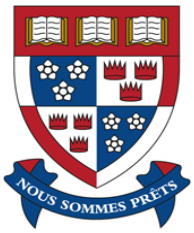


Simon Fraser University
Faculty of Statistics & Actuarial Science
Burnaby, British Columbia

Gaussian Processes

PRESENTED BY:

Barinder Thind

1 Introduction

Consider a group of people where each individual has given us their ratings (these are continuous variables) on the music genres of *folktronica* and *psychedelic rock*. Assume[1] that there exists some smooth (and possibly non-linear) relationship between these pairs of scores. Imagine that we are tasked to predict the score specified to *folktronica* from the rating given to *psychedelic rock*. Purely through intuition, under the assumption that some relationship exists, we would expect that on average, people who have similar ratings for *psychedelic rock* will likely have similar ratings for *folktronica*. This rationale is what underpins Gaussian Processes - we use some pre-defined measure of similarity between any pairs of data and then using this information, we can infer what we would expect the value of some response to be given the covariates of that particular observation. There are two main lens from which you can view this approach; I present the basics of Gaussian Processes through the Bayesian framework[2].

In Section 2, I highlight the mathematical underpinnings of Gaussian Processes beginning with linear regression & kernel functions and finishing with a concise methodology on how to make predictions using such a model. In Section 3, the methodology is shown in practice; predictions are made and compared with other models. Finally, some conclusions and other considerations are provided in Section 4.

2 Methodology

2.1 The Canonical Connection

We begin with the usual regression approach. Let $\vec{y} \in \mathbf{R}^n$ and consider the following model: $y_i = \vec{\beta}\mathbf{X} + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2 I)$, $\mathbf{X}$ has dimensionality n x p, and $\vec{\beta}$ is the vector of coefficients. We can generalize further and let $\mathbf{X}^{'}$ be some transformation(s) of the original data; that is to say, let the covariates be filtered through some basis functions: $\mathbf{X}^{'}_p = \phi(\mathbf{X}_p)$. Now, we can think of the *prior* for $\vec{\beta}$ has some draw from a multivariate normal distribution: $f(\beta) \sim \mathcal{MVN}(\vec{0}, \frac{1}{\alpha}\mathbf{I})$ where α is a hyperparameter that affects the magnitude of the variance. In essence, this has allowed us to define a distribution over functions as follows [1]: we get a draw from $f(\beta)$, this gets input into the original

[1] Pretend.
[2] Which I know you love.

regression model above, which induces a distribution for $f(x)$ - this distribution is also Gaussian as it is a linear combination of the draw from $f(\beta)$. We can derive the mean and covariance of $f(x)$'s joint Gaussian distribution as follows:

$$\mathbf{E}(\vec{y}) = \mathbf{E}(\vec{\beta}\mathbf{X}') = \mathbf{X}'\mathbf{E}(\vec{\beta}) = \vec{0}$$

$$\mathbf{cov}(\vec{y}) = \mathbf{E}(\vec{y}\vec{y}^T) - \underbrace{\mathbf{E}(\vec{y})\mathbf{E}(\vec{y}^T)}_{0} = \mathbf{X}' \underbrace{\mathbf{E}(\vec{\beta}\vec{\beta}^T)}_{\mathbf{cov}(\beta)-\mathbf{0}} \mathbf{X}'^T = \tfrac{1}{\alpha}\mathbf{X}'\mathbf{X}'^T$$

So, $\vec{y}$ is a draw from the distribution $f(x) \sim \mathcal{MVN}(\vec{0}, \frac{1}{\alpha}\mathbf{X}'\mathbf{X}'^T)$. An important point here is that we just derived the covariance *function* of $f(x)$ - in other words, we can input any desired combination of $x's$ and get the corresponding covariance matrix as defined by this function. Therefore, we theoretically have an infinite dimensional covariance matrix.

At this point, we can take a step back in the modelling process and note that we aren't limited to the covariance function we have described here and in fact, we can define whatever measure of *similarity* that we would like. This particular measure is known as the *linear kernel* and it limits us to lines. However, you can imagine that we would be interested in modelling non-linear relationships and for those, we would require more sophisticated *kernels*. There is a whole host of such functions and they will be discussed in the next section. The take away thus far is that (Bayesian) linear regression (i.e. using a linear kernel) is a special case of a Gaussian Process[3] [1]. While this will be formalized later, it would be advantageous to keep in mind that the flexibility in the choice of kernel function is what is at the core of this generality.

2.2 Kernel Functions

In the previous section, I introduced a specific kind of kernel - the linear one but it may not be apparent thus far what exactly a kernel function does. In short, kernel functions define some measure of a distance between two objects [2]. For our purposes, we could consider two observations: x_i and x_j where each is a p-dimensional vector - how *close* are these two vectors? This relates to the question posed in the opening paragraph - we have person i and j's rating of *psychedelic rock*; what function can we use to get the most appropriate measure of the relationship between these two such that it gives us the best prediction of the rating for *folktronica*? In Figure 1, we observe some draws of $f(x)$ using the prior distribution given two different kernels. Notice that if the data does not follow

[3]The Gaussian Process will be defined formally in the next section.

a linear relationship, using such a kernel will likely result in sub-par predictions in some subsets of the domain.

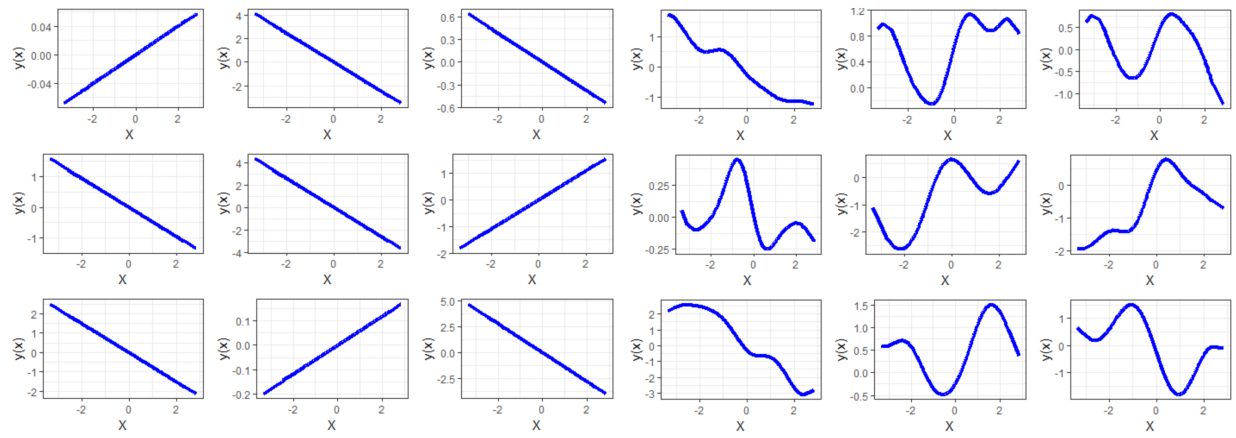


Figure 1: This image shows draws from $f(x)$. On the left hand side, we have draws from when the kernel is linear (the distribution of which is defined in the first section) whereas on the right hand side, the draws are made from when the kernel is a squared exponential one. These draws are made for a data set with one covariate.

Hence, it is important to decide an appropriate kernel function for the data that you are trying to do inference for. There are many choices from which to pick from but a particularly important one is the squared exponential kernel (also known as the radial basis kernel and the Gaussian kernel). This kernel has some properties that make it desirable for use. For example, there are only two hyperparameters which makes optimization a less costly process. It is defined as follows[4]:

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

The hyperparameter of σ^2 controls how variable the function is from its mean. A larger value of this hyperparameter would mean that we have a larger range of possibilities for any given draw at each of the input values. The other parameter l, often known as the length scale parameter, controls the width (or wiggliness) of the function [4]. Some examples of what happens when you vary these parameters are provided in Figure 2. Another useful property of this kernel is that it is infinitely differentiable everywhere i.e. it is a smooth function. This lends well to situations where optimization is required - we shall soon see this come into play.

There are a number of other kernel functions but the take away here should be that such functions exist and define some conditions that we expect our data to follow. Picking certain kernels may be

[4]This is the 1-dimensional case - there is a generalization that exists when $p > 1$.

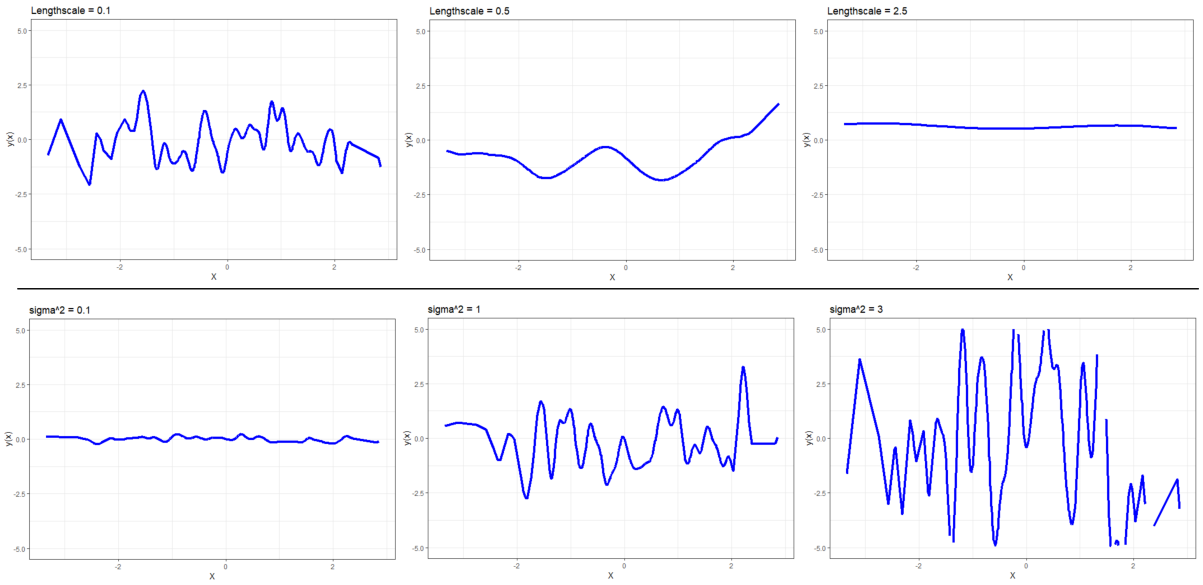


Figure 2: This figure shows what happens to draws from a $\mathcal{MVN}$ distribution with a squared exponential kernel when the hyperparameters are varied. The y scale was limited to [-5, 5]. The top three graphs correspond to a fixed value of σ^2 whereas the bottom three are for some fixed value of l. The other parameter is varied as indicated by the titles. You can see that the bias-variance trade-off will come into play here - function draws that are overly wiggly for data that doesn't warrant it will likely overfit, hence a greater increase in variance than decrease in bias.

more advantageous for the problem you are trying to solve. In the next section, we formalize the use of these kernels in the definition of a Gaussian Process.

2.3 Gaussian Processes

Thus far, we have discussed linear regression from the perspective of putting a prior on the coefficients along with the derivation of the linear kernel. We have also considered the possibility of other kernels and defined a particularly useful one. Conceptually, you actually already have some idea of what a Gaussian Process is but we can define it more rigorously as follows [4]:

Definition 2.3.1: A Gaussian Process is a a probability distribution over some set of functions, $f(x)$, such that the joint distribution of $f(x_i)$ evaluated at some arbitrary set of points $i = 1, 2, ..., n$, is multivariate normal with a mean vector $[\mathbf{E}(f(x_1)), ..., \mathbf{E}(f(x_n))]$ and n x n covariance matrix, $\mathbf{K}$.

In some sense, this is a generalization of the multivariate normal because the parameters of the Gaussian Process are functions themselves: $f(x) \sim \mathcal{GP}(\mathbf{E}(f(x)), k(x, x'))$. The multivariate normal

distribution that arises from the evaluation at some set number of points has a covariance function evaluated at those sets of points - this is now finite.

We have formalized what a Gaussian Process is, so we can now use the data that we have actually observed to get draws from a distribution that makes sense - i.e. update the parameterization of the Gaussian Process based on what we know. In Figure 1, what we really saw were draws from a Gaussian Process but notice that there seemed to be no rhyme or reason as to the way they arose. This is what is being adjusted for when we go from prior to posterior in this context. We define the Gaussian Process prior so that it has some particular properties (for example, by using the Gaussian kernel, we get smoothness), and then we can derive the posterior by conditioning on the data we have observed[5]. The implication here is that we need to construct a new mean and a new covariance matrix (that is conditioned on the observations) from which we can draw realizations. In Figure 3, you can see an example of a prior and a posterior.

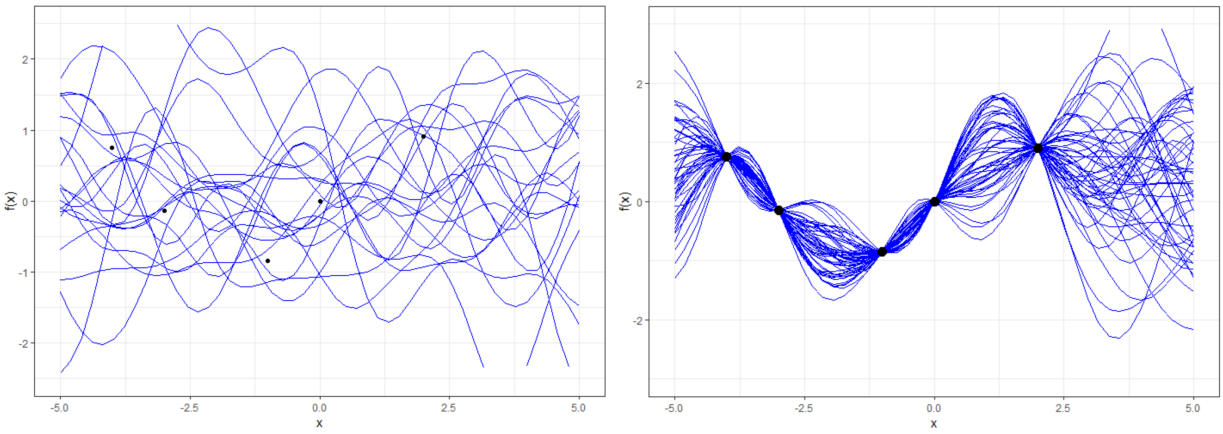


Figure 3: On the left are random draws from the prior Gaussian Process using a Gaussian kernel along with the observed data. Since we haven't conditioned on the data yet, the random draws from the Gaussian Process aren't following some pattern that the data would dictate. On the right, we can see the posterior. There is very little uncertainty as we get closer to the observed points and in fact, there is none at the exact observed points because the data was simulated from a noiseless data generating function.

The posterior can be derived for both noiseless and noisy predictions. The noisy predictions require nothing more than a simple modification of the simpler case so I will present the noiseless predictions first. The derivation of the posterior is omitted from this report but the exact formulation is given as follows: let $\mathbf{K}$ be covariance matrix found by putting your training points through the kernel function and let $\mathbf{K}_{**}$ be the covariance matrix formed by using the test points. The matrix $\mathbf{K}_{*-}$

[5]In the example of regression, the implication is that we condition on the data we have so that the coefficients we draw come from a distribution that makes sense for the data we have observed

is found by using the grid of values expanded by the test and train points. A note about the test points that may help with understanding is that they can be thought of as a sequence of points within the domain space specified by the covariates - since we would like to make a prediction anywhere within the domain[6], we would like to have Gaussian Process draw give a value at any point along this space; in Figure 3, the draws made from the prior are essentially our random predictions along the domain of x - the posterior will give us the post-training predictions. Since both the test points and the training points follow a Gaussian distribution, the joint distribution has the form [4]:

$$\begin{pmatrix} f(x) \\ f(x)_* \end{pmatrix} \sim \mathcal{GP}\left(\begin{pmatrix} \mathbf{E}(f(x)) \\ \mathbf{E}(f(x)_*) \end{pmatrix}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_{*-} \\ \mathbf{K}_{*-}^T & \mathbf{K}_{**} \end{pmatrix}\right)$$

This joint distribution can be derived easily by using the properties of the Gaussian distribution. With the joint distribution defined, we have all the tools required so that we can calculate the distribution we are specifically interested in: the conditional distribution of $f(x)_*$ given the prior and the information from the training data - this conditional distribution will be our posterior and is found to be [4]:

$$\begin{aligned} p\left(f(x)_* | \mathbf{X}_*, \mathbf{X}, f(x)\right) &= \mathcal{GP}\left(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*\right) \\ \boldsymbol{\mu}_* &= \mathbf{E}(f(x)_*) + \mathbf{K}_{*-}^T \mathbf{K}^{-1}(f(x) - \mathbf{E}(f(x))) \\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_{*-}^T \mathbf{K}^{-1} \mathbf{K}_{*-} \end{aligned}$$

For the kernel parameter, we see that we are subtracting out the covariance for which we have observed actual data. Using this distribution, we can get draws from the posterior - examples of these can be found on the right in Figure 3. To make the intuition clear on how exactly a prediction is made, let's consider the case when there is just 1 test point to be predicted - in this case, the posterior distribution derived above reduces to the univariate Gaussian distribution. Then, we can make predictions by taking the posterior mean. In Figure 4, a more detailed explanation is provided.

Finally, let's get back to figuring out the more practical case of when we need to do predictions for noisy data. Since the underlying data itself is noisy, it's clear that there is no longer a need for the model to perfectly predict the training points - there is just less uncertainty at those points when compared with parts of the domain for which there isn't data. Let $\epsilon_i \sim \mathcal{N}(0, \sigma_y^2)$ be the error associated with y where $y_i = f(x_i) + \epsilon_i$. Previously, the covariance of any two terms, y_i and y_j was simply the

[6] That is to say, for any combination of covariate values.

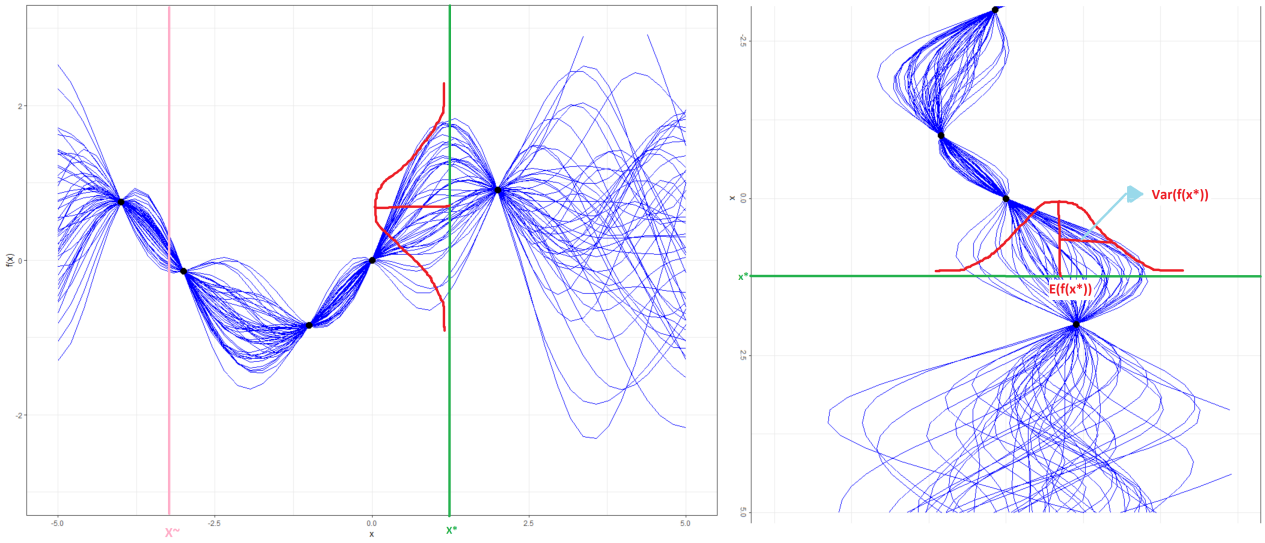


Figure 4: On the left hand side, you can see the point we are interested in making a prediction for: x^*. The black dots are points where we have actually observed data. The green line cuts through the blue curves at points that are possible values for $f(x^*)$ as observed through the realizations from the posterior - you can imagine these as draws from the distribution drawn in red. On the right hand side is the same plot but rotated - the point is to see that the prediction for x^* is going to be the mean of the univariate Gaussian distribution at that point i.e. the posterior mean. The variance associated with the distribution depends where the realizations fall - for points closer to an observed point (such as at the pink line corresponding to X~), there is less variance.

evaluation of the corresponding x_i and x_j through the kernel function but now, we also take into account the noise term as follows [5]: $\mathbf{K_{noise}} = \mathbf{cov}(y|x) = \mathbf{K} + \sigma_y^2\mathbf{I}$ where $\mathbf{K}$ is the previously defined covariance matrix and $\mathbf{I}$ is an n x n identity matrix. The conditional distribution is now found to be:

$$p\left(f(x)_*|\mathbf{X}_*, \mathbf{X}, f(x)\right) = \mathcal{GP}\left(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*\right)$$

$$\boldsymbol{\mu}_* = \mathbf{E}(f(x)_*) + \mathbf{K}_{*-}^T\mathbf{K_{noise}}^{-1}(f(x) - \mathbf{E}(f(x)))$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_{*-}^T\mathbf{K_{noise}}^{-1}\mathbf{K}_{*-}$$

The slight modification alluded to earlier is just made to the covariance of the posterior distribution where we now have taken into account for the error associated with each observation; it only affects the top left term of the covariance of the joint distribution ($\mathbf{K}$ becomes $\mathbf{K_{noise}}$); it also implies that we no longer will necessarily interpolate the true function value at the training points.

2.4 Hyperparameter Tuning

We have observed that we can obtain a posterior distribution from which we make predictions and that it (along with the prior) depends on the kernel we have chosen. As was stated earlier, the choice of kernel requires the handling of the associated hyperparameters. Thus far, the posterior distributions

we have derived have all arisen from some "default" or non-tuned values of the hyperparameters; sticking with the squared exponential, this means that we have not tuned the "wigglyness" parameter l, and the error noise parameter, σ^2 - the effect of changing these parameters was previously illustrated in Figure 2. There are a number of ways in which we can go about tuning these parameters. The brute force approach is to arrange for a large grid of the two parameters and measure the mean squared predicted error that we get after making predictions with the posterior; for example, we set the values of the kernel and then repeatedly create a training set from the full data. Using the training set we can build the posterior Gaussian Process, and predict on some other set of points in the domain - repeating this some number of times and taking the average of the predicted error will provide results in how *one* combination from the parameter grid performs; this is then repeated for every combination. Unfortunately, this can be very computationally inefficient. Alternatives to this approach include *MCMC* methods, *variational bayes*, and *empirical bayes* [4]; the exact procedures are left for the reader to explore.

3 Results

3.1 Data Description

For this part of the paper, I decided to compare the prediction results from a Gaussian Process to the other models we have seen in the class. In order to visualize results[7] and for computational efficiency[8], a model with just one explanatory variable was used the underlying function was:

$$f(x) = sin(4\pi x) + \exp(x) + x^2 + x^3 + tan(x) + \epsilon$$

Where $\epsilon \sim \mathcal{N}(0, 0.1)$, the response is standardized, and x $\in [0, 1]$. One advantage of Gaussian Processes is that they require very little data to obtain a competent model. In the next section, I will provide results from when only 5 data points are used in the training process.

3.2 Model Set Up & Predictions

All the other models for the comparison were tuned and compared to the default Gaussian Process[9]. This is purely due to time constraints and, as you will see in Figure 5, the default Gaussian

[7] The results from a possible box-plot run are shown in the last plot in the Appendix.
[8] The posterior requires an inversion which can be costly for data sets with large n.
[9] Where l and σ^2 were 0.1 and 1, respectively.

Process model bests the other (tuned) models for when 5 observations were used in the training set. In this comparison, the Gaussian Process was coded from scratch rather than using a package however, useful packages include: `GauPro` and `GPfit`.

The Gaussian Process model seemed to perform the best overall compared to the other methods. The regression model performed well, as well but this is due to the underlying nature of the data (it doesn't have a linear relationship but does follow a steady upward trend). The tuned random forest model seemed to be the most comparable to the Gaussian Process. As an interesting aside, some models seemed to have stagnated around an error near 1 - this needs to be explored further. The outliers that are present for the Gaussian Process are likely due to the small sample size of the training set - it is possible that the training points drawn were very close together which resulted in high variability of the draws from the posterior where there wasn't as many points.

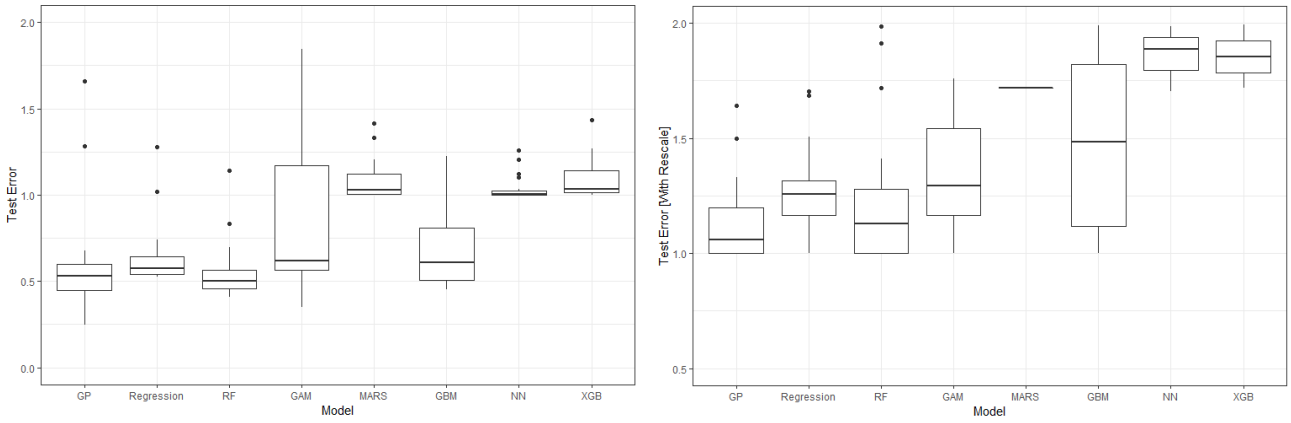


Figure 5: These are absolute (left) and relative error (right) box plots for various methods [3].

From these results, we have evidence that Gaussian Processes are competitive when it comes to data prediction problems - at least for when data is sparse. This is also a result that hasn't had its choice of kernel or the respective hyperparameters tuned implying that there is a large potential for improvement. In the Appendix, I present an implementation of Gaussian Processes from scratch and the corresponding results.

4 Conclusion

Gaussian Processes have proven to be invaluable tools for the purpose of prediction and inference. They have positively affected countless other fields and their versatility is latently evident by the fact that there are literal books written on the topic. In this short paper, I presented the fundamentals of

the method; more specifically, I motivated the approach with linear regression and showed it to be a special case of a Gaussian Process. Stepping back in the regression process naturally gave way to the bedrock of the methodology. This set in stone the tools needed to put this model to use in the way of an application to a simulated data set. Ultimately, Gaussian Processes were shown to perform excellently, besting most other models.

In the ever expanding zoo of predictive methodologies, Gaussian Processes have carved out a fairly prominent niche - particularly for sparse data sets. What was presented here is just a snippet of an expansive method for which there are a seemingly endless number of avenues to go down. Whether that be in deep learning (*deep Gaussian Processes*) or functional data analysis, Gaussian Processes have shown potential and prowess. All of this and more make Gaussian Processes a fascinating and growing field to do future research in!

5 References

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2017.

[3] Tom Loughin. *Lecture 13 Notes*. Stat 852 Regression Homework Questions. 2019.

[4] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[5] Carl Edward Rasmussen. «Gaussian processes in machine learning». In: *Summer School on Machine Learning*. Springer. 2003, pp. 63–71.

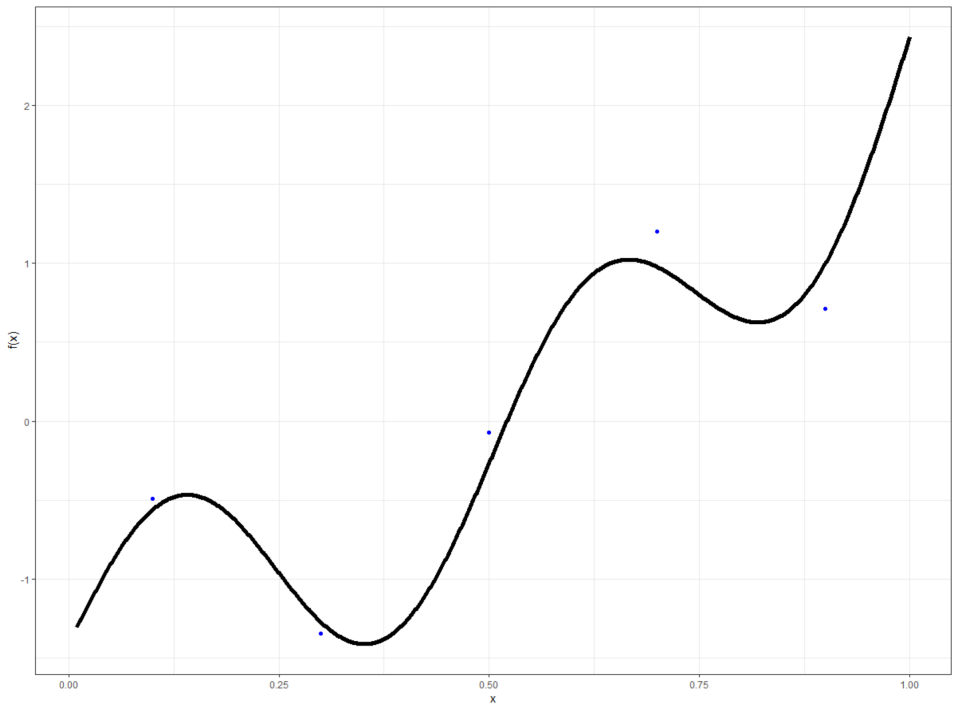
6 Appendix

6.1 A Gaussian Process from Scratch in R

Here, I just provide a quick implementation of the Gaussian process. I will provide clear connections to the mathematics presented earlier along with the R code. We begin by generating data as follows:

```
# Generating Data
x = seq(0.1, 1, 0.2)
y = sin(2*pi*x/0.5) + exp(x) + x^3 + x^2 + tan(x) + rnorm(length(x), mean = 0, sd =
↪  0.5)
df = data.frame(x = x, y = y)
# Normalizing y
df$y = (df$y - mean(df$y))/sd(df$y)
```

Note that here, we only generated 5 data points - this is going to be our training data. The true underlying function looks like (with the training points in blue):



The following code was used to generate this function:

```r
# Actual function
x_true = seq(0.01, 1, 0.001)
y_true = sin(2*pi*x_true/0.5) + exp(x_true) + x_true^3 + x_true^2 + tan(x_true)
df_true = data.frame(xt = x_true, yt = y_true)
# Normalizing y
df_true$yt = (df_true$yt - mean(df_true$yt))/sd(df_true$yt)
# Plotting data
ggplot(data = df, aes(x = x, y = y)) +
  geom_point(color = "blue") +
  geom_line(data = df_true, aes(x = x_true, y = y_true), color = "black", size = 2) +
  labs(y = "f(x)") +
  theme_bw()
```

Next, let's define the squared exponential kernel where the wiggly and noise parameter are built as options for the function:

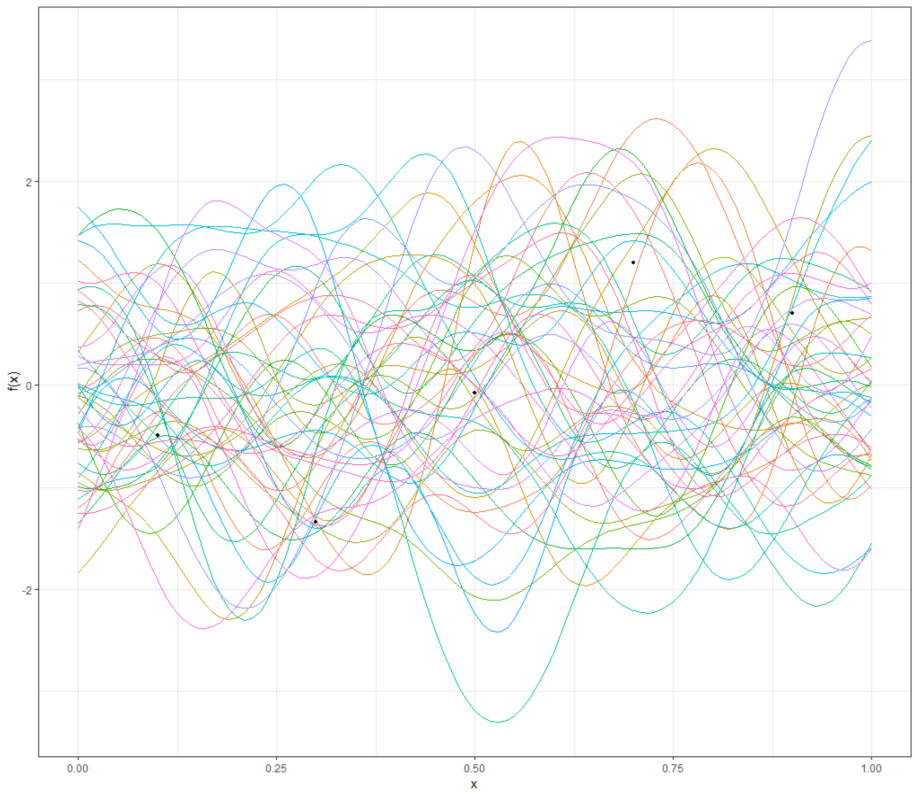
```r
# Gaussian kernel
gauss_kernel <- function(x, sigma, l){
  result = sigma^2*exp(-(x[1] - x[2])^2/(2*l^2))
  return(result)
}
```

We can now generate our testing points - this is like our $\mathbf{K}_{**}$ from before. Also, I give the code on how to generate random draws after building the covariate matrix using the kernel from before.

```r
# Random x's
x_ran = seq(0, 1, 0.01)
# Expanding grid
cov_grid = expand.grid(x_ran, x_ran)
# Putting through gaussian kernel to get results
cov_mat <- matrix(apply(cov_grid, 1, gauss_kernel, sigma = 1, l = 0.1), ncol =
↪  length(x_ran), nrow = length(x_ran))
# Drawing
draw = mvrnorm(50, mu = rep(0, length(x_ran)), Sigma = cov_mat)
```

The `mnvorm()` function draws from a multivariate normal distribution with some specified mean and covariance; in our case, we have a 0 mean (since we standardized) and the covariance matrix is generated with code before using the Gaussian kernel. In this case, we have only used the test points (meaning, we haven't trained the model yet), so the draws we are getting are from the prior. The following code generates the plots that shows draws from this prior:

```
# Melting
draw_melt = melt(draw)
colnames(draw_melt) <- c("Draw", "x", "y")
draw_melt$Draw = as.factor(draw_melt$Draw)
# Plotting
ggplot(data = draw_melt, aes(x = x_ran[x], y = y)) +
  geom_line(aes(color = Draw)) +
  geom_point(data = df, aes(x = df$x, y = df$y), size = 1) +
  labs(x = "x", y = "f(x)") +
  theme_bw()
```



Now, we can move on to getting the 4 covariance matrices that are required to generate the posterior. The comments indicate which is which:

```
# cov(train, test) = K*
cov_grid_star = expand.grid(df$x, x_ran)
cov_star = matrix(apply(cov_grid_star, 1, gauss_kernel, sigma = 1, l = 0.1), ncol =
↪  length(x_ran), nrow = length(df$x), byrow = F)
# cov(test, train) = K*^T
cov_grid_star_t = expand.grid(x_ran, df$x)
cov_star_t = matrix(apply(cov_grid_star_t, 1, gauss_kernel, sigma = 1, l = 0.1), ncol
↪  = length(df$x), nrow = length(x_ran), byrow = F)
# cov(train, train) = K
cov_train_grid = expand.grid(df$x, df$x)
cov_train = matrix(apply(cov_train_grid, 1, gauss_kernel, sigma = 1, l = 0.1), ncol =
↪  length(df$x), nrow = length(df$x))
# cov(test, test) = K**
cov_grid_starStar = expand.grid(x_ran, x_ran)
covStarStar = matrix(apply(cov_grid_starStar, 1, gauss_kernel, sigma = 1, l = 0.1),
↪  ncol = length(x_ran), nrow = length(x_ran))
```

We now have the joint distribution and all the tools necessary to go from the prior to the posterior. Sometimes, noise needs to be added to invert the matrix which is present in the next chunk of code where I get the posterior mean and posterior covariance.

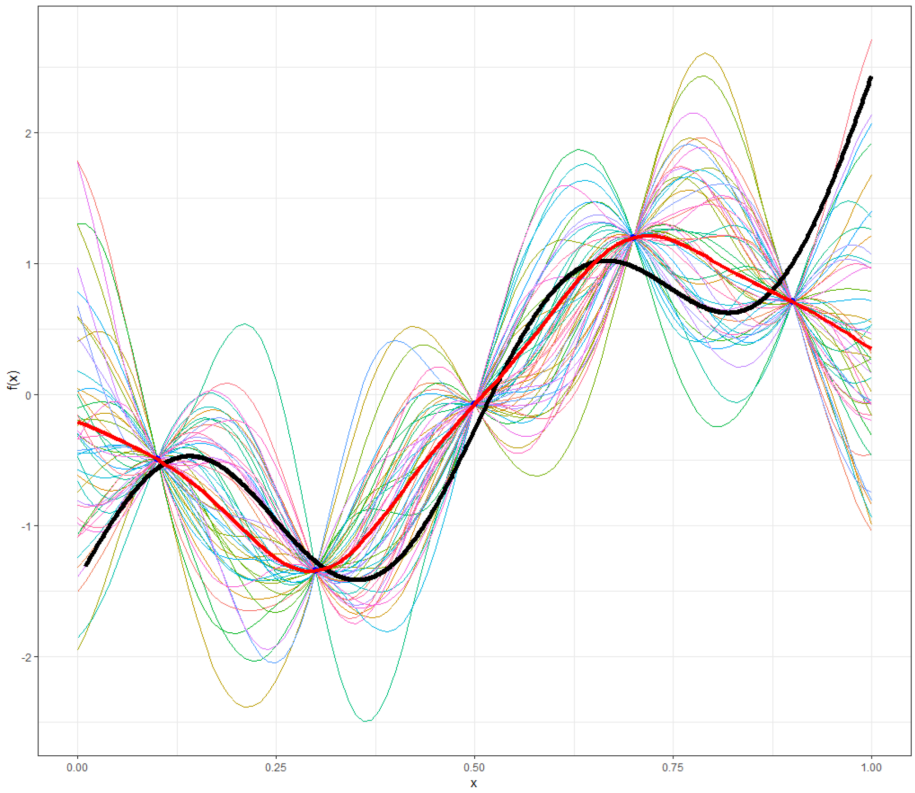
```
# Getting parameters
posterior_mean = (cov_star_t%*%solve(cov_train + diag(rep(x = 10^-7),
↪  nrow=nrow(cov_train), ncol = ncol(cov_train))))%*%df$y

posterior_cov = covStarStar - cov_star_t%*%solve(cov_train + diag(rep(x = 10^-7), nrow
↪  = nrow(cov_train), ncol = ncol(cov_train)))%*%cov_star

posterior_cov = ((posterior_cov+t(posterior_cov))/2)+diag(rep(x = 10^-7), nrow =
↪  nrow(posterior_cov),ncol = ncol(posterior_cov))
```

Now that we have the posterior, we are almost there! The rest is just a repeat of what we have seen before i.e. draw from the multivariate normal with this new distribution. The following chunk of code does this and plots the result.

```r
# Drawing again
draw_post = mvrnorm(50, mu = posterior_mean, Sigma = posterior_cov)
# Melting
draw_melt = melt(draw_post)
colnames(draw_melt) <- c("Draw", "x", "y")
draw_melt$Draw = as.factor(draw_melt$Draw)
# Plotting
ggplot(data = draw_melt, aes(x = x_ran[x], y = y)) +
  geom_line(aes(color = Draw)) +
  geom_point(data = df, aes(x = df$x, y = df$y), size = 2.5, color = "blue") +
  geom_line(data = df_true, aes(x = df_true$xt, y = df_true$yt), size = 2) +
  labs(x = "x", y = "f(x)") +
  theme_bw()
```



In this plot, we can see the true underlying data generating function in black, along with the posterior mean (our predictions) in red. We can also see that the draws from the distribution are much more directed as they were made from posterior (after training). And... there we go! We have now built a $\mathcal{GP}$ from scratch.