Machine Learning

Kernel Methods

Alberto Maria Metelli and Francesco Trovò

Definition of Different Models

What to do in the case the model you are considering is not performing well even by tuning properly the parameters (cross-validation)?

We have two opposite options:

- simplify the model
- **increase its complexity**

In the second hypothesis, one might see the problem in a more complex space:

- use handcrafted features
- look at the problem in the **kernel** space

Constructing Kernels

Constructing Kernels

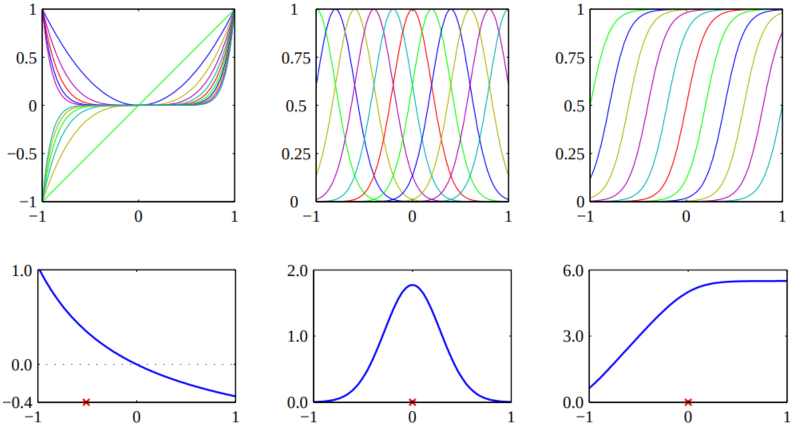
- To exploit kernel substitution need **valid** kernel functions
- First method
  - **Choose a feature space** mapping $\boldsymbol{\phi}(\mathbf{x})$ and use it to find corresponding kernel

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x})\phi_i(\mathbf{x}')$$

    - where $\boldsymbol{\phi}(\mathbf{x})$ are basis functions such as polynomial
    - for each i we choose $\phi_i(x) = x^i$ in the one-dimensional case

Construction of Kernel Functions from Basis Functions

One-dimensional input space



Testing whether a function is a valid kernel

- Second method
    - Without having to construct the function $\phi(\mathbf{x})$ explicitly
- **Necessary and sufficient** condition for a function $k(\mathbf{x}, \mathbf{x}')$ to be a kernel is
    - Gram matrix $\mathbf{K}$, whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$ is positive semi-definite for all possible choices of the set $\{\mathbf{x}_n\}$
        - Positive semi-definite is **not** the same thing as a matrix whose elements are non-negative
        - It means $\mathbf{y}^\top \mathbf{K} \mathbf{y} \geq 0$ for non-zero vectors $\mathbf{y}$ with real entries, i.e., $\sum_n \sum_m K_{n,m} y_n y_m \geq 0$ for any real numbers $\mathbf{y}$

Theorem

Mercer's theorem ***Any*** *continuous, symmetric, positive semi-definite kernel function $k(x, x')$ can be expressed as a* ***dot product*** *in a high-dimensional space*

- **New kernels** can be constructed from simpler kernels as **building blocks**

Techniques for Constructing Kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ the following new kernels will be valid

1. $k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}')$
2. $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$, where $f(\cdot)$ is any function
3. $k(\mathbf{x}, \mathbf{x}') = q(k_1(\mathbf{x}, \mathbf{x}'))$, where $q(\cdot)$ is a polynomial with non-negative coefficients
4. $k(\mathbf{x}, \mathbf{x}') = exp(k_1(\mathbf{x}, \mathbf{x}'))$
5. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
6. $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$
7. $k(\mathbf{x}, \mathbf{x}') = k_3(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}'))$, where $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$
8. $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T A\mathbf{x}'$, where A is a symmetric positive semidefinite matrix
9. $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b)$, where x_a and x_b are variables with $\mathbf{x} = (x_a, x_b)$
10. $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b)$

Gaussian Processes

GP Definition

- A Gaussian process is defined as a probability distribution over functions $y(\mathbf{x}_i)$
    - values of $y(\mathbf{x}_i)$ evaluated at any set of points $\mathbf{x}_1, \dots, \mathbf{x}_N$ jointly have a Gaussian distribution
- This distribution is completely specified by the mean and the covariance:
    - usually, we do not have any prior information about the mean of $y(\mathbf{x})$, so we take it to be zero
    - the covariance is given by the **kernel** function

$$\mathbb{C}\mathrm{ov}[y(\mathbf{x}_i), y(\mathbf{x}_j)|\mathbf{x}_i, \mathbf{x}_j] = \mathbb{E}[y(\mathbf{x}_i)\, y(\mathbf{x}_j)|\mathbf{x}_i, \mathbf{x}_j] = K(\mathbf{x}_i, \mathbf{x}_j)$$

- With this formulation, **Gaussian Process** (GP) are kernel methods that can be applied to solve regression problems

Output Modeling

- The **target** is $t = y(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is a noise independent on the point $\mathbf{x}$
- The conditional distribution of the targets $\mathbf{t}_N = (t_1, \dots, t_N)^\top$ of size N is:

$$p(\mathbf{t}_N | \mathbf{y}_N) = \mathcal{N}(\mathbf{t}_N | \mathbf{y}_N, \sigma^2 \mathbf{I}_N) \qquad \text{where } \mathbf{y}_N = (y(\mathbf{x}_1), \dots, y(\mathbf{x}_N))^\top$$

- The **prior** is $p(\mathbf{y}_N) = \mathcal{N}(\mathbf{0}, \mathbf{K}_N)$, where:

$$\mathbf{K}_N = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & \dots & K(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ K(\mathbf{x}_N, \mathbf{x}_1) & \dots & K(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

- Thus, the marginal distribution of the target is:

$$p(\mathbf{t}_N) = \int p(\mathbf{t}_N | \mathbf{y}_N) p(\mathbf{y}) \, \mathrm{d}\mathbf{y}_N = \mathcal{N}(\mathbf{t}_N | \mathbf{0}, \mathbf{C}_N) \qquad \text{where } \mathbf{C}_N = \mathbf{K}_N + \sigma^2 \mathbf{I}_N$$

Making Predictions

- We want to **predict** the target t_{N+1} corresponding to a specific unseen input $\mathbf{x}_{N+1}$
- From the definition we have:

$$p(\mathbf{t}_{N+1}) = \mathcal{N}(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}),$$

where:

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^\top & c \end{pmatrix}$$

$$\mathbf{k} = (K(\mathbf{x}_1, \mathbf{x}_{N+1}), \dots, K(\mathbf{x}_N, \mathbf{x}_{N+1}))^\top \qquad c = K(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma^2$$

- We need to compute $p(t_{N+1}|\mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_N) = \mathcal{N}(m(\mathbf{x}_{N+1}), \sigma^2(\mathbf{x}_{N+1}))$ where
  - Mean: $m(\mathbf{x}_{N+1}) = \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{t}$
  - Variance: $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^\top \mathbf{C}_N^{-1} \mathbf{k}$

GPs in Python

Model the relationship between petal length and width as a GP:

- Load the data and normalize them
- Select the values of:
  - noise variance $\sigma^2 = \mathbb{V}\mathrm{ar}[\varepsilon] = 0.2$
  - constant $k = 1$
  - lengthscale $l = 0.8$

$$K(\mathbf{x}_i, \mathbf{x}_j) = k \exp\left\{ -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2l^2} \right\}$$

- Initialize a GP regression model (`GaussianProcessRegressor`)
- Predict new values

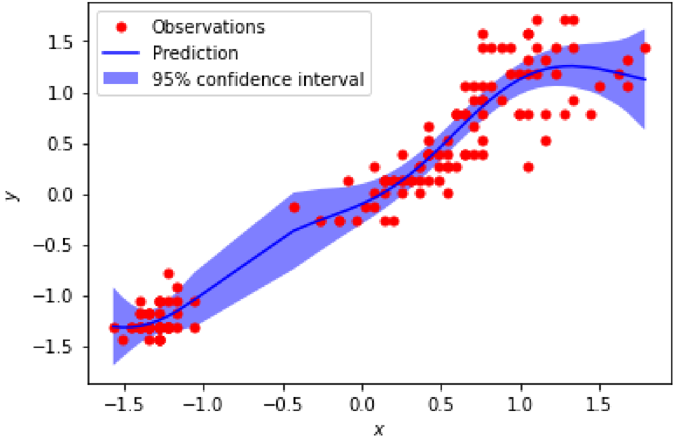
Hyperparameters

While GPs are a **non-parametric** methods, σ^2 and the parameters of the kernel (e.g., l and k) has to be estimated or set:

- using **a priori** information on the problem we are analyzing
- maximizing their **log-likelihood** on an independent dataset
- possibly improved as new data are collected

Caveat: most of the time you will see that they are estimated using the same data used for the prediction. This is clearly not a good ML practice (equivalent to overfitting)

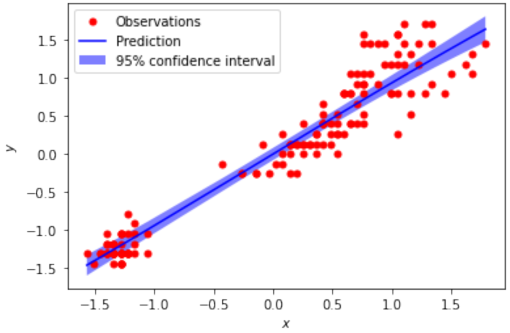
Results on the Iris Dataset

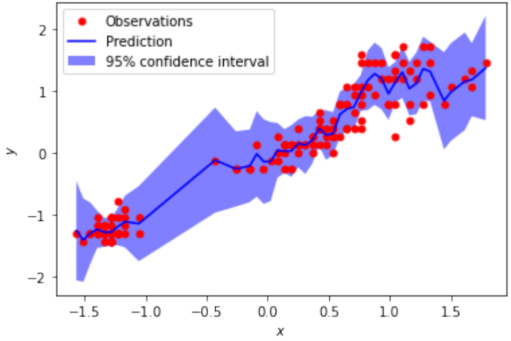


Parameters: $k = 3$, $l = 0.8$, and $\sigma^2 = 0.2$

Modify the Lengthscale

- Left: $k = 3$, $l = 8$, and $\sigma^2 = 0.2$
- Right: $k = 3$, $l = 0.08$, and $\sigma^2 = 0.2$

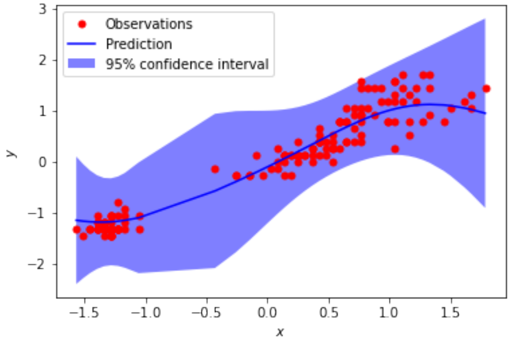


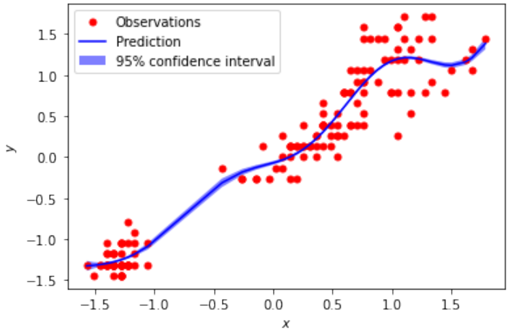


Controls the smoothness of the GP

Modify the Noise

- Left: $k = 3$, $l = 0.8$, and $\sigma^2 = 10$
- Right: $k = 3$, $l = 0.8$, and $\sigma^2 = 0.002$





Controls the target noise of the GP

Support Vector Machines

Support Vector Machines (SVM)

- Flexible and **theoretically supported** method
- Initially applied to classification only, over the years it has been extended to deal with regression, clustering and anomaly detection problems
- Idea: find the **hyperplane maximizing the margins** (distance between the boundary and the points)

Support Vector Machines (SVM)

- Hypothesis space: $y(\mathbf{x}) = f(\mathbf{x}; \mathbf{w}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$
- Loss function computed over $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N$ with $t_n \in \{-1, 1\}$:

$$\begin{aligned} \underset{\mathbf{w},\zeta_1,\dots\zeta_N}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N} \zeta_i \\ \text{s.t.} \quad & t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \zeta_i \qquad \forall n \in \{1, \dots, N\} \\ & \zeta_i \geq 0 \qquad \forall n \in \{1, \dots, N\} \end{aligned}$$

where $C > 0$ is a hyperparameter

- Optimization method: sequential quadratic optimization

Linear SVM in Python

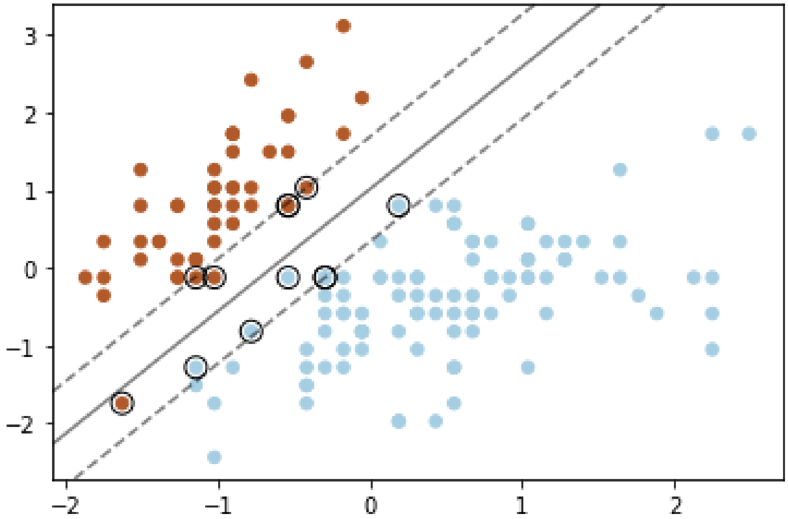
To train a linear classification SVM:

- Define an SVM: `SVM_model.svm.SVC(kernel='linear')`
- Train the SVM: `SVM_model.fit(input, target)`

We are interested to determine:

- Boundary $\mathbf{w}^T \mathbf{x}_n + b = 0$
- Margins $\mathbf{w}^T \mathbf{x}_n + b = \pm 1$
- Support vectors (`SVM_model.support_vectors_`)

Results on the Iris Dataset



Adding a Kernel

The use of kernels in the SVM is almost native (non-parametric method):

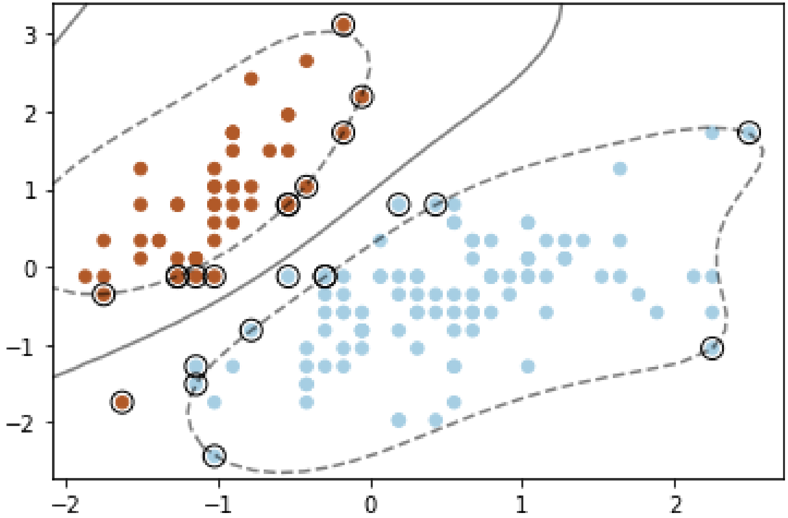
- Hypothesis space: $y(\mathbf{x}) = f(\mathbf{x}; \mathbf{w}) = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$
- Loss measure: loss function in the dual formulation
- Optimization method: quadratic optimization

In `Python`:

- Define an SVM: `SVM_model.svm.SVC()`
- Train the SVM: `SVM_model.fit(input, target)`

We do not have an explicit formula for the boundary and the margins anymore

Results on the Iris Dataset


3
2
1
0
−1
−2
−2
−1
0
1
2