Advanced Topics in Statistical Machine Learning

Kernel Methods for Regression

Oliver Bamford

July 1st 2018

M.Sc. in Mathematical Modelling and Scientific Computing

Contents

1 Introduction

Consider some data given by

$$(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, \quad i = 1, \ldots, n, \tag{1}$$

where the sets $\mathcal{X}$ and $\mathcal{Y}$ contain measurements and response variables, respectively. The objective of supervised machine learning is to identify some mathematical relationship between the measurements $\{\boldsymbol{x}_i\}_{i=1}^n$ and the response variables $\{y_i\}_{i=1}^n$. The exact structure of the set $\mathcal{Y}$ defines which of the two main groups of supervised learning a model fits into: if $\mathcal{Y}$ is a discrete set such as $\{-1, +1\}$ or $\{C_1, \ldots, C_k\}$, $k \in \mathbb{N}$, we are seeking *classification*, whereas if $\mathcal{Y}$ is some continuous set, such as the set of real numbers, we are looking at *regression*. More succinctly, we wish to learn a function

$$f : \mathcal{X} \to \mathcal{Y} \tag{2}$$

which can be used to accurately predict a response given new measurements. In the case where the relationship between measurements and their responses is linear, we can make accurate predictions using simple linear models. But what if the underlying relationship is non-linear?

In this report, we will explain 'the kernel trick', which allows us to learn non-linear functions using various linear models. Then, we will apply the kernel trick to three such models and apply the resulting kernel methods to some non-linear data.

2 The Kernel Trick

This section follows Sec. 2 of [2], with additional explanation and details added from other sources.

Underpinning all kernel methods is 'the kernel trick', which enables us to identify non-linear patterns in data using an entirely implicit transformation into a higher (possibly infinitely) dimensional space.

Consider the problem described in Sec. 1. As a basic example, imagine some empirical, possibly noisy, one-dimensional data which appears quadratic, i.e. $y \approx x^2$. In this case, linear regression analysis will not produce any accurate predictions. However, if we define a function $\Phi(x) = x^2$ and apply it to our measurements, we can apply linear regression in Φ-space to correctly model the data. This idea can be generalised to data which must be mapped to a higher, possibly infinitely dimensional

space to show an approximately-linear relationship. We formalise this concept by defining the *feature map*:

$$\Phi : \mathcal{X} \to \mathcal{H}, \tag{3}$$

where $\mathcal{H}$ is known as a *feature space*. To a practitioner, this should make alarm bells ring; it was mentioned previously that data may need to be mapped to a space with many or infinite dimensions, so how do we compute this map in practise? The answer is: we don't! For many machine learning problems (some of which will be discussed in this report), the measurements only appear as part of inner products with each other. This means that rather than computing $\Phi(x)$, we need only to compute

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle, \tag{4}$$

where $k(\cdot, \cdot)$ is known as a *kernel function*, and can be thought of as a similarity measure of the elements of $\mathcal{X}$. The brackets $\langle \cdot, \cdot \rangle$ represent an inner product in the feature space $\mathcal{H}$. Thus, by defining a kernel function

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \tag{5}$$

we implicitly define a feature map Φ without ever having to compute $\Phi(x)$, or even know the explicit form of Φ. Then, we may use Equation (4) in our supervised learning models, allowing us to work in potentially infinite dimensional feature spaces without having to explicitly transform our measurements. This substitution is known as the 'kernel trick'.

2.1 Reproducing Kernel Hilbert Spaces

The statement of Equation (4) imposes the condition that the kernel function $k(\cdot, \cdot)$ must be a valid inner product in the space $\mathcal{H}$. The following lemma from Sec. 16.2.2 of [8] gives a necessary and sufficient condition for this to be the case:

Lemma 1. *A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implements an inner product in some Hilbert space if and only if it is positive semi-definite; namely, for all $x_1, \ldots, x_n$, the Gram matrix $K_{ij} = k(x_i, x_j)$, is a positive semi-definite matrix.*

Proof. The positive semi-definiteness of the Gram matrix follows directly from the positive-definite property of an inner product. To prove the other direction, we first define the space of functions over $\mathcal{X}$ as $\mathbb{R}^{\mathcal{X}} = \{f : x \to \mathbb{R}\}$. Then, we define

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}} \quad \text{where} \quad x \mapsto k(\cdot, x), \tag{6}$$

so that $\Phi(x) = k(\cdot, x)$ assigns a value $k(x', x)$ to $x' \in \mathcal{X}$. Thus, applying Φ to a measurement x effectively represents it by its similarity to all other points in $\mathcal{X}$ [6]. In order to define an inner product space with our transformed measurements $\Phi(x_i)$, we first define a corresponding vector space by taking linear combinations of the elements $k(\cdot, x_i)$:

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i), \quad g(\cdot) = \sum_{j=1}^{n'} \beta_j k(\cdot, x'_j), \tag{7}$$

for any $\alpha_j, \beta_j \in \mathbb{R}$, $n, n' \in \mathbb{N}$ and $x_i, x'_j \in \mathcal{X}$. An inner product over this space can then be defined as

$$\langle f, g \rangle = \left\langle \sum_{i=1}^{n} \alpha_i k(\cdot, x_i), \sum_{j=1}^{n'} \beta_j k(\cdot, x'_j) \right\rangle = \sum_{i=1}^{n} \sum_{j=1}^{n'} \alpha_i \beta_j k(x_i, x'_j), \tag{8}$$

This object is both symmetric (because k is symmetric) and linear. Furthermore, it is positive definite because $k(x, x) \geq 0$ with equality only occurring if $\Phi(x)$ is the zero function. Thus, it is a valid inner product. By using Equation (8) and taking the sums and constants outside of the inner product, we can derive the relations

$$\langle k(\cdot, x), f \rangle = f(x) \quad \text{and} \quad \langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x'), \tag{9}$$

the latter of which allows us to conclude

$$\langle \Phi(x), \Phi(x') \rangle = \langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x'). \tag{10}$$

□

The properties given by Equations (9) define a *reproducing kernel*, which after completing the space of functions given by Equation (7) completes our construction of the *reproducing kernel Hilbert space* (RKHS), $\mathcal{H}_k$. It is shown in [1] that every positive semi-definite kernel defined on $\mathcal{X} \times \mathcal{X}$ has a unique RKHS and vice versa.

From a practitioners point of view, the important point to take away from this section is that any kernel function which has a symmetric, positive semi-definite Gram matrix defines an inner product in an RKHS. Thus, it can be used (via the kernel trick) to compute inner products in high dimensional spaces with low cost. Some problem-specific considerations which need to be made when constructing kernel function are explained in Sec. 7.

3 Introduction to Regression

We now take the general problem described in Sec. 1, and specify the spaces $\mathcal{X}$ and $\mathcal{Y}$ to define a *regression* problem:

Given n measurements of p features, and their response variables:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^n \in \mathbb{R}^p \times \mathbb{R}, \tag{11}$$

learn a function

$$f : \mathbb{R}^p \to \mathbb{R} \tag{12}$$

that accurately predicts new response variables given new measurements.

The most straightforward way to 'solve' this problem is to assume a linear relationship between the response variables y_i and the measurements $\boldsymbol{x}_i$, i.e. $y_i = \boldsymbol{x}_i^T \boldsymbol{w}$ for some *weight vector* $\boldsymbol{w} \in \mathbb{R}^p$. Then, we train the model by minimising the sum of the errors between the model predictions and the responses over the given measurements, $\left\|\boldsymbol{x}_i^T \boldsymbol{w} - y_i\right\|_2^2$, for all $i = 1, \ldots, n$. Using matrix notation, this means solving

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in \mathbb{R}^p} \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2, \tag{13}$$

where the *design matrix* $X \in \mathbb{R}^{n \times p}$ has the measurements $\{\boldsymbol{x}_i^T\}_{i=1}^n$ as its rows. This model is commonly generalised to data which do not intercept the origin by augmenting X with a column of ones, and increasing the length of $\boldsymbol{w}$ by one. Given a new measurement $\hat{\boldsymbol{x}}$, we may then predict the corresponding response using $\hat{y} = \hat{\boldsymbol{x}}^T \boldsymbol{w}^*$. This method is known as *ordinary least squares* (OLS) regression, and has the closed form solution

$$\boldsymbol{w}^* = (X^T X)^{-1} X^T \boldsymbol{y}. \tag{14}$$

Though having a closed form solution makes OLS straightforward to compute, the assumption of linearity will not hold in many cases, leading to inaccurate predictions. Furthermore, the matrix $X^T X$ may be singular or very poorly conditioned, corresponding to the case where two or more of the features are (nearly) linearly dependent. We alleviate this problem by introducing a Tikhonov regularisation term:

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in \mathbb{R}^p} \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2, \tag{15}$$

where $\lambda \in \mathbb{R}$ is known as a *regularisation* parameter. This problem is known as *ridge regression*, and has the closed form solution

$$\boldsymbol{w}^* = (X^T X + \lambda \mathbb{1})^{-1} X^T \boldsymbol{y}, \tag{16}$$

where $\mathbb{1}$ is the identity matrix. One can see that the effect of the regularisation term is to bound the minimal eigenvalue of the matrix being inverted away from zero, leading to a better condition number than $X^T X$.

4 Kernel Ridge Regression

Now that we are acquainted with linear regression models, it is time to learn some methods for predicting non-linear patterns in data. To this end, the first method we introduce is *kernel ridge regression* (KRR). As the name suggests, this model combines ridge regression with the kernel trick.

We begin by transforming our measurements into a feature space $\mathcal{H}$, using the feature map $\Phi : \mathbb{R}^p \to \mathcal{H}$. Applying this to our ridge regression model, Equation (15), gives

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}\in\mathcal{H}} \left(\sum_{i=1}^{n} \left(\langle \boldsymbol{w}, \Phi(\boldsymbol{x}_i) \rangle_{\mathcal{H}} - y_i \right)^2 + \lambda \|\boldsymbol{w}\|_{\mathcal{H}}^2 \right). \tag{17}$$

If we now specify a positive definite kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$, a RKHS is uniquely defined, allowing us to write:

$$f^* = \arg\min_{f\in\mathcal{H}_k} \left(\sum_{i=1}^{n} \left(\langle f, k(\cdot, \boldsymbol{x}_i) \rangle_{\mathcal{H}_k} - y_i \right)^2 + \lambda \|f\|_{\mathcal{H}_k}^2 \right), \tag{18}$$

where $f : \|\boldsymbol{w}\|_{\mathcal{H}} = \|f\|_{\mathcal{H}_k}$. Using the reproducing property of $k(\cdot, \boldsymbol{x})$, we may rewrite this as

$$f^* = \arg\min_{f\in\mathcal{H}_k} \left(\sum_{i=1}^{n} \left(f(\boldsymbol{x}_i) - y_i \right)^2 + \lambda \|f\|_{\mathcal{H}_k}^2 \right). \tag{19}$$

In this form, we see another interpretation of the regularisation term as a device to restrict the space from which we find minimising functions f. This is often necessary, as searching for functions in an infinitely large space can result in finding highly complex minimisers which fit training data very accurately, if not perfectly, but do not correctly learn the underlying relationship. This results in inaccurate predictions, and is known as *overfitting*.

To solve this minimisation problem, we make use of the following theorem from [6]:

Theorem 2 (Representer theorem)**.** *The minimisation problem*

$$\min_{f\in\mathcal{H}_k} \hat{R}(f) + g(\|f\|_{\mathcal{H}_k}), \tag{20}$$

where $\hat{R}$ is some risk functional and g is a strictly monotonically increasing function, has a solution of the form

$$f^*(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i). \tag{21}$$

Proof. Given $x_1, \ldots, x_n \in \mathcal{X}$, we may express any $f \in \mathcal{H}_k$ as the sum of a part which lies on the span of $\Phi(x_i) = k(\cdot, x_i)$ and a part which lies orthogonal to it, i.e.

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) + f_\perp, \tag{22}$$

for some $\alpha_i \in \mathbb{R}$ and $\langle f_\perp, k(\cdot, x_i)\rangle_{\mathcal{H}_k} = 0$ for all $i = 1, \ldots, n$. Thus, applying f to an arbitrary measurement x_j using Equation (9) gives

$$f(x_i) = \left\langle \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) + f_\perp, k(\cdot, x_j) \right\rangle_{\mathcal{H}_k} = \sum_{i=1}^{n} \alpha_i k(x_i, x_j), \tag{23}$$

which is independent of the orthogonal part, $f_\perp$. Thus, $\hat{R}(f)$ is independent of $f_\perp$. The regulariser term is given by

$$g\left(\|f\|_{\mathcal{H}_k}\right) = g\left(\left\langle \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) + f_\perp, \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) + f_\perp \right\rangle_{\mathcal{H}_k}\right) \tag{24}$$

$$= g\left(\sqrt{\left\| \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \right\|_{\mathcal{H}_k}^2 + \|f_\perp\|_{\mathcal{H}_k}^2}\right). \tag{25}$$

As g is a strictly monotonically increasing function, we may add that

$$g\left(\|f\|_{\mathcal{H}_k}\right) \geq g\left(\left\| \sum_{i=1}^{n} \alpha_i k(\cdot, x_i) \right\|_{\mathcal{H}_k}\right). \tag{26}$$

Thus, we have that $\hat{R}(f)$ is independent of $f_\perp$, and g is strictly increasing in $f_\perp$. Hence, any minimiser of Equation (20) must have $f_\perp = 0$. □

Substituting Equation (21) into the objective of the KRR minimisation problem (19) gives

$$\mathcal{J} := \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - y_i \right)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{27}$$

This can be written as the following matrix equation:

$$\mathcal{J} = \|K\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\boldsymbol{\alpha}^T K\boldsymbol{\alpha} \tag{28}$$

$$= \boldsymbol{\alpha}^T K^2\boldsymbol{\alpha} + \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T K\boldsymbol{\alpha} + \lambda\boldsymbol{\alpha}^T K\boldsymbol{\alpha} \tag{29}$$

$$= \boldsymbol{\alpha}^T(K + \lambda\mathbb{1})K\boldsymbol{\alpha} - 2\boldsymbol{y}^T K\boldsymbol{\alpha} + \boldsymbol{y}^T\boldsymbol{y}. \tag{30}$$

where $(K)_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, for $i, j = 1, \ldots, n$ and $\boldsymbol{\alpha}, \boldsymbol{y} \in \mathbb{R}^n$. As $\mathcal{J}$ is a quadratic equation with a positive definite Hessian matrix, differentiating it with respect to $\boldsymbol{\alpha}$ and setting it to zero gives us the global minimum:

$$\boldsymbol{\alpha}^* = (K + \lambda\mathbb{1})^{-1}\boldsymbol{y}. \tag{31}$$

Thus, after defining a suitable kernel function, we may find the coefficients $\boldsymbol{\alpha}^*$ using the training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. Then, given a new measurement $\hat{\boldsymbol{x}}$ we have that

$$\hat{y} = f^*(\hat{\boldsymbol{x}}) = \sum_{i=1}^{n} \alpha_i^* k(\hat{\boldsymbol{x}}, \boldsymbol{x}_i). \tag{32}$$

Or, for $\hat{n}$ new measurements $\hat{X} \in \mathbb{R}^{\hat{n}\times p}$:

$$\hat{\boldsymbol{y}} = K_{\hat{X}X}\boldsymbol{\alpha}^*, \tag{33}$$

where $K_{\hat{X}X} \in \mathbb{R}^{\hat{n}\times n} : (K_{\hat{X}X})_{ij} = k(\hat{\boldsymbol{x}}_i, \boldsymbol{x}_j)$.

5 Support Vector Regression

This section follows the first two sections of [9]

Though support vector machines are more widely known for their use in classification problems, a similar method can be used for regression. Following the same procedure as in Sec. 4, we begin by constructing a linear model, and then apply the kernel trick to extend our model to non-linear patterns.

5.1 Linear Support Vector Regression

We begin by restricting our search to functions of the form

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}. \tag{34}$$

Note that while we could include a constant term explicitly in f to account for functions which do not intercept the origin, the same outcome can be achieved (as

with OLS) by augmenting each of our measurements $\boldsymbol{x}_i$ with a one. Unlike in OLS, where we seek to minimise the difference between our prediction and the known values, here we seek to find the *flattest* line which has an error lower than some maximum, ε. This means we must solve the following constrained optimisation problem:

$$\begin{aligned} &\min_{\boldsymbol{w}\in\mathbb{R}^p} \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t. } & y_i - \boldsymbol{w}^T\boldsymbol{x}_i \leq \varepsilon, \\ & \boldsymbol{w}^T\boldsymbol{x}_i - y_i \leq \varepsilon. \end{aligned} \tag{35}$$

The problem with this formulation is that it may have an empty feasible region, making it unsolvable. We fix this by introducing two new variables, $\xi_i^+ > 0$ and $\xi_i^- > 0$, which relax the constraints:

$$\begin{aligned} &\min_{\substack{\boldsymbol{w}\in\mathbb{R}^p \\ \boldsymbol{\xi}^\pm\in\mathbb{R}^n}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^n(\xi_i^+ + \xi_i^-) \\ \text{s.t. } & y_i - \boldsymbol{w}^T\boldsymbol{x}_i \leq \varepsilon + \xi_i^+, \\ & \boldsymbol{w}^T\boldsymbol{x}_i - y_i \leq \varepsilon + \xi_i^-, \\ & \xi_i^+, \xi_i^- \geq 0. \end{aligned} \tag{36}$$

This allows our linear model to differ from the known values by more than ε, but seeks to minimise the amount we 'go over' by. As the constant $C \in \mathbb{R}$ tends to infinity, we retrieve the original optimisation problem (35).

As is standard in constrained optimisation problems, we form an unconstrained problem by forming a *Lagrangian*:

$$\begin{aligned} \mathcal{L} := \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^n(\xi_i^+ + \xi_i^-) - \sum_{i=1}^n(\mu_i^+\xi_i^+ + \mu_i^-\xi_i^-) \\ - \sum_{i=1}^n \lambda_i^+(\varepsilon + \xi_i^+ - y_i + \boldsymbol{w}^T\boldsymbol{x}_i) - \sum_{i=1}^n \lambda_i^-(\varepsilon + \xi_i^- + y_i - \boldsymbol{w}^T\boldsymbol{x}_i), \end{aligned} \tag{37}$$

where $\mu_i^+, \mu_i^-, \lambda_i^+, \lambda_i^- > 0$ are Lagrange multipliers. We know that necessary conditions for a minimum are given by:

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} = \boldsymbol{w} - \sum_{i=1}^n(\lambda_i^+ - \lambda_i^-)\boldsymbol{x}_i = 0, \tag{38}$$

$$\frac{\partial\mathcal{L}}{\partial\xi_i^\pm} = C - \mu_i^\pm - \lambda_i^\pm = 0. \tag{39}$$

Using Equation (39) we can eliminate the Lagrangian multipliers $\mu_i^{\pm}$ from our Lagrangian (this also imposes an upper bound of C on $\lambda_i^{\pm}$). Then, substituting Equation (38) into the the Lagrangian for $\boldsymbol{w}$ brings us to the *dual optimisation problem*:

$$\begin{aligned}\max_{\boldsymbol{\lambda}^{\pm}\in\mathbb{R}^n} \Big[&-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i^+ - \lambda_i^-)(\lambda_j^+ - \lambda_j^-)\boldsymbol{x}_i^T\boldsymbol{x}_j \\ &- \varepsilon\sum_{i=1}^{n}(\lambda_i^+ + \lambda_i^-) + \sum_{i=1}^{n} y_i(\lambda_i^+ - \lambda_i^-)\Big] \\ &\text{s.t. } \lambda_i^+, \lambda_i^- \in [0, C].\end{aligned} \tag{40}$$

Note that the dual problem is conventionally posed as a maximisation, but we could easily multiply the objective by -1 to get a minimisation problem. By combining Equations (34) and (38) we find

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\lambda_i^+ - \lambda_i^-)\boldsymbol{x}_i^T\boldsymbol{x}, \tag{41}$$

which tells us the form that our predictions on future data points will take.

5.2 Application of the Kernel Trick

As in the case of OLS, we have arrived at an optimisation problem which depends on our measurements x_i only through dot products. Thus, after defining a feature map $\Phi : \mathbb{R}^p \to \mathcal{H}$ and replacing the dot products in $\mathbb{R}^p$ with inner products in $\mathcal{H}$, we are ready to apply the kernel trick via Equation (4). This gives us the following dual maximisation problem:

$$\begin{aligned}\max_{\boldsymbol{\lambda}^{\pm}\in\mathbb{R}^n} \Big[&-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\lambda_i^+ - \lambda_i^-)(\lambda_j^+ - \lambda_j^-)k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &- \varepsilon\sum_{i=1}^{n}(\lambda_i^+ + \lambda_i^-) + \sum_{i=1}^{n} y_i(\lambda_i^+ - \lambda_i^-)\Big] \\ &\text{s.t. } \lambda_i^+, \lambda_i^- \in [0, C],\end{aligned} \tag{42}$$

which has a primal solution of the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(\lambda_i^+ - \lambda_i^-)k(\boldsymbol{x}_i, \boldsymbol{x}), \tag{43}$$

for some positive definite kernel function $k(\cdot, \cdot)$. Notice the similarity with the solution found via kernel ridge regression, Equation (32). The process of using support vector

regression thus consists of defining a suitable kernel function, fixing the constant C (which can be thought of in a similar way to the KRR regularisation parameter, λ), and solving the constrained optimisation problem (42). Predictions of future points are then given by Equation (43).

We have from the KKT conditions of our constrained optimisation that the Lagrange multipliers $\lambda_i^{\pm}$ are equal to zero if the primal constraints are satisfied. This means that, in contrast to the KRR objective which is affected by all data points, the SVR objective is affected by only the points which lie outside of the feasible region defined by ε. Furthermore, the systems which need to be solved to employ SVR are often sparse, leading to faster training and prediction than KRR.

6 Gaussian Process Regression

This section follows Sec. 2.1 of [5], with additional explanation and comparison with other methods covered in this report added.

For the previous two methods, we introduced linear regression methods, and applied the kernel trick to extend them to non-linear problems. Here, we take the same approach, but starting with *Bayesian linear regression* to produce a probabilistic model.

6.1 Bayesian Linear Regression

As with ordinary least squares and linear support vector regression, we model our data linearly. However, in the Bayesian approach we add a term accounting for possible noise in our measurements:

$$y_i = f(\boldsymbol{x}_i) + \epsilon_i, \quad f(\boldsymbol{x}_i) = \boldsymbol{w}^T \boldsymbol{x}_i \tag{44}$$

where $\boldsymbol{x}_i, y_i$ and $\boldsymbol{w}$ are as defined before. The noise is modelled by ϵ_i, which is assumed to follow an independent, identically distributed (i.i.d) Gaussian distribution:

$$\epsilon_i \sim \mathcal{N}(\mu, \sigma_n^2), \tag{45}$$

where σ_n^2 is the variance, and μ is the mean, which we take to be zero. From this, we can construct the *likelihood* of $\boldsymbol{w}$, which is the probability density of the responses $\boldsymbol{y}$, given the weights $\boldsymbol{w}$ and the measurements X, i.e. the likelihood that any given $\boldsymbol{w}$

could generate the responses $\boldsymbol{y}$ with measurements X:

$$p(\boldsymbol{y}|X, \boldsymbol{w}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i, \boldsymbol{w}). \tag{46}$$

As ϵ is normally distributed with zero mean, Equation (44) tells us that each y_i is also normally distributed, but with a mean of $\boldsymbol{w}^T\boldsymbol{x}_i$, allowing us to write

$$p(\boldsymbol{y}|X, \boldsymbol{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2}{2\sigma_n^2}\right] \tag{47}$$

$$= \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\sum_{i=1}^{n} \frac{(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2}{2\sigma_n^2}\right] \tag{48}$$

$$= \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{\|\boldsymbol{y} - X\boldsymbol{w}\|_2^2}{2\sigma_n^2}\right]. \tag{49}$$

Thus, we have that the likelihood of $\boldsymbol{w}$ is a joint normal distribution:

$$p(\boldsymbol{y}|X, \boldsymbol{w}) = \mathcal{N}(X\boldsymbol{w}, \sigma_n^2\mathbb{1}), \tag{50}$$

where the mean is now given by the vector $X\boldsymbol{w}$, which represents the mean of each feature $i = 1, \ldots, n$, and the variance is given by the *covariance* matrix $\sigma_n^2\mathbb{1}$, in which each element σ_{ij} represents the variance between the features i and j. The final ingredient we need to model the posterior $p(\boldsymbol{w}|X, \boldsymbol{y})$ is a prior over the weights vector $\boldsymbol{w}$. We assume this is also a multivariate normal distribution with zero mean:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{0}, \Sigma_p), \tag{51}$$

for covariance matrix $\Sigma_p \in \mathbb{R}^{p\times p}$. Our posterior is then given by Bayes' rule:

$$p(\boldsymbol{w}|X, \boldsymbol{y}) = \frac{p(\boldsymbol{y}, X|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}, X)} \tag{52}$$

$$= \frac{p(\boldsymbol{y}, X|X, \boldsymbol{w})p(\boldsymbol{w}|X)}{p(\boldsymbol{y}, X|X)} \tag{53}$$

$$= \frac{p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|X)}, \tag{54}$$

where in the final step, the fact that $\boldsymbol{w}$ is independent of X has been used. The *marginal likelihood* is given by

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})\,\mathrm{d}\boldsymbol{w}\,. \tag{55}$$

From Equation (50) we have

$$p(\boldsymbol{y}|X,\boldsymbol{w})p(\boldsymbol{w}) \propto \exp\left[-\frac{1}{2\sigma_n^2}(\boldsymbol{y}-X\boldsymbol{w})^T(\boldsymbol{y}-X\boldsymbol{w}) - \frac{1}{2}\boldsymbol{w}^T\Sigma_p^{-1}\boldsymbol{w}\right], \tag{56}$$

which, after completing the square, can be written as

$$p(\boldsymbol{y}|X,\boldsymbol{w})p(\boldsymbol{w}) \propto \exp\left[-\frac{1}{2}(\boldsymbol{w}-\bar{\boldsymbol{w}})^T\left(\frac{1}{\sigma_n^2}X^TX + \Sigma_p^{-1}\right)(\boldsymbol{w}-\bar{\boldsymbol{w}})\right], \tag{57}$$

where $\bar{\boldsymbol{w}} = \sigma_n^{-2}(\sigma_n^{-2}X^TX+\Sigma_p^{-1})^{-1}X^T\boldsymbol{y}$. Thus, we have another multivariate Gaussian distribution with mean $\bar{\boldsymbol{w}}$ and covariance A^{-1}:

$$p(\boldsymbol{w}|X,\boldsymbol{y}) = \mathcal{N}\left(\frac{1}{\sigma_n^2}A^{-1}X^T\boldsymbol{y}, A^{-1}\right), \tag{58}$$

where $A = \sigma_n^{-2}X^TX + \Sigma_p^{-1}$. After defining a suitable variance for the noise, σ_n^2, and a suitable covariance matrix for our prior over the weights, Σ_p, this fully defines our predictive model; the resulting lines can be made by drawing weights from $p(\boldsymbol{w}|X,\boldsymbol{y})$. Comparing this to Equation (16), we see that if we choose $\Sigma_p = \mathbb{1}$, which is to assume the weights are independent of each other, the mean of our Bayesian model coincides with the closed form solution of ridge regression with $\lambda = \sigma_n^{-2}$. Given a new measurement $\hat{\boldsymbol{x}}$, we may calculate a predictive distribution for $\hat{f} := f(\hat{\boldsymbol{x}})$ with

$$p(\hat{f}|\hat{\boldsymbol{x}},X,\boldsymbol{y}) = \int p(\hat{y}|\hat{\boldsymbol{x}},\boldsymbol{w})p(\boldsymbol{w}|X,\boldsymbol{y})\,\mathrm{d}\boldsymbol{w} \tag{59}$$

$$= \mathcal{N}\left(\frac{1}{\sigma_n^2}\hat{\boldsymbol{x}}^TA^{-1}X^T\boldsymbol{y}, \hat{\boldsymbol{x}}^TA^{-1}\hat{\boldsymbol{x}}\right). \tag{60}$$

6.2 Application of The Kernel Trick

Now that we have constructed a probabilistic linear regression model, we wish to use the same framework to predict non-linear patterns in data. Analogously to ridge regression and linear support vector regression, we use a feature map $\boldsymbol{\Phi} : \mathbb{R}^p \to \mathcal{H}$ to transform our data to some N-dimensional feature space. For the sake of generality, we consider not one new measurement, but $\hat{n}$ new measurements $\hat{X} \in \mathbb{R}^{\hat{n}\times p}$ for which we wish to predict the response. We let $\Phi(X) = \Phi \in \mathbb{R}^{n\times N}$ and $\Phi(\hat{X}) = \hat{\Phi} \in \mathbb{R}^{\hat{n}\times N}$ represent row-wise operations on the matrices X and $\hat{X}$, then carry out the exact same analysis as in the previous section, leading us to a very similar joint predictive distribution:

$$p(\hat{\boldsymbol{f}}|\hat{X},X,\boldsymbol{y}) = \mathcal{N}\left(\frac{1}{\sigma_n^2}\hat{\Phi}^TA^{-1}\Phi^T\boldsymbol{y}, \hat{\Phi}^TA^{-1}\hat{\Phi}\right), \tag{61}$$

where $A = \sigma_n^{-2}\Phi^T\Phi + \Sigma_p^{-1}$. Note that the covariance matrix Σ_p now has dimensions $N \times N$. We may rewrite the mean of our predictive distribution as:

$$\begin{aligned}
\boldsymbol{\mu}_{GP} &= \hat{\Phi}^T(\Phi^T\Phi + \sigma_n^2\Sigma_p^{-1})^{-1}\Phi^T\boldsymbol{y} && (62)\\
&= \hat{\Phi}^T\Sigma_p(\Phi^T\Phi\Sigma_p + \sigma_n^2\mathbb{1})^{-1}\Phi^T\boldsymbol{y} && (63)\\
&= \hat{\Phi}^T\Sigma_p(\Phi\Sigma_p + \sigma_n^2\Phi^{-T})^{-1}\boldsymbol{y} && (64)\\
&= \hat{\Phi}^T\Sigma_p\Phi^T(\Phi\Sigma_p\Phi^T + \sigma_n^2\mathbb{1})^{-1}\boldsymbol{y}. && (65)
\end{aligned}$$

To rewrite the variance, we use the *matrix inversion lemma*:

$$(Z + UWV^T)^{-1} = Z^{-1} - Z^{-1}U(W^{-1} + V^TZ^{-1}U)^{-1}V^TZ^{-1}, \tag{66}$$

with $Z^{-1} = \Sigma_p$, $W^{-1} = \sigma_n^2\mathbb{1}$, $V = U = \Phi^T$. This gives

$$\begin{aligned}
\boldsymbol{\sigma}_{GP} &= \hat{\Phi}^T(\sigma_n^{-2}\Phi^T\Phi + \Sigma_p^{-1})^{-1}\hat{\Phi} && (67)\\
&= \hat{\Phi}^T\Sigma_p\hat{\Phi} - \hat{\Phi}^T\Sigma_p\Phi^T(\Phi\Sigma_p\Phi^T + \sigma_n^2\mathbb{1})\Phi\Sigma_p\hat{\Phi}. && (68)
\end{aligned}$$

Once again, we are left with equations containing potentially infinite-dimensional vectors and matrices. However, a quick inspection of Equations (65) and (68) reveals that these objects only appear as part of $\Phi\Sigma_p\Phi^T$, $\hat{\Phi}^T\Sigma_p\Phi^T$ and $\hat{\Phi}^T\Sigma_p\hat{\Phi}$. As a covariance matrix is necessarily symmetric and positive definite, Σ_p defines an inner product. Thus, if we define $k(\boldsymbol{x}, \boldsymbol{x}') = \langle\Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}')\rangle_{\Sigma_p} = \Phi(\boldsymbol{x})^T\Sigma_p\Phi(\boldsymbol{x}')$, we have the following useful matrices:

$$\begin{aligned}
K_{XX} &= \Phi\Sigma_p\Phi^T \in \mathbb{R}^{n\times n}, \quad i, j = 1\ldots, n, && (69)\\
K_{X\hat{X}} &= \Phi\Sigma_p\hat{\Phi} \in \mathbb{R}^{n\times\hat{n}}, && (70)\\
K_{\hat{X}X} &= K_{X\hat{X}}^T = \Phi^T\Sigma_p\hat{\Phi}^T \in \mathbb{R}^{\hat{n}\times n}. && (71)
\end{aligned}$$

Once again we have that $k(\cdot,\cdot)$ must be a valid inner product, and must therefore have the same properties as proved in Sec. 2.1. In the context of Gaussian processes, $k(\cdot,\cdot)$ is often referred to as both a kernel function and a covariance function.

Our predictive distribution is finally given by

$$p(\hat{\boldsymbol{f}}|\hat{X}, X, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}_{GP}, \boldsymbol{\sigma}_{GP}), \tag{72}$$

where

$$\boldsymbol{\mu}_{GP} = K_{\hat{X}X}(K_{XX} + \sigma_n^2\mathbb{1})^{-1}\boldsymbol{y}, \tag{73}$$

and

$$\boldsymbol{\sigma}_{GP} = K_{\hat{X}\hat{X}} - K_{XX}(K_{XX} + \sigma_n^2\mathbb{1})K_{X\hat{X}}. \tag{74}$$

As the predictive distribution is normal, we know that the most probable value is the mean, so Equation (73) is used for predicting future data points. We note that this leads to the exact same prediction as kernel ridge regression (see Equations (31) and (32)). However, the Bayesian formulation used in Gaussian processes also leads to a formula for the variance of our predictive distribution, allowing us to easily calculate confidence intervals.

A further advantage that Gaussian process regression holds over KRR and SVR is found in the methods available for selecting hyper-parameters (including the noise variance σ_n^2, and any kernel parameters); continuing the use of Bayesian formalism allows us to use various optimisation methods to vary the hyper-parameters in order to maximise the likelihood of generating the data, i.e. we maximise

$$\log p(\boldsymbol{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T \tilde{K} \boldsymbol{y} - \frac{1}{2}\log|\tilde{K}| - \frac{n}{2}\log 2\pi, \tag{75}$$

where $\tilde{K} = K + \sigma_n^2 \mathbb{1}$, and $\boldsymbol{\theta}$ contains both σ_n and any parameters of the kernel function used. This is in contrast to kernel ridge regression (and many other methods), in which "optimal" hyper-parameters are usually found via grid search. For a good introduction to hyper-parameter estimation for Gaussian processes, see Sec. 3 of [10].

7 Results

In this section, we use the python package scikit-learn [4] to apply and compare the three methods derived in this report, and explain some of the factors which must be considered when constructing a kernel function.

For each method, we use the same $n = 100$ training data, drawn uniformly from $[0, 15]$, and then test our models in the range $[0, 25]$.

7.1 Kernel Ridge Regression

We begin by considering one dimensional noisy data with a quadratic underlying relationship with the response:

$$y = f_p(x) + \epsilon, \quad f_p(x) = 0.1x^2 - x, \tag{76}$$

where ϵ represents random noise. An extremely popular kernel function is the Gaussian radial basis function, or RBF, which is given by

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\gamma^2}\right], \tag{77}$$

for length-scale parameter γ. If we recall that a kernel function represents an inner product in some RKHS, we can see from the Taylor series of an exponential function that the RKHS defined by this kernel function is of infinite dimension. Its ability to view data in an infinite dimensional feature space is very powerful, and the only assumption we make in employing this kernel is that the function we are learning is infinitely smooth. This generality makes the RBF a great go-to kernel for many machine learning problems. Figure 1 (left) shows an attempt to fit our polynomial training data given by Equation (76) using kernel ridge regression with an RBF kernel. We find the optimal kernel parameter to be $\gamma = 10$ and the regularisation parameter

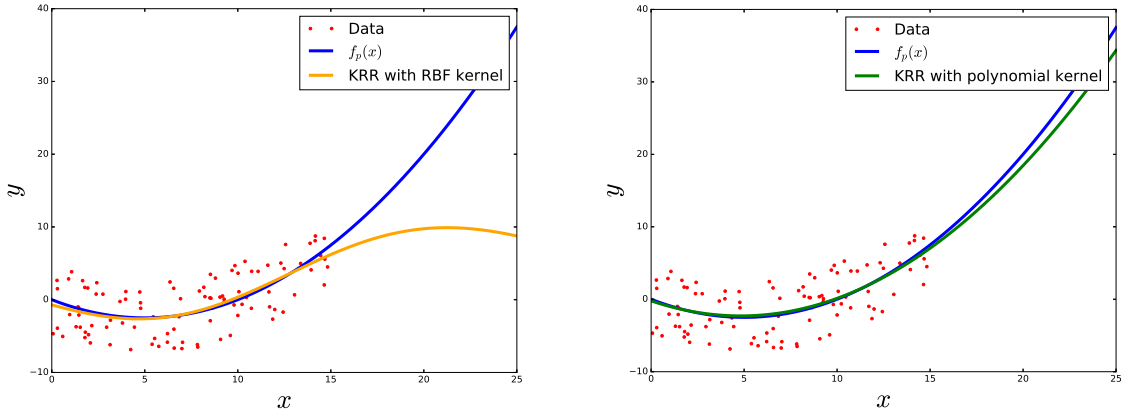

Figure 1: Comparison of prediction of $f_p(x)$ using KRR with an RBF kernel ($\gamma = 10$, $\lambda = 0.1$) to $f_p(x)$ (left). Comparison of prediction of $f_p(x)$ using KRR with a polynomial kernel ($\gamma = 2.15$, $c_0 = 2.5$, $\lambda = 400$) to $f_p(x)$ (right).

to be $\lambda = 0.1$. To do this, we use grid search, which simply tries every combination of different parameters within some given range, and picks the combination which scores the best against some criteria. Grid search thus grows exponentially in computational cost with the number of parameters being optimised over. We see that the function is predicted very accurately where training data is present, but poorly thereafter. This is because KRR constructs the response for each predicted point according to Equation (32), and as the RBF decays rapidly as the distance between two points increases, the predictions will always drop off to zero in the absence of any nearby training data. Thus, to make accurate predictions of test data, we must encode some more prior knowledge of the problem into our choice of kernel. To demonstrate this, we introduce the polynomial kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \left(\gamma \boldsymbol{x}^T \boldsymbol{x}' + c_0\right)^d, \tag{78}$$

where we once again have some more parameters which need to be tuned, γ, c_0 and the degree of the polynomial, d. As we know our function should be quadratic, we can fix $d = 2$ and grid search for the other three parameters, though in many cases it would be wise to include d in the grid search also. Using the polynomial kernel we learn $f_p(x)$ far more accurately, as shown in Figure 1 (right).

We now attempt to learn a periodic function, given by

$$y = f_s(x) + \epsilon, \quad f_s(x) = \sin(x). \tag{79}$$

If we wish to make accurate predictions of future data which follow this pattern, we will need a kernel function which captures the periodic nature of the training data. For this, we introduce the exponential sine squared (ESS) kernel (from [3]):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\frac{2\sin^2(\pi\|\boldsymbol{x} - \boldsymbol{x}'\|/p)}{\gamma^2}\right], \tag{80}$$

where we have a new parameter p, which controls the periodicity of the function. By using this kernel with KRR, and grid search to find the best parameters, we produce Figure 2. This shows the function $f_s(x)$ has been learned rather accurately despite the large amount of noise in the training data.

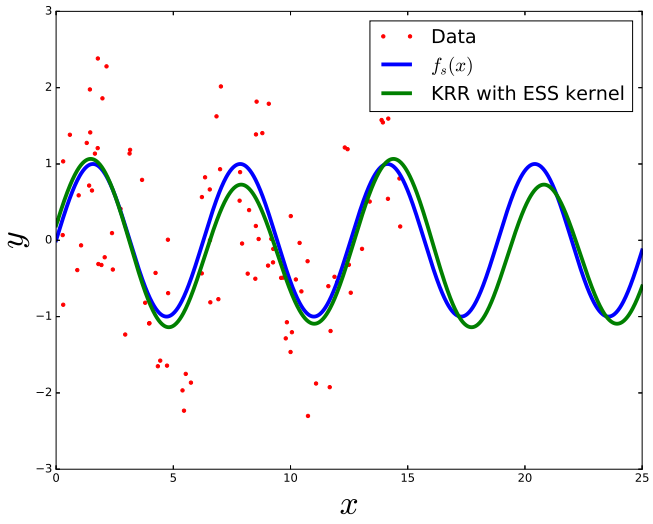

Figure 2: A graph comparing prediction of $f_s(x)$ using KRR with ESS kernel ($\gamma = 4.64$, $p = 12.9$, $\lambda = 0.001$) to $f_s(x)$.

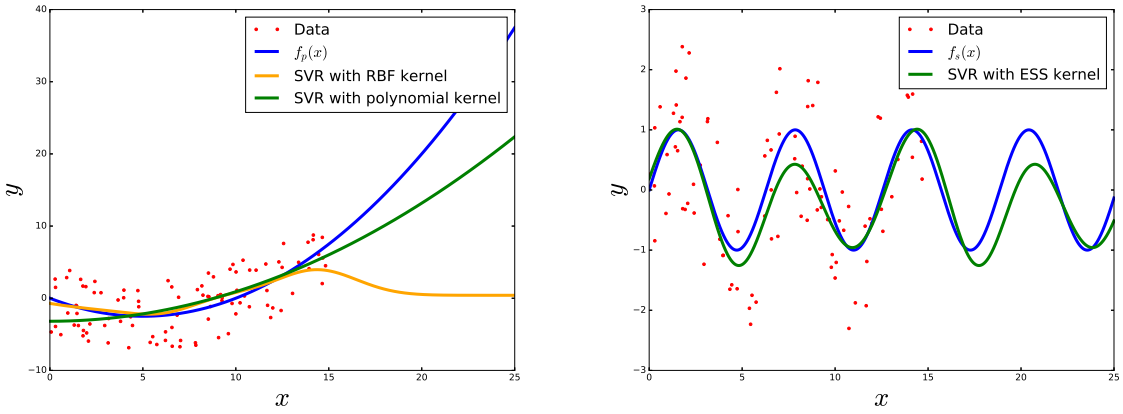

Figure 3: Comparison of prediction of $f_p(x)$ using SVR with RBF kernel and polynomial kernel ($\gamma = 0.13$, $c_0 = 0$, $\varepsilon = 0.15$, $C = 0.01$) to $f_p(x)$ (left). Comparison of prediction of $f_s(x)$ using SVR with ESS kernel ($\gamma = 1.67$, $p = 12.9$, $\varepsilon = 0.175$, $C = 7.50$) to $f_s(x)$ (right).

7.2 Support Vector Regression

We now repeat the analysis from the previous section, but using support vector regression. With SVR, we must decide on the parameters ε and C on top of any parameters of our selected kernel function (compared to KRR which just has one extra parameter, λ). Here, we use grid search to find ε and C. For the polynomial kernel we also search for γ, fixing $d = 2$ and $c_0 = 0$. For the ESS kernel, we search for γ and p. The results of this, as well as SVR with an RBF kernel, can be seen in Figure 3.

Here we see that SVR learns the functions $f_p(x)$ and $f_s(x)$ less accurately than KRR. There are, however, other factors to consider in evaluating the strength of a regression method; the sparse systems created by SVR are far less costly to solve than the dense systems in KRR. In fact, KRR and SVR take 0.030 and 0.025 seconds respectively to train on the data generated by $f_s(x)$. These numbers do not take into account grid searching, but in a real world case, the SVR parameter ε may have an obvious sensible value, meaning both methods would have an equally costly grid search. Furthermore, increasing ε creates even more sparse systems, further increasing the speed of SVR.

7.3 Gaussian Process Regression

Finally, we apply Gaussian Processes to our regression problems. For this method, we have just one extra parameter: the measurement noise level, σ_n. When applying KRR or SVR, we must employ a grid search to optimise the parameters. The cost of a grid search grows exponentially with the number of parameters being optimised over, meaning they are not viable for kernel functions with a large amount of parameters, or in cases where reasonable ranges of parameter values are not known *a priori*. With GPR, we can maximise the log marginal likelihood with respect to the parameters, in this case using the SciPy implementation [7] of the L-BFGS-B algorithm.

As we require the first derivative of our kernel function for parameter optimisation, the implementation of our polynomial kernel is slightly different in this section; we construct it by squaring the scikit-learn 'DotProduct' Gaussian process kernel. The result is the same as in the previous sections, but with $\gamma = 1$ fixed. When learning the function $f_p(x)$, we once again fix $d = 2$ on our polynomial kernel, and we use maximum log marginal likelihood to find optimal values of c_0 and σ_n (initial guesses: $c_0 = 0$, $\sigma_n = 0.1$). For the ESS kernel, we optimise over γ, p and σ_n with initial guesses $\gamma = 1$, $p = 5$ and $\sigma_n = 0.1$. As the log marginal likelihood is not a convex function, we are not guaranteed a global maximum, so we restart the optimiser twice from different starting points to achieve a better result. The resulting parameters and predictions are shown in Figure 4.

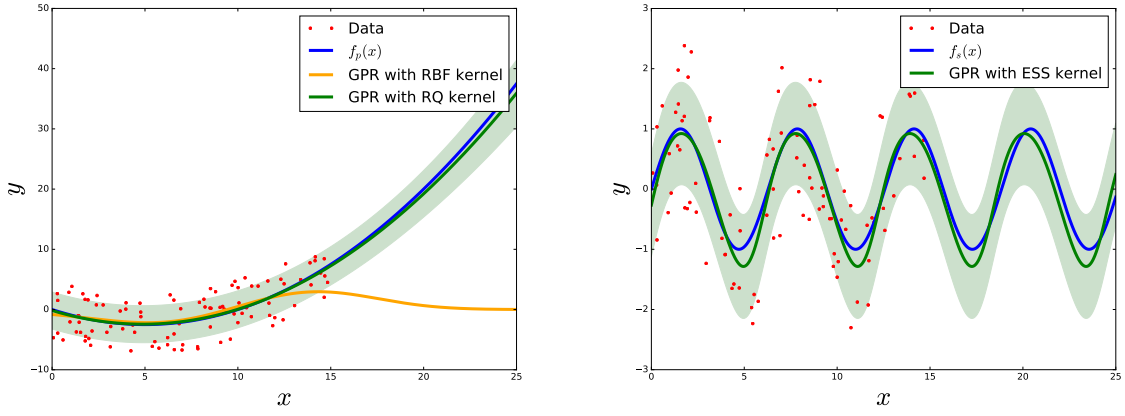

Figure 4: Comparison of prediction of $f_p(x)$ using GPR with RBF kernel and polynomial kernel ($d = 2$, $\gamma = 1$, $c_0 = 0.638$, $\sigma_n = 9.29$) to $f_p(x)$ (left). Comparison of prediction of $f_s(x)$ using GPR with ESS kernel ($\gamma = 1.53$, $p = 6.15$, $\sigma_n = 0.699$) to $f_s(x)$ (right), both with a confidence regions given by $\pm\sigma_{GP}$.

Finally, we find that GPR outperforms both KRR and SVR in terms of accuracy. Additionally, we are able to plot our predictions (given by Equation (73)) with a confidence region of $\pm\sigma_{GP}$ (from Equation (74)). As we have assumed our data to be normally distributed, this corresponds to a 68% confidence interval. On top of these benefits, our GPR model also fits the training data and finds optimal parameters in just 0.55 seconds, which is a huge improvement upon KRR and SVR which take 7.5 and 20.0 seconds respectively. The fast speed at which a GPR model can fit training data is by virtue of the fact that we can use standard optimisation algorithms to find parameters, as opposed to using grid searches.

8 Summary

In Sec. 2, it was explained what is meant by 'the kernel trick', and why one may wish to use it. In Sec. 2.1, the mathematics behind the kernel trick, and reproducing kernel Hilbert spaces was explored. After introducing the concept of linear regression and ordinary least squares, kernel ridge regression (KRR, Sec. 4), support vector regression (SVR, Sec. 5), and Gaussian process regression (GPR, Sec. 6) were derived by applying the kernel trick to their linear counterparts.

In Sec 7, the three kernel methods derived in this report were applied to two sets of noisy data: one with a polynomial pattern and one sinusoidal. In doing this, the importance of using a suitable kernel function was emphasised and some of the factors which should be considered when selecting a kernel function were discussed. KRR proved to be more accurate than SVR on our datasets. Though SVR is generally faster than KRR due to the sparse systems it creates, KRR was faster in our case, by virtue of having fewer parameters which needed to be included in a grid search. GPR outperformed the previous two methods both in terms of accuracy and speed. The superior speed of GPR can be attributed to the fact that we can select model parameters by optimising the log marginal likelihood instead of using grid search.

References

[1] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, 68:337–404, 1950.

[2] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *Ann. Statist.*, 36(3):1171–1220, June 2008.

[3] D. J.C. MacKay. Introduction to gaussian processes. *Neural Networks and Machine Learning*, 168, January 1998.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, Jan. 2006.

[6] B. Schölkopf, R. Herbrich, and A. J. Smola. A generalized representer theorem. In D. Helmbold and B. Williamson, editors, *Computational Learning Theory*, pages 416–426, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[7] SciPy.org. L-BFGS-B documentation. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html`. Accessed: 01-07-2018.

[8] S. Shalev-Schwartz and S. Ben-David. *Understanding Machine Learning: From Foundations to Algorithms*. Cambridge University Press, New York City, 1st edition, 2014.

[9] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, Aug 2004.

[10] C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 514–520. MIT Press, 1996.