UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Institut für Elektrotechnik und Informationstechnik
Universität Paderborn
Fachgebiet Nachrichtentechnik
Prof. Dr.-Ing. Reinhold Häb-Umbach

Topics in Pattern Recognition and Machine Learning

Gaussian Processes

von

Shankar Kumar Jeyakumar
Matr.-Nr.: 6809396

Betreuer: Prof. Dr. Reinhold Häb-Umbach
Abgabetermin: April 30, 2019

Abstract

In man's endeavour of *Artificial Intelligence*, Machine Learning is a stepping stone to achieve this goal. Machine Learning involves presenting observation to a machine, and having it learn a statistical model that it can see within the information in the observations, which it can later use to predict the outcomes of new unseen observations. This *Machine-Learning* can be categorized as Supervised Learning and unsupervised Learning of which Supervised Learning can be further categorized into *Classification* problems and *Regression* problems. *Gaussian Processes* are applied for the purpose of Regression and this report will focus on how and why we can do this.

This report is based mainly off the book "Gaussian Processes for Machine Learning" [RW06]. All credit and copyrights goes to the authors. I simply try to formulate the contents presented in the book into a summary for better understanding and provide visualizations using corresponding implementations of the ideas presented in the book.

Symbols

$\phi(x)$	Basis function
$\kappa(x, x`)$	Kernel function
σ^2	Variance
μ	Mean
l	Lengthscale
$P(x)$	Probability
ϵ	Noise parameter
$E(x)$	Expectation
$\mathcal{N}(\mu, \sigma)$	Normal distribution
$\mathcal{GP}(m, \kappa)$	Gaussian Process

Contents

1 Introduction

In Supervised Machine Learning[1], *parametric models*[2] such as Support Vector Machines[3] are often used to learn from presented data, store the *parameters* as a *model* and use this model to *predict* or perform *inference*[4]. This inference can be either predicting what newly presented data could be or predicting the function that produced the data. This works well when your data is simple enough to model parametrically. What happens when you data increases in complexity? This would mean more parameters to explain our data and hence increase in complexity of computation. As an example consider the graph shown in figure 1.1.

Figure 1.1: Least squares fit

In short, it is a straight line trying its best to pass through or fit/model the data presented to it. It doesn't seem to do a very good job of fitting even most of the data points. The line takes the form of

$$ax + b \tag{1.1}$$

[1] where you observe some input and its corresponding output, both of which are made available, and learn the mapping between them

[2] methods that formulate a model with fixed number of parameters

[3] a classification algorithm that differentiates between data using a dividing plane

[4] arriving at some conclusion based on some assumptions made a priory

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the only two parameters of the function that we could tweak to fit the given data. It is possible that only two parameters aren't enough. If we were to consider a quadratic function of the form

$$ax^2 + bx + c \tag{1.2}$$

this would probably perform a better since it is a quadratic equation with three parameters for tweaking and can curve around the data to fit itself. But then would have to learn three parameters instead of the earlier two. We see that this method kind of restricts the model to being specified by its parameters beforehand and doesn't offer much flexibility.

1.0.1 Weight Space View and Function Space View

This model formulated till now can be seen as a *Weight-Space* view where the output is a linear combination of input aided by some parameters or *weights* of the parameters. What if we cannot or do not want to specify these parameters beforehand? [Bai16]. Or we might want to consider which *function* actually generated these samples. Or a set of functions that are likely to have generated these samples. This is where *non-parametric*[5] models come to play. Unlike parametric models, non-parametric models are formulated by looking at the the data presented to it rather than specifying the parameters in advance. Although *non-parametric* does not necessarily mean no parameters at all, it's is that there are infinite number of possibilities for parameters [Bai16].

In the following sections we will focus on explaining this *Function-Space* view and correspondingly apply Gaussian Processes for Regression using this view. But before we get to that, we will formulate the Weight Space view and its limitations and see how it relates to the Function Space view.

[5]methods that rely on the distribution of the data rather than parameters alone

2 Regression

Application of supervised learning can fall under two categories – Classification and Regression. Classification involves finite discrete labeling to data whereas Regression involves fitting a continuous function around present data enabling us to predict future values. Consider we encounter some noisy data supposed to have been generated from an underlying function. The data that we have can be known as the *training data*. Our goal here would be to see through the noise and extract or approximate this original function based on this noisy data that we have from the function. Formulating Regression mathematically yields the steps [MW05]

- Our training data $D = (x_i, y_i)$ where $i = 1, 2...n$
- Each input is a vector of dimension x
- Target is a real valued scalar $y = f(x)$ + noise
- Input matrix $X = d$ x n, and targets in vector y
- $D = (X, y)$
- We would like to *infer* f^* for some test data, unseen before input x^* using $P(f^*|x^*, D)$

The term $P(f^*|x^*, D)$ in the above formulation represents a conditional probability which is part of the Bayesian approach to Regression, explained in later sections. But before that let us try to understand the simplest method to achieve Regression, the non Bayesian way, which is *Curve Fitting*. Lets take a simple example to demonstrate this [Fol13].

2.0.1 Curve Fitting Using Linear Regression

Consider you have data from a restaurant about tips received for meals served, shown in the table 2.0.1.

Meal	Cost
1	5
2	17
3	11
4	8
5	14
6	5
7	?

The only information you have is the amount of the tip you received. Now say you must predict the tip amount for the 7^{th} meal. How would we do that? With only one variable

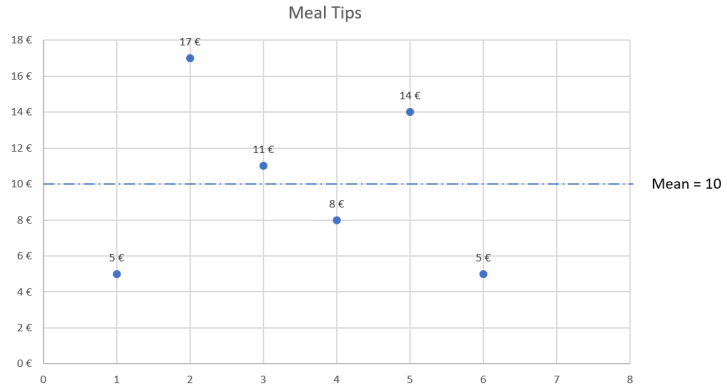


Figure 2.1: Meal tips scatter plot

and no other information, the best prediction for the next measurement is the mean of the samples themselves, shown in 2.1. The variance of the tip amounts can only be explained by the tip values themselves. Now that we have the mean or the "best fit line", we can measure how well the data actually fits. One way is to find the distance of the points from the mean. These distances can also be known as the errors, shown in 2.2. This is the first step towards measuring standard deviation. We can notice that

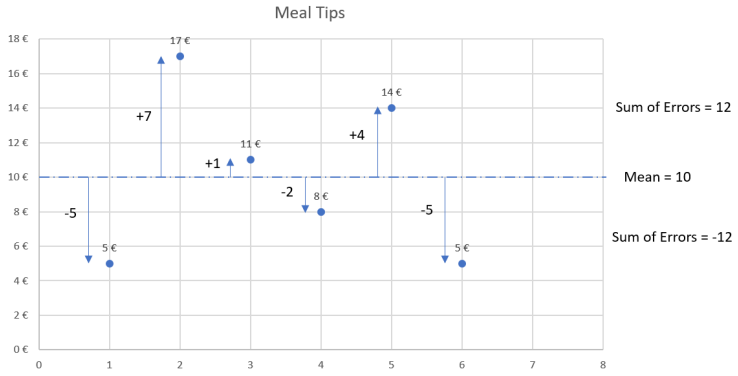


Figure 2.2: Deviation of meal tips

the sum of the errors on both sides of the line add up to zero. Let us proceed to square the error values to emphasize the larger deviations, shown in table 2.0.1 below.

Meal	Error	Error^2
1	-5	25
2	+7	49
3	+1	1
4	-2	4
5	+4	16
6	-5	25

The sum of the squared errors (SSE) turns out to be 120. The above is a simple Regression problem with the solution being a model to minimize the sum of squares of the errors. A Linear Regression model is good if it reduces this sum squared of errors by a large amount, here - finding the best fitting line through the data that minimizes the SSE. Now we keep talking about best fitting line and we during that and the introductory plot, we notice that the line doesn't actually fit our data samples well. Figure 2.3 represents this lack of expressiveness of the model we chose - a line.

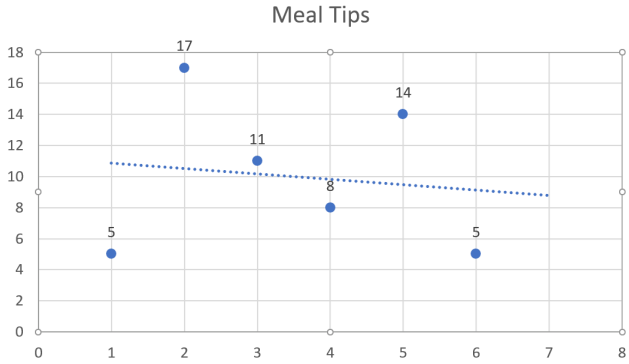


Figure 2.3: Meal tips scatter plot

2.0.2 Curve Fitting Using Non-Linear Regression

A step further in this direction would be to actually use curves instead of lines to fit our data. A line takes the form of

$$f(x) = ax + b \tag{2.1}$$

which only gives us a maximum of two coefficients to tweak until our line better fits the data. Instead, consider a second degree polynomial of the form

$$f(x) = ax^2 + bx + c \tag{2.2}$$

which improves our flexibility with three coefficients, $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$, to tweak. Figure 2.4 shows another example with noisy data collected from an underlying function and a line as a fitment around our data. Since we are considering polynomials, different fitment results for various orders are shown in figures 2.5a, 2.5b, 2.6a and 2.6b. How does one choose what order of the polynomial fits our data better? The simplest way

would be to count the number of points and set that as the order of the polynomial. But this is not very intuitive when the number of data points are too many to count. This is a non-statistical approach where we don't go on to *guess* beforehand what the curve might look like. A better approach is to look at the problem from a statistical viewpoint where we first guess what the curve might look like that fits our data, and then when we encounter the actual data, modify our *guess* accordingly. The *guess* can be a probabilistic one. This approach can be summed up as the *Bayesian* methodology.

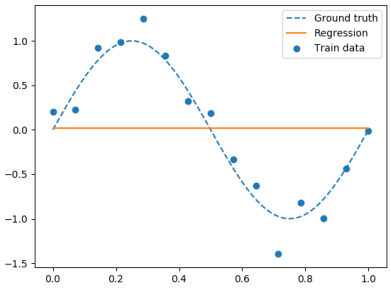
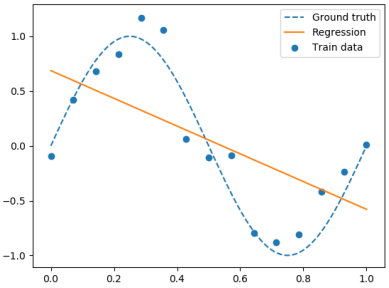
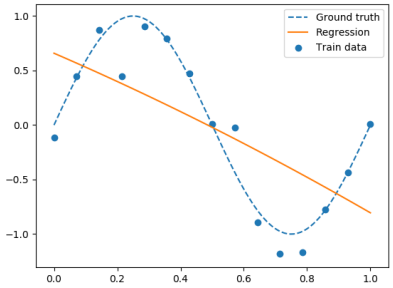


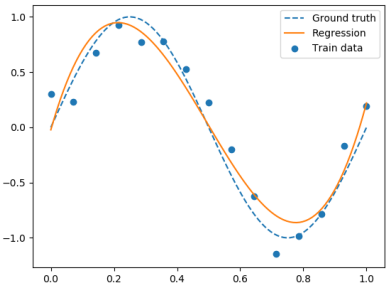
Figure 2.4: Order = 0



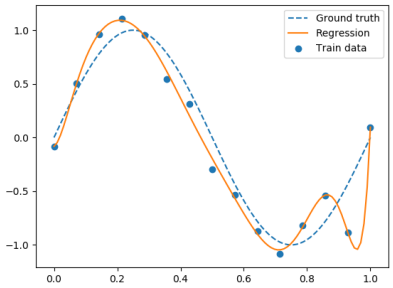
(a) Order = 1



(b) Order = 2



(a) Order = 3



(b) Order = 10

Figure 2.6: Different orders for polynomial curve fitting

2.0.3 Bayesian Perspective

Bayes theorem describes the probability of a future event given knowledge of the conditions that led to that particular event. Mathematically, consider A and B are events.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.3}$$

- $P(A|B)$ is a *conditional probability*[1], likelihood of A occurring, given B is True and vice versa for $P(B|A)$
- P(A) and P(B) are *marginal probabilities*[2] of events A and B, independent of each other

2.0.4 Weight Space View

Bayesian Regression is a statistical approach to find a distribution over the parameters. For this we need to specify a prior probability distribution or simply a *prior* over the parameters. These parameters can be the updated when you encounter new data. Let's revisit out initial Linear Regression model from a Bayesian perspective. Mathematically our Linear Regression model, with Gaussian noise can be written as

$$f(\mathbf{x}|\mathbf{w}) = \mathbf{x}^T \tag{2.4}$$

$$y = f(\mathbf{x}) + \epsilon \tag{2.5}$$

with the noise parameter distributed as a Gaussian with mean and variance

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{2.6}$$

The noise parameter and the model combined results in the *likelihood*, probability density taking all $\boldsymbol{n}$ observations into consideration, given by

$$p(\mathbf{y}|X, \mathbf{w}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w}) \tag{2.7}$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} exp(-\frac{(y_i - \mathbf{x}_i^T\mathbf{w})^2}{2\sigma_n^2}) \tag{2.8}$$

$$= \frac{1}{(2\pi\sigma_n^2)^{n/2}} exp(-\frac{1}{2\sigma_n^2}|\mathbf{y} - X^T\mathbf{w}|^2) \tag{2.9}$$

resulting in

$$= \mathcal{N}(X^T\mathbf{w}, \sigma^2 I) \tag{2.10}$$

According to Bayesian methodology, assuming a Gaussian prior distribution over our parameters

$$P(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \Sigma_p) \tag{2.11}$$

[1] probability of an event occurring, depending on other events
[2] probability of a lone event occurring, independent of other events

and according to Bayes' theorem to obtain the posterior distribution for *inference* or *prediction* which includes the likelihood and the prior

$$posterior = \frac{likelihood * prior}{marginalLikelihood} \tag{2.12}$$

$$P(\mathbf{w}|\mathbf{y}, X) \propto P(\mathbf{y}|X, \mathbf{w})P(\mathbf{w}) \tag{2.13}$$

$$P(\mathbf{w}|\mathbf{y}, X) = \frac{P(\mathbf{y}|X, \mathbf{w})P(\mathbf{w})}{P(\mathbf{y}|X)} \tag{2.14}$$

the marginal likelihood is given by

$$p(\mathbf{y}|X) = \int p(\mathbf{y}|X, \mathbf{w})p(\mathbf{w})d\mathbf{w} \tag{2.15}$$

the posterior takes into account the prior and our knowledge about the parameters. After some mathematical simplification we get

$$p(\mathbf{w}|X, \mathbf{y}) \propto exp(-\frac{1}{2\sigma_n^2}(\mathbf{y} - X^T\mathbf{w})^T(\mathbf{y} - X^T\mathbf{w}))exp(-\frac{1}{2}\mathbf{w}^T\Sigma_p^{-1}\mathbf{w}) \tag{2.16}$$

$$\propto exp(-\frac{1}{2}(w - \hat{w})^T(\frac{1}{\sigma_n^2}XX^T + \Sigma_p^{-1})(w - \hat{w})) \tag{2.17}$$

Equation 2.17 is of the form of a Gaussian which is our posterior with mean

$$\hat{w} = \sigma_n^{-2}(\sigma_n^{-2}XX^T + \Sigma_p^{-1})^-1 \tag{2.18}$$

and covariance matrix

$$A^{-1} = \sigma_n^{-2}XX^T + \Sigma_p^{-1} \tag{2.19}$$

$$P(\mathbf{w}|\mathbf{y}, X) = \mathcal{N}(\frac{1}{\sigma^2}A^{-1}X\mathbf{y}, A^{-1}) \tag{2.20}$$

The end purpose of all this is to predict when new unknown values $\mathbf{x}^*$ come in. Instead of plugging in values for $\mathbf{w}$ as in the previous method, we calculate the average for all possible values of $\mathbf{w}$ to get a *Predictive Distribution*. Hence the Predictive Distribution is obtained by computing the average over all the linear models with respect to the Gaussian posterior, given by

$$P(f^*|\mathbf{x}^*, X, \mathbf{y}) = \int f(\mathbf{x}^*|\mathbf{w})P(\mathbf{w}|X, \mathbf{y})d\mathbf{w} \tag{2.21}$$

$$= \mathcal{N}(\frac{1}{\sigma_n^2}\mathbf{x}_*^T A^{-1}X\mathbf{y}, \mathbf{x}^{*T}A^{-1}\mathbf{x}^*) \tag{2.22}$$

Equation 2.22 for prediction is also Gaussian whose mean is the mean of the posterior multiplied by some test values. The $\mathbf{x}_*$ terms in equation 2.22 make variance quadratic. The order of this quadratic increases with increase in input values due to the linear model formulation.

2.0.5 The Kernel Trick

We have already seen in the previous sections of curve fitting, how a linear function is inefficient to model the information from non linearly-separable data. Since the relationship is modelled by a linear function, the model will result in poor inference as seen in the introductory example. For this purpose we increase the dimensionality of the input by mapping it to a *feature space* by using some *Basis* function. After this higher dimensional mapping we can then proceed to apply the linear model formulated previously, to this *feature* space than on the original *input* space. We formulate that a scalar input $\boldsymbol{x}$ can be mapped to the feature space by using a function $\phi(\mathbf{x})$, with orders for $\boldsymbol{x}$ as 1, 2, 3... to facilitate Polynomial Regression. But then the question

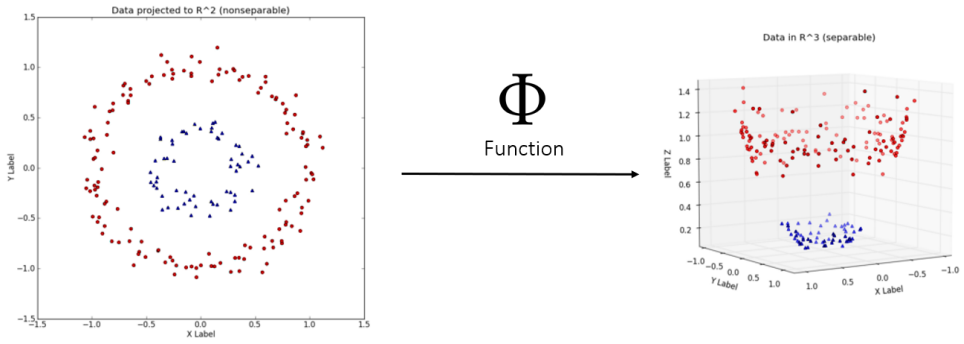


Figure 2.7: Increasing Dimensionality [Kan17]

remains how do we know which Basis function to choose for this purpose? This also turns out to be computationally intensive to determine this unknown basis function. For a mathematical view, rewrite equation 2.22 as

$$f(\mathbf{x}|\mathbf{w}) = \phi(x)^T w \tag{2.23}$$

Hence equation 2.11 can be rewritten including the formulation.

$$\mathcal{N}(\frac{1}{\sigma_n^2}\phi(x_*)^T A^{-1}\Phi y, \phi(x_*)^T A^{-1}\phi(x_*)) \tag{2.24}$$

where ϕ is called a *Basis function* that maps our point to a higher dimension. Let us revisit the Polynomial Regression example using the Bayesian formulation with a polynomial as the Basis function. Figures 2.8, 2.9a, 2.9b, 2.10a and 2.10b show the results of the same for various orders of our basis function. The region between the yellow dotted curves represents a confidence interval within which our observations points ought to be engulfed.

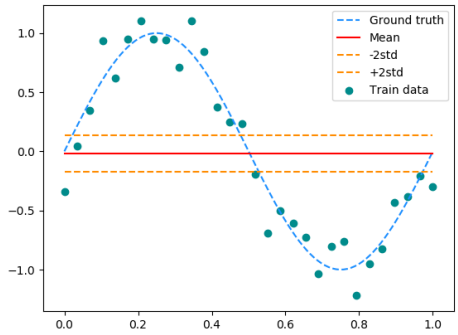
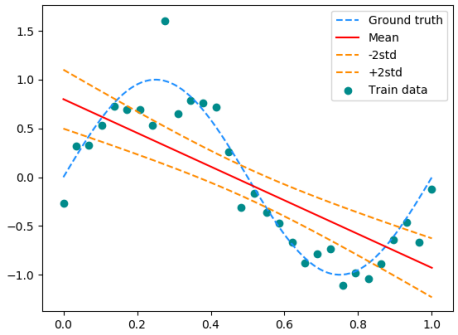
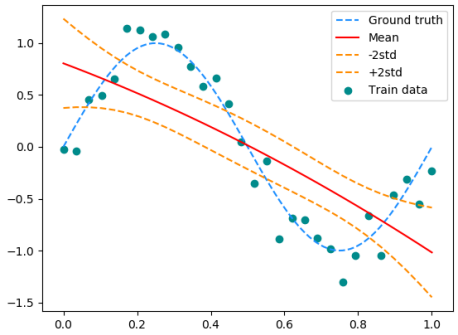


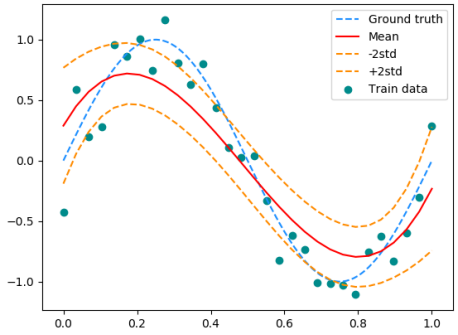
Figure 2.8: Order = 0



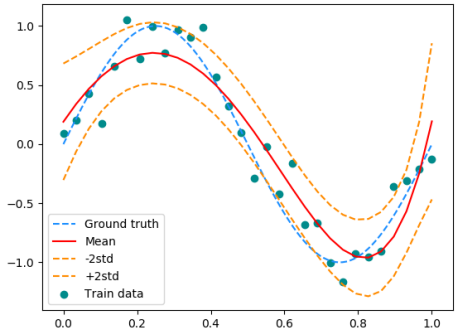
(a) Order = 1



(b) Order = 2



(a) Order = 3



(b) Order = 7

Figure 2.10: Different orders for Bayesian Polynomial Regression

But a question arises, how many of these basis functions do we use? The more the data points, there is a corresponding increase in the number of basis function we need to use. Complexity is increasing linearly with the number of points, which isn't efficient. Also

computation of the ϕ function and inversion of this matrix of $N*N$ dimensions, if N is large, is considered very complex and computationally intensive. As a work around for this, rewrite the equation 2.15 to include

$$\mathcal{N}(\phi_*^T\Sigma_p\Phi(K+\sigma_n^2I)^{-1}y, \phi_*^T\Sigma_p\phi_* - \phi_*^T\Sigma_p\Phi(K+\sigma_n^2I)^{-1}\Phi^T\Sigma_p\phi_*) \tag{2.25}$$

our point of interest in equation 2.16 being the term

$$\phi(x)^T\Sigma_p\phi(\mathbf{x}`) \tag{2.26}$$

which is an *inner product*. A mathematical rule [RW06] stating that

> "If an algorithm is described solely in terms of inner products in input space then it can be lifted into feature space by replacing occurrences of those inner products by $k(x, x`)$; this is sometimes called the kernel trick."

lets us replace the ϕ function by the kernel function k(x,x`) also called the covariance function, which is far more easier to compute than having to find a vector or the ϕ function that maps feature to the feature space.

$$k(\mathbf{x}, \mathbf{x}`) = \phi(\mathbf{x})^T\Sigma_p\phi(\mathbf{x}`) \tag{2.27}$$

This lets us realize that this *Kernel* function is of higher importance than the ϕ functions themselves. The next sections shows us how we can perform inference in this kernel function space itself by using the Kernel.

2.1 Function Space View

So far in the previous sections we have discussed parametric models for regression. These models, although benefit from their simplicity, lack in their ease of computation and increase in the degree of their complexity. The Kernel trick was an advantage for non linearly-separable data, but how many basis functions would we choose as the complexity of the data increases? For this purpose, we move to what is called the *Function space* rather than continue in the *Weight-Space*. Simply put the *Function-Space* view refers to all the possible functions that our original function might be. Remember our formulation of the Polynomial Regression in which we had to determine the coefficients of the polynomial equation we picked to model the curve according to our data. We saw that the coefficients kept increasing if we required better fitment. The Function-Space view involved performing inference directly in this function space or Kernel space. Instead of inference of a distribution over the parameters of a parametric function, we try to infer a distribution over functions directly.

2.1.1 Gaussian Processes

We wished to transit from mapping a set of observations used for training to a *function* that can predict where these samples or observations came from. The following sections help us understand this Function Space view and how we correspondingly formulate Gaussian Processes for *Regression*, better.

Gaussian Distribution

The univariate Gaussian distribution is given by

$$p(x|\mu,\sigma^2) = (2\pi\sigma^2)^{-1/2} exp(-\frac{1}{2\sigma^2}(x-\mu)^2) \tag{2.28}$$

and the Multivariate Gaussian distribution for multi-dimensional vectors is given by

$$p(x|\mu,\Sigma) = \mathcal{N}(\mu,\Sigma) = (2\pi)^{-D/2}|\Sigma|^{-1/2} exp(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)) \tag{2.29}$$

where

$$\mu = mean \tag{2.30}$$

and

$$\Sigma = covariance \tag{2.31}$$

A bi-variate version is shown in figure 2.11. The shape of the dome can be controlled

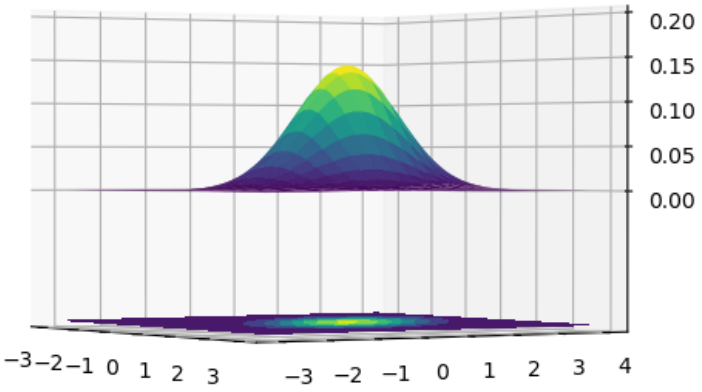


Figure 2.11: Bi-variate Gaussian Distribution

by the covariance matrix. The mean function, used to obtain the expected value of the function and the covariance matrix, defines a Gaussian Process. A generalized version of the multivariate Gaussian distribution to include infinite variables is a Gaussian process [RW06].

A process is a Gaussian Process if a set of random variables from this process has a multivariate normal distribution or every linear combination of a finite set of these variables are normally distributed. Simply put, if we were to pick any two points in from the process, the outputs at these points would be jointly Gaussian distributed. A Gaussian process is a *stochastic*[3] process, follows from the original concept of a Gaussian distribution, defined over vectors with the difference being for a Gaussian

[3] A process that can be imagined as the motion of a body with respect to time. The body could be at different points of space in particular instances of time, associating a probability of it being there, which allows us to infer the path it might follow at the next instance of time

process defined over functions, the mean and covariance are a vector and a matrix respectively. The mean and the covariance are all that we require to specify a Gaussian process, denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), \kappa(x, x^{'})) \tag{2.32}$$

where $m(\mathbf{x})$ is the mean and $\kappa(\mathbf{x}, \mathbf{x}^{'})$ is the covariance or the kernel function.

What makes a Gaussian Process so feasible to define a distribution over functions? The *Consistency*[4] property of a Gaussian Process, which mathematically states [MW05] that if the Gaussian Process specifies

$$y^{(1)}, y^{(2)} \sim \mathcal{N}(\mu, \Sigma) \tag{2.33}$$

then it also specifies

$$y^{(1)} \sim \mathcal{N}(\mu_1, \Sigma_{11}) \tag{2.34}$$

Consider a zero mean function with the covariance given by

$$K_{p,q} = Cov(f(x^{(p)}), f(x^{(q)})) \tag{2.35}$$

$$= K(x^{(p)}, x^{(q)}) \tag{2.36}$$

The Covariance function or the Kernel computes the joint distribution over the values of the original zero mean function, given as

$$f(x^{(1)}), f(x^{(2)})....f(x^{(n)}) \sim \mathcal{N}(\mathbf{0}, K) \tag{2.37}$$

from equation 2.32 and 2.33, we can say that a Gaussian Process hence specifies a distribution over functions.

Let us obtain a Gaussian Process from our Bayesian Linear model

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} \tag{2.38}$$

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p) \tag{2.39}$$

Mean is given by

$$E[f(\mathbf{x})] = \mathbf{x}^T E[w] = 0 \tag{2.40}$$

and the Covariance function is given by

$$E[f(\mathbf{x})f(\mathbf{x}^{'})] = \mathbf{x}^T E[\mathbf{w}\mathbf{w}^T]\mathbf{x}^{'} = \mathbf{x}^T\Sigma_p\mathbf{x}^{'} \tag{2.41}$$

The Kernel or the covariance function gives you the covariance between our observations. The idea is that if the kernel deems $x^{(p)}$ and $x^{(q)}$ to be noticeably similar, then the function also might produce similar values at these points [MW05], useful for prediction in our Regression.

An example of such a Kernel function is the Squared exponential covariance function

$$K(x^{(p)}, x^{(q)}) = exp(-\frac{1}{2}|x^{(p)} - x^{(q)}|) \tag{2.42}$$

[4]Kolmogorov Consistency Criterion

which is the most commonly used one. Other examples are Periodic, Linear, Hyperbolic, etc., According to previously defined Bayesian methodology, we consider a prior distribution by drawing samples over a set of our observations as

$$f(\mathbf{x}^{(1)}), f(\mathbf{x}^{(2)})...f(\mathbf{x}^{(n)}) \sim \mathcal{N}(\mathbf{0}, K) \tag{2.43}$$

Figure 2.12 [Wik19] shows 7 samples drawn from the Gaussian Priors with three different covariance functions.

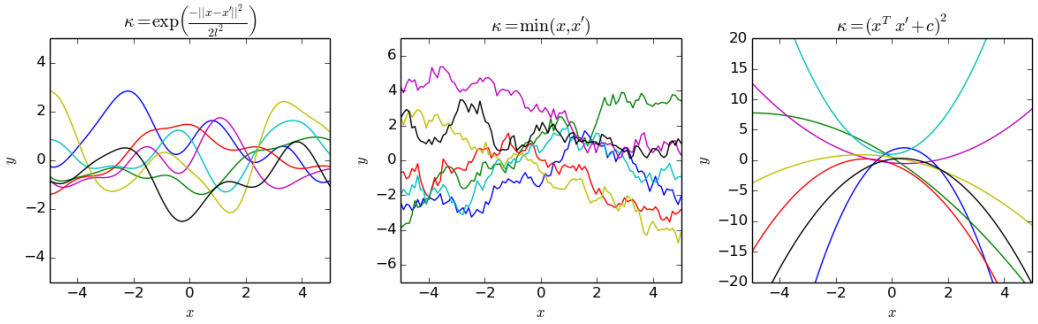


Figure 2.12: Gaussian Priors with different Kernels [Wik19]

Our aim was to be able to predict f^* at test points X^*. For this we compute the *Posterior*.

2.1.2 Gaussian Process Regression

From our previous mathematical formulation of the Gaussian Process for Regression we can come up with an algorithm for an implementation as follows. Figure 2.13 [RW06]

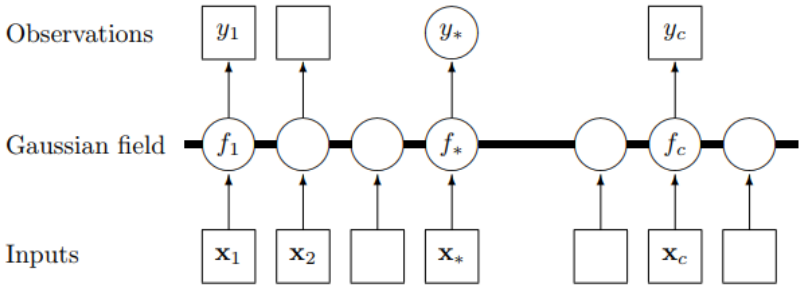


Figure 2.13: Gaussian Process Regression model

represent a graphical model for Gaussian Process Regression. The squares are input values that are known and the circles represent those that are unknown.

Noise Free Observations

With training data

$$\mathcal{D} = (\mathbf{x}^{(i)}, f^{(i)}) = (X, \mathbf{f}) \tag{2.44}$$

$$i = 1, 2, ...n \tag{2.45}$$

Inference

We would like to predict our function(s) $\mathbf{f}^*$ at test points X^*. The joint distribution of $\mathbf{f}$ and $\mathbf{f}^*$ is computed for $D = (\mathrm{X},\mathbf{f})$, restricting the prior for $(X^*, \mathbf{f}^*)$ for the posterior distribution for inference is given by the joint probability distribution[5] resulting in

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right).$$

function that agrees with D. Finally our Posterior is a Gaussian distribution shown in figure 2.16 with

$$\mathbf{f}_*|X_* \sim \mathcal{N}(K(X,X^*)K(X,X)^{-1}\mathbf{f}, K(X^*,X^*) - K(X,X^*)K(X,X)^{-1}K(X^*,X)) \tag{2.46}$$

where

$$mean = K(X,X^*)K(X,X)^{-1}\mathbf{f} \tag{2.47}$$

and

$$covariance = K(X^*,X^*) - K(X,X^*)K(X,X)^{-1}K(X^*,X) \tag{2.48}$$

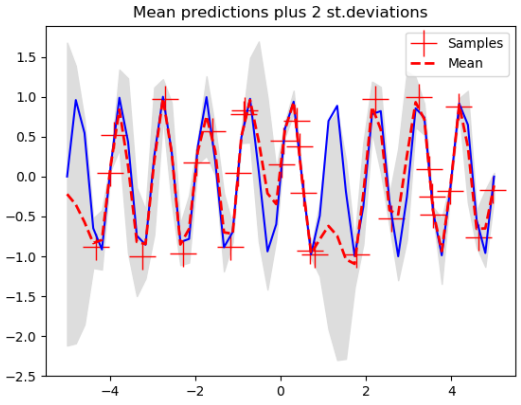


Figure 2.14: The original function to approximate

[5]probability that events occur simultaneously

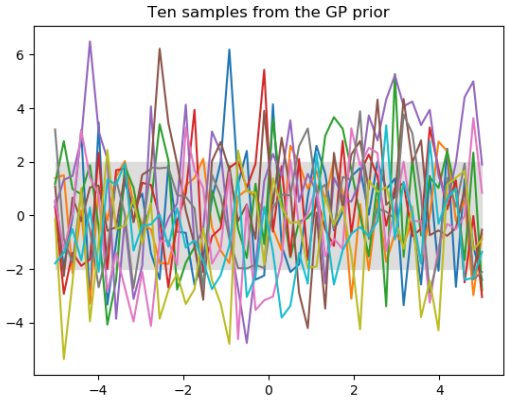


Figure 2.15: Gaussian Prior with 10 random samples

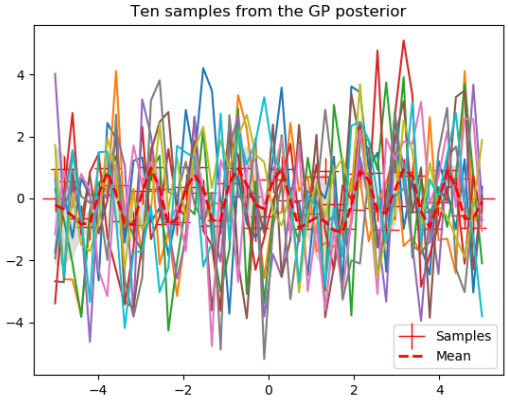


Figure 2.16: Gaussian Posterior with 10 random samples

Noisy Observations

In real life observations, we expect our samples to always contain some amount of noise. Hence our training set points become of the form

$$D = (X, \mathbf{y}) \tag{2.49}$$

where

$$\mathbf{y} = \mathbf{f} + \epsilon \tag{2.50}$$

the noise is assumed to be additive Gaussian distributed

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I) \tag{2.51}$$

Inference

The joint distribution of $\mathbf{f}$ and $\mathbf{f}^*$ then is for $D = (\mathrm{X},\mathbf{f})$, restrict the prior for $(X^*, \mathbf{f}^*)$ for

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X)+\sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

the posterior distribution, resulting in function that agrees with D. Finally our Posterior is a Gaussian distribution with

$$mean = K(X,X^*)[K(X,X)+\sigma^2 I]^{-1}\mathbf{y} \tag{2.52}$$

and

$$covariance = K(X^*,X^*) - K(X,X^*)[K(X,X)+\sigma^2 I]^{-1}K(X^*,X) \tag{2.53}$$

2.1.3 Weight Space and Function Space Correlation

From equation 2.24,

$$\mathcal{N}(\frac{1}{\sigma_n^2}\phi(\mathbf{x}_*)^T A^{-1}\Phi y, \phi(\mathbf{x}_*)^T A^{-1}\phi(\mathbf{x}_*)) \tag{2.54}$$

the inversion of matrix A is computationally intensive if its dimensions are large. Hence we rewrite it to the form

$$f_*|\mathbf{x}_*, X, \mathbf{y} \sim \mathcal{N}(\phi_*^T\Sigma_p\Phi(K+\sigma_n^2 I)^{-1})\mathbf{y}, \phi_*^T\Sigma_p\phi_* - \phi_*^T\Sigma_p\Phi(K+\sigma_n^2 I)^{-1}\Phi^T\Sigma_p\phi_*) \tag{2.55}$$

comparing to equation 2.53

$$covariance = K(X^*,X^*) - K(X,X^*)[K(X,X)+\sigma^2 I]^{-1}K(X^*,X) \tag{2.56}$$

shows that $K(\mathbf{x}(p),\mathbf{x}^{(q)}) = \Phi(\mathbf{x}^{(p)})^T\Sigma_p\Phi(\mathbf{x}^{(q)})$

In the weight space view, for a set of m basis functions $\phi(\mathbf{x})$, the corresponding kernel function is

$$K(x^{(p)},x^{(q)}) = \phi(x^{(p)})^T\Sigma_p\phi(x^{(q)}) \tag{2.57}$$

alternatively for every kernel function k there are infinite basis functions as

$$K(x^{(p)},x^{(q)}) = \Sigma_i\lambda_i\phi_i(x^{(p)})\phi(x^{(q)}) \tag{2.58}$$

$$i = 1, 2...\infty \tag{2.59}$$

in other words consider the mean function from equation 2.52.

$$mean = K(X,X^*)[K(X,X)+\sigma^2 I]^{-1}\mathbf{y} \tag{2.60}$$

which can be rewritten as

$$m_t(\mathbf{x}) = \sum_{i=1}^{t} w_i k(\mathbf{x}_i,\mathbf{x}) \tag{2.61}$$

where $\mathbf{x}_i$ is a previously obtained input in $\mathbf{X}$ and

$$w = [K(X,X)+\sigma_\epsilon^2\mathbf{I}]^{-1} \tag{2.62}$$

This shows that Gaussian Process Regression is the same as the linear regression model with $\phi(x)$ functions or *K basis* functions mapping out input points into feature space [SSK17].

3 Implementation

With all the mathematics from the previous sections adding up until now, we can propose a pseudo code [RW06] for an implementation, given by

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

$L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$

$\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$ } predictive mean eq. (2.25)

$\mathbf{v} := L \backslash \mathbf{k}_*$

$\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$ } predictive variance eq. (2.26)

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$ eq. (2.30)

return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

Figure 3.1: Pseudo code [RW06]

The steps then to visualize an example is as follows:

- Consider that the original function is unknown to us and we only have the samples generated from that function. For demonstration here we consider a ***Sine*** function shown in figure 3.2
- Define some points at which the function will be evaluated at. Lets say 30 points between the interval -5 and 5. These points are denoted by red pluses in figure 3.2
- Choose a kernel function - the RBF (Radial Basis Function) kernel in our case, which is quite commonly used
- Apply the kernel function to our data points
- Obtain a prior distribution by sampling some sets of standard normals and multiplying them with the covariance matrix's square root. Figure 3.3 shows 10 functions drawn from our prior.
- Compute mean and standard deviation at our test points, denoted by the red dashed line in figure 3.2
- Draw samples from the posterior at the evaluation points to see it match the original function it had to approximate, shown in figure 3.4.

Observe how the training points have helped reel in the Gaussian curves to the shape of the original Sine function in figure 3.4, which adheres to our Bayesian methodology.

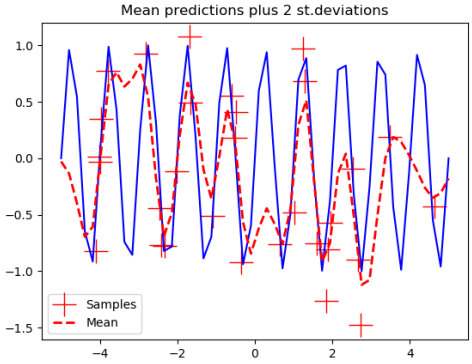


Figure 3.2: The original function to approximate

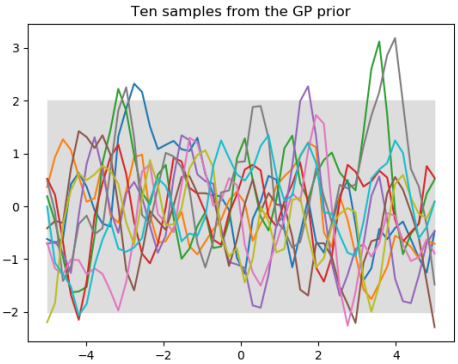


Figure 3.3: Gaussian Prior with 10 random samples

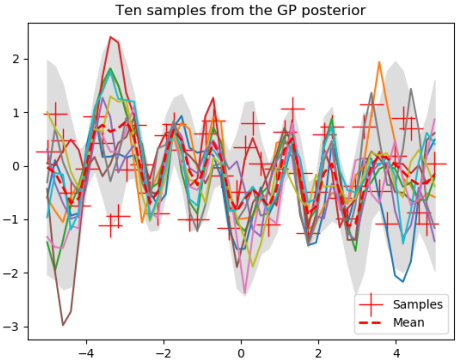


Figure 3.4: Gaussian Posterior with 10 random samples

As another example we can observe the function

$$f(x) = \frac{1}{4}x^2 \tag{3.1}$$

with the underlying ground truth and test samples shown in figure 3.5, its corresponding prior in figure 3.6 and posterior in figure 3.7

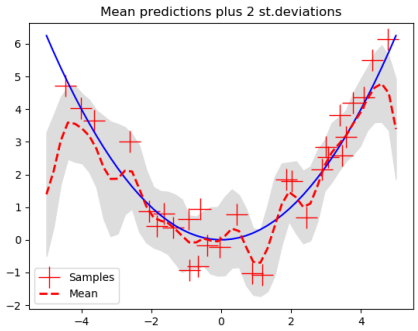


Figure 3.5: The original function to approximate

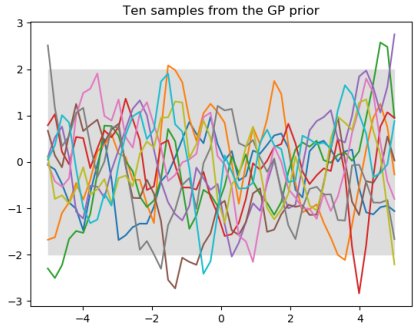


Figure 3.6: Gaussian Prior with 10 random samples

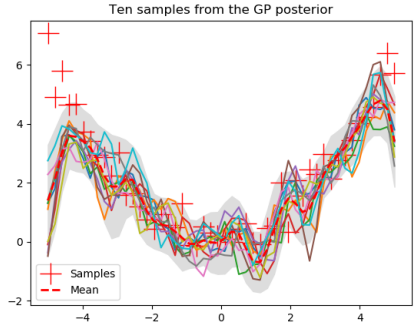


Figure 3.7: Gaussian Posterior with 10 random samples

4 Kernel Hyperparameter Tuning

Properties of the covariance function, also called the kernel function can be seen as *Hyperparameters*[1] [Sne07] which can then be specified by trial and error or chosen by a heuristic and the fitment can be observed. The most commonly used Squared exponential covariance function, also known as the Radial Basis Function kernel or Gaussian kernel takes the mathematical form

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 exp(-\frac{1}{2l^2}(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)) \tag{4.1}$$

A visual representation is shown in figure 4.1. The parameter $\boldsymbol{l}$, also known as the

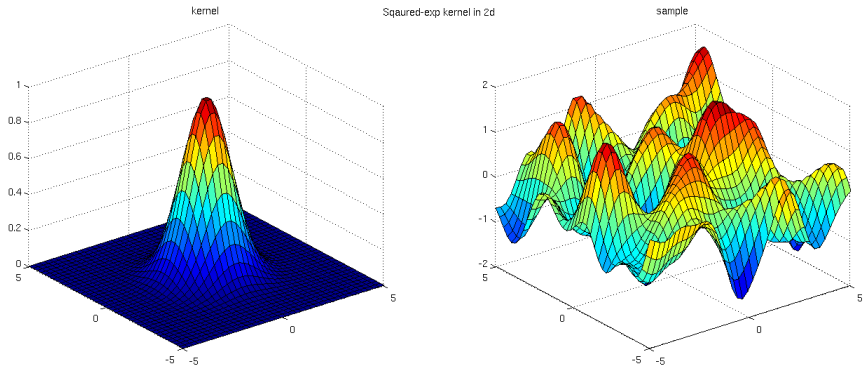
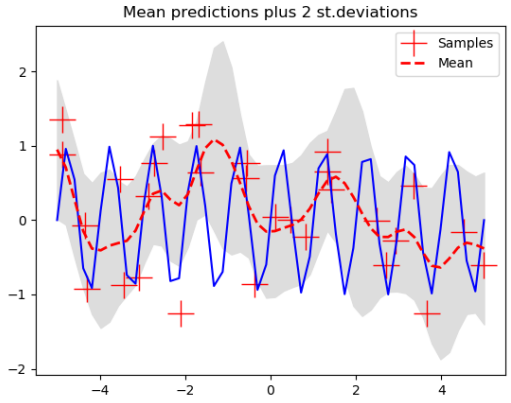


Figure 4.1: The Squared Exponential Kernel in 2-d [Duv14]

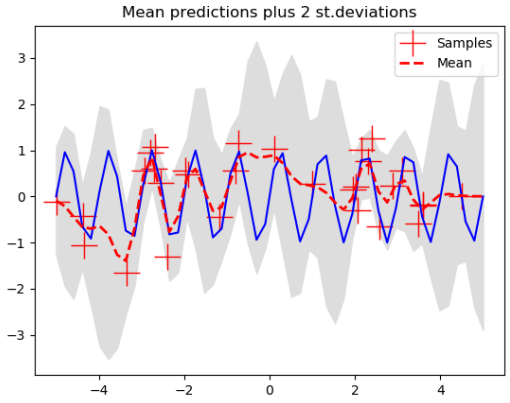
lengthscale controls the smoothness and σ_f controls vertical width. Varying these parameters allows us to obtain a better fit to our training data. Figures 4.2a, 4.2b, 4.3a, 4.3b, 4.4a and 4.4b are results of various values of $\boldsymbol{l}$ and σ_f.

Now how does one determine which values of $\boldsymbol{l}$, σ_f provide better fitment or approximation of our original function? Furthermore, in real life environments, out readings are most often always mired in noise. The samples that we collect of this unknown function will have a noise element in addition to it. The following figures 4.5a, 4.5b, 4.6a, 4.6b, 4.7a and 4.7b show effects of the noise parameter on the predictions of the posterior. Notice how as the noise increases, our mean does not seem to engulf all the evaluation points.

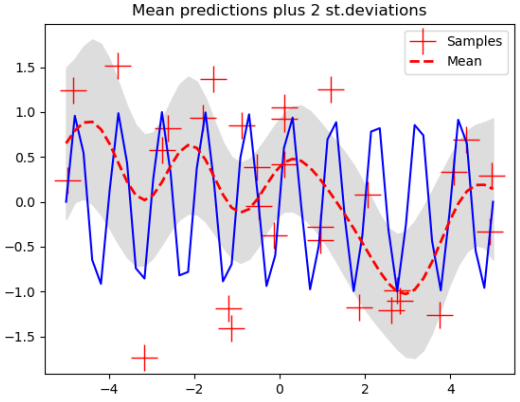
[1] whose value is independent of the data and is specified priory



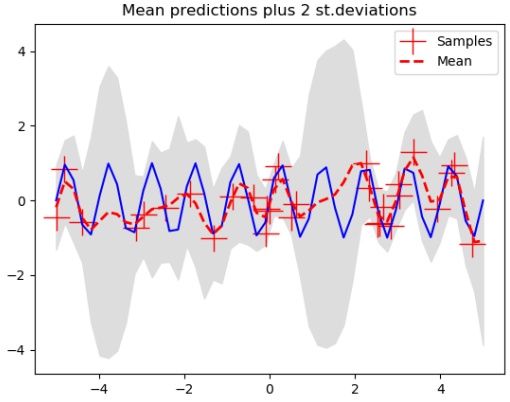
(a) l: 0.5, sigma_f: 1.0, noise: 0.4



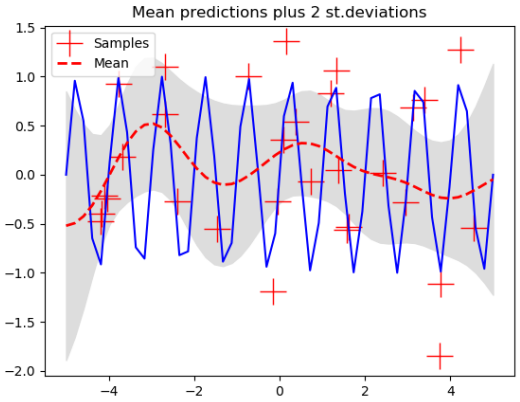
(b) l: 0.31622, sigma_f: 1.5, noise: 0.4



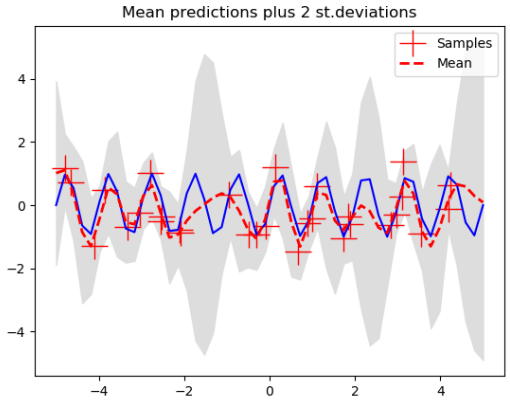
(a) l: 0.9, sigma_f: 1.0, noise: 0.4



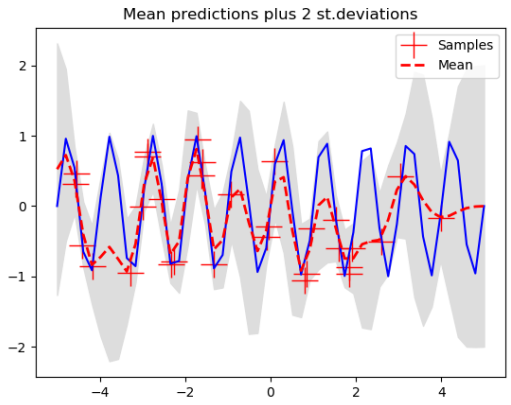
(b) l: 0.31622, sigma_f: 2.0, noise: 0.4



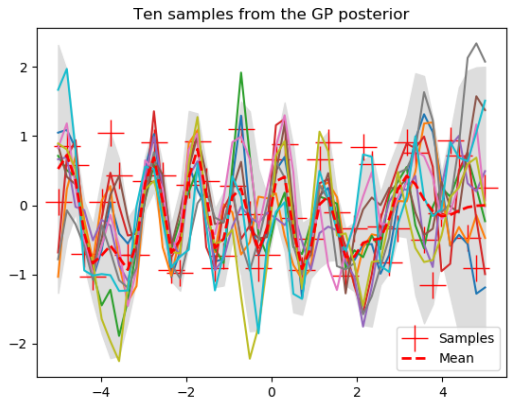
(a) l: 1.1, sigma_f: 1.0, noise: 0.4



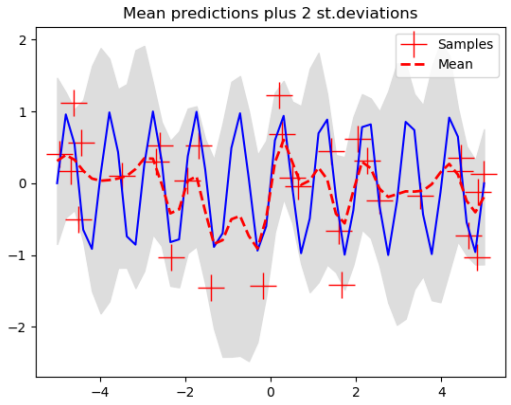
(b) l: 0.31622, sigma_f: 2.5, noise: 0.4



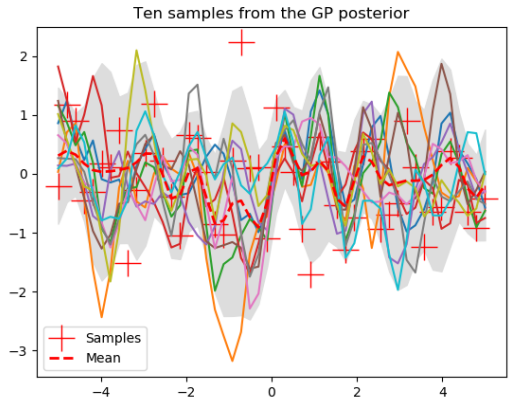
(a) l: 0.31622, sigma_f: 1.0, noise: 0.1



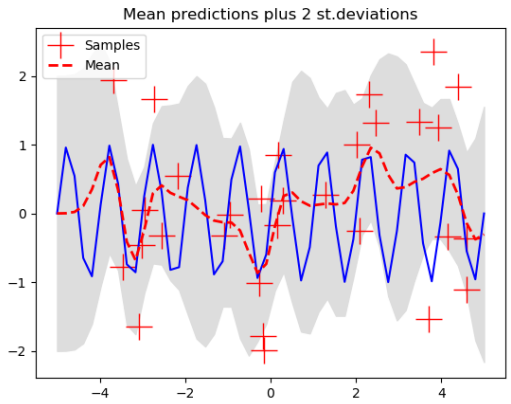
(b) l: 0.31622, sigma_f: 1.0, noise: 0.1



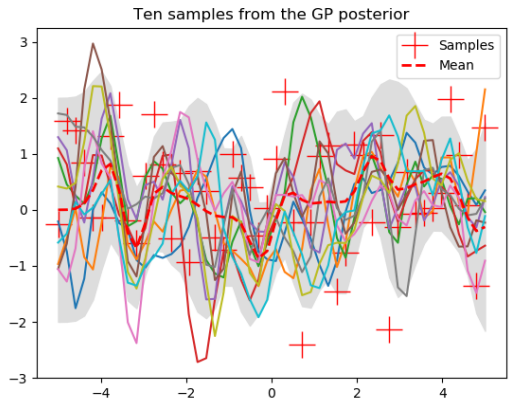
(a) l: 0.31622, sigma_f: 1.0, noise: 0.5



(b) l: 0.31622, sigma_f: 1.0, noise: 0.5



(a) l: 0.31622, sigma_f: 1.0, noise: 0.9



(b) l: 0.31622, sigma_f: 1.0, noise: 0.9

4.0.1 Optimal Kernel Hyperparameters

So far for our prior we chose random samples and assumed a zero mean prior, which would help in a generic method for predictions on the distributions. To strengthen this, we could either consider a non zero mean for the prior if we have enough data about our input samples, or proceed to take the relation that for the hyperparameter $\boldsymbol{l}$ the posterior is proportional to the marginal log likelihood as

$$P(l|X, \mathbf{y}) \propto P(\mathbf{y}|X, l) \tag{4.2}$$

Optimum kernel parameters for $\boldsymbol{l}$ and σ_f can be obtained by maximizing the marginal log likelihood function [Fol13] given by

$$logP(\mathbf{y}|\mathbf{X}, l) = log\mathcal{N}(y|0, \mathbf{K}_y) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}log|\mathbf{K}_y| - \frac{N}{2}log(2\pi) \tag{4.3}$$

which under the presence of noise, becomes

$$logP(\mathbf{y}|\mathbf{X}, l) = log\mathcal{N}(y|0, \mathbf{K}_y + \sigma_n^2\boldsymbol{I}) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma_n^2\boldsymbol{I})_y^{-1}\mathbf{y} - \frac{1}{2}log|\mathbf{K}_y + \sigma_n^2\boldsymbol{I}| - \frac{N}{2}log(2\pi) \tag{4.4}$$

After obtaining optimized values for the kernel hyperparameter l and setting the values of σ_f and σ_n to known values, a confidence interval[2] of over 92% is established over training data points even under high noise, shown in figure 4.8.

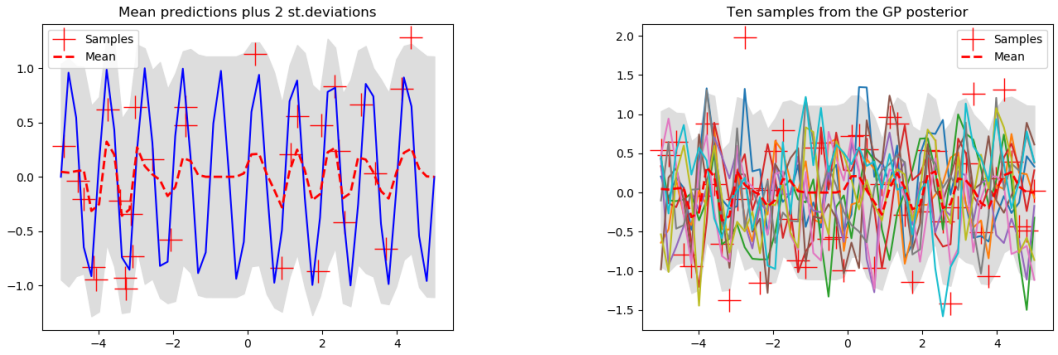


Figure 4.8: l_opt: 0.17874, sigma_f_opt: 0.59915

Choosing the right kernel

We can also use other common kernels such as the Rational Quadratic Kernel which takes the mathematical form

$$\kappa(x, x^{'}) = \sigma^2(1 + \frac{(x - x^{'})^2}{2\alpha l^2})^{-\alpha} \tag{4.5}$$

whose results can be seen in figure 4.9.

[2]A range within which we can be find our values of interest

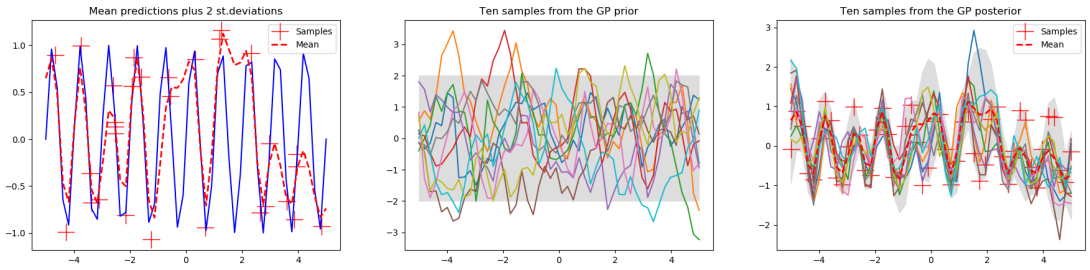


Figure 4.9: Results for Rational Quadratic Kernel

Based on how we think the shape of our function might be by observing the distribution of data, we could also consider the periodic kernel which takes the mathematical form

$$k(x, x`) = \sigma^2 exp(-\frac{2sin^2(\pi|x - x`|/p)}{l^2}) \tag{4.6}$$

In our example we considered a Sine function, which is periodic. Hence it is intuitive that a periodic kernel would be a near perfect fit for our model, whose results are shown in figure 4.10.

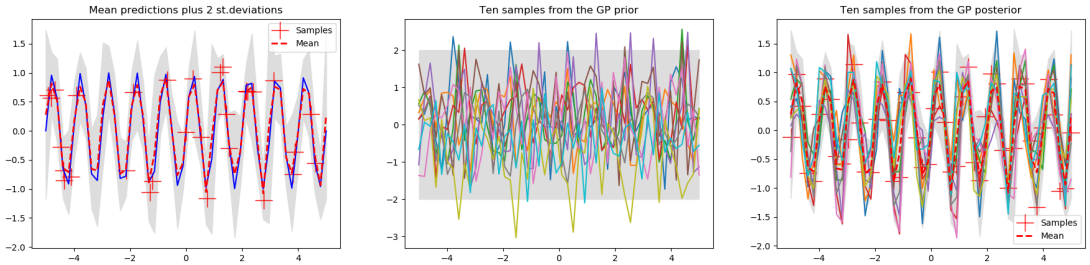


Figure 4.10: Results for Periodic Kernel

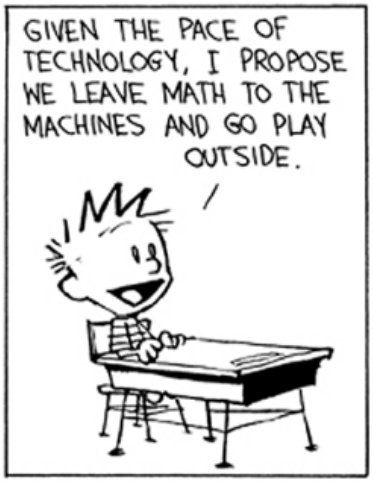
Summary

In this report I have tried my best to explain what the *Function-Space* view is and how it can be used for Regression problems using Gaussian Process.

To reach this, I have structured my report by first introducing what Regression problems are, proceeded with simple Regression methods that are mere curves fitted amongst our observations, shown their disadvantages and followed up with more robust methods that utilize parameters or weights and correspondingly the *Weight-Space* view to solve Regression problems which involves the Bayesian methodology of introducing probabilities and assumptions to support our observations and make predictions from them. The disadvantages of the previous methods points to a relatively computationally lax and generalized methods of the *Function-Space* view where Regression here utilizes the Bayesian methodology but without the parametric formulation. We then proceed to see the correspondence between these two views mathematically.

It always helps to visualize our concepts with an implementation rather than pages of mathematical formulations, hence we can also see some simple examples of Gaussian Process Regression being introduced as implementation results.

Latest research and advances on Gaussian Processes can be referred from the site http://www.gaussianprocess.org/.



List of Figures

Bibliography

[Bai16] K. Bailey. "Gaussian Processes for Dummies". In: (Aug. 2016). URL: http://katbailey.github.io/post/gaussian-processes-for-dummies/.

[Duv14] D. Duvenaud. "Automatic model construction with Gaussian processes". PhD thesis. Nov. 2014.

[Fol13] B. Foltz. "Statistics 101: Simple Linear Regression, The Very Basics". In: (Nov. 2013). URL: https://www.youtube.com/watch?v=ZkjP5RJLQF4.

[Kan17] H. Kandan. "Understanding the kernel trick". In: (Aug. 2017). URL: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78.

[MW05] H. M. Wallach. "Introduction to Gaussian Process Regression". In: (Jan. 2005).

[RW06] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Jan. 2006, p. 248.

[Sne07] E. L. Snelson. "Flexible and efficient Gaussian process models for machine learning". In: 2007.

[SSK17] E. Schulz, M. Speekenbrink, and A. Krause. "A tutorial on Gaussian process regression : Modelling , exploring , and exploiting functions". In: 2017.

[Wik19] Wikipedia. "Gaussian process". In: (Mar. 2019). URL: https://en.wikipedia.org/wiki/Gaussian_process.