4F10: Deep Learning and Structured Data

Support Vector Machines: Advanced Topics

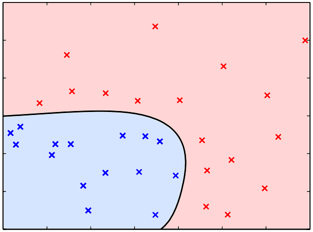
José Miguel Hernández–Lobato
Department of Engineering
University of Cambridge

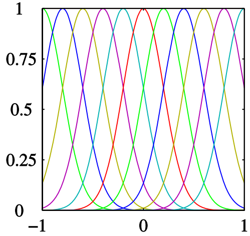
Michaelmas Term

Non-linear max-margin classifiers

So far we have only considered **linear** max-margin classifiers.

However, many classification problems have **non-linear** decision boundaries.





Solution: replace each feature vector $\mathbf{x}_n$ with a new one formed by applying non-linear functions to the original vector: $\boldsymbol{\phi}(\mathbf{x}_n) = (\phi_1(\mathbf{x}_n), \ldots, \phi_M(\mathbf{x}_n))^\mathrm{T}$.

The new classifier is given by

$$y(\mathbf{x}) = \mathbf{w}^\mathrm{T}\boldsymbol{\phi}(\mathbf{x}) + b\,,$$

and has a **non-linear decision border**.

The $\phi_m(\cdot)$, $m = 1, \ldots, M$, could be fixed **Gaussian basis functions**:

$$\phi_m(\mathbf{x}) = \exp\left\{-\frac{1}{2s}(\mathbf{x} - \mathbf{c}_m)^\mathrm{T}(\mathbf{x} - \mathbf{c}_m)\right\}\,.$$

Figure source: C. Bishop.

Gram matrix

The optimization objective for the max-margin classifier is then

$$\max_{\mathbf{a}} \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \phi(\mathbf{x}_n)^{\mathsf{T}} \phi(\mathbf{x}_m)\,,$$

For training, instead of $\mathbf{x}_1, \ldots, \mathbf{x}_N$, we only need the **gram matrix** $\mathbf{K}$, where

$$k_{n,m} = \phi(\mathbf{x}_n)^{\mathsf{T}} \phi(\mathbf{x}_m)\,.$$

No need to work with input features, only matrix $\mathbf{K}$ containing **dot products**!

After training, **predictions** can be made also using **dot products**:

$$y(\mathbf{x}) = \sum_{n \in \mathcal{S}} a_n t_n \phi(\mathbf{x})^{\mathsf{T}} \phi(\mathbf{x}_n) + b\,, \quad (\mathcal{S}: \text{ indexes of support vectors})$$

Cost scales linearly with the output dimension d of the feature mapping $\phi(\cdot)$.

Problem: d could be very large (even infinite). How can we avoid this cost in d?

Kernel functions

Computing entries $k_{n,m}$ of $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}$, where $\mathbf{\Phi} = (\phi(x_1), \ldots, \phi(x_N))^\mathsf{T}$, requires

1. Mapping inputs $\mathbf{x}_n$ and $\mathbf{x}_m$ to **feature-space** to obtain $\phi(\mathbf{x}_n)$ and $\phi(\mathbf{x}_m)$.
2. Computing dot product $\phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_m)$.

Main idea:

Instead, use **kernel function** $k(\cdot,\cdot)$ to **implicitly** map $\mathbf{x}_n$ and $\mathbf{x}_m$ to $\phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_m)$:

$$k_{n,m} = k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_m)\,.$$

$k(\cdot,\cdot)$ computes dot products in feature-space without explicitly performing the feature expansion.

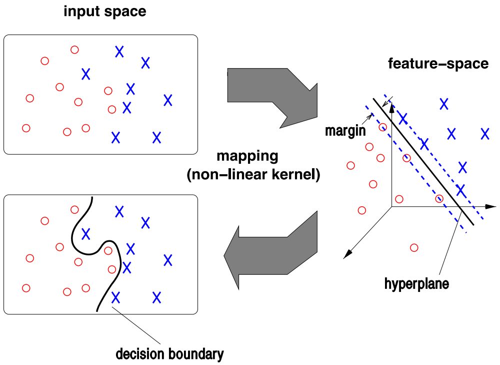


Figure source: M. Gales.

Example of kernel function

Let $\mathbf{x}_n \in \mathbb{R}^2$. The feature map

$$\phi(\mathbf{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2\right]^{\mathsf{T}} .$$

maps a 2-dimensional space to a 5-dimensional one.

The corresponding kernel function is given by

$$\begin{aligned}
k(\mathbf{x}_n, \mathbf{x}_m) &= \phi(\mathbf{x}_n)^{\mathsf{T}} \phi(\mathbf{x}_m) \\
&= \left[1, \sqrt{2}x_{n,1}, \sqrt{2}x_{n,2}, \sqrt{2}x_{n,1}x_{n,2}, x_{n,1}^2, x_{n,2}^2\right] \cdot \\
&\quad \left[1, \sqrt{2}x_{m,1}, \sqrt{2}x_{m,2}, \sqrt{2}x_{m,1}x_{m,2}, x_{m,1}^2, x_{m,2}^2\right]^{\mathsf{T}} \\
&= 1 + 2x_{n,1}x_{m,1} + 2x_{n,2}x_{m,2} + 2x_{n,1}x_{n,2}x_{m,1}x_{m,2} + x_{n,1}^2x_{m,1}^2 + x_{n,2}^2x_{m,2}^2 \\
&= (1 + x_{n,1}x_{m,1} + x_{n,2}x_{m,2})^2 \\
&= (1 + \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_m)^2 .
\end{aligned}$$

By working with $k(\cdot, \cdot)$ we avoid computing feature expansions through $\phi(\mathbf{x})$.

Example: XOR problem

Training data:

$$\mathbf{x}_1 = (1,1)\,, \qquad t_1 = 1$$
$$\mathbf{x}_2 = (-1,-1)\,, \quad t_2 = 1$$
$$\mathbf{x}_3 = (-1,1)\,, \qquad t_3 = -1$$
$$\mathbf{x}_4 = (1,-1)\,, \qquad t_4 = -1$$

Kernel function:

$$k(\mathbf{x}_n, \mathbf{x}_m) = (1 + \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m)^2\,.$$

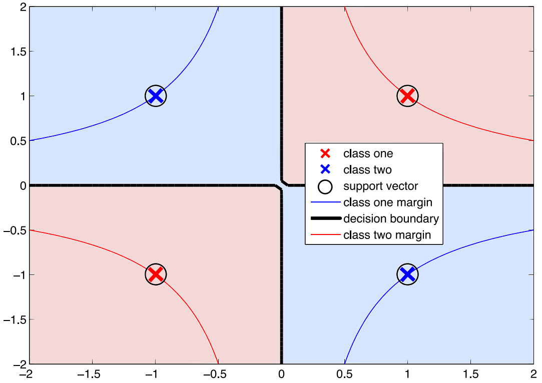


Figure source: M. Gales.

Examples of kernel functions and Mercer's condition

Kernel Name	$k(\mathbf{x}_n, \mathbf{x}_m)$
Linear	$\mathbf{x}_n^\mathsf{T}\mathbf{x}_m$
Polynomial	$(1 + \mathbf{x}_n^\mathsf{T}\mathbf{x}_m)^d$
Gaussian	$\exp\{-0.5s^{-1}\lvert\lvert\mathbf{x}_n - \mathbf{x}_m\rvert\rvert^2\}$

Question: When will any arbitrary function $k(\cdot,\cdot)$ be a valid kernel function?

$k(\cdot,\cdot)$ is a valid kernel iff there is a map $\phi(\cdot)$ such that $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_n)$ for any $\mathbf{x}_n$ and $\mathbf{x}_m$. In particular, iff for any data set, the gram matrix $\mathbf{K}$ obtained with $k(\cdot,\cdot)$ satisfies $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\mathsf{T}$, where $\mathbf{\Phi} = (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N))^\mathsf{T}$ for some $\phi(\cdot)$.

Mercer's condition: $k(\cdot,\cdot)$ is a valid kernel iff

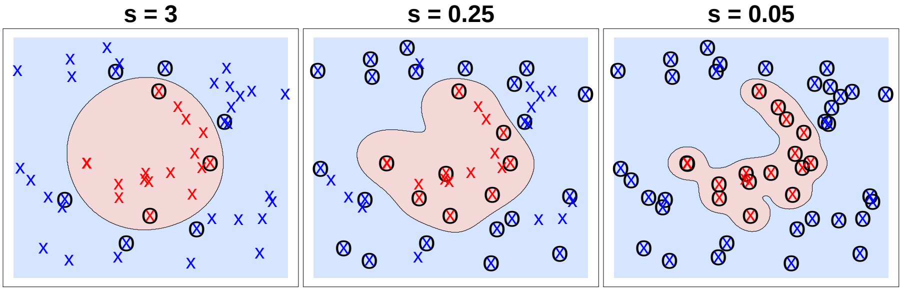
1. $k(\cdot,\cdot)$ is symmetric, that is, $k(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_m, \mathbf{x}_n)$.
2. **any** gram matrix $\mathbf{K}$ obtained with $k(\cdot,\cdot)$ is positive semi-definite:

$$\mathbf{g}^\mathsf{T}\mathbf{K}\mathbf{g} = \sum_{n,m} g_n k_{n,m} g_m \geq 0\,, \quad \text{for any vector } \mathbf{g} \text{ and any } \mathbf{x}_1, \ldots, \mathbf{x}_N\,.$$

Example with Gaussian kernel

$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\{-0.5s^{-1}||\mathbf{x}_n - \mathbf{x}_m||^2\}$ and $C = \infty$ (no constraint violations).

How does s affect the decision border? s controls smoothness of decision border.



The rank of $\mathbf{K} = \Phi\Phi^\mathsf{T}$, determines the **effective dimension** of the feature space.

As $s \to 0$, $\mathbf{K}$ approaches a diagonal matrix (rank$(K) = N$), wiggly decision border

As $s \to \infty$, all entries in $\mathbf{K}$ are the same (rank$(K) = 1$), smooth decision border.

s and C often tuned by a grid-search maximizing performance on validation data.

The kernel trick

Any algorithm that operates on the inputs $\mathbf{x}_1, \dots, \mathbf{x}_N$ by using only **dot products** can be implemented using kernels.

Example: kernelised least squares regression

Let $\mathbf{t} = (t_1, \dots, t_N)^N$. The regularized least squares objective is

$$\begin{aligned}
\mathrm{cost}(\mathbf{w}, \{(\phi(\mathbf{x}_n), t_n)\}_{n=1}^N) &= \frac{1}{2}(\mathbf{\Phi w} - \mathbf{t})^\mathsf{T}(\mathbf{\Phi w} - \mathbf{t}) + \frac{\lambda}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\
&= \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{\Phi}^\mathsf{T}\mathbf{\Phi w} + \frac{1}{2}\mathbf{t}^\mathsf{t}\mathbf{t} - \mathbf{t}^\mathsf{T}\mathbf{\Phi w} + \frac{\lambda}{2}\mathbf{w}^\mathsf{T}\mathbf{w}\,. \qquad (1)
\end{aligned}$$

The gradient is 0 if $\mathbf{w} = \mathbf{\Phi}^\mathsf{T}(\mathbf{t} - \mathbf{\Phi w})/\lambda = \mathbf{\Phi}^\mathsf{T}\mathbf{a}$. Replacing back into (1) yields

$$\mathrm{cost}(\mathbf{a}, \{(\phi(\mathbf{x}_n), t_n)\}_{n=1}^N) = \frac{1}{2}\mathbf{a}^\mathsf{T}\mathbf{\Phi\Phi}^\mathsf{T}\mathbf{\Phi\Phi}^\mathsf{T}\mathbf{a} + \frac{1}{2}\mathbf{t}^\mathsf{t}\mathbf{t} - \mathbf{t}^\mathsf{T}\mathbf{\Phi\Phi}^\mathsf{T}\mathbf{a} + \frac{\lambda}{2}\mathbf{a}^\mathsf{T}\mathbf{\Phi\Phi}^\mathsf{T}\mathbf{a}\,,$$

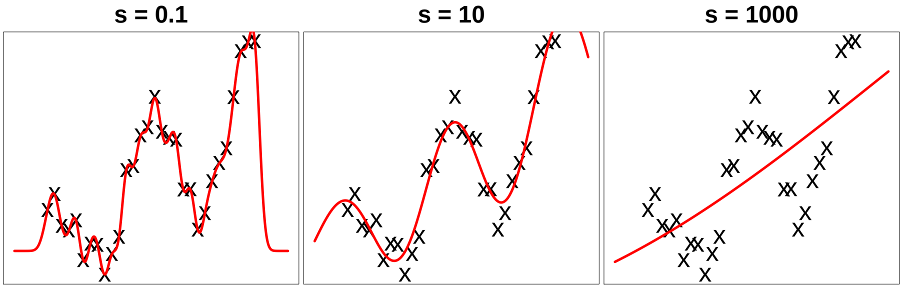
which is minimized (gradient is zero) if $\mathbf{a} = (\mathbf{\Phi\Phi}^\mathsf{T} + \lambda\mathbf{I})^{-1}\mathbf{t} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{t}$.

Predictions are then given by $y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\phi(\mathbf{x}) = \mathbf{a}^\mathsf{T}\mathbf{\Phi}\phi(\mathbf{x}) = \sum_{n=1}^N a_n k(\mathbf{x}_n, \mathbf{x})$.

Kernel least squares example

We use Gaussian kernel $k(\mathbf{x}_n, \mathbf{x}_m) = \exp\{-0.5s^{-1}||\mathbf{x}_n - \mathbf{x}_m||^2\}$ and fix $\lambda = 0.01$.

We try different values for s:

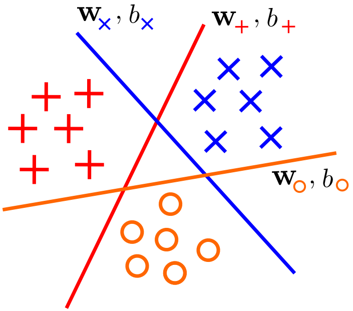


As s increases, the predictions of the model become linear and then constant.

When s is very small, the model overfits.

Multi-class max-margin classifiers

What to do when we have multiple classes?



One-vs-all classification:

Trains one classifier per class (**x**, **o** and $+$).

- **x** vs. $+$ and **o**, with parameters $\mathbf{w}_\mathbf{x}$ and $b_\mathbf{x}$.
- $+$ vs. **x** and **o**, with parameters $\mathbf{w}_+$ and b_+.
- **o** vs. $+$ and **x**, with parameters $\mathbf{w}_\mathbf{o}$ and $b_\mathbf{x}$.

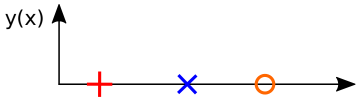
Predict label for new data point $\mathbf{x}_\star$ using

$$\hat{t}_\star = \underset{k\in\{\mathbf{x},\mathbf{o},+\}}{\arg\max}\; \mathbf{w}_k^\mathrm{T}\mathbf{x}_\star + b_k\,.$$

Problems:

- Classifiers may have different output scales.
- Many more training examples from one class than the other.

Could we solve this problem? (linear kernel)



Simultaneous learning of classifiers

Add constraints to enforce output of correct classifier to be larger than the others.

K possible classes ($t_n \in \{1, \ldots, K\}$). For each training point $(\mathbf{x}_n, t_n)$ we enforce

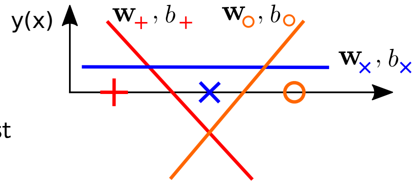
$$\underbrace{\mathbf{w}_{t_n}^\mathsf{T}\mathbf{x}_n + b_{t_n}}_{\text{prediction class } t_n} > \underbrace{\mathbf{w}_j^\mathsf{T}\mathbf{x}_n + b_j}_{\text{prediction class } j} \quad \text{for any } j \neq t_n\,.$$

The new objective to optimize with respect to $\mathbf{w}_1, \ldots, \mathbf{w}_K$ and $b_1, \ldots, b_K$ is

$$\frac{1}{2}\sum_{k=1}^{K} ||\mathbf{w}_k||^2 + C \sum_{n=1,\, j\neq t_n}^{N} \xi_{n,j} \quad \text{s.t.} \quad \begin{cases} \mathbf{w}_{t_n}^\mathsf{T}\mathbf{x}_n + b_{t_n} \geq \mathbf{w}_j^\mathsf{T}\mathbf{x}_n + b_j + 1 - \xi_{n,j} \\ \xi_{n,j} \geq 0 \end{cases}, \quad \forall\, n,\ j \neq t_n\,.$$

Predictions implemented using

$$\hat{t}_\star = \underset{k\in\{1,\ldots,K\}}{\arg\max}\ \mathbf{w}_k^\mathsf{T}\mathbf{x}_\star + b_k\,.$$



Disadvantage: very large computational cost (dual problem has many more variables). In practice, one-vs-all more widely used.

Summary

The gram matrix contains all dot products between data points in feature space.

Kernel functions sidestep feature expansion when computing dot products.

Any valid kernel must satisfy Mercer's condition.

Non-linear kernels allow us to obtain non-linear max-margin classifiers (MMC).

Any algorithm that operates on data only through dot products can be kernelised.

Multi-class MMCs can be easily obtained through the one-vs-all approach, which has some limitations but is computationally cheap and works well in practice.