Gaussian Process Regression with Motion Capture Data

Joel Walsh
University of Texas at Austin
joelawalsh@utexas.edu

Abstract

Gaussian Process Regression is a form of time series regression that exploits several unique properties of Gaussian distributions in order to model stochastic processes. In this paper we will show how Gaussian processes can be fit by a sliding window technique in order to obtain local hyper parameters.

1 Introduction

Gaussian processes are a tool that can be used for both regression and classification. They can best be thought of as a distribution over functions. They exploit important aspects of Gaussian distributions; namely closure under addition and multiplication. So if you multiply a Guassian by a Gaussian, the result is also Gaussian. The same is also true for addition. Gaussians also form a conjugate family with themselves, meaning that you can calculate a posterior distribution using a Gaussian prior

$$N\left(\boldsymbol{\theta}_0, 1/\tau_0\right)$$

and Gaussian likelihood

$$N\left(\boldsymbol{\theta}, 1/\tau_*\right)$$

, the result will be Gaussian.

$$N\left(\frac{\tau_0}{\tau_0+n\tau_*}\theta_0+\frac{n\tau_*}{\tau_0+n\tau_*}\bar{y}_*, 1/\left(\tau_0+n\tau_*\right)\right) \quad (1)$$

These properties allow for a surprisingly large amount of customization via kernel functions.

2 Methods

For this analysis, the Radial Basis Function or RBF Kernel was used, with an added noise term. The term δ_{pq} refers to the Kroennecker delta, which is equal to one if the entries in the matrix are equal and zero else. T

$$\operatorname{cov}\left(y_p, y_q\right)=k\left(\mathbf{x}_p, \mathbf{x}_q\right)+\sigma_n^2\delta_{pq} \text{ or } \operatorname{cov}(\mathbf{y})=K(X,X)+\sigma_n^2 I \quad (2)$$

he RBF was multiplied by a constant term σ_n^2 as well. These are all part of the kernel functions included within the Python package GaussianProcesses as ConstantKernel, RBF, and WhiteNoise. The first pass trained using a randomly sampled, 1030 frame vector of the x-postion of a finger; obtained through a motion capture suit. Throughout this frame of time one subject completed a traced circle with their finger five times. Those five traces were randomly sampled at each time frame to create a training set.

The second pass used this same randomly sampled vector. A sliding window technique was applied, meaning that local hyperparameters were calculated and stored within a vector. They were then plotted, which showed distinct intervals of time that could benefit from being enclosed within windows. After experimenting with a variety of window lengths, 20 frames seemed to give the most stable parameters. From this plot a few trends emerged that pointed to a specific demarcation of the frame vector.

3 Results

When using the entire random position vector to estimate a global kernel, the following plot (Figure 1) was obtained. The yellow band shows the prediction band, and the red line shows the random position vector. The yellow band was obtained via the following predictive distribution:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(X,X)+\sigma_n^2 I & K\left(X, X_*\right) \\ K\left(X_*, X\right) & K\left(X_*, X_*\right) \end{bmatrix}\right) \quad (3)$$

The five scatterplots in the background represent each of the five traces.

Using the aforementioned sliding window technique, and window sizes of varying size, hyperparamters for each window were obtained. The length scale paramter was perhaps the hardest to get to react in a stable manner, as it increased asymptotically at certain parts of the graph. Luckily the package had ways to restrict its value. Also, adjusting the window size eventually led to more stable hyperparamter values. Figure 2 shows how plotting these hyperparameters throughout the window pointed to three distinct sections of the graph where hyper parameters tended to acheive some measure of stability, cordoned off by dashed black vertical lines. The

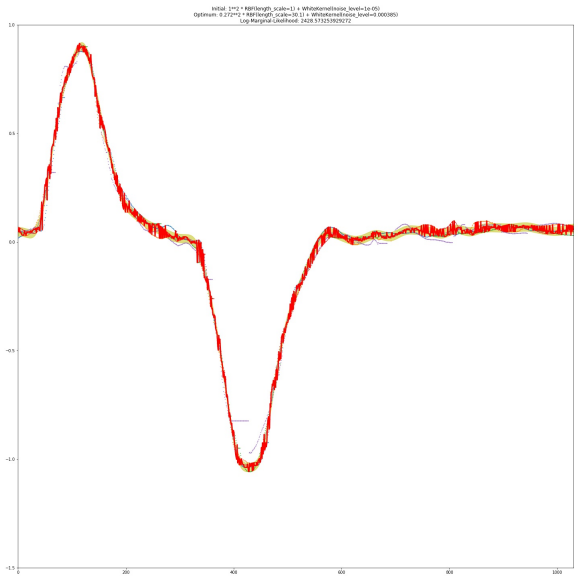

Figure 1: Predictive distribution using global kernel hyperparameters.

Table 1: Hyperparameters over three intervals

Parameters	(0-300)	(301-600)	(600-1030)
Sigma	0.331	0.432	0.045
Length Scale	55.68	54.48	94.06
Noise level	.00042	0.00046	0.00013

blue lines correspond to the length scale, which was espe cially unruly. The magenta line refers to the constant, an the flat noise term barely oscilated between -1 and 1. Table shows the average value of hyperparameters over these inter vals.

Once this division of the time interval became apparen the local hyperparamters were found interval by interval. Th prediction bands were calculated seperately, resulting in th distribution shown by Figure 3.

A truncated x- axis shows a more detailed view, in Figure 4 and 5.

4 Style and Format

After calculating the respective log marginal likelihoods, th global kernel yielded 2428.57 while the local kernel yielde 2526.86 units, a difference of 98.29. While this does mark a improvement, it's hard to tell how significant that is. If I ha more time I might have devised an interative method to max imize the difference in log marginal likelihoods by adjustin the window partitions.

5 Summary

Gaussian processes can be quite flexible models to complet time-series regression, with many well understood packages and kernel functions existing on popular software packages. While they can quickly become expensive, they are still a use- ful tool in Bayesian inference.

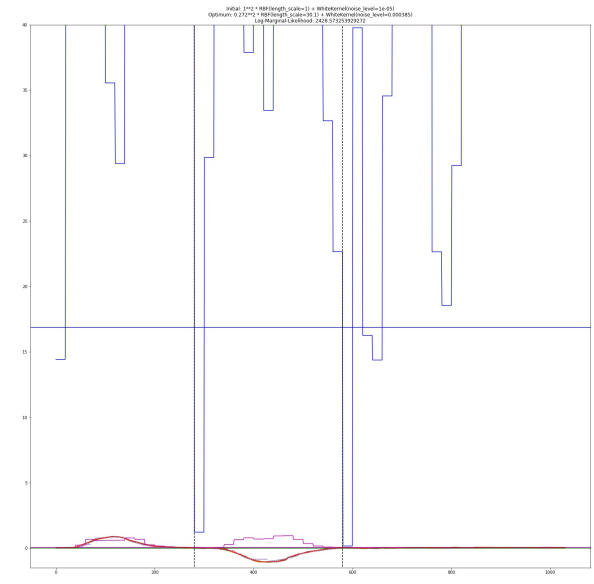

Figure 2: Kernel hyperparameters obtained via sliding window.

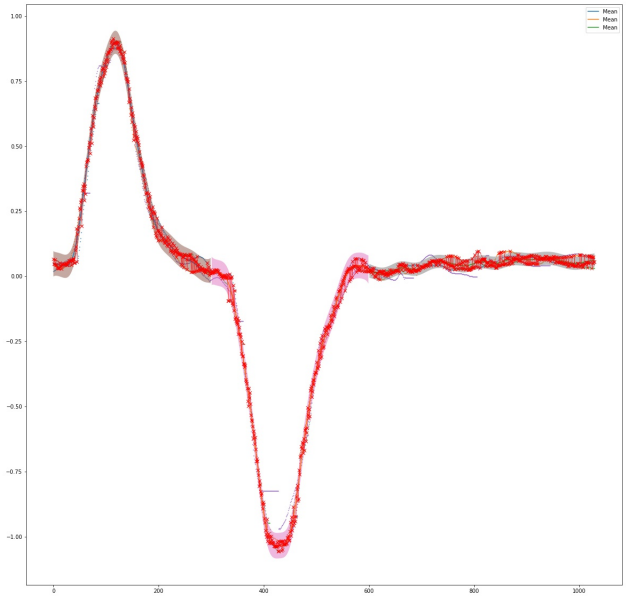

Figure 3: Local Kernel distribution.

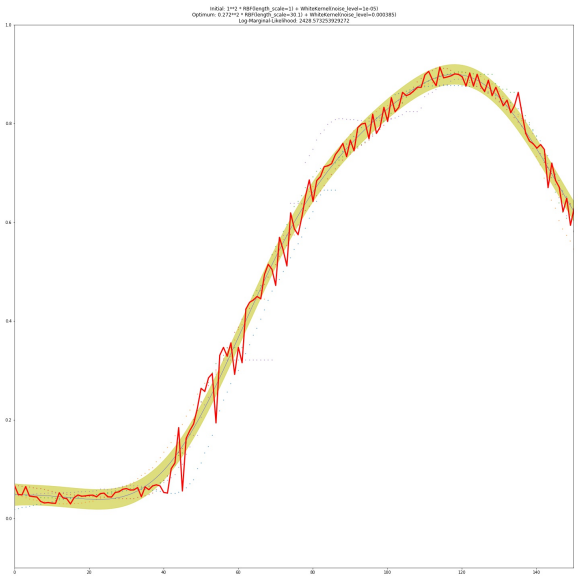

Figure 4: Global Kernel distribution.

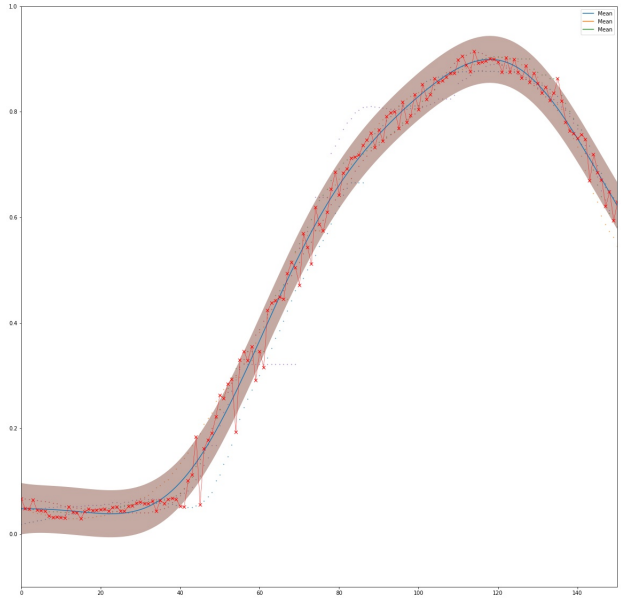

Figure 5: Local Kernel distribution.

References

Carl Edward Rasmussen and Christopher K. I. Williams. 2005. Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press.

Acknowledgments

Thank You Bayes God