UNIVERSITY OF OSLO

Data Analysis and Machine Learning
FYS-STK3155 / FYS4155

Assignment 3

Authors
Homa Priya Tarigopula
Mouhammad Abu Rasheed
Frederik Rogge

June 15, 2022

UiO : University of Oslo

Abstract

In this report, we discuss the application of Gaussian Process Regression (GPR) on a selected data from Simon Fraser University (SFU) virtual library [10], which is Dette & Pepelyshev three-dimensional exponential function. A domain of $x_i \in [0.01, 1]$ for $i = 1, 2, 3$ was chosen to do this analysis. GPR algorithm was implemented from Sickit-Learn library, and different types of kernels and kernel combinations were tried. R^2 was used to optimize the model and to define the level of uncertainty. Radial Basis Function (RBF), Constant kernel, Matern kernel, Dot-product kernel, as well as a second-order exponentiation of Dot-product kernel (a probe-specific covariance function) were used to train the model. White kernel was added to the previous functions in case of noisy models.

While high accuracy scores ($R^2 = 0.99, 0.99, 0.98$) were achieved using RBF, Matern and Exponentiated Dot-product kernels; respectively, the other form of kernel functions did not work well. The effect of adding noise to the estimations was also assessed, and shows that the Exponentiated Dot-product kernel had the best performance with ($R^2 = 0.91$) and outperformed all others including the RBF and white kernel combined. We find that the performance of the Exponentiated Dot-product kernel was also much better under the cases where the model was trained on a noisy train data and tested for true underlying values using non-noisy test set. In general, we see that the Exponentiated Dot-product kernel is less uncertain compared to the RBF kernel and more overconfident. Amongst the RBF kernel and RBF with white kernel fit and evaluated on noisy data, RBF with white kernel is more uncertain and less overconfident.

General information

The github repository consists of the following files :

- 01_visualization.ipynb - contains core analysis, all experiments discussed in report
- 02_analysis.ipynb - contains core analysis, all experiments discussed in report
- common.py - helper functions

Each of these notebook are well documented with comments.

Acronyms

GPR Gaussian Process Regression

GP Gaussian Processes

ML machine learning

MSE mean squared error

OLS ordinary least squares

RBF Radial Basis Function

SVD singular value decompositon

Introduction

In this assignment, we generate our dataset from the Dette & Pepelyshev exponential function [3] from the Simon Fraser University's virtual library. The function is defined as follows :

$$f(x) = 100\left(e^{-2/x_1^{1.75}} + e^{-2/x_2^{1.5}} + e^{-2/x_3^{1.25}}\right) \tag{1}$$

where $x_i \in [0, 1]$, for all i =1,2,3

It represents a cube within the limits for x_i as defined above. This takes a 3D input x_1, x_2, x_3 and returns the value of the function for that input. This function can be visualized in fig. 1 where the axis of the 3D plot represent the input dimensions and the color map at each location corresponds to the value of the function. The function assumes high values towards the upper limit of the sampling interval for each of the x_is.

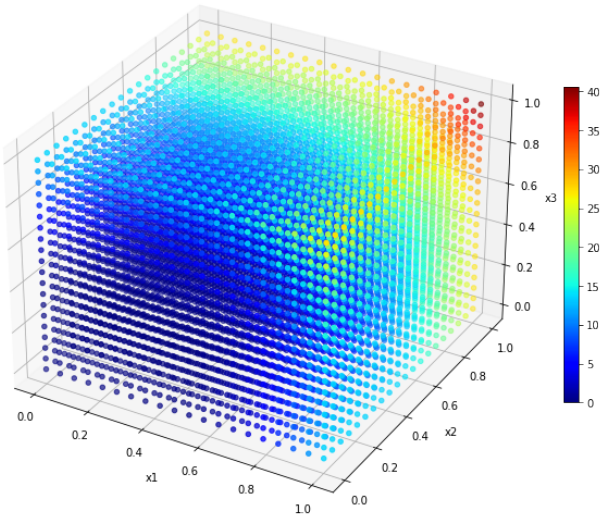

Figure 1: Dette & Pepelyshev exponential function

In this exercise, we use the dataset generated using the Dette & Pepelyshev exponential function described in eq. (1) to estimate the function values using Gaussian Process Regression.

1.1 Gaussian Processes Regression (GPR)

Linear regression is one of the extensively studied and used models for data prediction in machine learning. It offers a simple design where the output is a linear combination of the inputs. However, lack of flexibility and hyper-parameters over-dependency are major drawbacks leading to limited accuracy or overfitting.
Unlike other machine learning algorithms, Gaussian Processess (GPs) predict a distribution to reflect the state of uncertainty in predictions, making it more trustworthy in terms of decision-making based on the data. It also allows for an efficient hyper-parameter selection compared to the classic regression models.
Gaussian process regression is a Bayesian-based machine learning approach that is used particularly well in relatively small data-sets but can be used in general to provide uncertainty measurements on regression problems. Instead of assigning the parameters of the cost function through an optimization process, Gaussian Process Regressions (GPRs) infers a probability distribution over the domain of values for each parameter. Giving rise to more flexibility and providing a sign for the uncertainty in the models you have. In other words, GPRs aims for assigning the probability $p(y|X)$ rather than fitting a cost function [8]

Compared to the classic linear regression models, GPRs has several advantages [8][9]:

1. It gives high flexibility in terms of prediction as it produces a distribution of different possibilities to reflect the uncertainty of the model.

2. GPs can be implemented in a wide range of domains, from the simple scalar inputs to the complex graphic ones. While it only require the outputs to be scalar, it has a high flexibility in terms of the inputs nature.

3. It enables more efficient hyper-parameters selection compared to the classic linear regression algorithm.

There are several ways to explain GPs in regression problems. It could be defining a distribution over functions for each data-point and predicting a model directly from a space of functions (function-space view), or it could be by considering the function as a parametrized function with a weight vector w and having the distribution calculations over these parameters (weight-space view) [8].

1.2 GPR Algorithm

One basic hypothesis is known as Bayesian Occam's Razor (BOR) which states that the more the model is complex, the more flexibility it would get. Flexibility in this sense means that it can explain wider range of datasets [1].
As a rule of thumb, the sum of all probabilities for a model to describe all the possible datasets must sum up to 1 (Eq.2), and, consequently, flexible models that explain too many datasets must necessarily assign each of them a comparatively low probability. fig. 2 shows how the probability of different models to explain a given dataset works. Model-1 is too simple and unlikely to generate this data. Model-3 is overfitting and has low marginal probability. Model-2 in turn is the right model as it has the highest marginal likelihood [6].

$$\sum_{d=1}^{D} p(d|M) = 1 \tag{2}$$

GPR is a generative model that generate several functions, which can possibly fit the data we have and assign a probability distribution for each model to help choosing the most possible one. This probability distribution will also help assigning a range of predictions for the input that gives high probability rather than a specific point. This is the basic idea to explain how GPR works.
GPR follows the same aspects of Bayesian linear regression, with a primary generalization that it starts with a sample of some functions rather than linear functions.
The general approach that GPR follows are briefly as follows [7]:

1. Prior samples of functions will be generated and selected via some Kernel function. The Kernel function is chosen such that it decides how different inputs are similar. GP will then sample the functions that give close predictions (y_i) to the X values deemed similar by the kernel. Kernel selection is a crucial step in having a good convergence. There are several types of kernels that can be used, with the possibility to combine them together to better predict the data. Kernel selection and types will be further discussed later.

2. Bayes' theorem will be used to update the prior samples to Posterior predictive samples based on evidence from the data we have (Eq.3, 4).

3. Noise variance might/might not be added to posterior samples, and posterior predictive distribution will be generated.

4. Hyper-parameters of the kernel function as well as the noise will be selected such that it maximizes the likelihood of y for an infinite number of given prior functions f. In other words, hyper-parameters are selected such that they make the prior functions fit the data best. In practice, this can be done by using the function's values over an arbitrary finite set of points.

$$p(W|y,X) = \frac{p(y|X,W) \times p(W)}{p(y|X)}; posterior = \frac{likelihood \times prior}{marginallikelihood} \tag{3}$$

$$p(y|X) = \int p(y|X,W) \times p(W) \times dW \tag{4}$$

1.3 Mathematical Aspects of GPR [7]

Diving into the mathematics, let us consider the simple form of linear regression with a data-set $D = \{(X_i, y_i)\}_{i=1}^{N}$, where (X_i, y_i) is a datapoint with prediction as y_i and the dataset consists of N datapoints. The algorithm's task then is to provide predictive distributions of M test inputs $\{X_i{}^*\}_{i=1}^{M}$

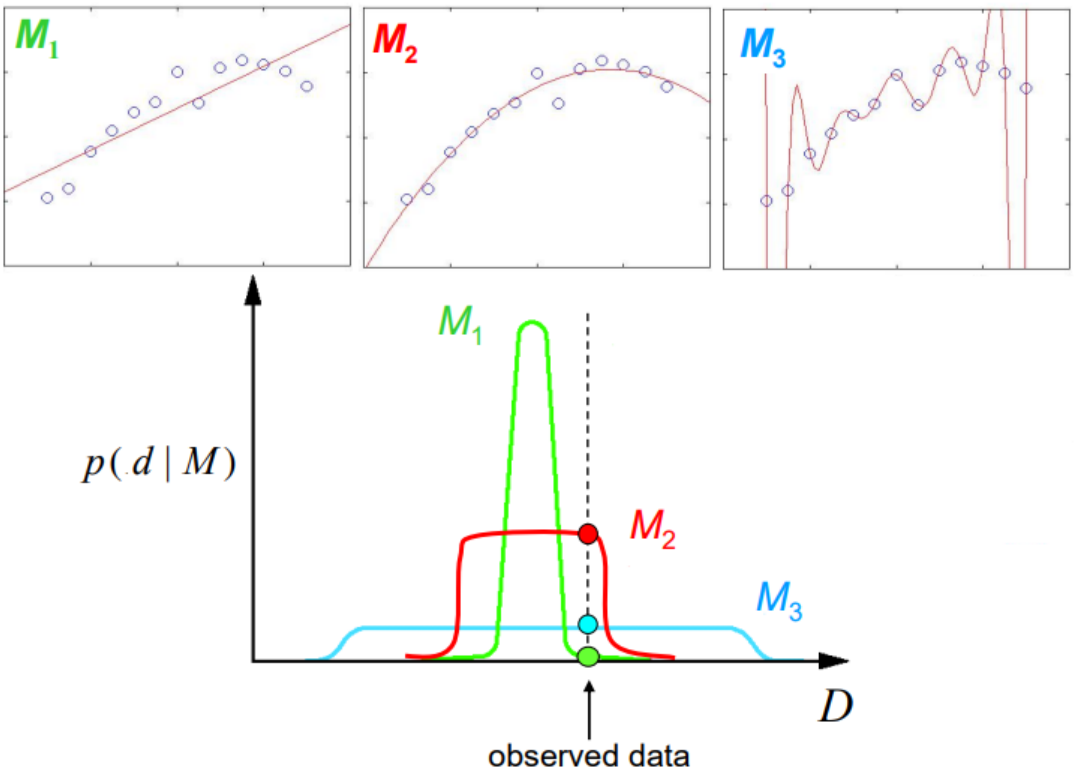

Figure 2: Marginal probabilities of different models in terms of flexibility.

We can assign a function $f(x)$ to describe the data D as follows (Eq.5):

$$y_i = f(X_i) + \epsilon_i; \text{ where } \epsilon_i \sim \mathcal{N}(0, \sigma_\epsilon^2) \tag{5}$$

In a more compact form, we can rewrite with X, X^*, y, f, f^*, where X, X^* are the features matrices of the whole and testing data-sets; respectively. y is a vector of labels (the outputs); f, f^* refers to the true function for the inputs (model predictions).
A Gaussian process is completely specified by its mean function and covariance function. We define mean function $m(x)$ and the covariance (Kernel) function $\kappa(X, X')$ of a real process $f(X)$ as:

$$m(x) = \mathbb{E}[f(X)] \tag{6}$$

$$\kappa(X, X') = \mathbb{E}[(f(X) - m(X)) \times (f(X') - m(X'))] \tag{7}$$

The Gaussian process then can be written as:

$$f(X) \sim \mathbb{GP}(m(X), \kappa(X, X')) \tag{8}$$

The mean function is commonly set to zero in practice, as the Gaussian process is very effective in modulating the mean of the prior based on the data. GPR algorithm starts by a basic assumption that vectors y, and f^* are $(N + M)$-dimensional multivariate normal (joint distribution) with a zero-mean and a covariance matrix made of kernel similarities between all inputs (the covariance matrix is an $(N + M) \times (N + M) - matrix$) (Eq.9). $K_{X,X}$ is an $N \times N$ matrix of all similarity scores out of the chosen kernel $\kappa(X_i, X_j|\tau)$. K_{X^*,X^*} is the same for the testing data.

$$\begin{bmatrix} y \\ f^* \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \hat{K}_{X,X} & K_{X,X^*} \\ K_{X^*,X} & K_{X^*,X^*} \end{bmatrix}) \; ; \quad \text{where} \quad \begin{aligned} \hat{K}_{X,X} &= K_{X,X} + \sigma_\epsilon^2 I \text{ , for noisy observations} \\ &= K_{X,X} \text{ , for noise-free observations} \end{aligned} \tag{9}$$

By applying the GP algorithm, and depending on whether the model chosen is a noisy or noise/free model, the posterior probability as well as distribution parameters are given in Equations 10, 11.

$$p(f^*|X^*, X, y) = \mathcal{N}(f^*|\mu^*, \Sigma^*) \tag{10a}$$

$$\mu^* = K_{X^*,X} \times \hat{K}_{X,X}^{-1} \times y \tag{10b}$$

$$\Sigma^* = K_{X^*,X^*} - K_{X^*,X} \times \hat{K}_{X,X}^{-1} \times K_{X,X^*} \tag{10c}$$

$$p(f^*|X^*, X, f) = \mathcal{N}(f^*|\mu^*, \Sigma^*) \tag{11a}$$

$$\mu^* = \mu(X^*) + K_{X^*,X} \times \hat{K}_{X,X}^{-1} \times (f - \mu(X^*)) \tag{11b}$$

$$\Sigma^* = K_{X^*,X^*} - K_{X^*,X} \times \hat{K}_{X,X}^{-1} \times K_{X,X^*} \tag{11c}$$

where; Eq. 10 are for noisy models, and Eq. 11 are for noise-free models. Figure 3 shows an example of how sample functions can be conditioned in the posterior based on the observed data.

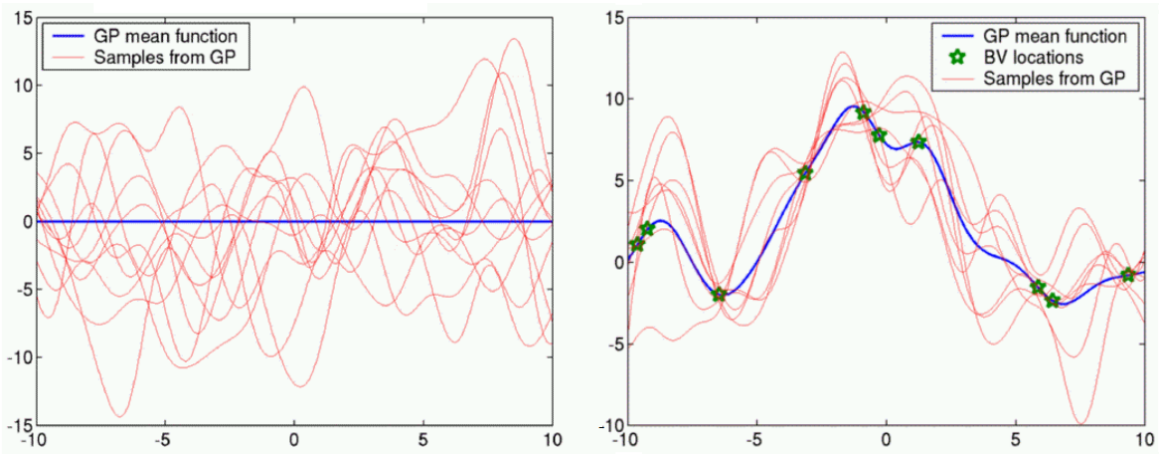

Figure 3: Gaussian process inference. (left) Samples from a zero-mean GP prior using RBF kernel, (right) Samples from a posterior GP. Figure retrieved from [9]

1.4 Kernels [7, 8]

We have seen that choosing the Kernel function plays a major role in GPR algorithm, as it affects the whole predictive performance of GPs. Kernel is a real-valued function of two arguments, $\kappa(x, x') \in R$, for $(x, x') \in X$. In general, the function is symmetric (i.e., $\kappa(x, x') = \kappa(x', x)$), and non-negative (i.e., $\kappa(x, x') \geq 0$), so it can be interpreted as a measure of similarity. The higher the number returned by the kernel is, the more similar the two inputs are.
What the Kernel does is basically telling the model how similar two data points (X, X') are. Several kernel functions are available for use with different types of data. These functions can be used separately, or as combinations for more complex trends. We will briefly discuss some of these functions below.

1.4.1 Squared Exponential Function

Squared exponential function, also known as RBF or Gaussian kernel, is one of the most commonly used kernel in GPR for 1D-inputs. The function form is given in Eq.12, and fig. 6 shows an example of the samples this function can produce.

$$\kappa(x, x'|\tau) = \sigma^2 \exp\left(-\frac{1}{2}\left(\frac{x - x'}{l}\right)^2\right) \tag{12}$$

The symbol τ is a two-element vector that includes the hyper-parameters (l, σ) that can be modified in this function. l is the length scale that define the length between the two inputs to be considered similar. The lower

l is, the less x values will be considered similar. On the other hand, σ is the output scale which determines the scale of y values of the samples of functions. The higher the σ is, the higher the spin of the sampled functions will be in YY' axis. fig. 4 and fig. 5 show a representation of prior function when modifying l and σ; respectively.

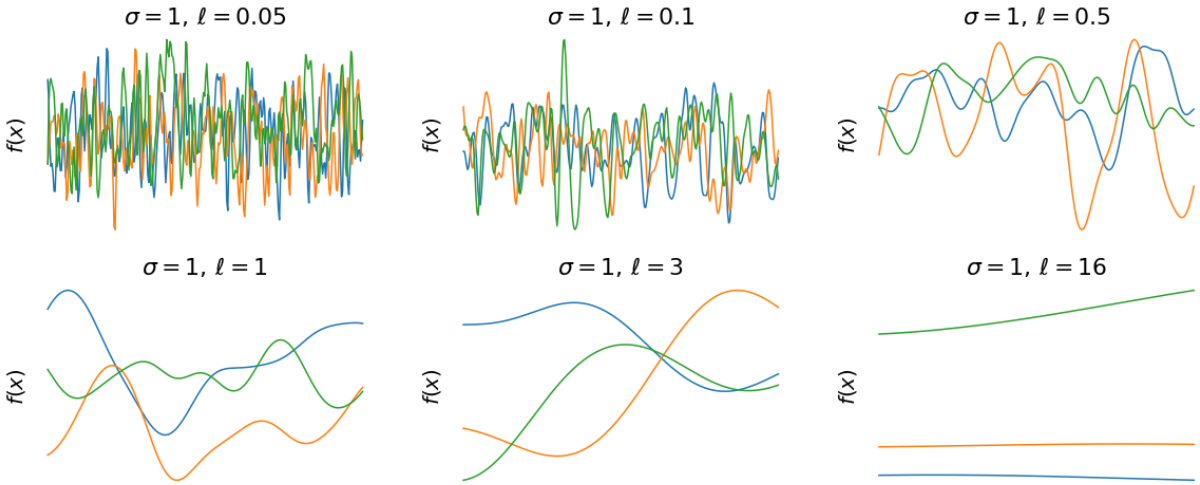

Figure 4: Prior samples of functions using RBF kernel with different length scale l and constant σ. Figure retrieved from [5]

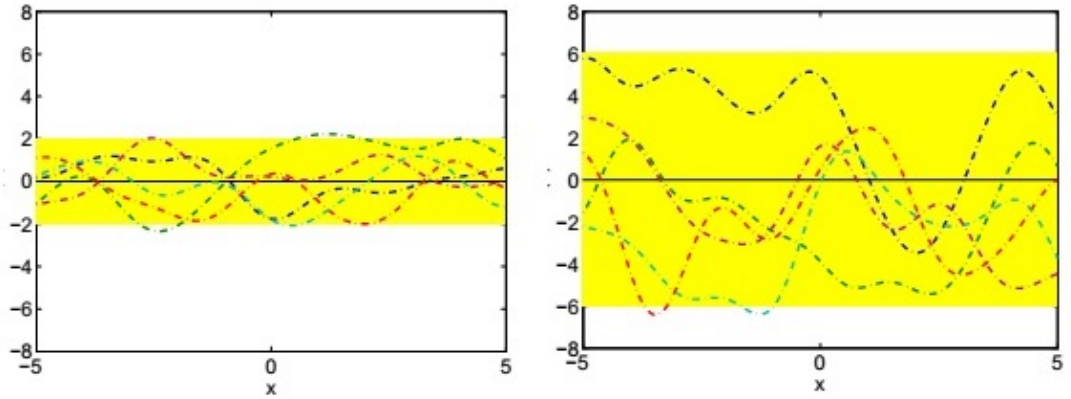

Figure 5: Prior samples of functions using RBF kernel with different output scale σ and constant l. Figure retrieved from [2]

1.4.2 Periodic Function

This kernel allows a repeatable function after a specific parameter. Which means that inputs with a specific distance apart will be considered similar (fig. 6). Eq.13 shows its form. p is the period hyper-parameter which determines this distance of the prior samples. The other hyper-parameters (l and σ^2) have similar effects on the periodic function as in the RBF.

$$\kappa(x, x'|\Upsilon) = \sigma^2 \exp\left(-\frac{2}{l^2}\sin^2\left(\frac{\pi|x - x'|}{p}\right)\right) \tag{13}$$

1.4.3 Linear Function

Also known as Dot-Product Kernel, this function is the simplest form of prior sampling, which produces samples of linear functions (fig. 6). This kernel is usually used in combination with other kernels to produce better priors, while when it is used alone, the prior produced will be similar to that of Bayessian Linear Regression's algorithm. The form of this kernel is shown in Eq.14. The offset c determines the lines' intercept of the priors, the term ${\sigma_b}^2$ determines the certainty around c, while ${\sigma_v}^2$ determines the average distance away from the function's mean (the slope range).

$$\kappa(x, x'|\upsilon) = {\sigma_b}^2 + {\sigma_v}^2(x - c)(x' - c) \tag{14}$$

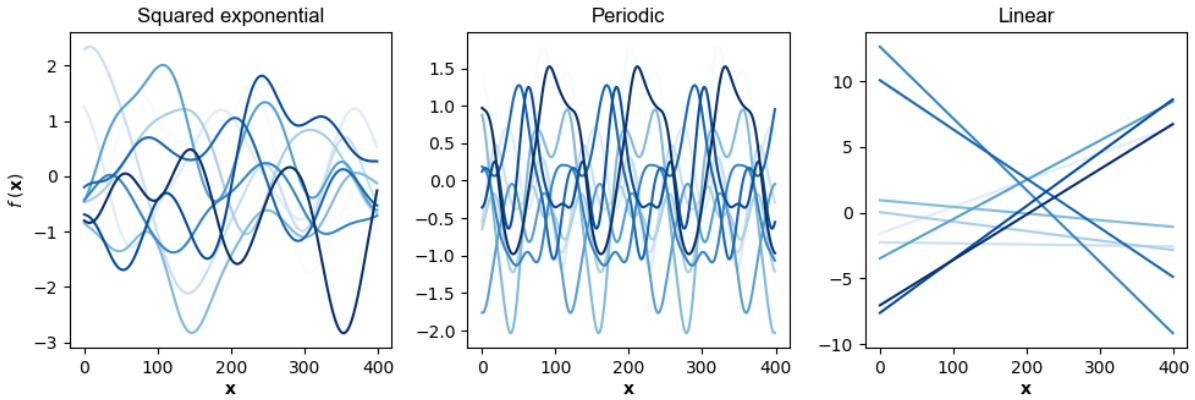

Figure 6: Prior samples of functions from a Gaussian process using different kernels. Figure retrieved from [4]

1.4.4 Kernel Combinations

In more complex models, a combination of some kernels can be used to produce better priors. This combination can simply be a summation of two other kernels $\kappa_{sum}(x, x') = \kappa_1(x, x') + \kappa_2(x, x')$, or a multiplication kernel $\kappa_{product}(x, x') = \kappa_1(x, x') \times \kappa_2(x, x')$, or an exponentiation kernel, which uses some other base kernel and a scalar parameter p and combines them exponentially via $\kappa_{exp}(x, x') = \kappa_{base}(x, x')^p$.

1.4.5 White Kernel

This kernel is used in the form of sum-combination with other kernels to explain noisy models, and it is given in the form of Eq.15, where noise-level equals the variance of this noise.

$$\begin{aligned} \kappa(x, x') &= noise - level \qquad if \quad x == x' \\ \kappa(x, x') &= 0 \qquad\qquad\qquad\quad otherwise \end{aligned} \tag{15}$$

1.5 Hyper-Parameters Selection

Kernel optimization in SVMs usually uses exhaustive search over a discrete grid of values for all parameters. This approach can be computationally expensive if applied to GPR. Instead, hyper-parameters can be selected such that they maximize the marginal likelihood given in Eq.16. In other words, it picks the set of hyper-parameters that makes the prior samples explain our dataset best (before doing any further conditioning).

$$\log p(y|X) = \log \mathcal{N}(y|0, \kappa_y) = -\frac{1}{2} y \kappa_y^{-1} y - \frac{1}{2} \log |\kappa_y| - \frac{N}{2} \log(2\pi) \tag{16}$$

Results & Discussion

This section discusses results of various experiments that were carried out during this project. If not stated otherwise, a dataset of size 3375 samples was used (15 steps in each dimension in the range $0 < x_i <= 1$ for $i \in \{1, 2, 3\}$). Out of this, 70% were used for training and the remaining 30% for testing.

2.1 Kernel function

We used a total of four different kernels, namely an RBF kernel, a matern kernel, a dot product kernel and a combination of a dot product kernel and an exponentiation of 2. While the RBF, matern and dot-product kernel are *off-the-shelf* kernels, the exponentiated dot product is a combined one. Here, we included some prior knowledge about the shape of the function. We know that the function has an exponential shape in all 3 dimensions, e.g. exponentially increasing function values for increasing x_is. One property of the combination of dot product and the exponentiation kernel is the covariance of function values over long distances which is likely to result in consistently increasing or decreasing values as opposed to oscillating patterns.

The r-square value on the test set for each of the kernels is shown in fig. 7. It can be seen that the RBF kernel, the matern kernel and the dot product with exponentiation all achieve an r square scores of almost 1. In contrast, using only the dot product kernel yields a score of just 0.74. It turns out that the function is fairly easy to predict for most of the kernels shown by the high r square values. On the other hand, it also shows that using our prior knowledge about the shape of the function yields high results and greatly improves the performance of the dot product kernel when combined with the exponentiation. In the following analysis, we will focus on the RBF kernel and the exponentiated dot product kernel.

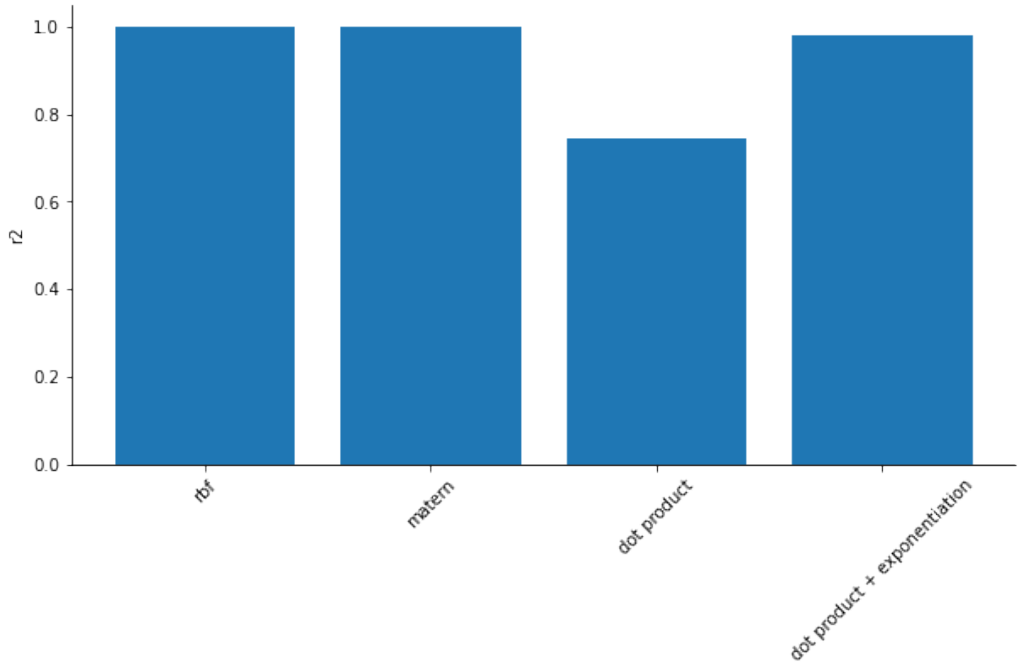

Figure 7: R square value on test set for different kernels.

2.2 Uncertainty and Generalization

Since the GPR model provides a probability distribution (a normal distribution) for each of the function values that we want to predict, we can measure the uncertainty of our prediction for each of the values which is given by the standard deviation. Figure 8 shows the uncertainty (represented by the standard deviation) at each datapoint for the model using the RBF kernel. It can be seen that the uncertainty is low for all datapoints (<0.2 standard deviations) suggesting a high confidence in the prediction. The r square value for this model is 1. The right plot in fig. 8 shows the same but for a model only trained on training samples for which $x_3 < 0.5$. The resulting r square value is now 1 for test samples with $x_3 < 0.5$ and -1.8 for test samples with $x_3 >= 0.5$. This shows that the model does not generalize well beyond the limits of the training samples. This is also

reflected in the uncertainty which gets higher for increasing x_3 in the region of $x_3 >= 0.5$, hence the model gets more uncertain about its predictions the further the samples are away from the training range. This is expected as GPR works good for interpolation between datapoints. However, the further we move away from the training region, the more will our predictions be driven by our prior. Here, the prior does not reflect the behavior of the function outside the training region at all and we are worse off than simply predicting the mean of the function values (which would result in an r square of 0).

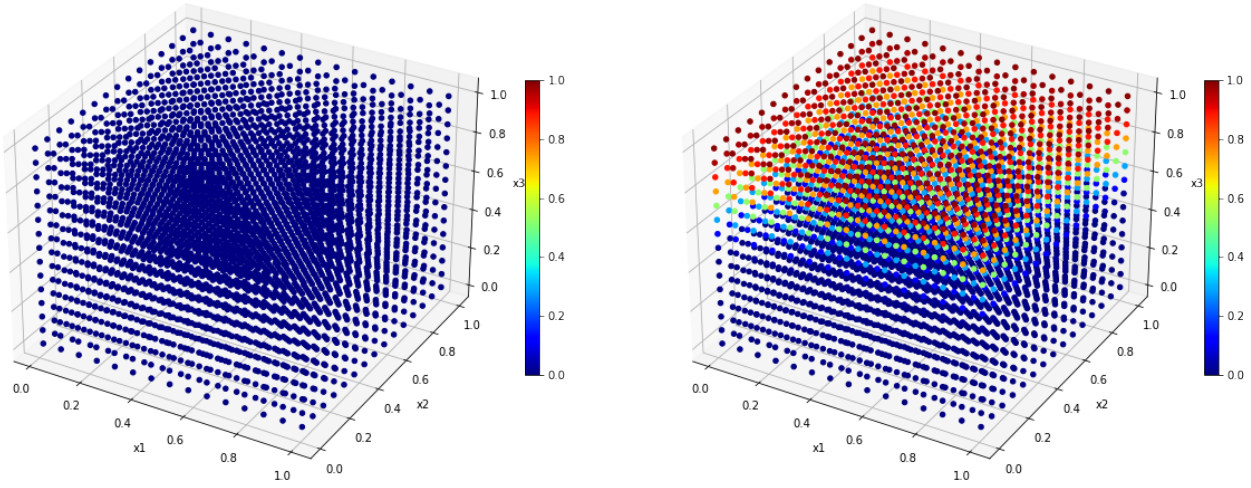

Figure 8: Uncertainty (represented by the standard deviation) of all datapoints for model with **RBF kernel** trained on the full training set (left) and one trained on training data with $x_3 < 0.5$ (right).

For comparison, we performed the same experiment for our designed prior (exponentiated dot product kernel). Here the r square values are 0.98 for the case where we trained on all the full range, 0.97 for $x_3 < 0.5$ and 0.33 for $x_3 >= 0.5$ when trained only on training samples with $x_3 < 0.5$. This shows that in this case our prediction is much better compared to the case with RBF kernel in the region without training samples, though the result is still much lower compared to the region with training samples. Figure 9 shows that the uncertainty did not increase much after leaving out half of the training set in contrast to what we have seen in the RBF case. This means we are both better but also more confident about in our results.

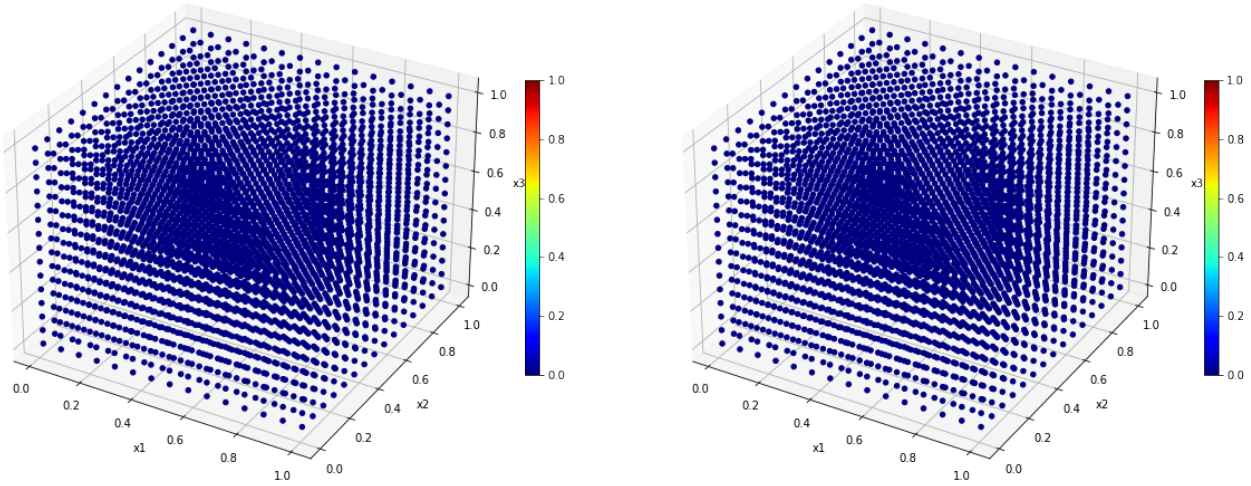

Figure 9: Uncertainty (represented by the standard deviation) of all datapoints for model with **dot product** and **exponentiation** trained on the full training set (left) and one trained on training data with $x_3 < 0.5$ (right).

Generally, one would want have a high prediction error to be reflected in a high uncertainty and low prediction

error in a low uncertainty. This would at least give use the opportunity to makes statements about how confident we are in the predictions we are making. If the model predicts a high standard deviation, we would know that we could likely make a high prediction error and vice versa. For real life examples, the standard deviation or uncertainty is all we have since we will not be given the ground truth data that the model is tested on. This raises the question if this relation actually holds true, or whether there are regions in which we are overconfident, e.g. we have a high prediction error even though we are very certain about the result. We study and discuss this in the next section.

2.3 Overconfidence

We measure the level of overconfidence in terms of the absolute value of the z-score of the true function value. Since our model outputs the mean μ_i and standard deviation σ_i of our prediction, we can represent the level of overconfidence as the absolute distance (the prediction error) of the true function value y_i from the mean prediction in terms of standard deviations using the following formula

$$z_i = \frac{|\mu_i - y_i|}{\sigma_i}. \tag{17}$$

Figure 10 shows this value for all test datapoints using the RBF kernel for both of the discussed cases in the previous section. It can be seen that for the model trained on the full range of datapoints (left), the score is generally smaller with most function values being within in range of 2 standard deviations away from the mean prediction. As a general trend, it can be seen that the level of overconfidence increases a bit with increasing values for all x dimensions. For the model trained on only half of the data (with $x_3 < 0.5$ (right), we observe that the model is highly overconfident for values further outside the training region with score of more than 10 standard deviations. This shows that even though our uncertainty about the prediction also increased (as shown in fig. 8), this still does not fully reflect how low the performance actually is for these datapoints (as indicated by the low r square value of -1.8 as discussed in the previous section).

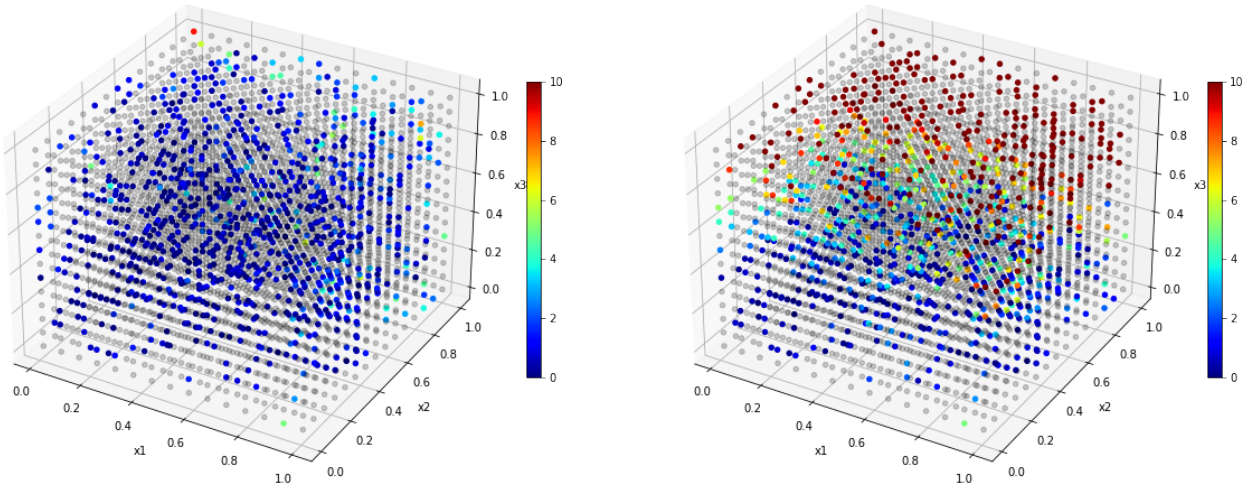

Figure 10: Level of overconfidence for the test samples represented by the absolute distance of the true function value to the mean prediction in terms of standard deviations. Shown for the **RBF kernel** trained on the full range of training samples (left) and training samples with $x_3 < 0.5$ (right). Training samples for which the overconfidence was not computed are shown in light grey.

When we look at the same scenarios for the exponentiated dot product kernel in fig. 11, we see that we are generally overconfident everywhere as indicated by the very high scores. However, these scores are not due to high prediction errors, but rather the very low level of uncertainty.

2.4 Noise

In this section, we will study the influence of noise on the predictions and the uncertainty. For this purpose, we will first add noise drawn from a standard distribution with 0 mean and a standard deviation of 2 to the entire dataset (including training and test set) in the first scenario. As before, we will use the RBF kernel and

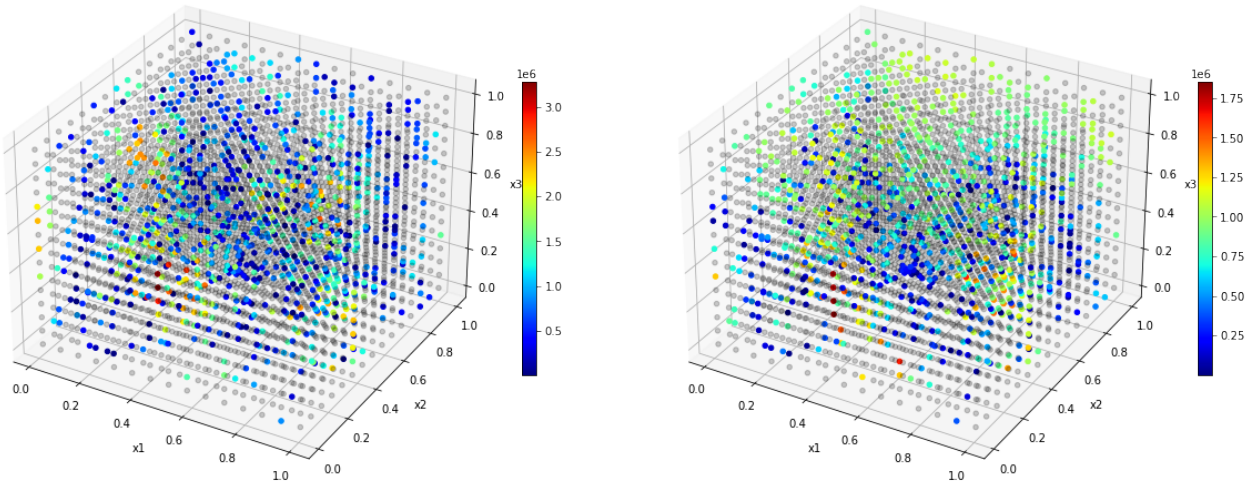

Figure 11: Level of overconfidence for the test samples represented by the absolute distance of the true function value to the mean prediction in terms of standard deviations. Shown for the **exponentiated dot product kernel** trained on the full range of training samples (left) and training samples with $x_3 < 0.5$ (right). Training samples for which the overconfidence was not computed are shown in light grey.

the exponentiated dot product kernel. In addition to that, we will study adding a white kernel to each of the kernels that models the added noise.

Figure 12 shows the r square values for each of these different models. Generally, it can be seen that the performance is worse than in the case without noise. For the RBF kernel, adding the white kernel improved the results. This means that here the white kernel acted as a regularization that prevents the GPR from adapting too much to the noise.

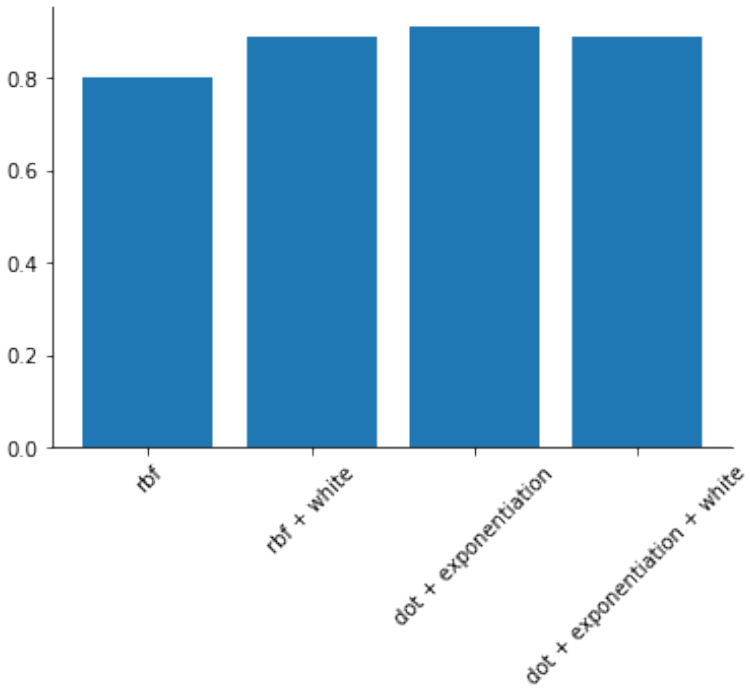

Figure 12: R square for different kernels on test set from a dataset with added noise sampled from a standard distribution with 0 mean and a standard deviation of 2.

The model does not actually get better at predicting the noise as it is impossible to predict it due to its

random and independent nature. For the exponentiated dot product kernel, the results are best (and best among all) without the added white kernel.

Next, we wanted to understand the impact of explicitly modeling the noise (through the white kernel) on the uncertainty and level of overconfidence. We studied this using the RBF kernel. Figure 13 shows the uncertainty (top) and the level of overconfidence (bottom) for the RBF kernel (left) and the RBF kernel with added white kernel (right). By looking at the case with only RBF kernel, it can be seen that the uncertainty is higher for the test samples (which are the colored ones in the plot below that). However, all in all the uncertainty is still low (mostly < 1.0) given the high standard deviation of the noise of 2. This mismatch is also reflected in the level of overconfidence (bottom left) that increased significantly compared to the case without noise (compare fig. 10). On the other hand, when looking at the case with added white kernel (right), it can be seen that the uncertainty increased a lot across all datapoints. This makes sense since the noise is random and not predictable and should therefore be reflected in the uncertainty about the prediction. The fact that now the model reflects this uncertainty makes it more realistic in its predictions and therefore, less overconfident. This can be seen in the plot below which shows an overall decreased level of overconfidence (bottom right) compared to the case without added white kernel.

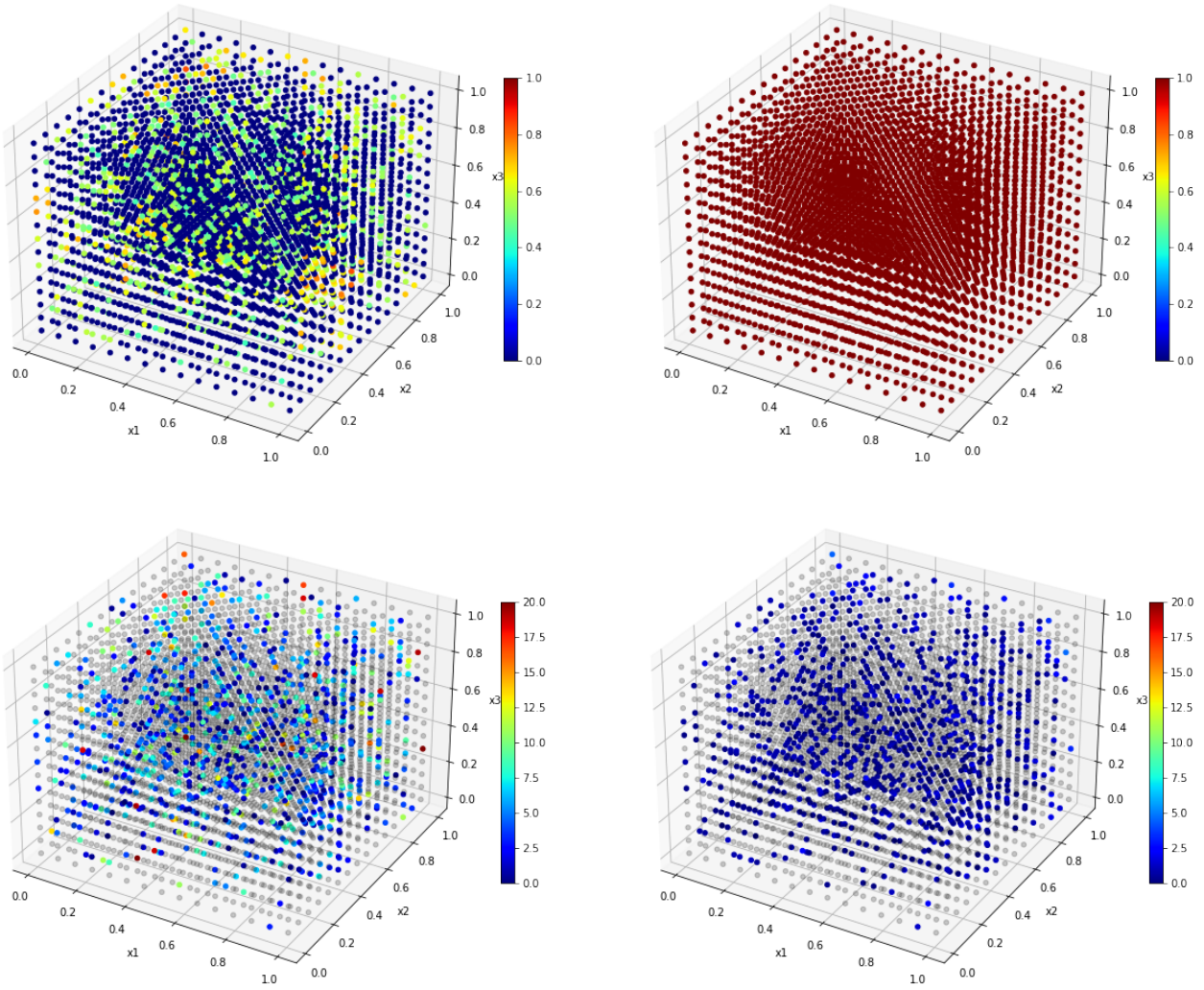

Figure 13: Uncertainty (top) and level of overconfidence (bottom) for the **RBF kernel** (left) and the **RBF and white kernel** (right). Gaussian noise with mean 0 and standard deviation of 2 was added to both the training and test samples. Uncertainty is measured as the standard deviation of the prediction. Overconfidence is measured as the absolute distance of the true value from the mean prediction in terms of standard deviations (see eq. (17))

In the last experiment, we trained a model on a noisy training dataset and tested it against the true underlying function values using a non noisy test set. Again, we compare the RBF kernel with the dot product

combined with the exponentiation kernel. The r square values on the test set can be seen in fig. 14. It shows that the performance of the dot product kernel is much better (0.98 vs 0.86). Hence, the dot product kernel is good at predicting the underlying true data generating function regardless of the noise added to the training samples. This again shows that this is a reasonable choice of a prior for this problem. A possible explanations is that the RBF kernel is adapting to the noise too much as it can easier model variations over shorter distances while the dot product kernel has long distance covariations. These results stress the importance of a proper choice of a prior which is the crucial design step in GPR modeling.

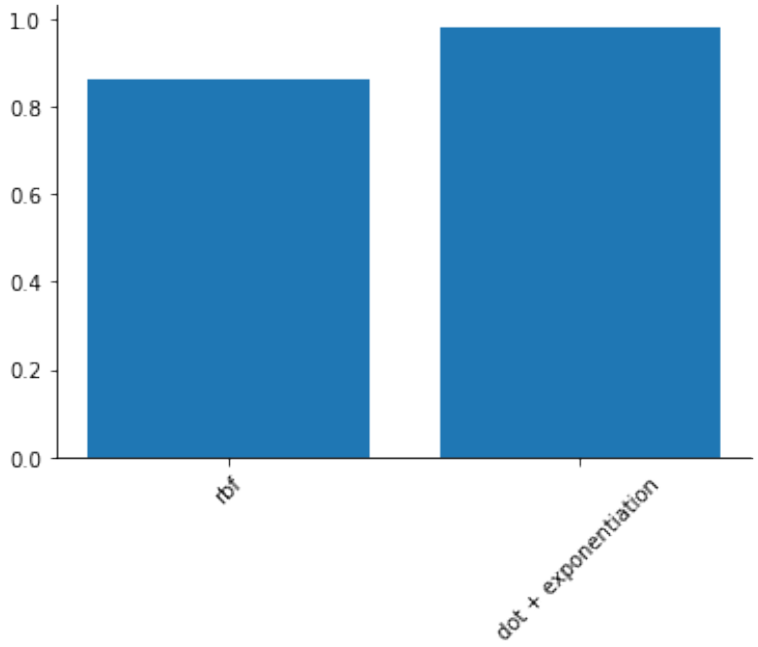

Figure 14: R square values for different kernels on non noisy test when trained on noisy training set (mean of 0 and standard deviation of 2)

References

[1] T. Blanchard, T. Lombrozo, and S. Nichols. Bayesian occam's razor is a razor of the people. *Cognitive Science*, 42:1345–1359, 2018. URL http://www.GaussianProcess.org/gpml.

[2] M. R. Chernick. what effect varying 2f and l2 has on a gaussian process model with rbf kernel? @ONLINE, 2017. URL https://stats.stackexchange.com/questions/275164/what-effect-varying-sigma-f2-and-l2-has-on-a-gaussian-process-model-wit.

[3] H. Dette and A. Pepelyshev. Generalized latin hypercube design for computer experiments. *Technometrics*, 52(4):421–429, 2010.

[4] G. Gundersen. Gaussian process regression with code snippets @ONLINE, 2019. URL https://gregorygundersen.com/blog/2019/06/27/gp-regression/.

[5] R. Guzman. Gaussian processes explained @ONLINE, 2018. URL https://relguzman.blogspot.com/2018/03/gaussian-processes-explained.html.

[6] B. Moghaddam. *Nonparmeteric Bayes Gaussian Processes*. CalTech, Lecture17. URL https://sites.astro.caltech.edu/~george/aybi199/OldLectures/Lec17_Moghaddam.pdf.

[7] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, London. URL http://noiselab.ucsd.edu/ECE228/Murphy_Machine_Learning.pdf.

[8] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT press, Massachusetts Institute of Technology. ISBN 026218253X. URL http://www.GaussianProcess.org/gpml.

[9] D. B. Scribe, V. Narayanan, K. M, and P. P. *Gaussian Processes*. Statistical Techniques in Robotics, (16-831, F12) Lecture21. URL http://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture22_venkat_koval_parashar.pdf.

[10] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets @ONLINE, 2013. URL https://www.sfu.ca/~ssurjano/detpep10exp.html.