

Machine Learning

A Bayesian and Optimization Perspective

Academic Press, 2015

Sergios Theodoridis¹

¹Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens, Athens, Greece.

Spring 2017, Version III

Chapter 11

Learning In Reproducing Kernel Hilbert Spaces

The Need for Nonlinear Models

- The focus in this part of the lectures will be on learning **nonlinear models**. For most of the problems met in real applications, linear models cannot model sufficiently well the data and lead to poor performance.
- For example, recall that given two jointly distributed random vectors $(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^k \times \mathbb{R}^l$, then we know that the optimal estimate of \mathbf{y} given $\mathbf{x} = \mathbf{x}$, in the mean-square-error sense (MSE), is the corresponding conditional mean, i.e., $\mathbb{E}[\mathbf{y}|\mathbf{x}]$, which in general is a **nonlinear function of \mathbf{x}** .

The Need for Nonlinear Models

- The focus in this part of the lectures will be on learning **nonlinear models**. For most of the problems met in real applications, linear models cannot model sufficiently well the data and lead to poor performance.
- For example, recall that given two jointly distributed random vectors $(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^k \times \mathbb{R}^l$, then we know that the optimal estimate of \mathbf{y} given $\mathbf{x} = \mathbf{x}$, in the mean-square-error sense (MSE), is the corresponding conditional mean, i.e., $\mathbb{E}[\mathbf{y}|\mathbf{x}]$, which in general is a **nonlinear function of \mathbf{x}** .

The Need for Nonlinear Models

- Our emphasis will be on a path through the so called **Reproducing Kernel Hilbert Spaces** (RKHS). The technique consists of a **mapping of the input variables** to a new space, such that the originally **nonlinear task is transformed into a linear one**.
- From a practical point of view, the beauty behind these spaces is that their rich structure allows to perform **inner product** operations in a very efficient way, with **complexity independent of the dimensionality** of the respective RKH space. Moreover, note that the dimension of such spaces can even be infinite.

The Need for Nonlinear Models

- Our emphasis will be on a path through the so called **Reproducing Kernel Hilbert Spaces** (RKHS). The technique consists of a **mapping of the input variables** to a new space, such that the originally **nonlinear task is transformed** into a linear one.
- From a practical point of view, the beauty behind these spaces is that their rich structure allows to perform **inner product** operations in a very efficient way, with **complexity independent of the dimensionality** of the respective RKH space. Moreover, note that the dimension of such spaces can even be infinite.

Generalized Linear Models

- Given $(y, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$, a **generalized linear estimator** \hat{y} of y has the form,

$$\hat{y} = f(\mathbf{x}) := \theta_0 + \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}), \quad (1)$$

where $\phi_1(\cdot), \dots, \phi_K(\cdot)$ are **preselected** (nonlinear) functions. A popular family of functions is the polynomial one, e.g.,

$$\hat{y} = \theta_0 + \sum_{i=1}^l \theta_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^l \theta_{im} x_i x_m + \sum_{i=1}^l \theta_{ii} x_i^2.$$

- Assuming $l = 2$ ($\mathbf{x} = [x_1, x_2]^T$), then the previous equation can be brought into the form of (1), by setting $K = 5$ and $\phi_1(\mathbf{x}) = x_1$, $\phi_2(\mathbf{x}) = x_2$, $\phi_3(\mathbf{x}) = x_1 x_2$, $\phi_4(\mathbf{x}) = x_1^2$, $\phi_5(\mathbf{x}) = x_2^2$.

Generalized Linear Models

- Given $(y, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$, a **generalized linear estimator** \hat{y} of y has the form,

$$\hat{y} = f(\mathbf{x}) := \theta_0 + \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}), \quad (1)$$

where $\phi_1(\cdot), \dots, \phi_K(\cdot)$ are **preselected** (nonlinear) functions. A popular family of functions is the polynomial one, e.g.,

$$\hat{y} = \theta_0 + \sum_{i=1}^l \theta_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^l \theta_{im} x_i x_m + \sum_{i=1}^l \theta_{ii} x_i^2.$$

- Assuming $l = 2$ ($\mathbf{x} = [x_1, x_2]^T$), then the previous equation can be brought into the form of (1), by setting $K = 5$ and $\phi_1(\mathbf{x}) = x_1$, $\phi_2(\mathbf{x}) = x_2$, $\phi_3(\mathbf{x}) = x_1 x_2$, $\phi_4(\mathbf{x}) = x_1^2$, $\phi_5(\mathbf{x}) = x_2^2$.

Generalized Linear Models

- The generalization to r -th order polynomials will contain products of the form $x_1^{p_1} x_2^{p_2} \cdots x_l^{p_l}$, with $p_1 + p_2 + \dots + p_l \leq r$. It turns out that the number of free parameters, K , for an r -th order polynomial is equal to

$$K = \frac{(l+r)!}{r!l!}.$$

- Just to get a feeling, for $l = 10$ and $r = 3$, $K = 286$. The use of polynomial expansions is justified by the **Weierstrass theorem**.

Generalized Linear Models

- The generalization to r -th order polynomials will contain products of the form $x_1^{p_1} x_2^{p_2} \cdots x_l^{p_l}$, with $p_1 + p_2 + \dots + p_l \leq r$. It turns out that the number of free parameters, K , for an r -th order polynomial is equal to

$$K = \frac{(l+r)!}{r!l!}.$$

- Just to get a feeling, for $l = 10$ and $r = 3$, $K = 286$. The use of polynomial expansions is justified by the **Weierstrass theorem**.

Generalized Linear Models

- A common characteristic of this type of models is that, the basis functions in the expansion are **preselected** and they are **fixed and independent of the data**. The advantage of such a path is that the associated models are **linear** with respect to the unknown **set of free parameters**.
- For an expansion involving K **fixed** functions, the squared approximation error **cannot** be made smaller than order $\left(\frac{1}{K}\right)^{\frac{2}{i}}$. In words, for high dimensional spaces, in order to get a **small enough error** one has to use **large values of K** .

Generalized Linear Models

- A common characteristic of this type of models is that, the basis functions in the expansion are **preselected** and they are **fixed and independent of the data**. The advantage of such a path is that the associated models are **linear** with respect to the unknown **set of free parameters**.
- For an expansion involving K **fixed** functions, the squared approximation error **cannot** be made smaller than order $\left(\frac{1}{K}\right)^{\frac{2}{t}}$. In words, for high dimensional spaces, in order to get a **small enough error** one has to use **large values of K** .

Generalized Linear Models

- One way to bypass the dependence of the approximation error on the input space dimensionality, l , is to employ an expansion in terms of **data-dependent functions**, which are **optimized with respect to the specific data set**. This is, for example, the case for a class of neural networks, which are treated in Chapter 18.
- In the sequel, a different path will be followed to “**embed**” linearity into the treatment of an originally nonlinear task, via the use of RKH spaces.

Generalized Linear Models

- One way to bypass the dependence of the approximation error on the input space dimensionality, l , is to employ an expansion in terms of **data-dependent functions**, which are **optimized with respect to the specific data set**. This is, for example, the case for a class of neural networks, which are treated in Chapter 18.
- In the sequel, a different path will be followed to “**embed**” linearity into the treatment of an originally nonlinear task, via the use of RKH spaces.

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- We have already justified the expansion in (1) by mobilizing arguments from the approximation theory. However, in the context of classification tasks, looking at this expansion from a different angle provides an alternative and important interpretation.
- Let us consider N points, $x_1, x_2, \dots, x_N \in \mathbb{R}^l$. We can say that these points are **in general position**, if **there is no** subset of $l + 1$ of them lying on a $(l - 1)$ -dimensional hyperplane. For example, in the two dimensional space, any three of these points are not permitted to lie on a straight line.

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- We have already justified the expansion in (1) by mobilizing arguments from the approximation theory. However, in the context of classification tasks, looking at this expansion from a different angle provides an alternative and important interpretation.
- Let us consider N points, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^l$. We can say that these points are **in general position**, if **there is no** subset of $l + 1$ of them lying on a $(l - 1)$ -dimensional hyperplane. For example, in the two dimensional space, any three of these points are not permitted to lie on a straight line.

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- **Cover's theorem:** The number of groupings, denoted as $\mathcal{O}(N, l)$, that can be formed by $(l - 1)$ -dimensional hyperplanes to separate the N points in **two** classes, exploiting **all possible combinations**, is given by,

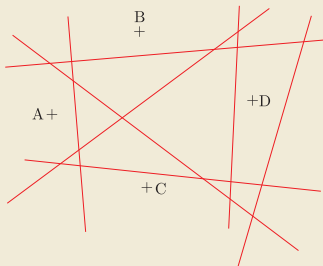
$$\mathcal{O}(N, l) = 2 \sum_{i=0}^l \binom{N-1}{i},$$

where

$$\binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!i!}.$$

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Each one of these groupings in two classes is also known as a **linear dichotomy**. The figure below illustrates the theorem for the case of $N = 4$ points in the two dimensional space. Observe that the possible groupings are: $[(ABCD)]$, $[A,(BCD)]$, $[B,(ACD)]$, $[C,(ABD)]$, $[D,(ABC)]$, $[(AB), (CD)]$ and $[(AC),(BD)]$. Each grouping is counted twice, since it can belong to either ω_1 or ω_2 class. Hence, the total number of groupings is 14, which is equal to $\mathcal{O}(4, 2)$. Note that the number of **all possible combinations of N points in two groups is 2^N** , which is 16 in our case. The grouping which is not counted is $[(BC),(AD)]$, which is not linearly separable.



Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note that if $N \leq l + 1$ then $\mathcal{O}(N, l) = 2^N$. That is, **all possible combinations in two groups are linearly separable**; verify it for the case of $N = 3$ in the two-dimensional space.
- Based on the previous theorem, given N points in the l -dimensional space, the probability of grouping these points in two **linearly separable** classes is,

$$P_N^l = \frac{\mathcal{O}(N, l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^l \binom{N-1}{i}, & N > l + 1, \\ 1 & N \leq l + 1. \end{cases}$$

Cover's Theorem: Capacity of a Space in Linear Dichotomies

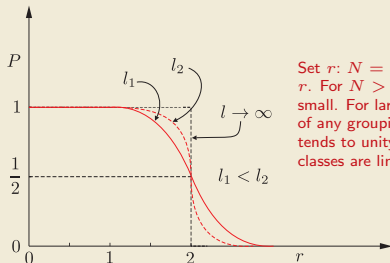
- Note that if $N \leq l + 1$ then $\mathcal{O}(N, l) = 2^N$. That is, **all possible combinations in two groups are linearly separable**; verify it for the case of $N = 3$ in the two-dimensional space.
- Based on the previous theorem, given N points in the l -dimensional space, the probability of grouping these points in two **linearly separable** classes is,

$$P_N^l = \frac{\mathcal{O}(N, l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^l \binom{N-1}{i}, & N > l + 1, \\ 1 & N \leq l + 1. \end{cases}$$

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note that if $N \leq l + 1$ then $\mathcal{O}(N, l) = 2^N$. That is, **all possible combinations in two groups are linearly separable**; verify it for the case of $N = 3$ in the two-dimensional space.
- Based on the previous theorem, given N points in the l -dimensional space, the probability of grouping these points in two **linearly separable** classes is,

$$P_N^l = \frac{\mathcal{O}(N, l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^l \binom{N-1}{i}, & N > l + 1, \\ 1 & N \leq l + 1. \end{cases}$$



Set $r: N = r(l + 1)$. The figure shows the probability as a function of r . For $N > 2(l + 1)$ the probability of linear separability becomes small. For large values of l , and provided $N < 2(l + 1)$, the probability of any grouping of the data into two classes to be linearly separable tends to unity. Also, if $N \leq (l + 1)$, all possible groupings in two classes are linearly separable.

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- The way the previous theorem is exploited in practice is the following: Given N feature vectors $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, a mapping

$$\phi : \mathbb{R}^l \ni \mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{R}^K, \quad K \gg l,$$

is performed. Then according to the theorem, the higher the value of K is the higher the probability becomes for the images of the mapping, $\phi(\mathbf{x}_n) \in \mathbb{R}^K$, $n = 1, 2, \dots, N$, to be **linearly separable** in the space \mathbb{R}^K .

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note that the expansion of a **nonlinear** classifier (that predicts the label in a **binary** classification task) is equivalent with a **linear** task after such a mapping. Indeed,

$$\hat{y} = \text{sgn}(f(\mathbf{x})), \quad f(\mathbf{x}) = \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}) + \theta_0 = \boldsymbol{\theta}^T \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}) \\ 1 \end{bmatrix}, \quad (2)$$

where $\text{sgn}(\cdot)$ is the sign function and

$$\boldsymbol{\phi}(\mathbf{x}) := [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_K(\mathbf{x})]^T.$$

- Provided that K is large enough, our task is linearly separable in the **new space**, \mathbb{R}^K , with **high probability**, which justifies the use of a linear classifier, $\boldsymbol{\theta}$, in (2).

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note that the expansion of a **nonlinear** classifier (that predicts the label in a **binary** classification task) is equivalent with a **linear** task after such a mapping. Indeed,

$$\hat{y} = \text{sgn}(f(\mathbf{x})), \quad f(\mathbf{x}) = \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}) + \theta_0 = \boldsymbol{\theta}^T \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}) \\ 1 \end{bmatrix}, \quad (2)$$

where $\text{sgn}(\cdot)$ is the sign function and

$$\boldsymbol{\phi}(\mathbf{x}) := [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_K(\mathbf{x})]^T.$$

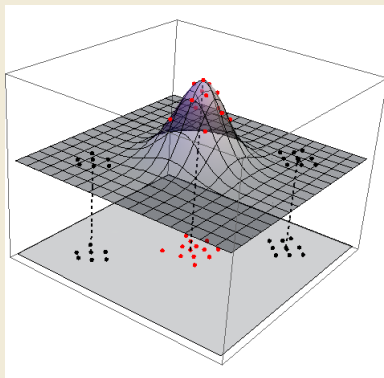
- Provided that **K is large enough**, our task is **linearly separable in the new space**, \mathbb{R}^K , with **high probability**, which justifies the use of a linear classifier, $\boldsymbol{\theta}$, in (2).

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- The previous procedure is illustrated in the figure below. The points in the two-dimensional space are **not** linearly separable. However, after the mapping in the three-dimensional space,

$$[x_1, x_2]^T \mapsto \phi(\mathbf{x}) = [x_1, x_2, f(x_1, x_2)]^T, \quad f(x_1, x_2) = 4 \exp(-(x_1^2 + x_2^2)/) + 5,$$

the points in the two classes become **linearly** separable.



Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note, however, that after the mapping, **the points lie on the surface of a paraboloid**. This surface is fully described in terms of **two free variables**. Loosely speaking, we can think of the two-dimensional plane, on which the data lie originally, to be **folded/transformed to form the surface of the paraboloid**.
- This is basically the idea behind the more general problem. After the mapping from the original l -dimensional space to the new K -dimensional one, the images of the points $\phi(x_n)$, $n = 1, 2, \dots, N$, **lie on an l -dimensional surface (manifold) in \mathbb{R}^K** .

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Note, however, that after the mapping, **the points lie on the surface of a paraboloid**. This surface is fully described in terms of **two free variables**. Loosely speaking, we can think of the two-dimensional plane, on which the data lie originally, to be **folded/transformed to form the surface of the paraboloid**.
- This is basically the idea behind the more general problem. After the mapping from the original l -dimensional space to the new K -dimensional one, the images of the points $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$, **lie on an l -dimensional surface (manifold) in \mathbb{R}^K** .

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Observe that we cannot fool nature. Since l variables were originally chosen to describe each pattern (dimensionality, number of free parameters) the **same number** of free parameters will be required to describe the same objects after the mapping in \mathbb{R}^K .
- In other words, after the mapping, we **embed an l -dimensional manifold in a K -dimensional space**, in such a way, so that the data in the two classes to **become linearly separable**.

Cover's Theorem: Capacity of a Space in Linear Dichotomies

- Observe that we cannot fool nature. Since l variables were originally chosen to describe each pattern (dimensionality, number of free parameters) the **same number** of free parameters will be required to describe the same objects after the mapping in \mathbb{R}^K .
- In other words, after the mapping, **we embed an l -dimensional manifold in a K -dimensional space**, in such a way, so that the data in the two classes to **become linearly separable**.

Reproducing Kernel Hilbert Spaces

- Consider a linear space \mathbb{H} , of real valued functions defined on a set \mathcal{X} . Furthermore, suppose that \mathbb{H} is a **Hilbert space**; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle$, that defines a corresponding norm $\| \cdot \|$.
- **Definition of RKHS:** A Hilbert space \mathbb{H} is called **Reproducing Kernel Hilbert Space**, if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

with the following properties:

- For every $x \in \mathcal{X}$, $\kappa(\cdot, x)$ belongs to \mathbb{H} .
- $\kappa(\cdot, \cdot)$ has the so called **reproducing property**, i.e.,

$$f(x) = \langle f, \kappa(\cdot, x) \rangle, \quad \forall f \in \mathbb{H}, \quad \forall x \in \mathcal{X}. \quad (3)$$

Reproducing Kernel Hilbert Spaces

- Consider a linear space \mathbb{H} , of real valued functions defined on a set \mathcal{X} . Furthermore, suppose that \mathbb{H} is a **Hilbert space**; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle$, that defines a corresponding norm $\| \cdot \|$.
- Definition of RKHS:** A Hilbert space \mathbb{H} is called **Reproducing Kernel Hilbert Space**, if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

with the following properties:

- For every $x \in \mathcal{X}$, $\kappa(\cdot, x)$ belongs to \mathbb{H} .
- $\kappa(\cdot, \cdot)$ has the so called **reproducing property**, i.e.,

$$f(x) = \langle f, \kappa(\cdot, x) \rangle, \quad \forall f \in \mathbb{H}, \quad \forall x \in \mathcal{X}. \quad (3)$$

Reproducing Kernel Hilbert Spaces

- Consider a linear space \mathbb{H} , of real valued functions defined on a set \mathcal{X} . Furthermore, suppose that \mathbb{H} is a **Hilbert space**; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle$, that defines a corresponding norm $\| \cdot \|$.
- Definition of RKHS:** A Hilbert space \mathbb{H} is called **Reproducing Kernel Hilbert Space**, if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

with the following properties:

- For every $x \in \mathcal{X}$, $\kappa(\cdot, x)$ belongs to \mathbb{H} .
- $\kappa(\cdot, \cdot)$ has the so called **reproducing property**, i.e.,

$$f(x) = \langle f, \kappa(\cdot, x) \rangle, \quad \forall f \in \mathbb{H}, \quad \forall x \in \mathcal{X}. \quad (3)$$

Reproducing Kernel Hilbert Spaces

- A direct consequence of the **reproducing property**, if we set $f(\cdot) = \kappa(\cdot, \mathbf{y})$, $\mathbf{y} \in \mathcal{X}$, is that

$$\langle \kappa(\cdot, \mathbf{y}), \kappa(\cdot, \mathbf{x}) \rangle = \kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{y}, \mathbf{x}).$$

- The RKHS, \mathbb{H} , can be **fully generated**, given $\kappa(\cdot, \cdot)$ and \mathcal{X} . Moreover, \mathbb{H} uniquely specifies $\kappa(\cdot, \cdot)$.

Reproducing Kernel Hilbert Spaces

- A direct consequence of the **reproducing property**, if we set $f(\cdot) = \kappa(\cdot, \mathbf{y})$, $\mathbf{y} \in \mathcal{X}$, is that

$$\langle \kappa(\cdot, \mathbf{y}), \kappa(\cdot, \mathbf{x}) \rangle = \kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{y}, \mathbf{x}).$$

- The RKHS, \mathbb{H} , can be **fully generated**, given $\kappa(\cdot, \cdot)$ and \mathcal{X} . Moreover, \mathbb{H} uniquely specifies $\kappa(\cdot, \cdot)$.

Reproducing Kernel Hilbert Spaces

- **Feature Map:** Let \mathbb{H} be an RKHS, associated with a kernel function $\kappa(\cdot, \cdot)$, and \mathcal{X} a set of elements. Then, the mapping

$$\mathcal{X} \ni \mathbf{x} \longmapsto \phi(\mathbf{x}) := \kappa(\cdot, \mathbf{x}) \in \mathbb{H},$$

is known as **feature map** and the space, \mathbb{H} , the **feature space**.

- In other words, if \mathcal{X} is the set of our observation vectors, the feature mapping maps each vector to the RKHS \mathbb{H} . Note that, in general, \mathbb{H} can be of infinite dimension and its elements can be **functions**. That is, each training point is mapped to a function.

Reproducing Kernel Hilbert Spaces

- **Feature Map:** Let \mathbb{H} be an RKHS, associated with a kernel function $\kappa(\cdot, \cdot)$, and \mathcal{X} a set of elements. Then, the mapping

$$\mathcal{X} \ni \mathbf{x} \longmapsto \phi(\mathbf{x}) := \kappa(\cdot, \mathbf{x}) \in \mathbb{H},$$

is known as **feature map** and the space, \mathbb{H} , the **feature space**.

- In other words, if \mathcal{X} is the set of our observation vectors, the feature mapping maps each vector to the RKHS \mathbb{H} . Note that, in general, \mathbb{H} can be of infinite dimension and its elements can be **functions**. That is, each training point is mapped to a function.

Reproducing Kernel Hilbert Spaces

- In special cases, where \mathbb{H} becomes a (finite dimensional) Euclidean space, \mathbb{R}^K , the image is a **vector** $\phi(\mathbf{x}) \in \mathbb{R}^K$. From now on, the general infinite dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$.
- After mapping from the original space to a high dimensional **RKHS** one, we have gained something very important. Let $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. Then, the inner product of the respective mapping images is written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle \kappa(\cdot, \mathbf{x}), \kappa(\cdot, \mathbf{y}) \rangle = \kappa(\mathbf{x}, \mathbf{y}).$$

- In words, employing this type of mapping to our problem, we can **perform inner product operations in \mathbb{H}** , in a very efficient way; that is, via a **function evaluation in the original low dimensional space!** This property is also known as the **kernel trick**, and it facilitates significantly the computations.

Reproducing Kernel Hilbert Spaces

- In special cases, where \mathbb{H} becomes a (finite dimensional) Euclidean space, \mathbb{R}^K , the image is a **vector** $\phi(\mathbf{x}) \in \mathbb{R}^K$. From now on, the general infinite dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$.
- After mapping from the original space to a high dimensional **RKHS** one, we have gained something very important. Let $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. Then, the inner product of the respective mapping images is written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle \kappa(\cdot, \mathbf{x}), \kappa(\cdot, \mathbf{y}) \rangle = \kappa(\mathbf{x}, \mathbf{y}).$$

- In words, employing this type of mapping to our problem, we can perform inner product operations in \mathbb{H} , in a very efficient way; that is, via a function evaluation in the original **low dimensional space**! This property is also known as the **kernel trick**, and it facilitates significantly the computations.

Reproducing Kernel Hilbert Spaces

- In special cases, where \mathbb{H} becomes a (finite dimensional) Euclidean space, \mathbb{R}^K , the image is a **vector** $\phi(\mathbf{x}) \in \mathbb{R}^K$. From now on, the general infinite dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$.
- After mapping from the original space to a high dimensional **RKHS** one, we have gained something very important. Let $\mathbf{x}, \mathbf{y} \in \mathcal{X}$. Then, the inner product of the respective mapping images is written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle \kappa(\cdot, \mathbf{x}), \kappa(\cdot, \mathbf{y}) \rangle = \kappa(\mathbf{x}, \mathbf{y}).$$

- In words, employing this type of mapping to our problem, we can **perform inner product operations in \mathbb{H}** , in a very efficient way; that is, via a **function evaluation in the original low dimensional space!** This property is also known as the **kernel trick**, and it facilitates significantly the computations.

The Kernel Trick

- The way this property is exploited in practice involves the following steps:

- 1 Map (implicitly) the input training data to an RKHS,

$$\mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

- 2 Solve a **linear** estimation task in \mathbb{H} , **involving the images** $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.

- 3 **Cast the algorithm** that solves for the unknown parameters **in terms of inner product** operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

- 4 **Replace each inner product by a kernel evaluation**, i.e.,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

The Kernel Trick

- The way this property is exploited in practice involves the following steps:

- 1 Map (implicitly) the input training data to an RKHS,

$$\mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

- 2 Solve a linear estimation task in \mathbb{H} , involving the images $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.

- 3 Cast the algorithm that solves for the unknown parameters in terms of inner product operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

- 4 Replace each inner product by a kernel evaluation, i.e.,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

The Kernel Trick

- The way this property is exploited in practice involves the following steps:

- 1 Map (implicitly) the input training data to an RKHS,

$$\mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

- 2 Solve a **linear** estimation task in \mathbb{H} , **involving the images** $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.

- 3 **Cast the algorithm** that solves for the unknown parameters **in terms of inner product** operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

- 4 **Replace each inner product by a kernel evaluation**, i.e.,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

The Kernel Trick

- The way this property is exploited in practice involves the following steps:

- ➊ Map (implicitly) the input training data to an RKHS,

$$\mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

- ➋ Solve a **linear** estimation task in \mathbb{H} , **involving the images** $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.

- ➌ **Cast the algorithm** that solves for the unknown parameters **in terms of inner product** operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

- ➍ Replace each inner product by a kernel evaluation, i.e.,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

The Kernel Trick

- The way this property is exploited in practice involves the following steps:

- ➊ Map (implicitly) the input training data to an RKHS,

$$\mathbf{x}_n \mapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

- ➋ Solve a **linear** estimation task in \mathbb{H} , **involving the images** $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.

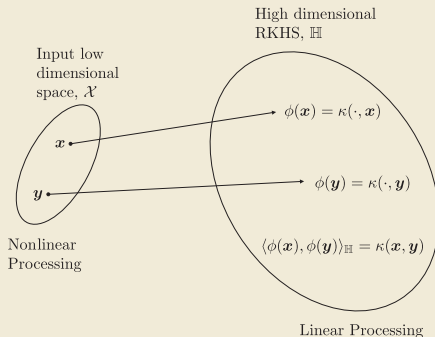
- ➌ **Cast the algorithm** that solves for the unknown parameters **in terms of inner product** operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

- ➍ **Replace each inner product by a kernel evaluation**, i.e.,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

The Kernel Trick



The nonlinear task in the original low dimensional space is mapped to a linear one in the high dimensional RKHS \mathbb{H} . Using the feature mapping, inner product operations are efficiently performed via kernel evaluations in the original low dimensional spaces.

The Kernel Trick

- One **does not need** to perform any **explicit mapping** of the data. All is needed is to perform the **kernel operations at the final step**. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$.
- As we will see, **different choices for $\kappa(\cdot, \cdot)$** , correspond to **different types of nonlinearity**.
- In practice, the four steps referred before are equivalent with:
 - Work in the **original space** and express all operations in terms of **inner products**.
 - At the final step **substitute the inner products with kernel evaluations**.

The Kernel Trick

- One **does not need** to perform any **explicit mapping** of the data. All is needed is to perform the **kernel operations at the final step**. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$.
- As we will see, **different choices for $\kappa(\cdot, \cdot)$** , correspond to **different types of nonlinearity**.
- In practice, the four steps referred before are equivalent with:
 - Work in the **original space** and express all operations in terms of **inner products**.
 - At the final step **substitute the inner products with kernel evaluations**.

The Kernel Trick

- One **does not need** to perform any **explicit mapping** of the data. All is needed is to perform the **kernel operations at the final step**. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$.
- As we will see, **different choices for $\kappa(\cdot, \cdot)$** , correspond to **different types of nonlinearity**.
- In practice, the four steps referred before are equivalent with:
 - Work in the **original space** and express all operations in terms of **inner products**.
 - At the final step **substitute the inner products with kernel evaluations**.

The Kernel Trick

- One **does not need** to perform any **explicit mapping** of the data. All is needed is to perform the **kernel operations at the final step**. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$.
- As we will see, **different choices for $\kappa(\cdot, \cdot)$** , correspond to **different types of nonlinearity**.
- In practice, the four steps referred before are equivalent with:
 - Work in the **original space** and express all operations in terms of **inner products**.
 - At the final step **substitute the inner products with kernel evaluations**.

The Kernel Trick: An Example

- Consider the case of the two-dimensional space and the mapping,

$$\mathbb{R}^2 \ni \mathbf{x} \mapsto \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3.$$

Then, given two vectors \mathbf{x} and \mathbf{y} , it is straightforward to see that,

$$\phi^T(\mathbf{x})\phi(\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

That is, the inner product in the new space is given in terms of a function of the variables in the original space,

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

The Kernel Trick: An Example

- Consider the case of the two-dimensional space and the mapping,

$$\mathbb{R}^2 \ni \mathbf{x} \mapsto \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3.$$

Then, given two vectors \mathbf{x} and \mathbf{y} , it is straightforward to see that,

$$\phi^T(\mathbf{x})\phi(\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

That is, the inner product in the new space is given in terms of a function of the variables in the original space,

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

- We present some typical examples of kernel functions, which are commonly used in various applications.
 - The **Gaussian kernel** is among the most popular ones and it is given by our familiar form

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp \left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2} \right),$$

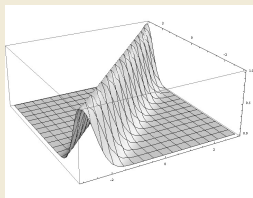
with $\sigma > 0$ being a parameter. The dimension of the corresponding RKHS is **infinite**.

Examples of Kernel Functions

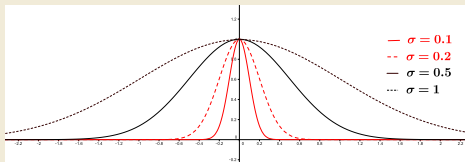
- We present some typical examples of kernel functions, which are commonly used in various applications.
 - The **Gaussian kernel** is among the most popular ones and it is given by our familiar form

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right),$$

with $\sigma > 0$ being a parameter. The dimension of the corresponding RKHS is **infinite**.



The Gaussian kernel for $\mathcal{X} = \mathbb{R}$, $\sigma = 0.5$.



The element $\phi(0) = \kappa(\cdot, 0)$ for different values of σ .

Examples of Kernel Functions

- Another popular example of a kernel function is the following:
 - The **inhomogeneous** polynomial kernel is given by

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^r,$$

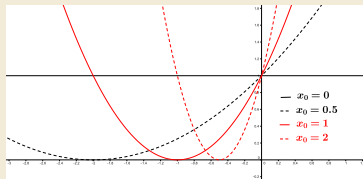
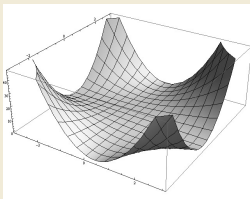
where $c \geq 0$ and $r > 0$, $r \in \mathbb{N}$ parameters. The dimensionality of the RKHS associated with polynomial kernels is **finite**.

Examples of Kernel Functions

- Another popular example of a kernel function is the following:
 - The **inhomogeneous** polynomial kernel is given by

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^r,$$

where $c \geq 0$ and $r > 0$, $r \in \mathbb{N}$ parameters. The dimensionality of the RKHS associated with polynomial kernels is **finite**.



The inhomogeneous polynomial kernel for $\mathcal{X} = \mathbb{R}$, $r = 2$.

The element $\phi(x) = \kappa(\cdot, x_0)$, for different values of x_0 .

Examples of Kernel Functions: String Kernels

- So far, our discussion has been focussed on input data, which were vectors in a Euclidean space. However, the input data need not to be necessarily vectors, and they can be **elements of more general sets**.
- **String kernels**: Let us denote by \mathcal{S} an alphabet set; that is, a set with a finite number of elements, which we call **symbols**. For example, this can be the set of all capital letters in the Latin alphabet. A **string** is a finite sequence, of any length, of symbols from \mathcal{S} . For example, two examples of strings are:

$T_1 = \text{"MYNAMEISSERGIOS"} , T_2 = \text{"HERNAMEISDESPOINA"} .$

- In a number of applications, such as in text mining, spam filtering, text summarization, bioinformatics, it is important to see how **"similar"** two strings are.

Examples of Kernel Functions: String Kernels

- So far, our discussion has been focussed on input data, which were vectors in a Euclidean space. However, the input data need not to be necessarily vectors, and they can be **elements of more general sets**.
- **String kernels**: Let us denote by \mathcal{S} an alphabet set; that is, a set with a finite number of elements, which we call **symbols**. For example, this can be the set of all capital letters in the Latin alphabet. A **string** is a finite sequence, of any length, of symbols from \mathcal{S} . For example, two examples of strings are:

$T_1 = \text{"MYNAMEISSERGIOS"} , T_2 = \text{"HERNAMEISDESPOINA"} .$

- In a number of applications, such as in text mining, spam filtering, text summarization, bioinformatics, it is important to see how **"similar"** two strings are.

Examples of Kernel Functions: String Kernels

- So far, our discussion has been focussed on input data, which were vectors in a Euclidean space. However, the input data need not to be necessarily vectors, and they can be **elements of more general sets**.
- **String kernels**: Let us denote by \mathcal{S} an alphabet set; that is, a set with a finite number of elements, which we call **symbols**. For example, this can be the set of all capital letters in the Latin alphabet. A **string** is a finite sequence, of any length, of symbols from \mathcal{S} . For example, two examples of strings are:

$T_1 = \text{"MYNAMEISSERGIOS"} , T_2 = \text{"HERNAMEISDESPOINA"} .$

- In a number of applications, such as in text mining, spam filtering, text summarization, bioinformatics, it is important to see how **"similar"** two strings are.

Examples of Kernel Functions: String Kernels

- However, kernels, by their definition are **similarity measures**; they are constructed so that to express inner products in the high dimensional feature space. An inner product is a similarity measure.
- Two vectors are most similar if they point to the same direction. Starting from this observation, there has been a lot of activity on defining kernels that measure **similarity** between strings.

Examples of Kernel Functions: String Kernels

- However, kernels, by their definition are **similarity measures**; they are constructed so that to express inner products in the high dimensional feature space. An inner product is a similarity measure.
- Two vectors are most similar if they point to the same direction. Starting from this observation, there has been a lot of activity on defining kernels that measure **similarity** between strings.

Examples of Kernel Functions: String Kernels

- However, kernels, by their definition are **similarity measures**; they are constructed so that to express inner products in the high dimensional feature space. An inner product is a similarity measure.
- **Two vectors are most similar if they point to the same direction.** Starting from this observation, there has been a lot of activity on defining kernels that measure **similarity** between strings.

- Let us denote by \mathcal{S}^* the set of all possible strings that can be constructed using symbols from \mathcal{S} . Also, a string, s , is said to be a **substring** of x if $x = bsa$, where a and b are other strings (possibly empty) from the symbols of \mathcal{S} . Given two strings $x, y \in \mathcal{S}^*$, define,

$$\kappa(x, y) := \sum_{s \in \mathcal{S}^*} w_s \phi_s(x) \phi_s(y),$$

where, $w_s \geq 0$, and $\phi_s(x)$ is the number of times substring s appears in x . It turns out that this is indeed a kernel, in the sense that it complies with the properties of a kernel definition; such kernels constructed from strings are known as **string kernels**.

- Obviously, a number of different variants of this kernel are available. The so-called **k -spectrum** kernel considers common substrings only of length k . For example, for the two strings given before in the previous slide, the value of the 6-spectrum string kernel, as defined before, is equal to one (one common substring of length 6 is identified: "NAMEIS").

- Let us denote by \mathcal{S}^* the set of all possible strings that can be constructed using symbols from \mathcal{S} . Also, a string, s , is said to be a **substring** of x if $x = bsa$, where a and b are other strings (possibly empty) from the symbols of \mathcal{S} . Given two strings $x, y \in \mathcal{S}^*$, define,

$$\kappa(x, y) := \sum_{s \in \mathcal{S}^*} w_s \phi_s(x) \phi_s(y),$$

where, $w_s \geq 0$, and $\phi_s(x)$ is the number of times substring s appears in x . It turns out that this is indeed a kernel, in the sense that it complies with the properties of a kernel definition; such kernels constructed from strings are known as **string kernels**.

- Obviously, a number of different variants of this kernel are available. The so-called **k -spectrum** kernel considers common substrings only of length k . For example, for the two strings given before in the previous slide, the value of the 6-spectrum string kernel, as defined before, is equal to one (one common substring of length 6 is identified: "NAMEIS").

Representer Theorem

- **Representer Theorem:** This theorem is of major importance from a practical point of view. It allows to perform **empirical** function optimization, based on a **finite set of training points**, in a very efficient way **even if** the function to be estimated belongs to a **very high (even infinite) dimensional RKHS, \mathbb{H}** .

- Let

$$\Omega : [0, +\infty) \mapsto \mathbb{R},$$

be an arbitrary **strictly monotonic** increasing function.

- Let also

$$\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function.

Representer Theorem

- **Representer Theorem:** This theorem is of major importance from a practical point of view. It allows to perform **empirical** function optimization, based on a **finite set of training points**, in a very efficient way **even if** the function to be estimated belongs to a **very high (even infinite) dimensional RKHS, \mathbb{H}** .

- Let

$$\Omega : [0, +\infty) \mapsto \mathbb{R},$$

be an arbitrary **strictly monotonic** increasing function.

- Let also

$$\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function.

Representer Theorem

- **Representer Theorem:** This theorem is of major importance from a practical point of view. It allows to perform **empirical** function optimization, based on a **finite set of training points**, in a very efficient way **even if** the function to be estimated belongs to a **very high (even infinite) dimensional RKHS, \mathbb{H}** .

- Let

$$\Omega : [0, +\infty) \mapsto \mathbb{R},$$

be an arbitrary **strictly monotonic** increasing function.

- Let also

$$\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function.

Representer Theorem

- Then each minimizer, $f \in \mathbb{H}$, of the **regularized minimization** task,

$$\min_{f \in \mathbb{H}} J(f) := \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \Omega(\|f\|^2)$$

admits an elegant representation of the form,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \quad (4)$$

where $\theta_n \in \mathbb{R}$, $n = 1, 2, \dots, N$.

Representer Theorem

- Then each minimizer, $f \in \mathbb{H}$, of the **regularized minimization** task,

$$\min_{f \in \mathbb{H}} J(f) := \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \Omega(\|f\|^2)$$

admits an elegant representation of the form,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \quad (4)$$

where $\theta_n \in \mathbb{R}$, $n = 1, 2, \dots, N$.

Representer Theorem

- **Proof of the theorem.** The linear span, $A := \text{span}\{\kappa(\cdot, \mathbf{x}_1), \dots, \kappa(\cdot, \mathbf{x}_N)\}$, forms a closed subspace. Then, each $f \in \mathbb{H}$ can be decomposed into two parts, i.e.,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + f_{\perp},$$

where f_{\perp} is the part of f which is orthogonal to A . From the reproducing property, we obtain

$$f(\mathbf{x}_m) = \langle f, \kappa(\cdot, \mathbf{x}_m) \rangle = \left\langle \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \kappa(\cdot, \mathbf{x}_m) \right\rangle = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}_m, \mathbf{x}_n),$$

where we used the fact that $\langle f_{\perp}, \kappa(\cdot, \mathbf{x}_n) \rangle = 0$, $n = 1, 2, \dots, N$.

- In other words, the expansion guarantees that at the **training points**, the value of $f(\cdot)$ does not depend on f_{\perp} . Hence, the first term in the associated cost function, corresponding to the empirical loss, **does not** depend on f_{\perp} .

Moreover, for all f_{\perp} we have,

$$\Omega(\|f\|^2) = \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2 + \|f_{\perp}\|^2\right) \geq \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2\right).$$

Thus, for **any** choice of θ_n , $n = 1, 2, \dots, N$, the cost function is minimized for $f_{\perp} = 0$. Thus, the claim is proved.

Representer Theorem

- **Proof of the theorem.** The linear span, $A := \text{span}\{\kappa(\cdot, \mathbf{x}_1), \dots, \kappa(\cdot, \mathbf{x}_N)\}$, forms a closed subspace. Then, each $f \in \mathbb{H}$ can be decomposed into two parts, i.e.,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + f_{\perp},$$

where f_{\perp} is the part of f which is orthogonal to A . From the reproducing property, we obtain

$$f(\mathbf{x}_m) = \langle f, \kappa(\cdot, \mathbf{x}_m) \rangle = \left\langle \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \kappa(\cdot, \mathbf{x}_m) \right\rangle = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}_m, \mathbf{x}_n),$$

where we used the fact that $\langle f_{\perp}, \kappa(\cdot, \mathbf{x}_n) \rangle = 0$, $n = 1, 2, \dots, N$.

- In other words, the expansion guarantees that at the **training points**, the value of $f(\cdot)$ does not depend on f_{\perp} . Hence, the first term in the associated cost function, corresponding to the empirical loss, **does not** depend on f_{\perp} .

Moreover, for all f_{\perp} we have,

$$\Omega(\|f\|^2) = \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2 + \|f_{\perp}\|^2\right) \geq \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2\right).$$

Thus, for **any** choice of θ_n , $n = 1, 2, \dots, N$, the cost function is minimized for $f_{\perp} = 0$. Thus, the claim is proved.

- **Proof of the theorem.** The linear span, $A := \text{span}\{\kappa(\cdot, \mathbf{x}_1), \dots, \kappa(\cdot, \mathbf{x}_N)\}$, forms a closed subspace. Then, each $f \in \mathbb{H}$ can be decomposed into two parts, i.e.,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + f_{\perp},$$

where f_{\perp} is the part of f which is orthogonal to A . From the reproducing property, we obtain

$$f(\mathbf{x}_m) = \langle f, \kappa(\cdot, \mathbf{x}_m) \rangle = \left\langle \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \kappa(\cdot, \mathbf{x}_m) \right\rangle = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}_m, \mathbf{x}_n),$$

where we used the fact that $\langle f_{\perp}, \kappa(\cdot, \mathbf{x}_n) \rangle = 0$, $n = 1, 2, \dots, N$.

- In other words, the expansion guarantees that at the **training points**, the value of $f(\cdot)$ does not depend on f_{\perp} . Hence, the first term in the associated cost function, corresponding to the empirical loss, **does not** depend on f_{\perp} .

Moreover, for all f_{\perp} we have,

$$\Omega(\|f\|^2) = \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2 + \|f_{\perp}\|^2\right) \geq \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2\right).$$

Thus, for **any** choice of θ_n , $n = 1, 2, \dots, N$, the cost function is minimized for $f_{\perp} = 0$. Thus, the claim is proved.

Representer Theorem

- The importance of this theorem is that in order to optimize the cost function with respect to f , one uses the expansion in (4) and minimization is carried out with respect to the **finite set of parameters**, θ_n , $n = 1, 2, \dots, N$.
- Note that when working in high (even infinite) dimensional spaces, the presence of a **regularizer** can hardly be avoided; otherwise, the obtained solution will **suffer from overfitting**, since only a finite number of data samples are used for training.

Representer Theorem

- The importance of this theorem is that in order to optimize the cost function with respect to f , one uses the expansion in (4) and minimization is carried out with respect to the **finite set of parameters**, θ_n , $n = 1, 2, \dots, N$.
- Note that when working in high (even infinite) dimensional spaces, the presence of a **regularizer** can hardly be avoided; otherwise, the obtained solution will **suffer from overfitting**, since only a finite number of data samples are used for training.

Representer Theorem

- Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation,

$$\tilde{f} = f + b, \quad f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n).$$

In practice, the use of a bias term (which does not enter in the regularization) turns out to improve performance.

- First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization that is imposed by the regularizing term, $\Omega(\|f\|^2)$, the minimizer pushes the values, which the function takes at the training points, to smaller values. The existence of b tries to “absorb” some of this action.

Representer Theorem

- Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation,

$$\tilde{f} = f + b, \quad f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n).$$

In practice, the use of a bias term (**which does not enter in the regularization**) turns out to improve performance.

- First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization that is imposed by the regularizing term, $\Omega(\|f\|^2)$, the minimizer pushes the values, which the function takes at the training points, to smaller values. The existence of b tries to “absorb” some of this action.

Representer Theorem

- Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation,

$$\tilde{f} = f + b, \quad f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n).$$

In practice, the use of a bias term (**which does not enter in the regularization**) turns out to improve performance.

- First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization that is imposed by the regularizing term, $\Omega(\|f\|^2)$, the minimizer pushes the values, which the function takes at the training points, to smaller values. The existence of b tries to “absorb” some of this action.

Kernel Ridge Regression

- Ridge regression has been discussed in Chapters 3 and 7. Here, we will state the task in a general RKH space. The path to be followed is a **typical one used to extend techniques, which have been developed for linear models, to the more general RKH spaces.**
- We assume that the generation mechanism of the data, represented by the training set $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, is modelled via a **nonlinear regression** task,

$$y_n = g(\mathbf{x}_n) + \eta_n, \quad n = 1, 2, \dots, N.$$

Kernel Ridge Regression

- Ridge regression has been discussed in Chapters 3 and 7. Here, we will state the task in a general RKH space. The path to be followed is a **typical one used to extend techniques, which have been developed for linear models, to the more general RKH spaces.**
- We assume that the generation mechanism of the data, represented by the training set $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, is modelled via a **nonlinear regression** task,

$$y_n = g(\mathbf{x}_n) + \eta_n, \quad n = 1, 2, \dots, N.$$

Kernel Ridge Regression

- Let us denote by f the **estimate** of the unknown g . Sometimes, f is called the **hypothesis** and the space \mathbb{H} , in which f is searched, is known as the **hypothesis space**. We will further assume that f lies in an RKHS, associated with a kernel,

$$\kappa : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

- The adopted **regularized** cost function to be minimized is chosen to be:

$$f = \arg \min_{\tilde{f} \in \mathbb{H}} J(\tilde{f}) = \sum_{n=1}^N (y_n - \tilde{f}(x_n))^2 + C \|\tilde{f}\|^2.$$

Kernel Ridge Regression

- Let us denote by f the **estimate** of the unknown g . Sometimes, f is called the **hypothesis** and the space \mathbb{H} , in which f is searched, is known as the **hypothesis space**. We will further assume that f lies in an RKHS, associated with a kernel,

$$\kappa : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

- The adopted **regularized** cost function to be minimized is chosen to be:

$$f = \arg \min_{\tilde{f} \in \mathbb{H}} J(\tilde{f}) = \sum_{n=1}^N (y_n - \tilde{f}(\mathbf{x}_n))^2 + C \|\tilde{f}\|^2.$$

Kernel Ridge Regression

- **The solution:** The obtained kernel ridge regression **predictor** turns out to be,

$$\hat{y}(\mathbf{x}) = \mathbf{y}^T (\mathcal{K} + CI)^{-1} \boldsymbol{\kappa}(\mathbf{x}).$$

where

$$\boldsymbol{\kappa}(\mathbf{x}) := [\kappa(\mathbf{x}, \mathbf{x}_1), \dots, \kappa(\mathbf{x}, \mathbf{x}_N)]^T.$$

and \mathcal{K} is the corresponding **kernel matrix** defined as,

$$\mathcal{K} := \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \quad (5)$$

Kernel Ridge Regression

- **Proof of the result:** Motivated by the representer theorem, we adopt the following expansion

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n).$$

- Constraining \tilde{f} in the cost to be of the above form, the unknown coefficients are estimated by the following task

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{n=1}^N \left(y_n - \sum_{m=1}^N \theta_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \right)^2 + C \langle f, f \rangle,$$

- The last cost function can be rewritten as,

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathcal{K}\boldsymbol{\theta})^T (\mathbf{y} - \mathcal{K}\boldsymbol{\theta}) + C \boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta},$$

where

$$\mathbf{y} = [y_1, \dots, y_N]^T, \quad \boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T,$$

and \mathcal{K} is the kernel matrix defined before; the latter is completely specified by the set of training points and the kernel function that defines the corresponding RKHS, \mathbb{H} .

Kernel Ridge Regression

- **Proof of the result:** Motivated by the representer theorem, we adopt the following expansion

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n).$$

- Constraining \tilde{f} in the cost to be of the above form, the unknown coefficients are estimated by the following task

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{n=1}^N \left(y_n - \sum_{m=1}^N \theta_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \right)^2 + C \langle f, f \rangle,$$

- The last cost function can be rewritten as,

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathcal{K}\boldsymbol{\theta})^T (\mathbf{y} - \mathcal{K}\boldsymbol{\theta}) + C \boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta},$$

where

$$\mathbf{y} = [y_1, \dots, y_N]^T, \quad \boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T,$$

and \mathcal{K} is the kernel matrix defined before; the latter is completely specified by the set of training points and the kernel function that defines the corresponding RKHS, \mathbb{H} .

Kernel Ridge Regression

- **Proof of the result:** Motivated by the representer theorem, we adopt the following expansion

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n).$$

- Constraining \tilde{f} in the cost to be of the above form, the unknown coefficients are estimated by the following task

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{n=1}^N \left(y_n - \sum_{m=1}^N \theta_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \right)^2 + C \langle f, f \rangle,$$

- The last cost function can be rewritten as,

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathcal{K}\boldsymbol{\theta})^T (\mathbf{y} - \mathcal{K}\boldsymbol{\theta}) + C \boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta},$$

where

$$\mathbf{y} = [y_1, \dots, y_N]^T, \quad \boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T,$$

and \mathcal{K} is the kernel matrix defined before; the latter is completely specified by the **set of training points and the kernel function** that defines the corresponding RKHS, \mathbb{H} .

Kernel Ridge Regression

- Minimizing $J(\boldsymbol{\theta})$ w.r. to $\boldsymbol{\theta}$ leads to

$$(\mathcal{K}^T \mathcal{K} + C \mathcal{K}^T) \hat{\boldsymbol{\theta}} = \mathcal{K}^T \mathbf{y},$$

or

$$(\mathcal{K} + CI) \hat{\boldsymbol{\theta}} = \mathbf{y},$$

where $\mathcal{K}^T = \mathcal{K}$ has been assumed to be invertible.

- Once $\hat{\boldsymbol{\theta}}$ has been obtained, given an unknown vector, $\mathbf{x} \in \mathbb{R}^l$, the corresponding prediction value of the dependent variable is given by

$$\hat{y} = \sum_{n=1}^N \hat{\theta}_n \kappa(\mathbf{x}, \mathbf{x}_n) = \hat{\boldsymbol{\theta}}^T \boldsymbol{\kappa}(\mathbf{x}),$$

which readily leads to the previously stated result.

Kernel Ridge Regression

- Minimizing $J(\boldsymbol{\theta})$ w.r. to $\boldsymbol{\theta}$ leads to

$$(\mathcal{K}^T \mathcal{K} + C \mathcal{K}^T) \hat{\boldsymbol{\theta}} = \mathcal{K}^T \mathbf{y},$$

or

$$(\mathcal{K} + CI) \hat{\boldsymbol{\theta}} = \mathbf{y},$$

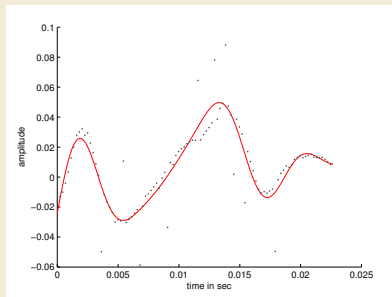
where $\mathcal{K}^T = \mathcal{K}$ has been assumed to be invertible.

- Once $\hat{\boldsymbol{\theta}}$ has been obtained, given an unknown vector, $\mathbf{x} \in \mathbb{R}^l$, the corresponding prediction value of the dependent variable is given by

$$\hat{y} = \sum_{n=1}^N \hat{\theta}_n \kappa(\mathbf{x}, \mathbf{x}_n) = \hat{\boldsymbol{\theta}}^T \boldsymbol{\kappa}(\mathbf{x}),$$

which readily leads to the previously stated result.

- In this example, the prediction power of the kernel ridge regression, in the **presence of Gaussian noise as well as of outliers**, will be tested. The original data were samples from a music recording from Blade Runner by Vangelis Papathanasiou. A white Gaussian noise was then added at a 15dB level and a number of outliers were intentionally randomly introduced and “hit” some of the values (10%). The kernel ridge regression method was used, employing the Gaussian kernel with $\sigma = 0.004$. A bias term was also present, as discussed before. The prediction (fitted) curve, $\hat{y}(x)$, for various value of x , is shown in the figure below, together with the (noisy) data used for training.



Loss Functions For Robust Learning

- **Learning in the presence of outliers:** The least squares cost function is not always the best criterion for optimization, in spite of its merits. In the case of the presence of a **non-Gaussian noise with long tails**, and hence with an increased number of noise **outliers**, the square dependence of the LS criterion **gets biased** towards values associated with the **presence of outliers**.
- The method of Least-Squares is equivalent to the maximum likelihood estimation under the assumption of **white Gaussian noise**. Under this assumption, the LS estimator achieves the Cramer-Rao bound and it becomes a **minimum variance estimator**. However, under other noise scenarios, one has to look for alternative criteria.

Loss Functions For Robust Learning

- **Learning in the presence of outliers:** The least squares cost function is not always the best criterion for optimization, in spite of its merits. In the case of the presence of a **non-Gaussian noise with long tails**, and hence with an increased number of noise **outliers**, the square dependence of the LS criterion **gets biased** towards values associated with the **presence of outliers**.
- The method of Least-Squares is equivalent to the maximum likelihood estimation under the assumption of **white Gaussian noise**. Under this assumption, the LS estimator achieves the **Cramer-Rao bound and it becomes a minimum variance estimator**. However, under other noise scenarios, one has to look for alternative criteria.

Loss Functions For Robust Learning

- The task of optimization in the **presence of outliers** was studied by Huber, whose goal was to obtain a strategy for choosing a loss function that **matches best to the noise model**. He proved that, under the assumption that the noise has a symmetric pdf, the optimal minimax strategy for regression is obtained via the following loss function,

$$\mathcal{L}(y, f(\mathbf{x})) = |y - f(\mathbf{x})|,$$

which gives rise to the **least modulus** method.

Loss Functions For Robust Learning

- **Huber loss function**: Huber also showed that if the noise comprises two components, one corresponding to a Gaussian and another to an arbitrary pdf (which remains symmetric), then the best in the **minimax** sense loss function is given by,

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} \epsilon|y - f(\mathbf{x})| - \frac{\epsilon^2}{2}, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ \frac{1}{2}|y - f(\mathbf{x})|^2, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases}$$

for some parameter ϵ . This is known as the **Huber loss** function.

Loss Functions For Robust Learning

- **Linear ϵ -insensitive loss function:** This function is an approximation to the Huber loss one and it is defined as,

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})| - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases} \quad (6)$$

Note that for $\epsilon = 0$, it coincides with the absolute value loss function, and it is close to the Huber loss for small values of $\epsilon < 1$.

Loss Functions For Robust Learning

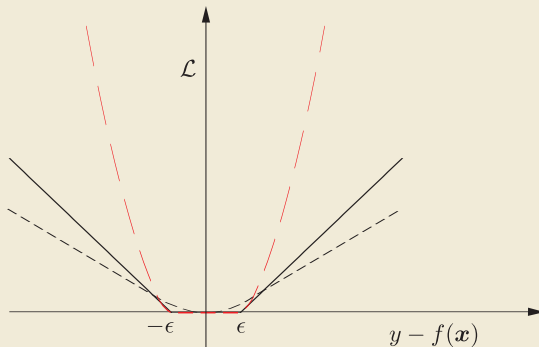
- Quadratic ϵ -insensitive loss function: This is defined by

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})|^2 - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases}$$

which coincides with the squared error loss for $\epsilon = 0$.

Loss Functions For Robust Learning

- The following figure shows all three previously defined loss functions.



The Huber loss function (dotted-gray), the linear ϵ -insensitive (full-gray) and the quadratic ϵ -insensitive (red) loss functions, for $\epsilon = 0.7$.

- **Linear regression and the linear ϵ -insensitive loss function:** For the linear regression, the model is

$$y_n = f(\mathbf{x}_n) + \eta_n = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 + \eta_n,$$

and the task is to estimate $(\theta_0, \boldsymbol{\theta})$ by minimizing the regularized cost

$$J(\theta_0, \boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}(y_n, \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0),$$

where the loss function is the linear ϵ -insensitive one. Note that C multiplies the error misfit term; this is equivalent with the regularizer being multiplied by $1/C$.

- **Introducing slack variables:** Note that the the linear ϵ -insensitive loss is **not differentiable**. Although we could optimize the non-smooth convex function directly (employing subgradients), we will follow another path. To this end, let us now introduce two sets of **auxiliary variables**.

- **Linear regression and the linear ϵ -insensitive loss function:** For the linear regression, the model is

$$y_n = f(\mathbf{x}_n) + \eta_n = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 + \eta_n,$$

and the task is to estimate $(\theta_0, \boldsymbol{\theta})$ by minimizing the regularized cost

$$J(\theta_0, \boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}(y_n, \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0),$$

where the loss function is the linear ϵ -insensitive one. Note that C multiplies the error misfit term; this is equivalent with the regularizer being multiplied by $1/C$.

- **Introducing slack variables:** Note that the the linear ϵ -insensitive loss is **not differentiable**. Although we could optimize the non-smooth convex function directly (employing subgradients), we will follow another path. To this end, let us now introduce two sets of **auxiliary variables**.

- We consider the following two cases:

- If

$$y_n - \theta^T x_n - \theta_0 \geq \epsilon,$$

define $\tilde{\xi}_n \geq 0$, such as

$$y_n - \theta^T x_n - \theta_0 \leq \epsilon + \tilde{\xi}_n.$$

Note that **ideally**, we would like to select θ, θ_0 , so that $\hat{\xi}_n = 0$, since this would make the **contribution of the respective term in the loss function equal to zero**.

- If

$$y_n - \theta^T x_n - \theta_0 \leq -\epsilon,$$

define $\xi_n \geq 0$, such as

$$\theta^T x_n + \theta_0 - y_n \leq \epsilon + \xi_n.$$

Once more, we would like to select our unknown set of parameters so that ξ_n to be zero.

- We consider the following two cases:

- If

$$y_n - \theta^T \mathbf{x}_n - \theta_0 \geq \epsilon,$$

define $\tilde{\xi}_n \geq 0$, such as

$$y_n - \theta^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n.$$

Note that **ideally**, we would like to select θ, θ_0 , so that $\hat{\xi}_n = 0$, since this would make the **contribution of the respective term in the loss function equal to zero**.

- If

$$y_n - \theta^T \mathbf{x}_n - \theta_0 \leq -\epsilon,$$

define $\xi_n \geq 0$, such as

$$\theta^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n.$$

Once more, we would like to select our unknown set of parameters so that ξ_n **to be zero**.

Support Vector Regression

- The auxiliary variables are also known as **slack variables**. We will now employ them to formulate our optimization task in a form that leads to a **closed form** solution.
- A careful look reveals that minimizing the empirical cost around the linear ϵ -insensitive loss function is equivalent to:

$$\begin{aligned} \text{minimize} \quad & J(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right), \\ \text{subject to} \quad & y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n, \quad n = 1, 2, \dots, N, \\ & \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n, \quad n = 1, 2, \dots, N, \\ & \tilde{\xi}_n \geq 0, \quad \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

Support Vector Regression

- The auxiliary variables are also known as **slack variables**. We will now employ them to formulate our optimization task in a form that leads to a **closed form** solution.
- A careful look reveals that minimizing the empirical cost around the linear ϵ -insensitive loss function is equivalent to:

$$\begin{aligned} \text{minimize} \quad & J(\boldsymbol{\theta}, \boldsymbol{\theta}_0, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right), \\ \text{subject to} \quad & y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n, \quad n = 1, 2, \dots, N, \\ & \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n, \quad n = 1, 2, \dots, N, \\ & \tilde{\xi}_n \geq 0, \quad \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

Support Vector Regression

- **The solution and the support vectors:** The solution of the optimization task is obtained by **introducing Lagrange multipliers and forming the corresponding Lagrangian.**
- Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\theta} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n,$$

where $\tilde{\lambda}_n, \lambda_n, n = 1, 2, \dots, N$, are the Lagrange multipliers associated with **each one of the constraints.**

- The Lagrange multipliers are **nonzero only** for those of the points, \mathbf{x}_n , which correspond to error values **either equal or larger than ϵ .** These are known as **support vectors.**

Support Vector Regression

- **The solution and the support vectors:** The solution of the optimization task is obtained by **introducing Lagrange multipliers and forming the corresponding Lagrangian**.
- Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n,$$

where $\tilde{\lambda}_n, \lambda_n, n = 1, 2, \dots, N$, are the Lagrange multipliers associated with **each one of the constraints**.

- The Lagrange multipliers are **nonzero only** for those of the points, \mathbf{x}_n , which correspond to error values either equal or larger than ϵ . These are known as **support vectors**.

Support Vector Regression

- **The solution and the support vectors:** The solution of the optimization task is obtained by **introducing Lagrange multipliers and forming the corresponding Lagrangian**.
- Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n,$$

where $\tilde{\lambda}_n, \lambda_n, n = 1, 2, \dots, N$, are the Lagrange multipliers associated with **each one of the constraints**.

- The Lagrange multipliers are **nonzero only** for those of the points, \mathbf{x}_n , which correspond to error values **either equal or larger than ϵ** . These are known as **support vectors**.

Support Vector Regression

- The bias term can be obtained by any one from the set of equations

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 = \epsilon,$$

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n = \epsilon,$$

where n above runs over the points which are associated with $\tilde{\lambda}_n > 0$ ($\lambda_n > 0$) and $\tilde{\xi}_n = 0$ ($\xi_n = 0$) (note that these points form a subset of the support vectors). In practice, $\hat{\theta}_0$ is obtained as the average from all the previous equations.

Support Vector Regression

- **Casting a nonlinear task in an RKHS:** In this case of nonlinear modeling, the path to follow is:

- Assume an (implicit) mapping to the RKHS associated with a kernel, $\kappa(\cdot, x)$, $x \in \mathbb{R}^l$, via the **feature map**

$$x \longmapsto \phi(x) = \kappa(\cdot, x),$$

- Approximate the nonlinear function, $g(x)$, in the regression task, as a **linear one in the respective RKHS**, i.e.,

$$g(x) \approx f(x) = \langle \theta, \phi(x) \rangle + \theta_0,$$

where θ is now **treated as a function** in the RKHS, \mathcal{H} .

Support Vector Regression

- **Casting a nonlinear task in an RKHS:** In this case of nonlinear modeling, the path to follow is:

- Assume an (implicit) mapping to the RKHS associated with a kernel, $\kappa(\cdot, \mathbf{x})$, $\mathbf{x} \in \mathbb{R}^l$, via the **feature map**

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

- Approximate the nonlinear function, $g(\mathbf{x})$, in the regression task, as a **linear one in the respective RKHS**, i.e.,

$$g(\mathbf{x}) \approx f(\mathbf{x}) = \langle \theta, \phi(\mathbf{x}) \rangle + \theta_0,$$

where θ is now **treated as a function** in the RKHS, \mathcal{H} .

Support Vector Regression

- **Casting a nonlinear task in an RKHS:** In this case of nonlinear modeling, the path to follow is:

- Assume an (implicit) mapping to the RKHS associated with a kernel, $\kappa(\cdot, \mathbf{x})$, $\mathbf{x} \in \mathbb{R}^l$, via the **feature map**

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

- Approximate the nonlinear function, $g(\mathbf{x})$, in the regression task, as a **linear one in the respective RKHS**, i.e.,

$$g(\mathbf{x}) \approx f(\mathbf{x}) = \langle \theta, \phi(\mathbf{x}) \rangle + \theta_0,$$

where θ is now **treated as a function in the RKHS**, \mathbb{H} .

- Since the task remains linear in \mathbb{H} , all we said before carries on to the new space and we obtain,

$$\hat{\theta}(\cdot) = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \mathbf{x}_n).$$

- The bias term $\hat{\theta}_0$ is obtained as discussed before. Once $\hat{\theta}$, $\hat{\theta}_0$ have been obtained, we are ready to perform prediction.
- Given a value \mathbf{x} , we first perform the (implicit) mapping using the feature map ($\mathbf{x} \mapsto \kappa(\cdot, \mathbf{x})$) and we get

$$\hat{y}(\mathbf{x}) = f(\mathbf{x}) = \langle \hat{\theta}, \kappa(\cdot, \mathbf{x}) \rangle + \hat{\theta}_0, \quad \text{or}$$

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0. \quad (7)$$

where $N_s \leq N$, is the number of **nonzero** Lagrange multipliers.

- Since the task remains linear in \mathbb{H} , all we said before carries on to the new space and we obtain,

$$\hat{\theta}(\cdot) = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \mathbf{x}_n).$$

- The bias term $\hat{\theta}_0$ is obtained as discussed before. Once $\hat{\theta}$, $\hat{\theta}_0$ have been obtained, we are ready to perform prediction.
- Given a value \mathbf{x} , we first perform the (implicit) mapping using the feature map ($\mathbf{x} \mapsto \kappa(\cdot, \mathbf{x})$) and we get

$$\hat{y}(\mathbf{x}) = f(\mathbf{x}) = \langle \hat{\theta}, \kappa(\cdot, \mathbf{x}) \rangle + \hat{\theta}_0, \quad \text{or}$$

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0. \quad (7)$$

where $N_s \leq N$, is the number of **nonzero** Lagrange multipliers.

- Since the task remains linear in \mathbb{H} , all we said before carries on to the new space and we obtain,

$$\hat{\theta}(\cdot) = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \mathbf{x}_n).$$

- The bias term $\hat{\theta}_0$ is obtained as discussed before. Once $\hat{\theta}$, $\hat{\theta}_0$ have been obtained, we are ready to perform prediction.
- Given a value \mathbf{x} , we first perform the (implicit) mapping using the feature map ($\mathbf{x} \mapsto \kappa(\cdot, \mathbf{x})$) and we get

$$\hat{y}(\mathbf{x}) = f(\mathbf{x}) = \langle \hat{\theta}, \kappa(\cdot, \mathbf{x}) \rangle + \hat{\theta}_0, \quad \text{or}$$

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0. \quad (7)$$

where $N_s \leq N$, is the number of **nonzero** Lagrange multipliers.

Support Vector Regression

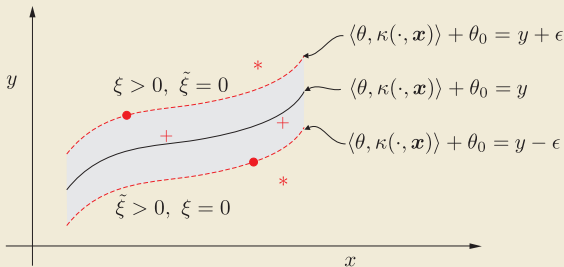
- Observe that the last equation, i.e.,

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0. \quad (8)$$

is an **expansion in terms of nonlinear (kernel) functions**.

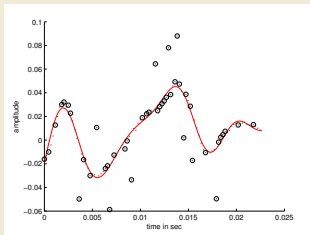
Moreover, since only a **fraction of the points** is involved (N_s), the use of the ϵ -insensitive loss function achieves a form of **sparsification** on the general expansion that would involve all the training points, as dictated by the representer theorem.

- The figure below illustrates $\hat{y}(x)$ for a choice of $\kappa(\cdot, \cdot)$. Observe that the values of ϵ form a “tube” around the respective graph. Points lying outside the tube correspond to values of the slack variables larger than zero. Because from all the training points, the solution **depends only on the support vectors**, the method is known as **support vector regression (SVR)**.

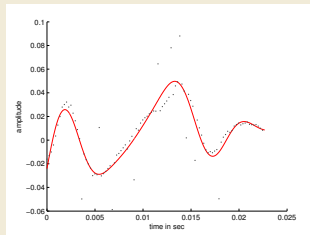


The tube around the nonlinear regression curve. Points outside the tube have either $\tilde{\xi} > 0$ and $\xi = 0$ or $\xi > 0$ and $\tilde{\xi} = 0$. The rest of the points have $\xi = \tilde{\xi} = 0$. Points which are inside the tube correspond to zero Lagrange multipliers.

- Consider the same time series used for the nonlinear prediction task, used in the kernel ridge regression example. This time, the SVR method was used and optimized around the linear ϵ -insensitive loss function, with $\epsilon = 0.003$. The same Gaussian kernel, with $\sigma = 0.004$, was employed, as in the kernel ridge regression case. Figure (a) below shows the resulting prediction curve, $\hat{y}(x)$, as a function of x given in (8). The curve fits the data samples much better compared to the kernel ridge regression (Figure (b)), exhibiting the **enhanced robustness of the SVR method, relative to the KKR**, in the presence of **outliers**.



(a)

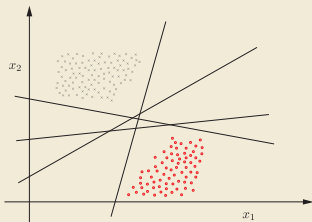


(b)

The improved performance compared to the kernel ridge regression used is readily observed, by simply observing the two figures. The encircled points are the support vectors resulting from the optimization, using the ϵ -insensitive loss function.

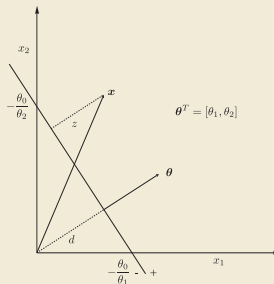
Maximum Margin Classifiers

- **Margin between two classes:** If the classes are **linearly separable**, there is an infinity of hyperplanes that classify correctly **all** the points, Figure (a). Each hyperplane is defined in terms of its **direction and its position** in the respective space, Figure (b).



(a)

There is an infinite number of linear classifiers which can classify correctly all the patterns, in a linearly separable class task.



(b)

The direction of the hyperplane, $\theta^T x + \theta_0 = 0$, is determined by θ and its position in space by θ_0 .

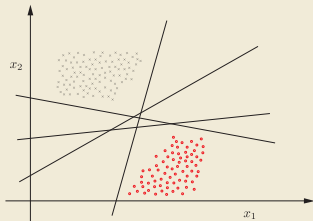
- The distance of a point x from a hyperplane, Figure (b), is given by,

$$z = \frac{|\theta^T x + \theta_0|}{\|\theta\|},$$

which is obviously zero if the point lies on the hyperplane.

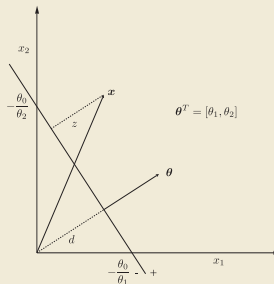
Maximum Margin Classifiers

- **Margin between two classes:** If the classes are **linearly separable**, there is an infinity of hyperplanes that classify correctly **all** the points, Figure (a). Each hyperplane is defined in terms of its **direction and its position** in the respective space, Figure (b).



(a)

There is an infinite number of linear classifiers which can classify correctly all the patterns, in a linearly separable class task.



(b)

The direction of the hyperplane, $\theta^T x + \theta_0 = 0$, is determined by θ and its position in space by θ_0 .

- The distance of a point x from a hyperplane, Figure (b), is given by,

$$z = \frac{|\theta^T x + \theta_0|}{\|\theta\|},$$

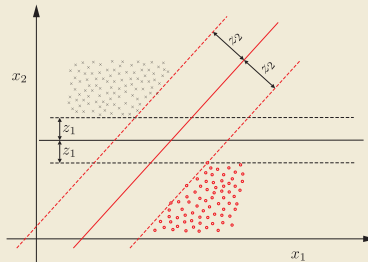
which is obviously **zero** if the point lies on the hyperplane.

Maximum Margin Classifiers

- From the set of all classifiers that solve the task exactly and have certain direction (i.e., they share a common θ), we select θ_0 so that to place the classifier in between the two classes, such that its distance from the **nearest points** from each one of the two classes to be same.

Maximum Margin Classifiers

- The previous statement is illustrated in the figure below that shows the linear classifiers (hyperplanes) in two different directions (**full lines** in gray and red). Both of them have been placed so that to have the **same distance** from the **nearest points** in both classes. Moreover note that, the distance z_1 associated with the “gray” classifier is smaller than the z_2 associated with the “red” one.



Maximum Margin Classifiers

- After an appropriate scaling, we can always make the **distance of the nearest points from the two classes** to the hyperplane to be **equal to $z = \frac{1}{\|\theta\|}$** ; equivalently, the scaling guarantees that $f(x) = \pm 1$ if x is a nearest to the hyperplane point and depending on whether the point belongs to ω_1 (+1) or ω_2 (-1).
- The two hyperplanes, defined by $f(x) = \pm 1$, are shown in the figure in the previous slide as **dotted lines**, for both the “gray” and the “red” directions. The **pair of these hyperplanes** defines the corresponding **margin**, for each direction, whose **width** is equal to $\frac{2}{\|\theta\|}$.

Maximum Margin Classifiers

- After an appropriate scaling, we can always make the **distance of the nearest points from the two classes** to the hyperplane to be **equal to $z = \frac{1}{\|\theta\|}$** ; equivalently, the scaling guarantees that $f(x) = \pm 1$ if x is a nearest to the hyperplane point and depending on whether the point belongs to ω_1 (+1) or ω_2 (-1).
- The two hyperplanes, defined by $f(x) = \pm 1$, are shown in the figure in the previous slide as **dotted lines**, for both the “gray” and the “red” directions. The **pair of these hyperplanes** defines the corresponding **margin**, for each direction, whose **width is equal to $\frac{2}{\|\theta\|}$** .

Maximum Margin Classifiers

- Thus, each classifier constructed as explained before and which solves the task, satisfies the following two properties:
 - It has a **margin** of width equal to $\frac{2}{\|\theta\|}$
 - In addition,

$$\begin{aligned}\theta^T \mathbf{x}_n + \theta_0 &\geq 1, & \mathbf{x}_n &\in \omega_1, \\ \theta^T \mathbf{x}_n + \theta_0 &\leq -1, & \mathbf{x}_n &\in \omega_2.\end{aligned}$$

- We are now ready to state the task that defines the maximum margin classifier, for linearly separable classes.

Maximum Margin Classifiers

- Thus, each classifier constructed as explained before and which solves the task, satisfies the following two properties:
 - It has a **margin** of width equal to $\frac{2}{\|\theta\|}$
 - In addition,

$$\begin{aligned}\theta^T \mathbf{x}_n + \theta_0 &\geq 1, & \mathbf{x}_n &\in \omega_1, \\ \theta^T \mathbf{x}_n + \theta_0 &\leq -1, & \mathbf{x}_n &\in \omega_2.\end{aligned}$$

- We are now ready to state the task that defines the maximum margin classifier, for linearly separable classes.

- **Maximum margin classifier:** The corresponding optimization task is cast as follows:

$$\begin{aligned} &\text{minimize w.r. to } \boldsymbol{\theta} && \frac{1}{2} \|\boldsymbol{\theta}\|^2 \\ &\text{subject to} && y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N. \end{aligned}$$

- In other words, from the **infinity of linear classifiers**, which can solve the task and classify correctly all the training patterns, our optimization task selects the one which has **minimum norm**.
- However, as we have just seen, minimizing the norm $\|\boldsymbol{\theta}\|$ is equivalent with **maximizing the respective margin**! This is done in a way that guarantees that **all** the training points are classified correctly, as dictated by the **associated constraints**.
- Thus, in this context, **regularization** that controls the norm of the estimate, is dressed up with the **geometric interpretation of the margin**.

- **Maximum margin classifier**: The corresponding optimization task is cast as follows:

$$\begin{aligned} &\text{minimize w.r. to } \boldsymbol{\theta} && \frac{1}{2} \|\boldsymbol{\theta}\|^2 \\ &\text{subject to} && y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N. \end{aligned}$$

- In other words, from the **infinity of linear classifiers**, which can solve the task and classify correctly all the training patterns, our optimization task selects the one which has **minimum norm**.
- However, as we have just seen, minimizing the norm $\|\boldsymbol{\theta}\|$ is equivalent with **maximizing the respective margin**! This is done in a way that guarantees that **all** the training points are classified correctly, as dictated by the **associated constraints**.
- Thus, in this context, **regularization** that controls the norm of the estimate, is dressed up with the **geometric interpretation of the margin**.

- **Maximum margin classifier**: The corresponding optimization task is cast as follows:

$$\begin{aligned} &\text{minimize w.r. to } \boldsymbol{\theta} && \frac{1}{2} \|\boldsymbol{\theta}\|^2 \\ &\text{subject to} && y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N. \end{aligned}$$

- In other words, from the **infinity of linear classifiers**, which can solve the task and classify correctly all the training patterns, our optimization task selects the one which has **minimum norm**.
- However, as we have just seen, minimizing the norm $\|\boldsymbol{\theta}\|$ is equivalent with **maximizing the respective margin**! This is done in a way that guarantees that **all** the training points are classified correctly, as dictated by the **associated constraints**.
- Thus, in this context, **regularization** that controls the norm of the estimate, is dressed up with the **geometric interpretation of the margin**.

- **Maximum margin classifier**: The corresponding optimization task is cast as follows:

$$\begin{aligned} &\text{minimize w.r. to } \boldsymbol{\theta} && \frac{1}{2} \|\boldsymbol{\theta}\|^2 \\ &\text{subject to} && y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N. \end{aligned}$$

- In other words, from the **infinity of linear classifiers**, which can solve the task and classify correctly all the training patterns, our optimization task selects the one which has **minimum norm**.
- However, as we have just seen, minimizing the norm $\|\boldsymbol{\theta}\|$ is equivalent with **maximizing the respective margin**! This is done in a way that guarantees that **all** the training points are classified correctly, as dictated by the **associated constraints**.
- Thus, in this context, **regularization** that controls the norm of the estimate, is dressed up with the **geometric interpretation of the margin**.

Maximum Margin Classifiers

- The previous task is a typical one of a **convex cost** under **convex inequality constraints** and can be solved by mobilizing standard methods via the introduction of Lagrange multipliers.
- **Support vectors**: The solution turns out to be in an **elegant and simple** form and it is given as a **linear combination** of a **subset** of the training samples, i.e.,

$$\hat{\theta} = \sum_{n=1}^{N_s} \lambda_n y_n \mathbf{x}_n,$$

where, N_s are the **nonzero** Lagrange multipliers.

Maximum Margin Classifiers

- The previous task is a typical one of a **convex cost** under **convex inequality constraints** and can be solved by mobilizing standard methods via the introduction of Lagrange multipliers.
- **Support vectors**: The solution turns out to be in an **elegant and simple** form and it is given as a **linear combination of a subset** of the training samples, i.e.,

$$\hat{\theta} = \sum_{n=1}^{N_s} \lambda_n y_n \mathbf{x}_n,$$

where, N_s are the **nonzero** Lagrange multipliers.

Maximum Margin Classifiers

- It turns out that, **only** the Lagrange multipliers associated with **the nearest points to the (hyperplane) classifier**, i.e., those points satisfying the constraints with equality ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1$), are **nonzero**. These are known as the **support vectors**. The Lagrange multipliers corresponding to the **points outside the margin** ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) > 1$) are **zero**.
- Because from all the training points, the solution depends **only on the support vectors**, this type of classifiers are known as **support vector machines (SVM)**. In other words, for this case, the solution depends only on the nearest to the classifier points. Points that lie **outside the margin** have not a say in the formation of the corresponding hyperplane.

Maximum Margin Classifiers

- It turns out that, **only** the Lagrange multipliers associated with **the nearest points to the (hyperplane) classifier**, i.e., those points satisfying the constraints with equality ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1$), are **nonzero**. These are known as the **support vectors**. The Lagrange multipliers corresponding to the **points outside the margin** ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) > 1$) are **zero**.
- Because from all the training points, the solution depends **only on the support vectors**, this type of classifiers are known as **support vector machines (SVM)**. In other words, for this case, the solution depends only on the nearest to the classifier points. Points that lie **outside the margin** have not a say in the formation of the corresponding hyperplane.

Maximum Margin Classifiers

- **The nonlinear case in an RKHS:** For the more general RKHS case, the solution becomes,

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \mathbf{x}_n),$$

which leads to the following prediction rule.

- Given an unknown \mathbf{x} , its class label is predicted according to the sign of

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0.$$

Maximum Margin Classifiers

- **The nonlinear case in an RKHS:** For the more general RKHS case, the solution becomes,

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \mathbf{x}_n),$$

which leads to the following prediction rule.

- Given an unknown \mathbf{x} , its class label is predicted according to the **sign** of

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0.$$

Maximum Margin Classifiers

- The bias, $\hat{\theta}_0$, is obtained by selecting all constraints with $\lambda_n \neq 0$, corresponding to (the counterpart of $(y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1$ for the linear case)

$$y_n \left(\sum_{m=1}^N \lambda_m y_m \kappa(\mathbf{x}_m, \mathbf{x}_n) + \hat{\theta}_0 \right) - 1 = 0, \quad n = 1, 2, \dots, N_s.$$

and $\hat{\theta}_0$ is computed as the average of the values obtained from each one of these constraints.

- The task has a **unique solution**; however, the corresponding Lagrange multipliers **may not** be unique.

Maximum Margin Classifiers

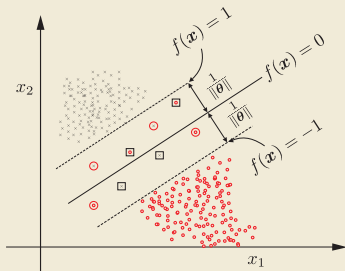
- The bias, $\hat{\theta}_0$, is obtained by selecting all constraints with $\lambda_n \neq 0$, corresponding to (the counterpart of $(y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1$ for the linear case)

$$y_n \left(\sum_{m=1}^N \lambda_m y_m \kappa(\mathbf{x}_m, \mathbf{x}_n) + \hat{\theta}_0 \right) - 1 = 0, \quad n = 1, 2, \dots, N_s.$$

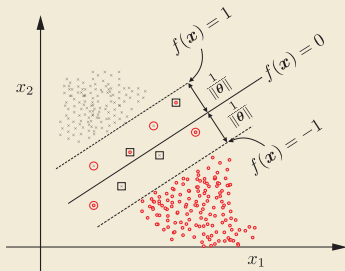
and $\hat{\theta}_0$ is computed as the average of the values obtained from each one of these constraints.

- The task has a **unique solution**; however, the corresponding Lagrange multipliers **may not** be unique.

- **The nonseparable class case:** We now turn our attention to the more realistic case of **overlapping classes**. In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. The following figure shows the respective geometry for a linear classifier, $f(\mathbf{x}) = 0$ and the corresponding margin that is defined, as before, by the two hyperplanes, $f(\mathbf{x}) = \pm 1$. For all the points, define the respective slack variables, ξ_n , $n = 1, 2, \dots, N$. There are three types of points:



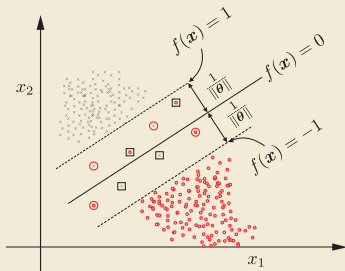
- **The nonseparable class case:** We now turn our attention to the more realistic case of **overlapping classes**. In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. The following figure shows the respective geometry for a linear classifier, $f(\mathbf{x}) = 0$ and the corresponding margin that is defined, as before, by the two hyperplanes, $f(\mathbf{x}) = \pm 1$. For all the points, define the respective slack variables, ξ_n , $n = 1, 2, \dots, N$. There are three types of points:



$$\text{a) } y_n f(\mathbf{x}_n) \geq 1, \quad \xi_n = 0$$

a) Points that lie **outside or on the borders** of the margin and are classified correctly ($\xi_n = 0$),

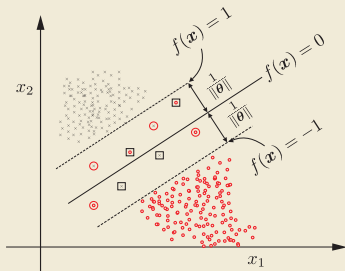
- **The nonseparable class case:** We now turn our attention to the more realistic case of **overlapping classes**. In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. The following figure shows the respective geometry for a linear classifier, $f(\mathbf{x}) = 0$ and the corresponding margin that is defined, as before, by the two hyperplanes, $f(\mathbf{x}) = \pm 1$. For all the points, define the respective slack variables, ξ_n , $n = 1, 2, \dots, N$. There are three types of points:



- a) $y_n f(\mathbf{x}_n) \geq 1$, $\xi_n = 0$
- b) $y_n f(\mathbf{x}_n) \geq 1 - \xi_n$, $0 < \xi_n < 1$

- a) Points that lie **outside or on the borders** of the margin and are classified correctly ($\xi_n = 0$),
- b) points **inside the margin** and classified correctly ($0 < \xi_n < 1$) denoted by circles,

- **The nonseparable class case:** We now turn our attention to the more realistic case of **overlapping classes**. In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. The following figure shows the respective geometry for a linear classifier, $f(\mathbf{x}) = 0$ and the corresponding margin that is defined, as before, by the two hyperplanes, $f(\mathbf{x}) = \pm 1$. For all the points, define the respective slack variables, ξ_n , $n = 1, 2, \dots, N$. There are three types of points:



- a) $y_n f(\mathbf{x}_n) \geq 1$, $\xi_n = 0$
- b) $y_n f(\mathbf{x}_n) \geq 1 - \xi_n$, $0 < \xi_n < 1$
- c) $y_n f(\mathbf{x}_n) \leq 0$, $1 \leq \xi_n$

- a) Points that lie **outside or on the borders** of the margin and are classified correctly ($\xi_n = 0$),
- b) points **inside the margin** and classified correctly ($0 < \xi_n < 1$) denoted by circles,
- and c) **misclassified** points denoted by a square, ($\xi_n \geq 1$).

Support Vector Machines

- Our desire would be to estimate a classifier, so that to **maximize the margin and at the same time** to keep the number of **margin errors** as small as possible.
- Margin errors include points that lie on the **correct side** of the classifier but **inside** the margin. Mobilizing the **indicator** function, $I(\cdot)$, (recall: $I(\xi) = 1$ (0), if $\xi > 0$ ($\xi = 0$)), this goal can be expressed via the following optimization task:

$$\begin{aligned} \text{minimize w.r. to } \boldsymbol{\theta}, \theta_0, \boldsymbol{\xi} \quad & J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N I(\xi_n), \\ \text{subject to} \quad & y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1 - \xi_n, \\ & \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

Support Vector Machines

- Our desire would be to estimate a classifier, so that to **maximize the margin and at the same time** to keep the number of **margin errors** as small as possible.
- Margin errors include points that lie on the **correct side** of the classifier but **inside** the margin. Mobilizing the **indicator** function, $I(\cdot)$, (recall: $I(\xi) = 1$ (0), if $\xi > 0$ ($\xi = 0$)), this goal can be expressed via the following optimization task:

$$\begin{aligned} \text{minimize w.r. to } \boldsymbol{\theta}, \theta_0, \boldsymbol{\xi} \quad & J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N I(\xi_n), \\ \text{subject to} \quad & y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1 - \xi_n, \\ & \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

Support Vector Machines

- However, in such a case the task becomes a combinatorial one. So, we relax the task and use ξ_n in place of the indicator function, leading to the following:

$$\begin{array}{ll} \text{minimize w.r. to } \boldsymbol{\theta}, \theta_0, \boldsymbol{\xi} & J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \xi_n, \\ \text{subject to} & y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1 - \xi_n, \\ & \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{array}$$

Support Vector Machines

- Note that, optimization is achieved in a **trade off rationale**; the user-defined parameter, C , controls the influence of each of the two contributions to the minimization task.
- If C is **large**, the resulting **margin** (the distance between the two hyperplanes defined by $f(x) = \pm 1$) will be **small in order to include a smaller number of margin errors**. If C is **small**, the **opposite is true**. As we will see from the simulation examples, **the choice of C is very crucial**.

Support Vector Machines

- Note that, optimization is achieved in a **trade off rationale**; the user-defined parameter, C , controls the influence of each of the two contributions to the minimization task.
- If C is **large**, the resulting **margin** (the distance between the two hyperplanes defined by $f(\mathbf{x}) = \pm 1$) will be **small in order to include a smaller number of margin errors**. If C is **small**, the **opposite is true**. As we will see from the simulation examples, **the choice of C is very crucial**.

Support Vector Machines

- **The solution:** Once more, the solution is given as a linear combination of a subset of the training points,

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \mathbf{x}_n),$$

where λ_n , $n = 1, 2, \dots, N_s$ are the nonzero Lagrange multipliers associated with the **support vectors**. In this case, support vectors are all points that lie either:

- on the pair of the hyperplanes that define the margin, or
 - inside the margin or,
 - outside the margin but on the wrong side of the classifier.
- That is, **correctly classified points which lie outside the margin do not contribute to the solution**, since the corresponding Lagrange multipliers are zero.

Support Vector Machines

- **The solution:** Once more, the solution is given as a linear combination of a subset of the training points,

$$\hat{\theta}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \mathbf{x}_n),$$

where λ_n , $n = 1, 2, \dots, N_s$ are the nonzero Lagrange multipliers associated with the **support vectors**. In this case, support vectors are all points that lie either:

- on the pair of the hyperplanes that define the margin, or
 - inside the margin or,
 - outside the margin but on the wrong side of the classifier.
- That is, **correctly classified points which lie outside the margin do not contribute to the solution**, since the corresponding Lagrange multipliers are zero.

- The class prediction rule is the same as before; that is, it depends on the sign of

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0$$

where $\hat{\theta}_0$ is computed from the constraints corresponding to $\lambda_n \neq 0$ and $\xi_n = 0$; these correspond to the points, which lie **on the hyperplanes that define the margin, and on the correct side of the classifier**. These classifiers also belong to the family of **support vector machines (SVM)**.

- It must be pointed out that the **number of support vectors** is related to the **generalization performance** of the classifier. The **smaller** the number of support vectors is, the **better** the generalization is expected to be.

- The class prediction rule is the same as before; that is, it depends on the sign of

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0$$

where $\hat{\theta}_0$ is computed from the constraints corresponding to $\lambda_n \neq 0$ and $\xi_n = 0$; these correspond to the points, which lie **on the hyperplanes that define the margin, and on the correct side of the classifier**. These classifiers also belong to the family of **support vector machines (SVM)**.

- It must be pointed out that the **number of support vectors** is related to the **generalization performance** of the classifier. The **smaller** the number of support vectors is, the **better** the generalization is expected to be.

Support Vector Machines

- The margin interpretation of the regularizing term has been raised in the context of statistical learning theory in the **pioneering work of Vapnik-Chervonenkis**; the latter establishes elegant performance bounds on the generalization performance of such classifiers.
- These bounds are **dimension-free**; this is the reason of the good performance of the SVMs, in spite of the fact that the design can take place in infinite dimensional spaces, while the size of the training set remains finite.

Support Vector Machines

- The margin interpretation of the regularizing term has been raised in the context of statistical learning theory in the **pioneering work of Vapnik-Chervonenkis**; the latter establishes elegant performance bounds on the generalization performance of such classifiers.
- These bounds are **dimension-free**; this is the reason of the good performance of the SVMs, in spite of the fact that the design can take place in infinite dimensional spaces, while the size of the training set remains finite.

The Hinge Loss Function and SVMs

- **The hinge loss function:** The design of SVMs can also be seen from an alternative point of view, without it being necessary to mobilize the concept of margin.
- Recall that the goal of designing any classifier is to estimate a function, classifier, f , so that to predict the labels for each one of the classes. Then, the classifier is estimated according to the empirical cost, based on a loss function $\mathcal{L}(\cdot, \cdot)$, i.e.,

$$J(f) = \sum_{n=1}^N \mathcal{L}(y_n, f(x_n)), \quad y_n = \begin{cases} +1, & \text{if } x_n \in \omega_1, \\ -1, & \text{if } x_n \in \omega_2, \end{cases}$$

for the two-class case.

The Hinge Loss Function and SVMs

- **The hinge loss function:** The design of SVMs can also be seen from an alternative point of view, without it being necessary to mobilize the concept of margin.
- Recall that the goal of designing any classifier is to estimate a function, classifier, f , so that to predict the labels for each one of the classes. Then, the classifier is estimated according to the empirical cost, based on a loss function $\mathcal{L}(\cdot, \cdot)$, i.e.,

$$J(f) = \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)), \quad y_n = \begin{cases} +1, & \text{if } \mathbf{x}_n \in \omega_1, \\ -1, & \text{if } \mathbf{x}_n \in \omega_2, \end{cases}$$

for the two-class case.

The Hinge Loss Function and SVMs

- For a binary classification task, the first loss function that comes into the mind is,

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} 1, & \text{if } yf(\mathbf{x}) \leq 0, \\ 0, & \text{otherwise,} \end{cases}$$

which is also known as the **(0, 1)-loss function**. However, this is a **discontinuous function and its optimization is a hard task**. To this end, a number of alternative loss functions has been adopted in an effort to approximate the (0, 1)-loss function.

- Recall that the LS loss can also be employed but, this is not well suited for classification tasks and **bears little resemblance** with the (0, 1)-loss function.

The Hinge Loss Function and SVMs

- For a binary classification task, the first loss function that comes into the mind is,

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} 1, & \text{if } yf(\mathbf{x}) \leq 0, \\ 0, & \text{otherwise,} \end{cases}$$

which is also known as the **(0, 1)-loss function**. However, this is a **discontinuous function and its optimization is a hard task**. To this end, a number of alternative loss functions has been adopted in an effort to approximate the (0, 1)-loss function.

- Recall that the LS loss can also be employed but, this is not well suited for classification tasks and **bears little resemblance** with the (0, 1)-loss function.

The Hinge Loss Function and SVMs

- In the SVM context, we turn our attention to the **hinge** loss function defined as,

$$\mathcal{L}_\rho(y, f(\mathbf{x})) = \max \{0, \rho - yf(\mathbf{x})\}.$$

- In other words, if the sign of the product between the true label, y , and the predicted by the discriminant function value, $f(\mathbf{x})$, is **positive and larger than a threshold/margin** (user-defined) value $\rho \geq 0$, the loss is zero. If not, the loss exhibits a linear increase.
- We say that a **margin error** is committed if $yf(\mathbf{x})$ cannot achieve a value at least ρ .

The Hinge Loss Function and SVMs

- In the SVM context, we turn our attention to the **hinge** loss function defined as,

$$\mathcal{L}_\rho(y, f(\mathbf{x})) = \max \{0, \rho - yf(\mathbf{x})\}.$$

- In other words, if the sign of the product between the true label, y , and the predicted by the discriminant function value, $f(\mathbf{x})$, is **positive and larger than a threshold/margin** (user-defined) value $\rho \geq 0$, the loss is zero. **If not, the loss exhibits a linear increase.**
- We say that a **margin error** is committed if $yf(\mathbf{x})$ cannot achieve a value at least ρ .

The Hinge Loss Function and SVMs

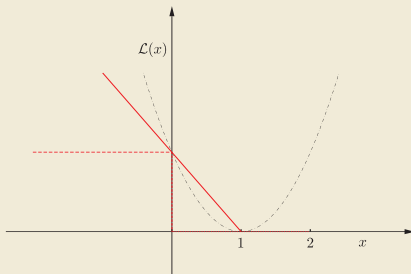
- In the SVM context, we turn our attention to the **hinge** loss function defined as,

$$\mathcal{L}_\rho(y, f(\mathbf{x})) = \max \{0, \rho - yf(\mathbf{x})\}.$$

- In other words, if the sign of the product between the true label, y , and the predicted by the discriminant function value, $f(\mathbf{x})$, is **positive and larger than a threshold/margin** (user-defined) value $\rho \geq 0$, the loss is zero. **If not, the loss exhibits a linear increase.**
- We say that a **margin error is committed** if $yf(\mathbf{x})$ cannot achieve a value at least ρ .

The Hinge Loss Function and SVMs

- The hinge loss function is shown in the figure below, together with $(0, 1)$ and squared error loss functions.



The $(0, 1)$ -loss (dotted red), the hinge loss (red) and the squared error (dotted black) functions tuned to pass through the $(0, 1)$ point for comparison. For the hinge loss, $\rho = 1$.

The Hinge Loss Function and SVMs

- Let us now see how we can view the SVM design via the hinge loss function. We will constrain ourselves to linear discriminant functions in the input space, \mathbb{R}^l . The extension to the more general RKHS space can be carried out, at the final stage, by mobilizing the **kernel trick**.
- Thus, our classifier has the form,

$$f(x) = \theta_0 + \theta^T x.$$

The Hinge Loss Function and SVMs

- Let us now see how we can view the SVM design via the hinge loss function. We will constrain ourselves to linear discriminant functions in the input space, \mathbb{R}^l . The extension to the more general RKHS space can be carried out, at the final stage, by mobilizing the **kernel trick**.
- Thus, our classifier has the form,

$$f(\mathbf{x}) = \theta_0 + \boldsymbol{\theta}^T \mathbf{x}.$$

The Hinge Loss Function and SVMs

- The goal of designing a linear classifier now becomes equivalent with minimizing the cost,

$$J(\boldsymbol{\theta}, \theta_0) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}_{\rho} \left(y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \right).$$

- However, employing slack variables, and following a similar reasoning as before, minimizing the previous cost **becomes equivalent** to

$$\begin{aligned} &\text{minimize w.r. to } \boldsymbol{\theta}, \theta_0, \boldsymbol{\xi} & J(\boldsymbol{\theta}, \boldsymbol{\xi}) &= \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \xi_n, \\ &\text{subject to} & y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) &\geq \rho - \xi_n, \\ & & \xi_n &\geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

The Hinge Loss Function and SVMs

- The goal of designing a linear classifier now becomes equivalent with minimizing the cost,

$$J(\boldsymbol{\theta}, \theta_0) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}_{\rho} \left(y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \right).$$

- However, employing slack variables, and following a similar reasoning as before, minimizing the previous cost **becomes equivalent** to

$$\begin{aligned} \text{minimize w.r. to } \boldsymbol{\theta}, \theta_0, \boldsymbol{\xi} \quad & J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \xi_n, \\ \text{subject to} \quad & y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq \rho - \xi_n, \\ & \xi_n \geq 0, \quad n = 1, 2, \dots, N. \end{aligned}$$

The Hinge Loss Function and SVMs

- Indeed, assuming $\rho = 1$ without harming generality, a margin error is committed if $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \leq 1$, corresponding to $\xi_n > 0$. On the other hand, if $\xi_n = 0$, then $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1$. The latter corresponds to **the region where the hinge loss becomes zero**.
- The goal of the second optimization task is to drive as many of the ξ_n 's to zero as possible; however, this is equivalent with estimating the parameters so that the hinge loss-based task has as many points scoring zero as possible.

The Hinge Loss Function and SVMs

- Indeed, assuming $\rho = 1$ without harming generality, a margin error is committed if $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \leq 1$, corresponding to $\xi_n > 0$. On the other hand, if $\xi_n = 0$, then $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1$. The latter corresponds to **the region where the hinge loss becomes zero**.
- The goal of the second optimization task is to drive as many of the ξ_n 's to zero as possible; however, this is equivalent with estimating the parameters so that the hinge loss-based task has as many points scoring zero as possible.

The Hinge Loss Function and SVMs

- The hinge loss interpretation paves the way for using alternative algorithms for convex optimization. The use of the slack variables and the respective solution, as discussed before, leads to solving the task in the so called **dual formulation**, which provides the solution in terms of inner products; this allows the extension to RKH spaces via the use of the kernel trick.
- However, for nonlinear tasks, this can also be made possible, if one uses the hinge loss formulation and takes advantage of the representer theorem, to replace f in terms of N parameters.

The Hinge Loss Function and SVMs

- The hinge loss interpretation paves the way for using alternative algorithms for convex optimization. The use of the slack variables and the respective solution, as discussed before, leads to solving the task in the so called **dual formulation**, which provides the solution in terms of inner products; this allows the extension to RKH spaces via the use of the kernel trick.
- However, for nonlinear tasks, this can also be made possible, if one uses the hinge loss formulation and takes advantage of the representer theorem, to replace f in terms of N parameters.

- In this example, the performance of the SVM is tested in the context of a two-class two-dimensional classification task. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\mathbf{x}_n = [x_{n,1}, x_{n,2}]^T$, $n = 1, 2, \dots, N$, we compute

$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + 1 + \eta,$$

where η stands for zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. The point is assigned to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies. That is, if $y_n > f(x_{n1})$ the point is assigned to class ω_1 otherwise is assigned to class ω_2 .

- The Gaussian kernel was used with $\sigma = 10$, since this resulted in the best performance. In the next slide, Figure (a) shows the obtained classifier for $C = 20$ and Figure (b) for $C = 1$. Observe how the obtained classifier, and hence the performance, **depends heavily on the choice of C** . In the former case, the number of the support vectors was equal to 64 and for the latter equal to 84.

- In this example, the performance of the SVM is tested in the context of a two-class two-dimensional classification task. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\mathbf{x}_n = [x_{n,1}, x_{n,2}]^T$, $n = 1, 2, \dots, N$, we compute

$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + 1 + \eta,$$

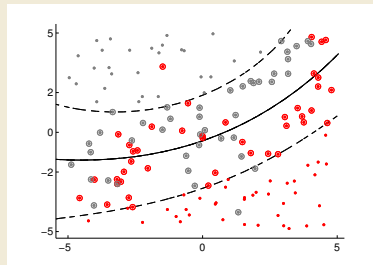
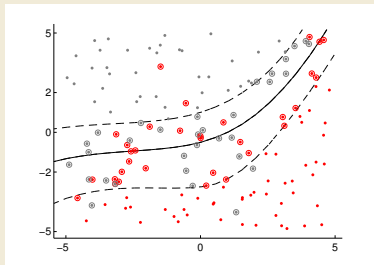
where η stands for zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. The point is assigned to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies. That is, if $y_n > f(x_{n1})$ the point is assigned to class ω_1 otherwise is assigned to class ω_2 .

- The Gaussian kernel was used with $\sigma = 10$, since this resulted in the best performance. In the next slide, Figure (a) shows the obtained classifier for $C = 20$ and Figure (b) for $C = 1$. Observe how the obtained classifier, and hence the performance, **depends heavily on the choice of C** . In the former case, the number of the support vectors was equal to 64 and for the latter equal to 84.

An SVM-based Classification Example



a) The training data points for the two classes (red and gray respectively). The full line is the graph of the obtained SVM classifier and the dotted lines indicate the margin, for $C = 20$. b) The result for $C = 1$. For both cases, the Gaussian kernel with $\sigma = 20$ was used.

- A notable characteristic of the support vector machines is that the complexity is independent of the dimensionality of the respective RKHS. The need of having a large number of parameters is bypassed and this has an influence on the generalization performance; SVMs exhibit very good generalization performance in practice. Theoretically, such a claim is substantiated by their maximum margin interpretation, in the framework of the elegant **structural risk minimization theory**, developed by Vapnik and Chernovenkis.
- Various studies concerning the comparative study of the performance of SVMs against other popular classifiers demonstrate that the SVMs rank at the very top among the most popular of the classifiers. However, there are cases for which other methods score better performance.

- A notable characteristic of the support vector machines is that the complexity is independent of the dimensionality of the respective RKHS. The need of having a large number of parameters is bypassed and this has an influence on the generalization performance; SVMs exhibit very good generalization performance in practice. Theoretically, such a claim is substantiated by their maximum margin interpretation, in the framework of the elegant **structural risk minimization theory**, developed by Vapnik and Chernovenkis.
- Various studies concerning the comparative study of the performance of SVMs against other popular classifiers demonstrate that the SVMs rank at the very top among the most popular of the classifiers. However, there are cases for which other methods score better performance.

Choice of Hyperparameters

- One of the main issues associated with SVM/SVRs is the choice of the parameter, C , which controls the relative influence of the loss and the regularizing parameter in the cost function.
- Although some efforts have been done in developing theoretical tools for the respective optimization, the path that has survived in practice is that of **cross-validation** techniques against a test data set. Different values of C are used to train the model, and the value, which results in the best performance over the test set, is selected.

Choice of Hyperparameters

- One of the main issues associated with SVM/SVRs is the choice of the parameter, C , which controls the relative influence of the loss and the regularizing parameter in the cost function.
- Although some efforts have been done in developing theoretical tools for the respective optimization, the path that has survived in practice is that of **cross-validation** techniques against a test data set. Different values of C are used to train the model, and the value, which results in the best performance over the test set, is selected.

Choice of Hyperparameters

- The other main issue is the choice of the kernel function. Different kernels lead to different performance. Once more, the recipe is to use different kernels and different values for the involved parameters, within a kernel family, and keep the one that results in the best performance. For example, in the case of a Gaussian kernel, different values for the parameter σ are used.
- A line of research is to design kernels, which match the data at hand, based either on some prior knowledge or via some optimization path. However, in general, this methodology leads to non-convex optimization tasks.

Choice of Hyperparameters

- The other main issue is the choice of the kernel function. Different kernels lead to different performance. Once more, the recipe is to use different kernels and different values for the involved parameters, within a kernel family, and keep the one that results in the best performance. For example, in the case of a Gaussian kernel, different values for the parameter σ are used.
- A line of research is to design kernels, which match the data at hand, based either on some prior knowledge or via some optimization path. However, in general, this methodology leads to non-convex optimization tasks.

Computational Considerations

- Solving a quadratic programming task, in general, requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory operations. To cope with such demands, a number of **decomposition techniques** have been devised, which “break” the task into a sequence of smaller ones. The **sequential minimal optimization** (SMO) algorithm breaks the task into a sequence of problems comprising two points, which **can be solved analytically**. Efficient implementation of such schemes lead to an empirical training time that scales between $\mathcal{O}(N)$ and $\mathcal{O}(N^{2.3})$.
- The activity on the computational front in the context of SVM/SVRs has been very intensive. A more detailed description is provided in the book, where some major directions are provided. The LibSVM is one among the most popular publicly available packages.

Computational Considerations

- Solving a quadratic programming task, in general, requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory operations. To cope with such demands, a number of **decomposition techniques** have been devised, which “break” the task into a sequence of smaller ones. The **sequential minimal optimization** (SMO) algorithm breaks the task into a sequence of problems comprising two points, which **can be solved analytically**. Efficient implementation of such schemes lead to an empirical training time that scales between $\mathcal{O}(N)$ and $\mathcal{O}(N^{2.3})$.
- The activity on the computational front in the context of SVM/SVRs has been very intensive. A more detailed description is provided in the book, where some major directions are provided. The LibSVM is one among the most popular publicly available packages.

- The SVM classification task has been introduced in the context of a two-class classification task. The more general M -class case can be treated in various ways:
 - **One-against-All:** One solves M two-class problems. Each time, one of the classes is classified against all the others using a different SVM each time. Thus, M classifiers are estimated, i.e.,

$$f_m(\mathbf{x}) = 0, \quad m = 1, 2, \dots, M,$$

which are trained so that $f_m(\mathbf{x}) > 0$ for $\mathbf{x} \in \omega_m$ and $f_m(\mathbf{x}) < 0$ if \mathbf{x} otherwise. Classification is achieved via the rule,

$$\text{assign } \mathbf{x} \text{ in } \omega_k : \text{ if } k = \arg \max_m f_m(\mathbf{x}).$$

According to this method, there may be regions in space where more than one of the discriminant functions score a positive value. Moreover, another disadvantage of this approach is the so called **class imbalanced problem**; this is caused by the fact that the number of training points in one of the classes (which comprises the data from $M - 1$ classes) can be much larger than the points in the other; this can introduce inaccuracies during the optimization process.

- The SVM classification task has been introduced in the context of a two-class classification task. The more general M -class case can be treated in various ways:
 - **One-against-All:** One solves M two-class problems. Each time, one of the classes is classified against all the others using a different SVM each time. Thus, M classifiers are estimated, i.e.,

$$f_m(\mathbf{x}) = 0, \quad m = 1, 2, \dots, M,$$

which are trained so that $f_m(\mathbf{x}) > 0$ for $\mathbf{x} \in \omega_m$ and $f_m(\mathbf{x}) < 0$ if \mathbf{x} otherwise. Classification is achieved via the rule,

$$\text{assign } \mathbf{x} \text{ in } \omega_k : \text{ if } k = \arg \max_m f_m(\mathbf{x}).$$

According to this method, there may be regions in space **where more than one of the discriminant functions score a positive value**, Moreover, another disadvantage of this approach is the so called **class imbalanced problem**; this is caused by the fact that the number of training points in one of the classes (which comprises the data from $M - 1$ classes) can be much larger than the points in the other; this can introduce inaccuracies during the optimization process.

Multiclass Generalizations

- Other alternatives are:
 - **One-against-One**. According to this method, one solves $\frac{M(M-1)}{2}$ binary classification tasks, by considering all classes in pairs. The final decision is taken on the basis of the majority rule.
 - The SVM rationale can be extended in estimating simultaneously M hyperplanes. However, this technique ends up with a large number of parameters, equal to $N(M-1)$, which have to be estimated via a single minimization task; this turns out to be rather prohibitive for most practical problems.
 - The multiclass task is treated in the context of **error correcting codes**. Each class is associated with a binary code word. If the code words are properly chosen, an **error resilience** is “embedded” into the process.

Multiclass Generalizations

- Other alternatives are:
 - **One-against-One**. According to this method, one solves $\frac{M(M-1)}{2}$ binary classification tasks, by considering all classes in pairs. The final decision is taken on the basis of the majority rule.
 - The SVM rationale can be extended in estimating simultaneously M hyperplanes. However, this technique ends up with a large number of parameters, equal to $N(M-1)$, which have to be estimated via a single minimization task; this turns out to be rather prohibitive for most practical problems.
 - The multiclass task is treated in the context of **error correcting codes**. Each class is associated with a binary code word. If the code words are properly chosen, an **error resilience** is “embedded” into the process.

Multiclass Generalizations

- Other alternatives are:
 - **One-against-One**. According to this method, one solves $\frac{M(M-1)}{2}$ binary classification tasks, by considering all classes in pairs. The final decision is taken on the basis of the majority rule.
 - The SVM rationale can be extended in estimating simultaneously M hyperplanes. However, this technique ends up with a large number of parameters, equal to $N(M-1)$, which have to be estimated via a single minimization task; this turns out to be rather prohibitive for most practical problems.
 - The multiclass task is treated in the context of **error correcting codes**. Each class is associated with a binary code word. If the code words are properly chosen, an **error resilience** is “embedded” into the process.