end{aligned}$$

where

$$V(y_i, f(\mathbf{x}_i)) = |1 - y_i f(\mathbf{x}_i)|_+, \text{ where } |t|_+ = \begin{cases} t, & t \geq 0 \\ 0, & t < 1 \end{cases}$$

The objective function of SVMs (Contd)

- We usually write down the objective of SVMs as follows:

$$\min_{f \in \mathcal{H}} R_{\text{reg}}[f] = \frac{1}{2}\|f\|^2_{\mathcal{H}} + C \sum_{i=1}^{n} V(y_i, f(\mathbf{x}_i))$$

  where $C = \frac{1}{2n\lambda}$
- When $\lambda = 0$, i.e., $C \propto \infty$, the hard-margin SVMs recovery.

Reproducing Property

- Define the following reproducing kernel map for a kernel $k(\cdot, \cdot)$:

$$\phi : \ \mathbf{x} \rightarrow k(\mathbf{x}, \cdot)$$

  i.e., to each point $\mathbf{x}$ in the original space we associate a function $k(\mathbf{x}, \cdot)$ (example)

- We now construct a vector space containing all linear combinations of the functions $k(\mathbf{x}, \cdot)$

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$$

  for arbitrarily $\{\mathbf{x} \in \mathcal{X}\}_{i=1}^{n}$. This will be used to construct our RKHS $\mathcal{H}$.

Reproducing Kernel Hilbert Space (RKHS)

- The Hilbert space L_2 is too “big" for our purposes, containing too many non-smooth functions. One approach to obtaining restricted, smooth spaces is the Reproducing Kernel Hilbert Space (RKHS) approach. A RKHS is “smaller" than a general Hilbert space.
- We now define an inner product. Let $f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$ and $g(\cdot) = \sum_{j=1}^{n} \beta_j k(\mathbf{x}_j', \cdot)$, such that $f, g \in \mathcal{H}$ and define

$$< f, g >= \sum_{ij} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j')$$

- For any $f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot)$, we have

$$< f, k(\mathbf{x}, \cdot) >= \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = f(\mathbf{x})$$

Representer Theory

(Kimeldorf and Wahba, 1970).

- Assume $\mathcal{H}$ is the RKHS associated to the kernel k, each minimizer $f \in \mathcal{H}$ of the regularized risk

$$c(y_i, f(\mathbf{x}_i)) + \lambda\Omega[f]$$

admits a representation of the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Feature Spaces

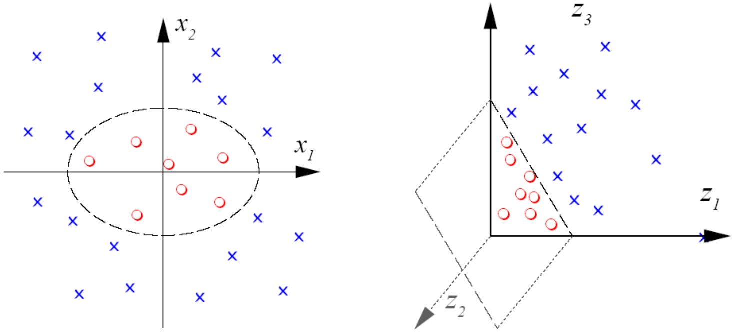
- Preprocess the data with

$$\begin{aligned} \phi : \mathcal{X} &\to \mathcal{H} \\ \mathbf{x} &\to \phi(\mathbf{x}), \end{aligned}$$

  where $\mathcal{H}$ is a dot product space and learn the mapping from $\phi(\mathbf{x})$ to the output y
- Usually, $\dim(\mathcal{X}) \ll \dim(\mathcal{H})$, capacity control or complexity of the hypothesis space

Nonlinear Mapping (Example)

$$
\begin{aligned}
\phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\
(x_1, x_2) &\rightarrow (z_1, z_2, z_3) \; : (x_1^2, x_2^2, \sqrt{2}x_1 x_2) \\
\phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2) &= (\mathbf{x}_1^\top \mathbf{x}_2)^2
\end{aligned}
$$



Nonlinear Mapping (Example, Contd)

$$
\begin{aligned}
\phi :& \mathbb{R}^2 \rightarrow \mathbb{R}^3 \\
& (x_1, x_2) \rightarrow (z_1, z_2, z_3) \ : (x_1^2, x_2^2, \sqrt{2}x_1 x_2) \\
& \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2)^2 \\
\phi :& \mathbb{R}^2 \rightarrow \mathbb{R}^4 \\
& (x_1, x_2) \rightarrow (z_1, z_2, z_3, z_4) \ : (x_1^2, x_2^2, x_1 x_2, x_2 x_1) \\
& \phi(\mathbf{x}_1)^\top \phi(\mathbf{x}_2) = (\mathbf{x}_1^\top \mathbf{x}_2)^2
\end{aligned}
$$

Primal Formulation

- If the kernel exists, we can define the mapping

$$\phi(\mathbf{x}) = k(\cdot, \mathbf{x})$$

- By the Representer theorem, we have

$$f(\cdot) = \sum_{i=1}^{n} \beta_i k(\cdot, \mathbf{x}_i)$$

and with the reproducing property, we have

$$f(\mathbf{x}) =< f(\cdot), k(\cdot, \mathbf{x}) >= \sum_{i=1}^{n} \beta_i k(\mathbf{x}, \mathbf{x}_i) \tag{1}$$

Primal Formulation (Contd)

- Taking (1) into the objective function of SVMs, we have

$$
\begin{aligned}
&\frac{1}{2}\sum_{i=1}^{n}\beta_i k(\mathbf{x}_i,\cdot)\sum_{j=1}^{n}\beta_j k(\cdot,\mathbf{x}_j) + C\sum_{i=1}^{n} V\left(y_i, \sum_{j=1}^{n}\beta_j k(\mathbf{x}_i,\mathbf{x}_j)\right)\\
= \quad & \frac{1}{2}\beta^\top \mathbf{K}\beta + C\sum_{i=1}^{n} V\left(y_i, \mathbf{K}_i^\top \beta\right)
\end{aligned}
$$

  where $\mathbf{K}_i = [k(\mathbf{x}_i,\mathbf{x}_j)]_{j=1}^{n} \in \mathbb{R}^{n\times 1}$ is the i^{th} column of $\mathbf{K}$
- If $V(\cdot,\cdot)$ is differentiable, the optimum value β^* can be obtained by gradient descent with respect to β

Newton Method for Nonlinear Primal SVM

Define $\mathbf{I}^0$ an $n \times n$ diagonal matrix with the first n_{sv} entries being 1 and the others 0 [**?**].

$$I^0 \equiv \begin{pmatrix} 1 & & & & & \\ & \ddots & & & 0 & \\ & & 1 & & & \\ & & & 0 & & \\ & 0 & & & \ddots & \\ & & & & & 0 \end{pmatrix}$$

Gradient

$$\begin{aligned} \nabla &= 2\lambda K\boldsymbol{\beta} + \sum_{i=1}^{n_{\text{sv}}} K_i \frac{\partial L}{\partial t}(y_i, K_i^\top \boldsymbol{\beta}) \\ &= 2\lambda K\boldsymbol{\beta} + 2\sum_{i=1}^{n_{\text{sv}}} K_i y_i (y_i K_i^\top \boldsymbol{\beta} - 1) \\ &= 2(\lambda K\boldsymbol{\beta} + K I^0 (K\boldsymbol{\beta} - Y)), \end{aligned}$$

Hessian: $H = 2(\lambda K + K\mathbf{I}^0 K)$

Newton step: $\boldsymbol{\beta} \Leftarrow \boldsymbol{\beta} - H^{-1}\nabla$

$$\begin{aligned} \boldsymbol{\beta} &= \begin{pmatrix} (\lambda I_{n_{\text{sv}}} + K_{\text{sv}})^{-1} & 0 \\ 0 & 0 \end{pmatrix} Y, \\ &= \begin{pmatrix} (\lambda I_{n_{\text{sv}}} + K_{\text{sv}})^{-1} Y_{\text{sv}} \\ 0 \end{pmatrix}. \end{aligned}$$

Data Preprocessing

- When the labeled samples $(\mathbf{x}_i)_{i=1}^n$ are mapped to the feature space, they span a vectorial subspace $\mathcal{S}$ ($\mathcal{S} \subset \mathcal{H}$), whose dimension is at most n.
- By choosing a basis for $\mathcal{S}$ and expressing the coordinates of all the points in that basis, we can then directly work in $\mathcal{S}$.
- Let $(\mathbf{v}_1, \cdots, \mathbf{v}_n) \in \mathcal{S}$ be orthonormal basis of $\mathcal{S}$ with $\mathbf{v}_i$ expressed as:

$$\mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp}\Phi(\mathbf{x}_j), \quad 1 \leq p \leq n$$

- Since $(\mathbf{v}_1, \cdots, \mathbf{v}_n)$ is an orthonormal basis (i.e., $\mathbf{v}_p^\top \mathbf{v}_q = \delta_{pq}$), we have

$$\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$$

Data Preprocessing

- As we have $\mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp}\Phi(\mathbf{x}_j), \;\; 1 \leq p \leq n$,

$$\begin{aligned} \mathbf{v}_p^\top \mathbf{v}_q = \sum_{i=1}^{n}\sum_{j=1}^{n} \mathbf{A}_{ip}\mathbf{A}_{jq}\Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j) \\ &= (\mathbf{A}^\top \mathbf{K}\mathbf{A})_{pq} \\ &= \delta_{pq} \end{aligned}$$

- This is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$

Data Preprocessing

- If we can use the following map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and recalling that $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$, we can express the transformed data based on a kernel function:

$$\tilde{\mathbf{x}}_i = \psi(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top \mathbf{A}_i$$

- The p-th component of $\psi(\mathbf{x}_i)$ can be derived as follows:

$$\psi_p(\mathbf{x}_i) = \Phi(\mathbf{x}_i)^\top \mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp} k(\mathbf{x}_i, \mathbf{x}_j), 1 \leq p \leq n$$

- It is easy to check that:

$$\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)$$

Data Preprocessing

- If we can use the following map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and recalling that $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$, we can express the transformed data based on a kernel function:

$$\tilde{\mathbf{x}}_i = \psi(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top \mathbf{A}_i$$

- The p-th component of $\psi(\mathbf{x}_i)$ can be derived as follows:

$$\psi_p(\mathbf{x}_i) = \Phi(\mathbf{x}_i)^\top \mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp} k(\mathbf{x}_i, \mathbf{x}_j), 1 \leq p \leq n$$

- It is easy to check that:

$$\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) = (k(\mathbf{x}_i, \cdot)^\top \mathbf{A}_i)^\top k(\mathbf{x}_j, \cdot)^\top \mathbf{A}_j$$

Data Preprocessing

- If we can use the following map $\psi : \mathbb{R}^d \to \mathbb{R}^n$ and recalling that $\Phi : \mathbb{R}^d \to \mathcal{H}$, we can express the transformed data based on a kernel function:

$$\tilde{\mathbf{x}}_i = \psi(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top \mathbf{A}_i$$

- The p-th component of $\psi(\mathbf{x}_i)$ can be derived as follows:

$$\psi_p(\mathbf{x}_i) = \Phi(\mathbf{x}_i)^\top \mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp} k(\mathbf{x}_i, \mathbf{x}_j), 1 \leq p \leq n$$

- It is easy to check that:

$$\begin{aligned}\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) &= (k(\mathbf{x}_i, \cdot)^\top \mathbf{A}_i)^\top k(\mathbf{x}_j, \cdot)^\top \mathbf{A}_j \\ &= \mathbf{A}_i^\top k(\mathbf{x}_i, \cdot)^\top k(\mathbf{x}_j, \cdot) \mathbf{A}_j\end{aligned}$$

Data Preprocessing

- If we can use the following map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and recalling that $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$, we can express the transformed data based on a kernel function:

$$\tilde{\mathbf{x}}_i = \psi(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top \mathbf{A}_i$$

- The p-th component of $\psi(\mathbf{x}_i)$ can be derived as follows:

$$\psi_p(\mathbf{x}_i) = \Phi(\mathbf{x}_i)^\top \mathbf{v}_p = \sum_{j=1}^{n} \mathbf{A}_{jp} k(\mathbf{x}_i, \mathbf{x}_j), 1 \leq p \leq n$$

- It is easy to check that:

$$\begin{aligned}
\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) &= (k(\mathbf{x}_i, \cdot)^\top \mathbf{A}_i)^\top k(\mathbf{x}_j, \cdot)^\top \mathbf{A}_j \\
&= \mathbf{A}_i^\top k(\mathbf{x}_i, \cdot)^\top k(\mathbf{x}_j, \cdot) \mathbf{A}_j \\
&= k(\mathbf{x}_i, \mathbf{x}_j)
\end{aligned}$$

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e.,

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since $\mathbf{A}$ must satisfy $\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$,

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., **K** = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since **A** must satisfy $\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$, this is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since $\mathbf{A}$ must satisfy $\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$, this is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$
3. With the map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$,

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since **A** must satisfy $\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$, this is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$
3. With the map $\psi : \mathbb{R}^d \to \mathbb{R}^n$ and $\Phi : \mathbb{R}^d \to \mathcal{H}$, we have $\tilde{\mathbf{X}}_l = \mathbf{K}^\top \mathbf{A} = (\mathbf{A}^\top)^{-1}$

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since $\mathbf{A}$ must satisfy $\mathbf{A}^\top\mathbf{K}\mathbf{A} = \mathbf{I}$, this is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$
3. With the map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$, we have $\tilde{\mathbf{X}}_l = \mathbf{K}^\top\mathbf{A} = (\mathbf{A}^\top)^{-1}$
4. So we can get the new representation of the input data with $\tilde{\mathbf{X}}_l$ by the Cholesky decomposition on the kernel function $\mathbf{K}$, i.e., $\tilde{\mathbf{X}}_l = \text{chol}(\mathbf{K})$

Computing **A**: Cholesky decomposition of **K**

1. Computing the kernel matrix in terms of the training set $\mathbf{X}_l$ and the related kernel parameters α, i.e., $\mathbf{K}$ = Compute_Kernel($\mathbf{X}_l, \alpha$)
2. Since $\mathbf{A}$ must satisfy $\mathbf{A}^\top \mathbf{K} \mathbf{A} = \mathbf{I}$, this is equivalent to $\mathbf{K} = (\mathbf{A}\mathbf{A}^\top)^{-1}$
3. With the map $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ and $\Phi : \mathbb{R}^d \rightarrow \mathcal{H}$, we have $\tilde{\mathbf{X}}_l = \mathbf{K}^\top \mathbf{A} = (\mathbf{A}^\top)^{-1}$
4. So we can get the new representation of the input data with $\tilde{\mathbf{X}}_l$ by the Cholesky decomposition on the kernel function $\mathbf{K}$, i.e., $\tilde{\mathbf{X}}_l = \text{chol}(\mathbf{K})$

With the new representation of the input data set $\tilde{\mathbf{X}}$, we can train the linear SVMs in the primal.