Machine Learning

Kernel Methods

Mirco Mutti
Credits to 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 feature space
- increase its complexity

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 feature space
- increase its complexity

Increase its complexity

By means of kernel methods:

- define a kernel function for a d-dimensional feature space:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

- such that, even if $d \to \infty$, it is still feasible to compute $k(\boldsymbol{x}, \boldsymbol{x}')$

Gaussian Process

GP Definition

- A Gaussian process is defined as a probability distribution over functions $\mathbf{y}(\mathbf{x})$ such that the set of values of $\mathbf{y}(\mathbf{x})$ evaluated at an arbitrary 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 $\mathbf{y}(\mathbf{x})$, so we take it to be zero
  - the covariance is given by the kernel function $\mathbb{E}[t(x_i)\, t(x_j)] = K(x_i, x_j)$
- With this formulation, Gaussian Process (GP) are kernel methods that can be applied to solve regression problems

Output Modeling

- Assume to have a target $t_n = y(\mathbf{x}_n) + \varepsilon_n$, where ε_n is a measurement noise which is not dependent on the specific point $\mathbf{x}_n$
- The joint distributions of the targets $\mathbf{t}$ of dimensions N is:

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \sigma^2 I_N)$$

- since $p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, K)$, we have:

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\, d\mathbf{y} = \mathcal{N}(\mathbf{t}|\mathbf{0}, C)$$

where $C(\mathbf{x}_n, \mathbf{x}_m) = K(\mathbf{x}_n, \mathbf{x}_m) + \delta_{nm}\sigma^2$ and δ_{nm} is the Dirac delta, i.e., $\delta_{nm} = 1 \iff n = m$

Making Predictions

We would like to predict the target t_{N+1} corresponding to a specific unseen input $\mathbf{x}_{N+1}$

From the definition we have:

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

where $C_{N+1} = \begin{pmatrix} C_N & \mathbf{k} \\ \mathbf{k}^\top & c \end{pmatrix}$, $\mathbf{k}$ is the vector of $K(\mathbf{x}_i, \mathbf{x}_{N+1})$, for each $i \in \{1, \dots, N\}$ and c is the covariance $C(\mathbf{x}_{N+1}, \mathbf{x}_{N+1})$

We need to compute: $p(t_{N+1}|\mathbf{t}_N, \mathbf{x}_1, \dots, \mathbf{x}_N)$

- Mean: $m(\mathbf{x}_{N+1}) = \mathbf{k}^\top C_N^{-1}\mathbf{t}$
- Variance: $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^\top C_N^{-1}\mathbf{k}$

GPs in Python

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

- Load the data and normalize them
- Select the value for the GP:
  - noise variance $\sigma^2 = Var[\varepsilon_n] = 0.2$
  - constant $\phi = 1$
  - lengthscale $l = 0.8$
- Initialize a GP regression model (`GaussianProcessRegressor`)
- Predict new values

Kernel:

$$\mathbf{K}_{ij} = \phi \exp\left\{-\frac{||x_i - x_j||_2}{2l^2}\right\}$$

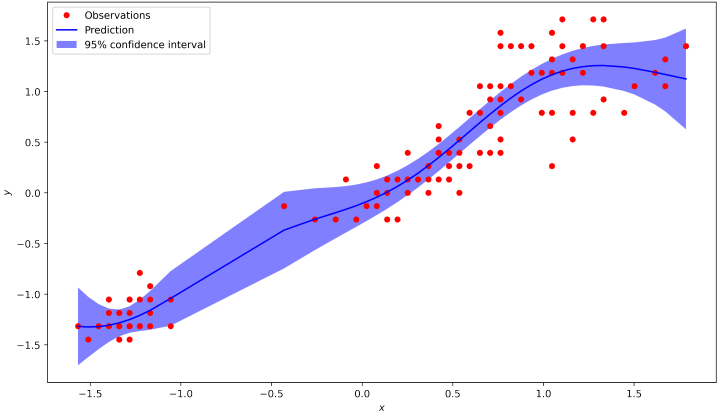
Hyperparameters

While GPs are a non-parametric methods, the parameters of the kernel has to be estimated or set:

- using a priori information on the problem we are analysing
- maximizing their loglikelihood 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)

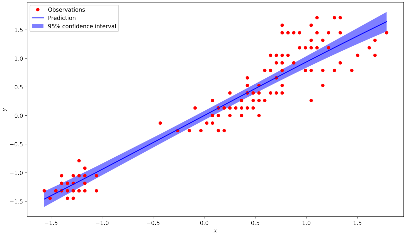
Results on the Iris Dataset

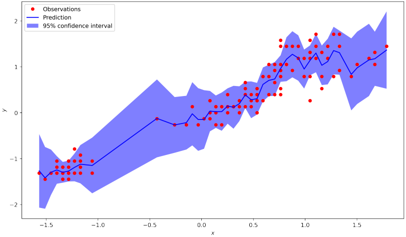


Parameters: $\phi = 3$, $l = 0.8$, and $\sigma^2 = 0.2$

Modify the Lengthscale

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

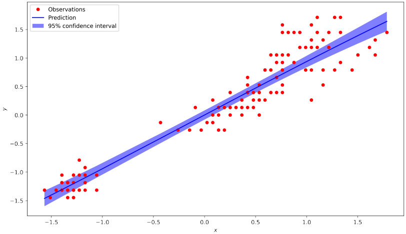


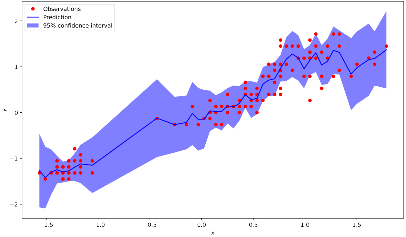


Controls the smoothness of the GP

Modify the Lengthscale

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

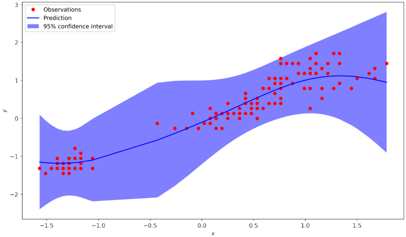


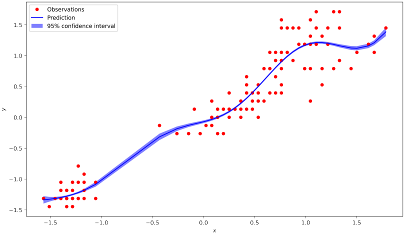


Controls the smoothness of the GP

Modify the Noise

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

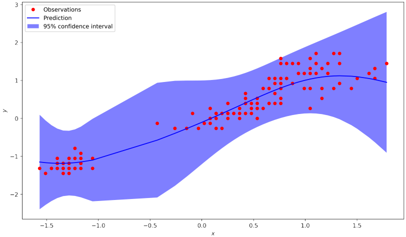


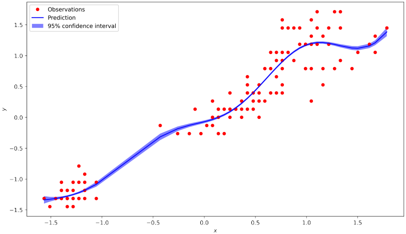


Controls the point noise of the GP

Modify the Noise

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





Controls the point noise of the GP

SVM

Support Vector Machines

- Flexible and theoretically supported method
- Initially only applied to classification, 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)

- Hypothesis space: $y_n = f(\mathbf{x}_n, \mathbf{w}) = sign\left(\mathbf{w}^T\mathbf{x}_n + b\right)$
- Loss measure: $||\mathbf{w}||^2 + C\sum_i \zeta_i$ s.t. $t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \zeta_i \ \forall n$
- Optimization method: quadratic optimization

Linear SVM in Python

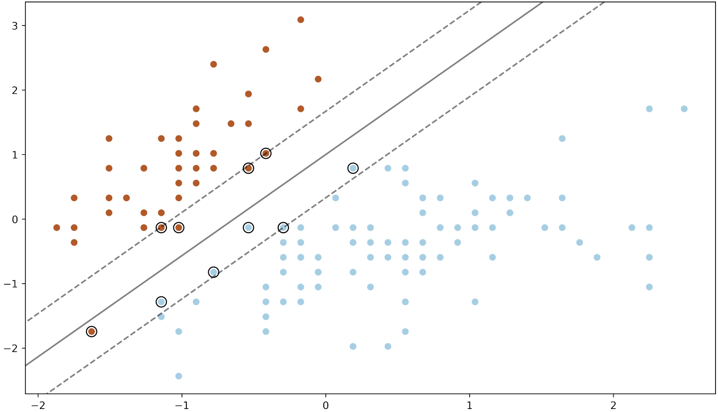
To train a linear classification SVM:

- 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: $y_n = f(\mathbf{x}_n, \mathbf{w}) = sign\left(\sum_n \alpha_i t_i K(\mathbf{x}_i, \mathbf{x}_n) + b\right)$
- 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