AARHUS
UNIVERSITET

Introduction to kernel methods: With special focus on Gaussian Process Regression

Gunnar Schmitz
Aarhus universitet

Aarhus, 25 of September, 2017

Motivation

- We want to get familiar with more different flavors of Machine Learning (ML). There is more than Neural Networks!!
- One popular branch of statistical of ML algorithms are kernel based methods.
- Therefore we should know what the so called "kernel trick" is and which popular kernel based methods are available.

The kernel trick: What is a kernel?

A kernel is the generalization of a positive function or matrix.

Definition

Let $\mathcal{X}$ be a nonempty index set. A symmetric function $\boldsymbol{K} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a positive definite kernel on $\mathcal{X}$ if

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0 \tag{1}$$

holds for any $n \in \mathbb{N}, \boldsymbol{x}_1, ..., \boldsymbol{x}_n \in \mathcal{X}, c_1, ..., c_n \in \mathbb{R}$

Some examples:

- Linear kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$
- Polynomial kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + r)^n$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d, r > 0$
- Exponential kernel (aka RBF kernel): $k(\boldsymbol{x}, \boldsymbol{y}) = e^{-\frac{(\boldsymbol{x}-\boldsymbol{y})^2}{2l^2}}$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d, l > 0$
- Sigmoid kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = \tanh(\boldsymbol{x}^T \boldsymbol{y} + r)$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d, r > 0$
- Matérn kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{|\boldsymbol{x} - \boldsymbol{y}|}{l} \right)^{\nu} K_\nu \left(\sqrt{2\nu} \frac{|\boldsymbol{x} - \boldsymbol{y}|}{l} \right)$

The kernel trick: What is a kernel?

- And there are many more kernel functions and you can create your own new one by for example addition or multiplication.
- As shown the kernel functions depend on parameters like σ and l. These parameters can have a large influence on the performance of the ML algorithm.
- However this parameters are not parameters in the classical sense and are therefore usually referred to as hyper parameters.
- Depending on the context of ML algorithm the hyper parameters can be more or less automatically adjusted.

The kernel trick: What is a then the kernel trick?

Definition

An algorithm solely defined in terms of inner products in the input space, can be lifted implicitly into a higher dim. feature space by replacing the dot products by $k(\boldsymbol{x}, \boldsymbol{y})$.

Mercer's theorem: Assuming to a kernel k a linear operator on functions defined by

$$[T_k f](\boldsymbol{t}) = \int_a^b k(\boldsymbol{t}, \boldsymbol{s}) f(\boldsymbol{s}) d\boldsymbol{s} \tag{2}$$

is associated. Then the kernel can be represented as

$$k(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j=1}^{\infty} \lambda_j e_j(\boldsymbol{x}) e_j(\boldsymbol{y}) \tag{3}$$

using the eigenfunction e_j and eigenvalues λ_j of T_k. This is equivalent

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle \tag{4}$$

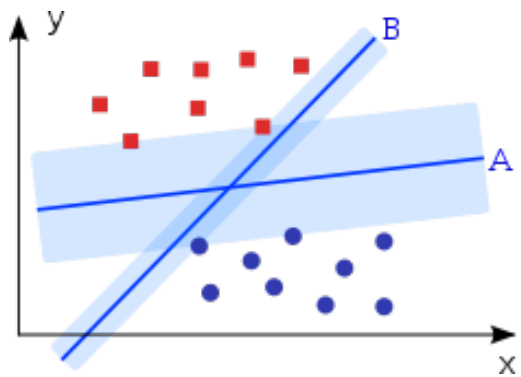
to a dot product in a different feature space using the mapping $\phi(\boldsymbol{x})$

Support Vector Machines as an illustrative example

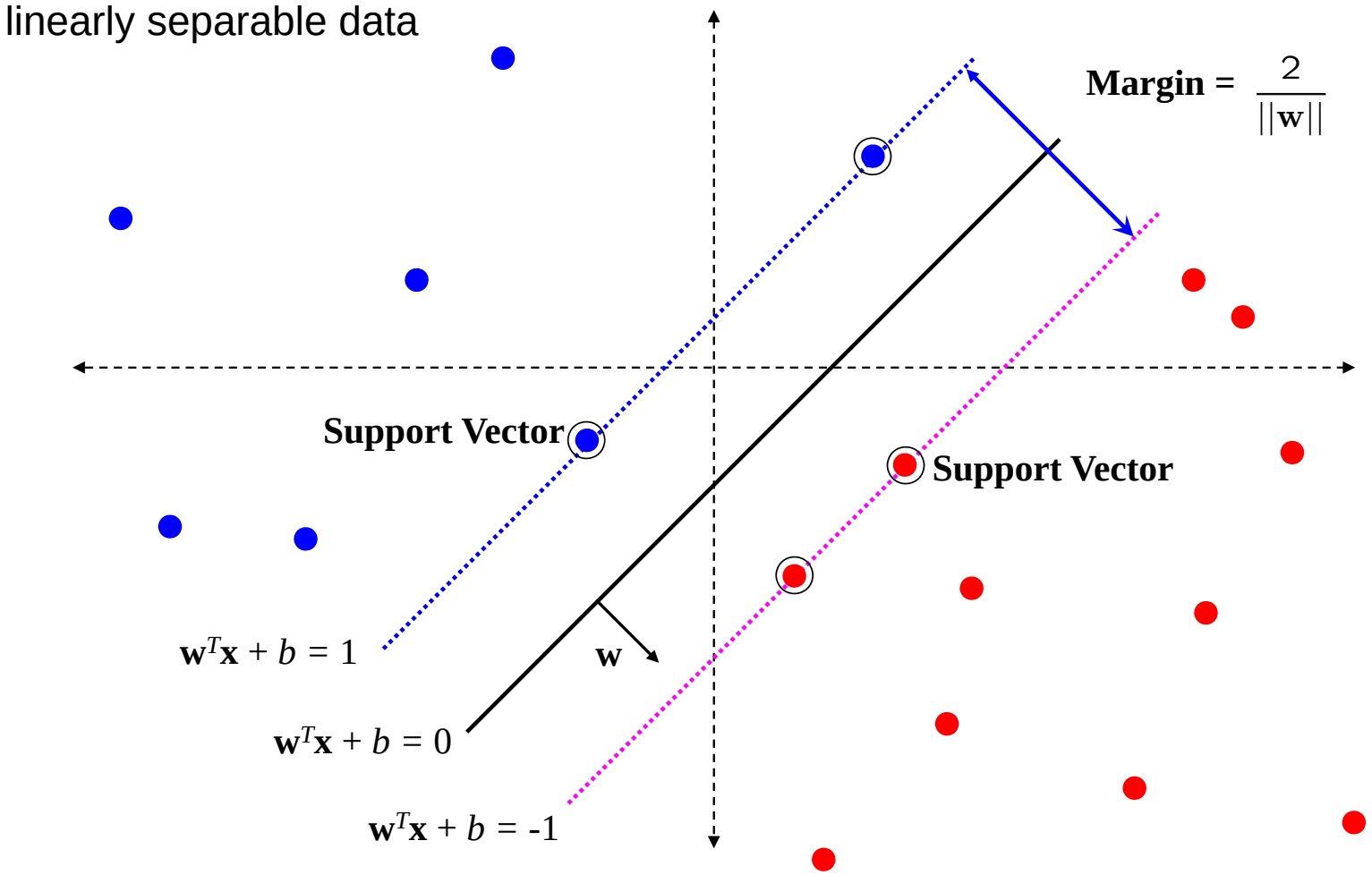
- With this trick we can turn algorithms for linear problems into algorithms for non-linear problems.
- Still this sounds a bit abstract.... let's take an example.
- We will use the **Support Vector Machines** (SVMs) as an illustrative example to demonstrate this concept quite visually.

Support Vector Machines as an illustrative example

- SVMs can be used for classification (most prominently binary).
- SVMs are also (less usually) used for regression. (Not covered today)
- Given training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, where $\boldsymbol{x}_i$ are $\mathbb{R}^d$ vectors and $y_i = 1/-1$ is the classifier, we search for the hyper plane with the maximum margin which separates the two data-sets. (Here plane A)



Support Vector Machines as an illustrative example



Support Vector Machines as an illustrative example

- Training the SVM can be formulated as an optimization problem:

$$\max_{\boldsymbol{\omega}} \frac{2}{||\boldsymbol{\omega}||} \quad \text{subject to} \quad \boldsymbol{\omega}^T \boldsymbol{x}_i + b \begin{matrix} \geq +1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{matrix} \text{ for } i = 1, \ldots, N \tag{5}$$

- Or equivalently

$$\min_{\boldsymbol{\omega}} ||\boldsymbol{\omega}||^2 \quad \text{subject to} \quad y_i \left(\boldsymbol{\omega}^T \boldsymbol{x}_i + b \right) \geq 1 \text{ for } i = 1, \ldots, N \tag{6}$$

- The $\boldsymbol{\omega}$ and b which solve this problem give our classifier

$$\boldsymbol{x} \rightarrow \operatorname{sgn} \left(\boldsymbol{\omega}^T \cdot \boldsymbol{x} - b \right) \tag{7}$$

- Actually usually a penalty term is introduced to soften the constraint

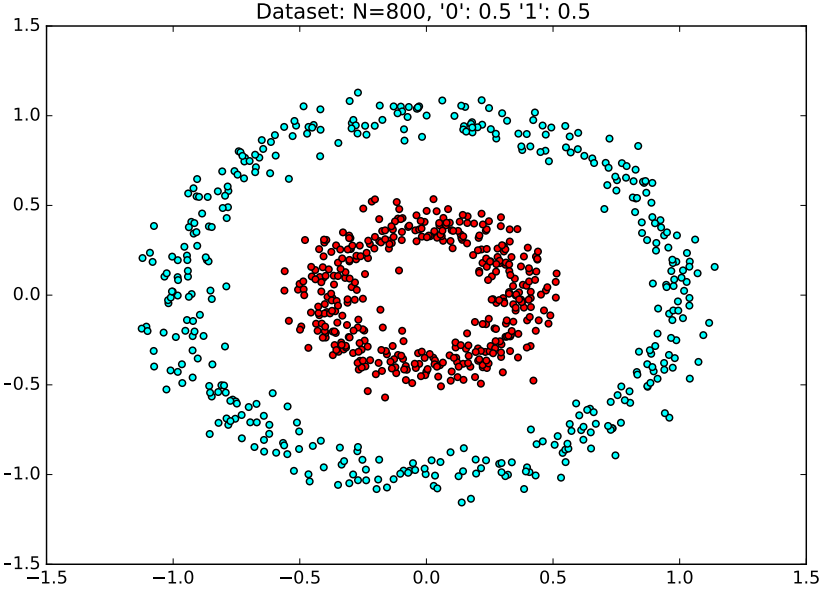
$$\min_{\boldsymbol{\omega}} ||\boldsymbol{\omega}||^2 + \lambda \sum_i^N \zeta_i \quad \text{subject to} \quad y_i \left(\boldsymbol{\omega}^T \boldsymbol{x}_i + b \right) \geq 1 - \zeta_i \text{ for } i = 1, \ldots, N \tag{8}$$

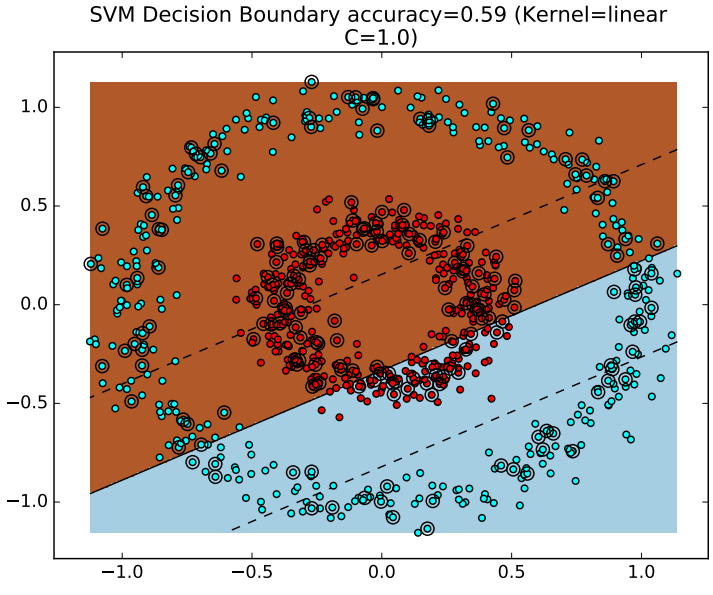
Support Vector Machines as an illustrative example

However the data must be "linearly separable".

What is in the problematic cases?

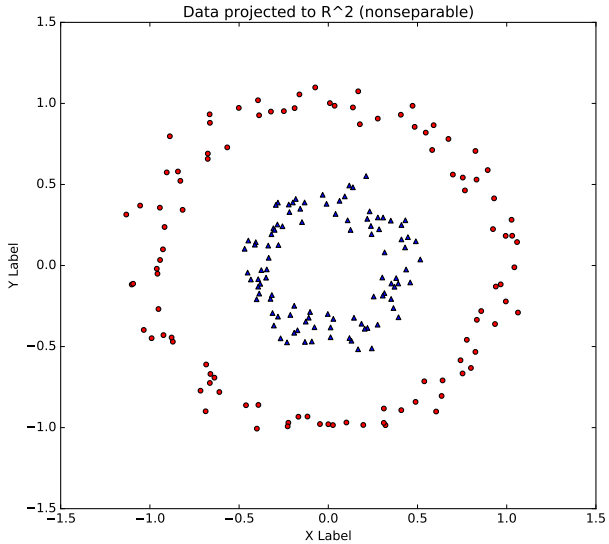
Then we just get a poor classification.

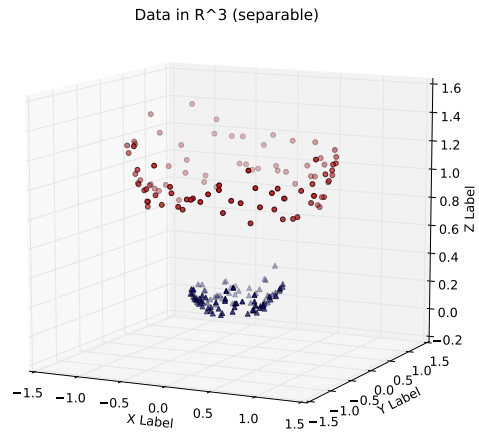




Support Vector Machines as an illustrative example

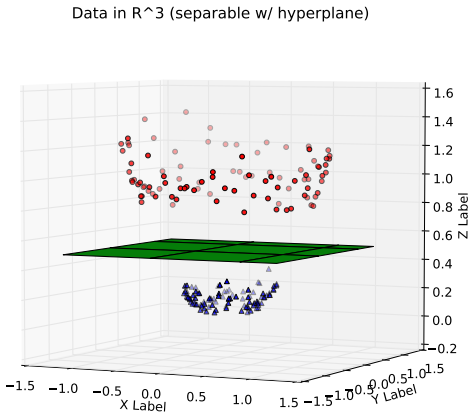
We can just give up or we can find a transformation ϕ to a higher dimensional space, where the data is separable.

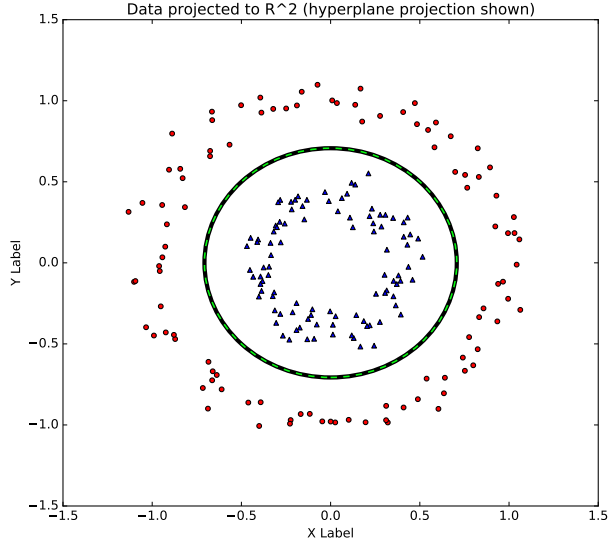




Support Vector Machines as an illustrative example

If we found this transformation ϕ we can define a plane, which separates the data sets. Afterwards we can back-transform this plane and have in the original space a non linear separator.





Support Vector Machines as an illustrative example

What has this to do with the kernel trick???

- Nothing.
- In the current scheme we have to find a transformation ϕ, which might be complicated.
- And anyway we increase the memory demands: Instead of $\mathbb{R}^N$ vectors we have to deal with $\mathbb{R}^M$ vectors with $M > N$.
- Here the kernel trick comes into play to do this implicitly without increasing the memory demand.

Support Vector Machines as an illustrative example

For example we have a polynomial kernel $k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^2$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^2$. So far no transformation involved. We just calculate a scalar:

$$k(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^2 = (x_1 y_1 + x_2 y_2)^2 = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 \quad (9)$$

However, this is nothing else but a dot product of two vectors $(x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$ and $(y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2)$.

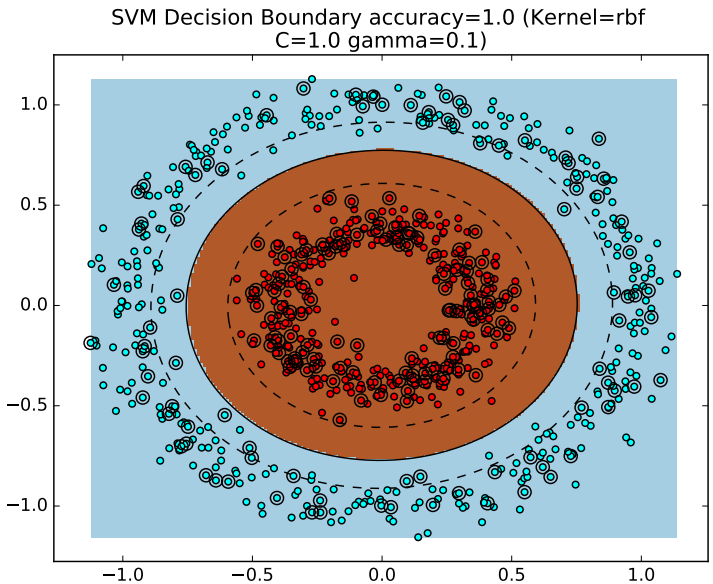
Without knowing the transformation ϕ from the input space to this this higher dimensional feature space, we calculated the dot product

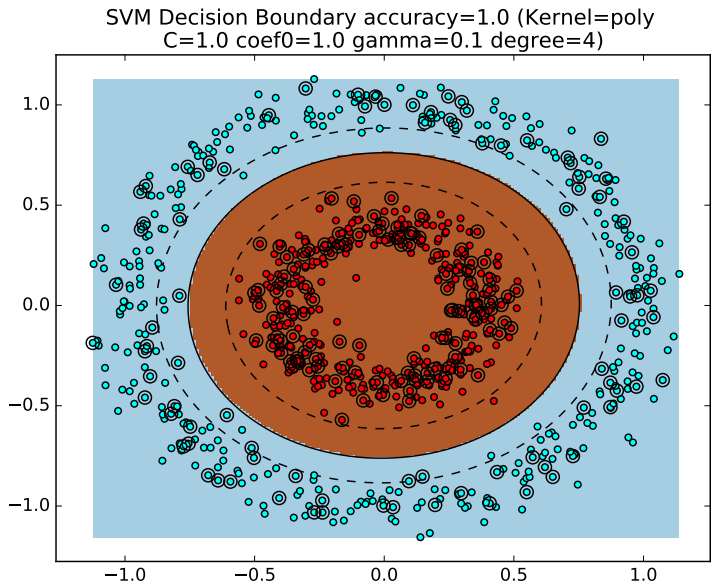
$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle \quad (10)$$

In general, the kernel function is equivalent to a dot product in a feature space. (Mercer's Theorem)

Support Vector Machines as an illustrative example

If we replace the dot products in the linear SVM algorithm we can use different kernel functions and get better classifications:





Gaussian Process regression

We will know talk about our hopefully future work horse **Gaussian Process Regression** (GPR) a bit more in detail.

Gaussian Process regression: Definition

Definition

A Gaussian process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution $\mathcal{N}$.

Let us assume that the elements $\boldsymbol{f}$ of a function f have a multivariant Gaussian distribution. The joint probability density is given as

$$p(\boldsymbol{f}; \boldsymbol{m}, \boldsymbol{K}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{K}|^{1/2}} \exp\left(-\frac{1}{2} (\boldsymbol{f} - \boldsymbol{m}) \boldsymbol{K}^{-1} (\boldsymbol{f} - \boldsymbol{m})\right) \tag{11}$$

for which we use the short hand notation

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K}) \tag{12}$$

Often a constant or zero mean function $\boldsymbol{m} = (m(\boldsymbol{x_1}), m(\boldsymbol{x_2}), ..., m(\boldsymbol{x_n}))$ is used and $\boldsymbol{K}$ is the co-variance matrix.

Gaussian Process regression: Definition

Having set the initial mean function to zero or a constant value the only thing which defines the GP is the co-variance function/ matrix:

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_i, \mathbf{x}_i) & k(\mathbf{x}_i, \mathbf{x}_j) & \cdots & k(\mathbf{x}_i, \mathbf{x}_n) \\ k(\mathbf{x}_j, \mathbf{x}_i) & k(\mathbf{x}_j, \mathbf{x}_j) & \cdots & k(\mathbf{x}_j, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_i) & k(\mathbf{x}_n, \mathbf{x}_j) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \tag{13}$$

which is constructed from kernel functions.

➡ The kernel functions describe how close the input data is.

Gaussian Process regression: Definition

In order to make predictions it is exploited that the training data $\boldsymbol{f}$ and the unknown data $\boldsymbol{f}_* = \left(f(\boldsymbol{x}_1^*), f(\boldsymbol{x}_2^*), ..., f(\boldsymbol{x}_n^*)\right)$ is distributed according to

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m} \\ \boldsymbol{m}_* \end{bmatrix}, \begin{bmatrix} K(\boldsymbol{X}, \boldsymbol{X}) & K(\boldsymbol{X}, \boldsymbol{X}_*) \\ K(\boldsymbol{X}_*, \boldsymbol{X}) & K(\boldsymbol{X}_*, \boldsymbol{X}_*) \end{bmatrix}\right) \quad , \tag{14}$$

By conditioning the joint Gaussian prior distribution on the observations one obtains the Bayesian conditional probability (or posterior distribution)

$$p(\boldsymbol{f}_* | \boldsymbol{f}; \boldsymbol{m}, \boldsymbol{K}) = \frac{p(\boldsymbol{f}, \boldsymbol{f}_*; \boldsymbol{m}, \boldsymbol{K})}{\int p(\boldsymbol{f}, \boldsymbol{f}_*; \boldsymbol{m}, \boldsymbol{K}) d\boldsymbol{f}_*} = \frac{p(\boldsymbol{f}, \boldsymbol{f}_*; \boldsymbol{m}, \boldsymbol{K})}{p_1(\boldsymbol{f}; \boldsymbol{m}, \boldsymbol{K})} \tag{15}$$

- The term in the nominator is the joint probability density.
- The term in the denominator is the marginal likelihood.

Gaussian Process regression: How to make Predictions

Or in short form

$$\boldsymbol{f}_* | \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{16}$$

with new mean and co-variance function

$$\boldsymbol{\mu} = \boldsymbol{m}_* + K(\boldsymbol{X}_*, \boldsymbol{X}) K(\boldsymbol{X}, \boldsymbol{X})^{-1} (\boldsymbol{f} - \boldsymbol{m}) \tag{17}$$

$$\boldsymbol{\Sigma} = K(\boldsymbol{X}_*, \boldsymbol{X}_*) - K(\boldsymbol{X}_*, \boldsymbol{X}) K(\boldsymbol{X}, \boldsymbol{X})^{-1} K(\boldsymbol{X}, \boldsymbol{X}_*) \tag{18}$$

which can be used to make predictions for not observed data. Our estimate for the function is

$$f(\boldsymbol{x}_i) = m_i + \sum_{j=1}^{N_{train}} \omega_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{19}$$

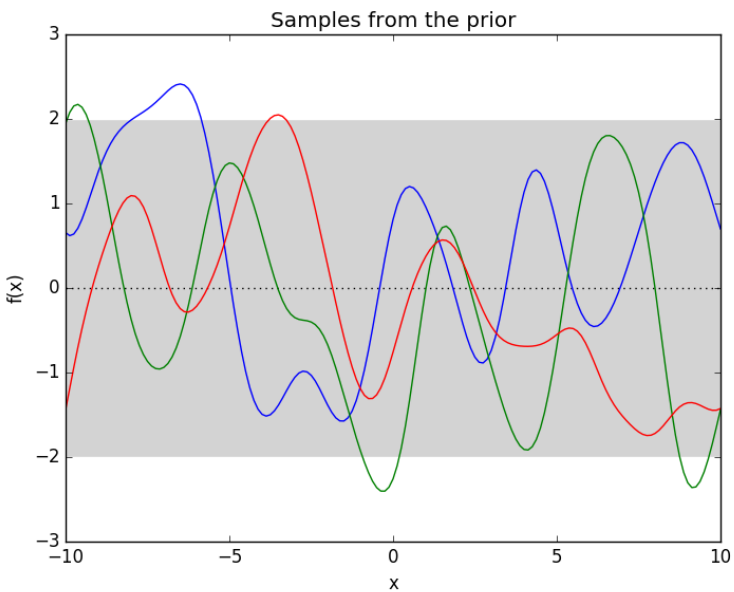
with

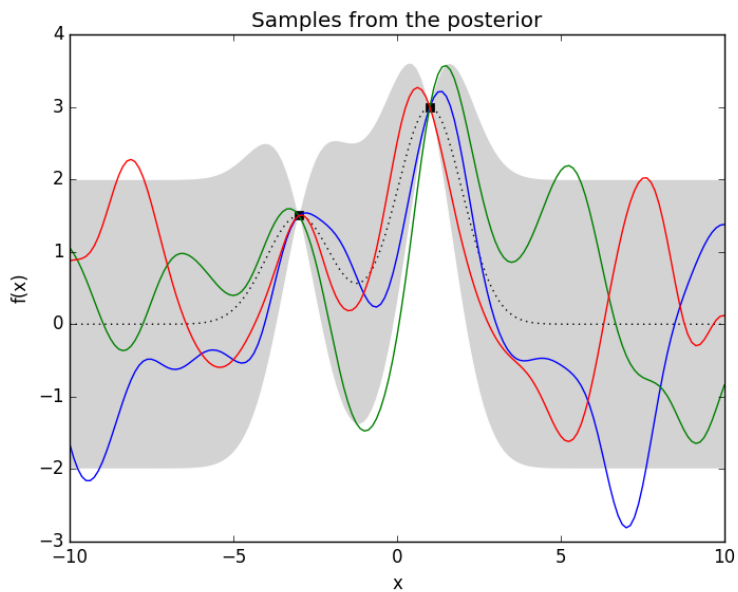
$$\boldsymbol{\omega} = \boldsymbol{K}^{-1} (\boldsymbol{f} - \boldsymbol{m}) \tag{20}$$

Gaussian Process regression: How to make Predictions

An example:

- The prior distribution has zero mean for $f(x)$ and same variance everywhere.
- If data is observed the posterior distribution is obtained
  - ... which has a different mean and variance.
  - ... we don't predict one value!
  - ... we predict a distribution and can only make statements on the average.





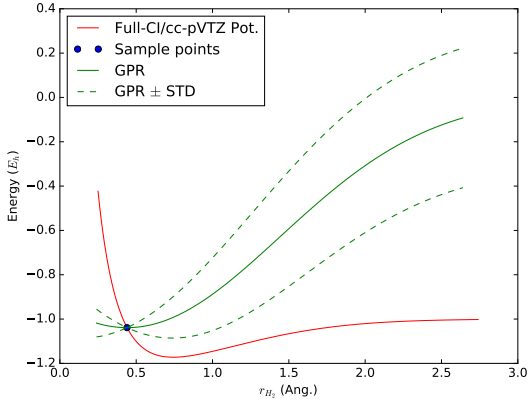
Gaussian Process regression: Adding noise

Often measurements or even calculations carry an error bar or some features of the function are more important than others. This can be achieved by adding noise to the GP in the co-variance matrix:

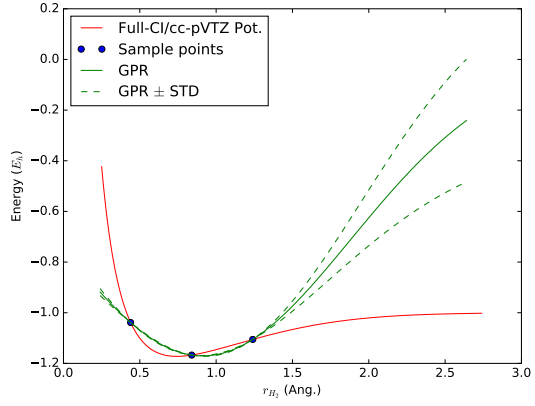
$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K} + \sigma \boldsymbol{I}) \tag{21}$$

- This can also be seen as a regularization.
- The GPR is not forced anymore to go exactly through the data points. This can give some flexibility to make the function actually smoother.

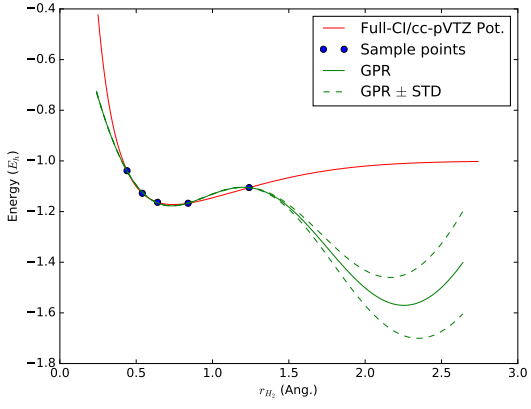
Gaussian Process regression: An example



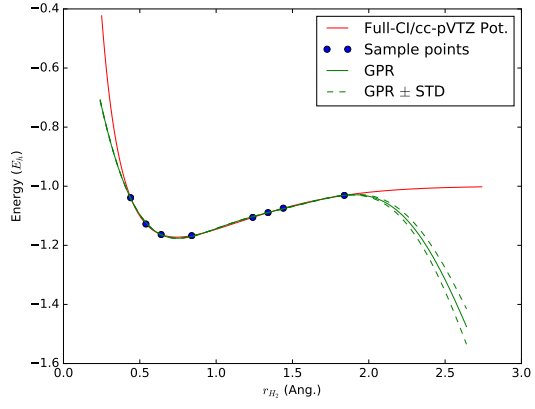
GPR with 1 point



GPR with 3 points



GPR with 5 points



GPR with 8 points

Gaussian Process regression: Kernels & Hyper parameters

Let's recall the kernel function like the squared exponential kernel:

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{22}$$

- It depends on the hyper parameters σ and l.
- They are not meaningless for the performance of the predictor:
  - σ^2: average distance of function away from mean.
  - l: length of the 'wiggles'. Max. possible to extrapolate l units away from the data.
  - Chose hyper parameters wisely or optimize them!

Gaussian Process regression: Kernels & Hyper parameters

There are different kernel functions for different purposes:

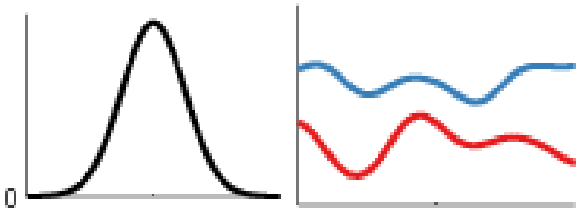


Figure: Squared exponential kernel

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \qquad (23)$$

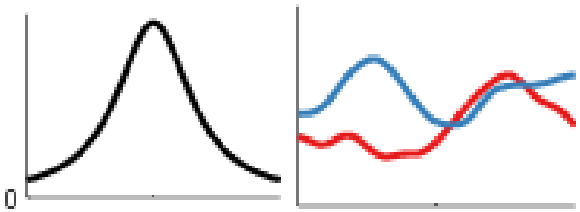


Figure: Rational Quadratic Kernel

$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha l^2}\right)^{-\alpha} \qquad (24)$$

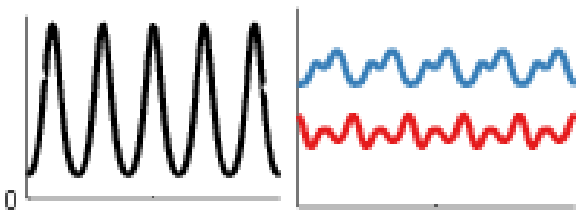
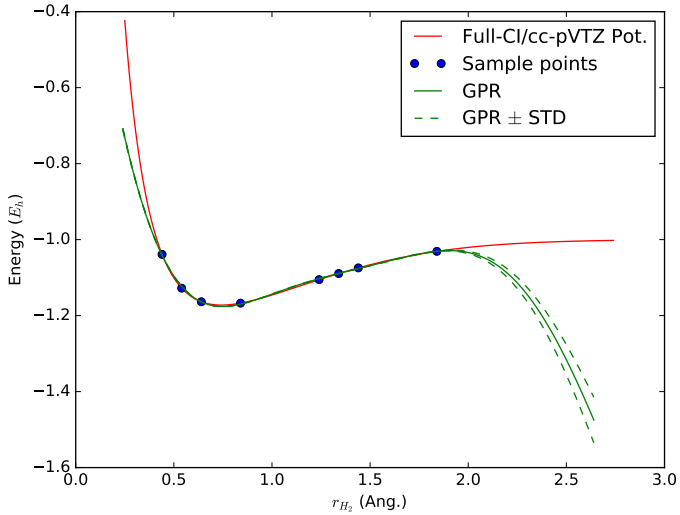


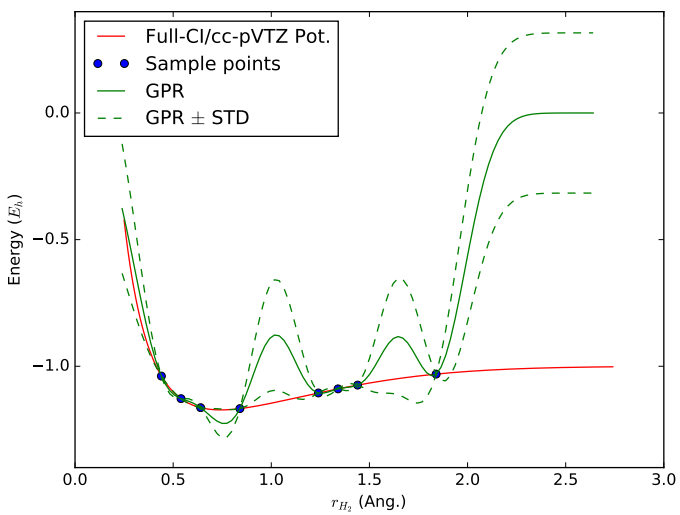
Figure: Periodic Kernel

$$k_{\mathrm{Rer}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi\,|x - x'|\,/p)}{l^2}\right) \qquad (25)$$

Gaussian Process regression: Hyper parameters



$l = 1$



$l = 0.15$

Figure: Example of the influence of the choice of the length scale parameter l for the squared exponential kernel when representing the potential for H_2. The red curves is the actual function. The green curve the GPR representation and the green doted lines illustrate the confidence interval.

Gaussian Process regression: Optimizing hyper parameters

The hyper parameters θ have to be chosen wisely or can be tuned (in GPR):

Maximize the probability of the posterior distribution:

$$p(\boldsymbol{f}_*|\boldsymbol{f};\boldsymbol{m},\boldsymbol{K}) = \frac{p(\boldsymbol{f},\boldsymbol{f}_*;\boldsymbol{m},\boldsymbol{K})}{\int p(\boldsymbol{f},\boldsymbol{f}_*;\boldsymbol{m},\boldsymbol{K})d\boldsymbol{f}_*} = \frac{p(\boldsymbol{f},\boldsymbol{f}_*;\boldsymbol{m},\boldsymbol{K})}{p_1(\boldsymbol{f};\boldsymbol{m},\boldsymbol{K})} \tag{26}$$

➡ Maximize the log of the marginal likelihood...

$$\log p_1(\boldsymbol{f};\boldsymbol{m},\boldsymbol{K};\theta) = -\frac{1}{2}(\boldsymbol{f}-\boldsymbol{m})^T \boldsymbol{K}^{-1}(\boldsymbol{f}-\boldsymbol{m}) - \frac{1}{2}\log|\boldsymbol{K}| - \frac{n}{2}\log 2\pi \tag{27}$$

- The dominant cost for the evaluation is the matrix inversion.
- Using SVD or Cholesky decomposition the log of the determinant is easily accessible.

Gaussian Process regression: Optimizing hyper parameters

What about derivatives for

$$\log p_1(\boldsymbol{f}; \boldsymbol{m}, \boldsymbol{K}; \theta) = -\frac{1}{2}(\boldsymbol{f} - \boldsymbol{m})^T \boldsymbol{K}^{-1}(\boldsymbol{f} - \boldsymbol{m}) - \frac{1}{2}\log|\boldsymbol{K}| - \frac{n}{2}\log 2\pi \tag{28}$$

???

A simple equation for the partial derivatives for the hyper parameters

$$\frac{\partial}{\partial \theta_i}\log p_1(\boldsymbol{f}; \boldsymbol{m}, \boldsymbol{K}; \theta) = \frac{1}{2}\mathrm{tr}\left(\left(\boldsymbol{\omega}\boldsymbol{\omega}^{\mathrm{T}} - \boldsymbol{K}^{-1}\right)\frac{\partial \boldsymbol{K}}{\partial \theta_{\mathrm{i}}}\right) \quad , \tag{29}$$

where $\boldsymbol{\omega} = \boldsymbol{K}^{-1}(\boldsymbol{f} - \boldsymbol{m})$ can be derived given rise to gradient based optimization algorithms

➡ In a nutshell the GP **learns** how the hyper parameters look like.

Gaussian Process regression: Optimizing hyper parameters

The hyper parameters can also be optimized using grid search and cross validation.

Grid search:

- For each hyper parameter θ_i reasonable values are preselected e.g. $\theta_1 \in \{10, 100, 1000\}$, $\theta_2 \in \{0.1, 0.2, 0.5, 1.0\}$
- Each combination is tested and assessed using cross validation.

Cross validation:

- The training data is partitioned in k subsets.
- The model is trained for each subset.
- For each run the error is calculated using the other $k-1$ subsets.
- Of course there are **many** ways to do this and here we just life with the basic idea.

Gaussian Process Regression: A small outlook

We (succesfully?) used GPR in geoemtry optimizations, which don't have analytical derivatives. However, we use Δ-Learning:

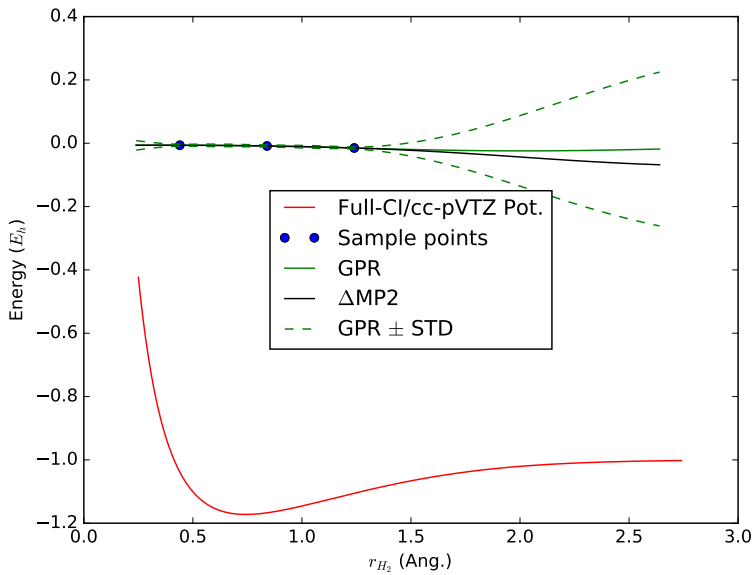


Figure: Illustration of delta learning.

- We don't train the GPR with the total energy, but keep a large fraction exact (either HF or MP2).
- The GPR is trained on the difference, which is more constant → less points are required.

Gaussian Process Regression: A small outlook

Table: Number of performed SPs until convergence for RIPLEY and the conventional numerical gradient (NUMGRAD).

	GPR			
	(HF,CCSD(F12*)(T))	(MP2,CCSD(F12*)(T))	NUMGRAD	dim
NH_3	22	19	78	6
CH_3COF	82	47	124	15
CH_3COOH	44	47	185	18
CH_3NH_2	40	38	155	15
C_2H_6	43	43	222	18
C_2H_5OH	134	54	344	21
H_2CO	21	19	65	6
Hydrosulphane	136	42	208	6
SiH_4	20	20	57	9
transbutadiene	54	53	196	24
transCHFCHF	33	31	125	12
H_2O	15	14	42	3
Furan	108	67	387	21

Kernel Ridge Regression

One other popular kernel based method is **Kernel Ridge Regression** (KRR).
Although different background: Final working equations are similar (identical) to GPR.

We approximate the function as

$$f(\boldsymbol{x}_i) = \sum_{j=1}^{N_{train}} \omega_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{30}$$

and determine the weights by minimizing

$$\min_{\omega} \sum_{j}^{N_{train}} \left[f(\boldsymbol{x}_j) - f^{ref}(\boldsymbol{x}_j) \right] + \lambda \boldsymbol{\omega}^T \boldsymbol{K} \boldsymbol{\omega} \tag{31}$$

which has the analytical solution

$$\boldsymbol{\omega} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{f}^{ref} \tag{32}$$

If same kernel is used and λ equals the noise in the GPR $\rightarrow$ same prediction.

Kernel Ridge Regression

However KRR is missing some nice properties of GPR:

- It has no statistic confidence interval.
- It has no intrinsic procedure to optimize the hyper parameters and one needs to rely on grid search and cross validation.
- Therefore KRR and GPR usually give not the same results!
- In my very personal view GPR has more advantages.

Outlook and Conclusion

- We learned what the kernel trick is and applied to different algorithms.
- We saw how to use Supporting Vector Machines for binary classification
- We studied Gaussian Process regression in detail and will hopefully it apply successfully in fitting potential energy surfaces.
- We compared GPR to kernel ridge regression.

Some useful things

- For information on Gaussian Processes go to http://www.gaussianprocess.org/.
- If you want to learn more on SVM: http://www.robots.ox.ac.uk/ az/lectures/ml/.
- There are good python implementations for the discussed methods (e.g. scikit-learn).

Thank you for your attention

Image: CC Licence, Steve H. Link: www.flickr.com/photos/desteve/5824438515/