A Sub-bagging Approach to Gaussian Process Models for Deterministic Outputs

Dustin Johnson

University of British Columbia
STAT 547L

Introduction

Bootstrap Aggregation (Bagging) with replacement has been shown to improve the stability and prediction accuracy of Gaussian Process regression models for data with inherent variability from an underlying stochastic structure (T. Chen, 2009). On the other hand, Gaussian Processes (GP) are commonly used for emulating deterministic computer simulator outputs. These GP models interpolate the data, thereby rendering bootstrap sampling with replacement unsuitable. In order to account for deterministic behaviour of the simulator outputs, we require the sampling technique to treat each sample outcome as mutually exclusive. Sampling without replacement results in deliberate avoidance of choosing any member of the population more than once. It will be demonstrated that, by incorporating bootstrap re-sampling *without replacement* in a bagging routine, we can improve the prediction accuracy and stability of a Gaussian Process model of a deterministic simulator output.

Methodology

This section will outline the model and methodology on which the experiments are based. To reiterate, we will focus our attention on a Gaussian Process (GP) for a deterministic simulator output, thereby assuming noiseless outputs.

The Model

A benefit of using a GP is the fact that it can be completely determined by its mean and covariance function, thereby requiring the specification of only the first and second-order moments. Notationally, we could denote a GP as $\mathbf{y} \sim GP(m, K)$, meaning $\mathbf{y}$ is a

Gaussian process distributed with a mean function m and covariance function K. The covariance function is referred to as the kernel. From this point, we will assume the random variable $\mathbf{y}$ to have mean zero, thereby simplifying calculations without the loss of generality, allowing the process to be determined entirely by the kernel K.

Say we have a number of points with an input vector $\mathbf{x}$, producing some target outputs y. A GP draws the assumption that those inputs $\mathbf{x}$ that are close to one another have similar outputs y. Along these lines, we could then assume that training points that are near a test point will provide information regarding the prediction of that point. It is therefore crucial that a kernel be specified in a way to capture the *similarity* between the points in function space and the degree to which the similarity diffuses as points become further away. There are many types of kernels, but we will focus our attention on the widely used power-exponential kernel. Let us begin by refining the Gaussian process notationally.

Given a training data set of N data points, $\{x_i, y_i; 1, \ldots, N\}$, our GP prior is:

$$\mathbf{y} = (y_1, \ldots, y_N)^T \sim G(\mathbf{0}, \mathbf{K})$$

Where $\mathbf{K}$ is an $N \times N$ matrix defined as follows:

$$\mathbf{Cor}(y(x), y(x')) = K(x, x') = \exp(-\frac{1}{\theta}|x - x'|^2) = \exp(-\frac{1}{\theta}|x^{(i)} - x^{(j)}|^2)$$

The hyper-parameters θ define the covariance function and dictate the strength of the correlation between nearby points. A low theta would indicate higher correlation with nearby points, whereas a high theta would indicate less inter-point correlation. The power-exponential kernel is infinitely differentiable and provides a smooth decay in correlation as we move further away from a points in close proximity, as indicated by the nature of the second power. This smoothness assumes continuity in the deterministic function we attempt to model.

For a new data point with an input vector $\mathbf{x}^*$, we condition its predictive distribution on the training data providing us with a posterior mean and variance as follows:

$$\hat{y^*} = \mathbf{k}^T(\mathbf{x}^*)\mathbf{C}^{-1}\mathbf{y}$$
$$\sigma^2_{\hat{y^*}} = C(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T(\mathbf{x}^*)\mathbf{C}^{-1}\mathbf{k}(\mathbf{x}^*)$$

where $\mathbf{k}(\mathbf{x}^*) = [C(\mathbf{x}^*, \mathbf{x_1}), \ldots, C(\mathbf{x}^*, \mathbf{x_N})]^T$. These will be vital components in the aggregation procedure proposed in the following sections.

Parameter Estimation

To estimate the parameters of the GP model, we can use the approach of maximum likelihood estimation (MLE), which says that we should select the hyper-parameters that make the data most probable. This can be accomplished by maximising the log-likelihood function with respect to the hyper-parameters θ and σ^2.

$$\mathbf{L}(\mathbf{y}|\sigma^2, \theta_1, \ldots, \theta_d) = -\frac{1}{2}\log\det\mathbf{K} - \frac{1}{2}\mathbf{y}^T \det\mathbf{K}^{-1}\mathbf{y} - \frac{N}{2}\log 2\pi$$

Essentially, we would like to take the partial derivatives of the log-likelihood function and set them to zero, as follows:

$$\frac{\partial \mathbf{L}}{\partial \theta} = -\frac{1}{2}tr\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\right) + \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\mathbf{K}^{-1}\mathbf{y}$$

In the majority of cases, optimisation of the likelihood function cannot be accomplished analytically. In such cases, numerical methods are required.

Bootstrap AGGregatING (Bagging)

Bagging, short for "Bootstrap AGGregatING", is a method of improving the robustness and accuracy of models that are inherently unstable by obtaining bootstrapped re-samples of the training data. Bagging involves drawing a number of bootstrapped samples from the training set and fitting each sample with the model in question. The models are then combined or "aggregated" together and evaluated for predictive accuracy. Sub-bagging refers to bootstrapped sample sets that are smaller than the number of observations in the training set.

Typically, the bootstrapped samples are conducted *with replacement*, thereby permitting repeated draws of the same training points. As outlined in the introduction, this is not practical for deterministic computer simulations. In order to consider the case of a deterministic output, a sub-bagging method without replacement is proposed.

Sub-bagging Without Replacement

Suppose $Z = \{x_i, y_i; 1, \ldots, N\}$ represents our original training data. To take a bootstrapped sample, we conduct the following:

1. Randomly sample t data points out of the total N training points *without replacement* from Z with probability $\frac{1}{N}$, where $t < N$. The t sampled data points will be denoted as Z_1.
2. Repeat the procedure K times and obtain K re-sampled data sets: $Z_1, \ldots, Z_K$.

3. Fit K separate models to each re-sampled training set, providing us with $\mathcal{M} = \{M_1, \ldots, M_K\}$ models.
4. Aggregate the $\mathcal{M}$ models together by simply averaging their aforementioned predictive distributions:

$$p(y^*|\mathcal{M}) = \frac{1}{K}\sum_{k=1}^{K} p(y^*|\mathcal{M}_k)$$

The resulting predictive model derived from the simple averaging method is known as a Gaussian Mixture Model, making it no longer Gaussian distributed. Fortunately, the calculation of the predictive mean and variance can be derived quite easily through the properties of the Gaussian Mixture Model.

$$E(y^*) = \frac{1}{K}\sum_{k=1}^{K} \hat{y}^*(k),$$

$$Var(y^*) = \frac{1}{K}\sum_{k=1}^{K} \sigma^2_{\hat{y}^*}(k) + \frac{1}{K}\sum_{k=1}^{K} (\hat{y}^*(k) - E(\hat{y}^*))^2$$

Using simple averaging, we can obtain an aggregated posterior mean and variance from a set of individual priors.

G-Protein Computer Experiment

Objective

In this section, we will be comparing and contrasting two models on the basis of their fit and prediction accuracy:

- Single GP model
- Bagged GP ensemble

The aim of this experiment is to determine whether or not a sub-bagging routine for a GP model will outperform a single GP model fit in the case of a deterministic output. In order to achieve valid results, we will monitor the stability of the root-mean-square error (RMSE) and absolute error (AE) over a number of iterations, identify an optimal bootstrap sample size, and examine the diagnostics of the two models.

Experimental Structure

In this section, we will apply the sub-bagging procedure without replacement on a biosystems model for ligand activation of G-protein in yeast. The deterministic output y is the normalised concentration of part of the complex and there are four input variables that can be controlled. The computer experiment is structured as follows:

- $log(x), log(u_1), log(u_2)$, and $log(u_3)$ are the log-transformed input variables corresponding to the concentration of ligand and 3 kinetic parameters, respectively.
- Input variable ranges have been normalised to $[0, 1]$ on the log scale.
- A training set of 40 has been randomly sampled from the data's total training set of 4000.
- The test set contains 5000 observations, randomly sampled from the total test set of 10000.
- The DiceKriging package was used to fit the GP models.

Due to the high accuracy of this model, the size of the training set has been chosen to be 40. The package *DiceKriging* offered the most flexibility and speed for the computational aspect. Parallel computation from the package *doMC* was required to boost the speed of some time-consuming iterations. It is important to note that we will be using DiceKriging's power-exponential kernel, providing us with an assumption of smoothness (power of 2) and parameterising the kernel's hyper-parameters as $\frac{1}{\theta}$, as denoted in our model section.

Selecting a Kernel

As outlined in the methodology section, the kernel is crucial in a Gaussian process predictor. Many kernels offer different properties in how to structure the inter-point correlation relationship. The Matérn class of kernels allow specification on the nature of the correlation by means of the parameter v. For a Matérn1_2, with $v = \frac{1}{2}$, the covariance between points decays rapidly making the process very rough and nowhere differentiable. On the other hand, increasing the parameter $v \to \infty$, the Matérn converges to the power-exponential (C. E. Rasmussen, 2006). We will concentrate on two cases of Matérn kernels: Matérn5_2 (twice differentiable), and Matérn3_2 (once differentiable). The remaining two analysed are the exponential and gauss kernels. The exponential kernel can be thought of as the power-exponential without the squared value, making the process rough. The gauss is infinitely differentiable, just as the power-exponential, but tends to decay at a slower rate creating a highly smooth process.

Table 1 compares the RMSE and AE of a single and bagged GP fit according to five different kernels. The single models were fit to the entire training data, while the bagged models were conducted by aggregating 15 bootstrap re-sample GP fits of size 35. The details of this choice will become apparent in the next few sub-sections. We notice that the Matérn3_2 and Matérn5_2 kernels have the best *single* GP fit prediction performance - The RMSE of Matérn3_2 is 0.00252, followed by 0.00397 for Matérn5_2, while the AE remains lower for Matérn5_2. The power-exponential follows suit with an RMSE and AE of 0.00510 and 135.86922, respectively. The interesting aspect is not the performance of the Matérn kernels, which are shown to work well in low-dimensional input space (Durrande, 2010), but the remarkable improvement in the RMSE of the power-exponential under the bagged GP method. The predictability of the power-exponential kernel, as indicated by the RMSE and AE, outperforms the others when bagging is considered.

Kernel	RMSE Single	AE Single	RMSE Ensemble	AE Ensemble
matern5_2	0.00397	130.76906	0.00489	141.90141
matern3_2	0.00252	134.84567	0.00307	155.32412
powexp	0.00510	135.86922	0.00284	126.26690
exp	0.00614	294.02023	0.00811	371.75638
gauss	0.00530	138.21515	0.00453	126.28858

Table 1: RMSE and AE according to kernel type conducted over 15 iterations with a bootstrap sub-sampling size of 35.

The Matérn kernels seemed to work well with a single GP fit, indicating lower inter-point correlation between nearby points. This could also mean that points further away are playing a significant role that the other kernels were unable to pick-up because of their emphasis in nearby points. However, the Matérn kernels performed worse under bagging. This could have been caused by the reduction in the training sample on which the models were fit. As some points would naturally be spaced further apart, the Matérn kernels under-compensate for these gaps, reducing predictability. Due to the dramatic reduction in prediction error for the power-exponential under the bagging method, we will devote our attention to this specific kernel.

Number of Bootstrap Samples

Obviously, resources are limited. It is therefore infeasible to sample until the error has perfectly stabilised due to lack computational power, money, etc. We would like to know how many iterations to perform in order to achieve reasonably stable results. *Figure 1.(a)* (left) and *Figure 1.(b)* (right) below represent the RMSE and AE of a series of cumulatively aggregated GP fits over 30 iterations using the simple averaging of their posterior means. These iterations were based on a bootstrap sub-sample size of 35 points

(without replacement) obtained from the training set containing 40 experimental trials. It will become more apparent why we chose 35 in the upcoming section. We notice that, as the number of iterations increase - the more models we aggregate - the better our predictive performance becomes. The red dashed line corresponds to the error of a single GP fit to the G-protein data.

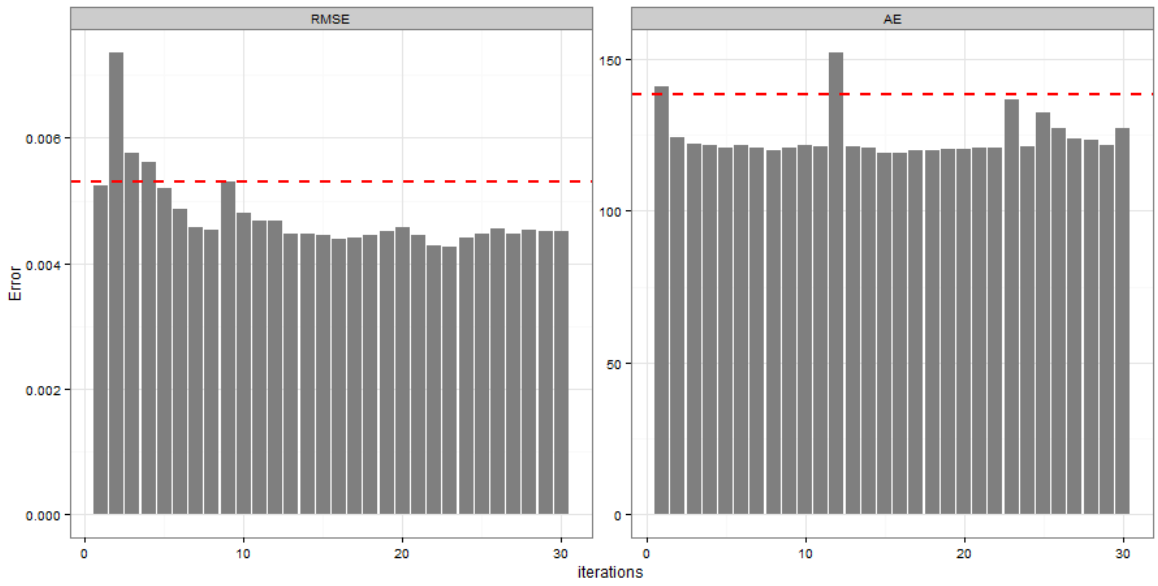


Figure 1: *RMSE and AE over 30 cumulatively aggregated bootstrap samples (single GP fit error denoted as red line). Left:* RMSE steadily declines over 30 sample iterations. **Right:** AE stabilises after one iteration with a few apparent spikes.

In *Figure 1.(a)*, we can see that the RMSE of our ensemble model drops below that of the single model fit and stabilises at roughly 13-15 iterations. If we continue to aggregate beyond this point, there is little to no improvement in the RMSE. The AE in *Figure 1.(b)*, on the other hand, sharply drops below the AE of the single GP fit and (nearly) stabilises after only one iteration. We do notice a sudden spike or two after 10 iterations, but this is likely due to a poor bootstrapped re-sample fit to an outlier, severely pulling up the sensitive absolute error.

Overall, it seems to be safe to conduct 15 iterations, as both the RMSE and AE have stabilised at this point and litter improvement will be gained with a few more bootstrap sample fits.

Size of Bootstrap Samples

From the previous section, we determined that 15 iterations resulted in reasonably stable values of the bagged GP's prediction error. We are now interested in identifying what bootstrap sample size would be optimal. Regarding bootstrapping with replacement,

it will always be beneficial to sample a size greater or equal to the size of the training set if it is computationally feasible to do so. Unfortunately, we do not have the luxury when it comes to sampling without replacement. We must collect a sample size less then the size of the training set (sub-sample). It is thus important to determine the optimal re-sample size to minimise the prediction error of the bagged GP model.

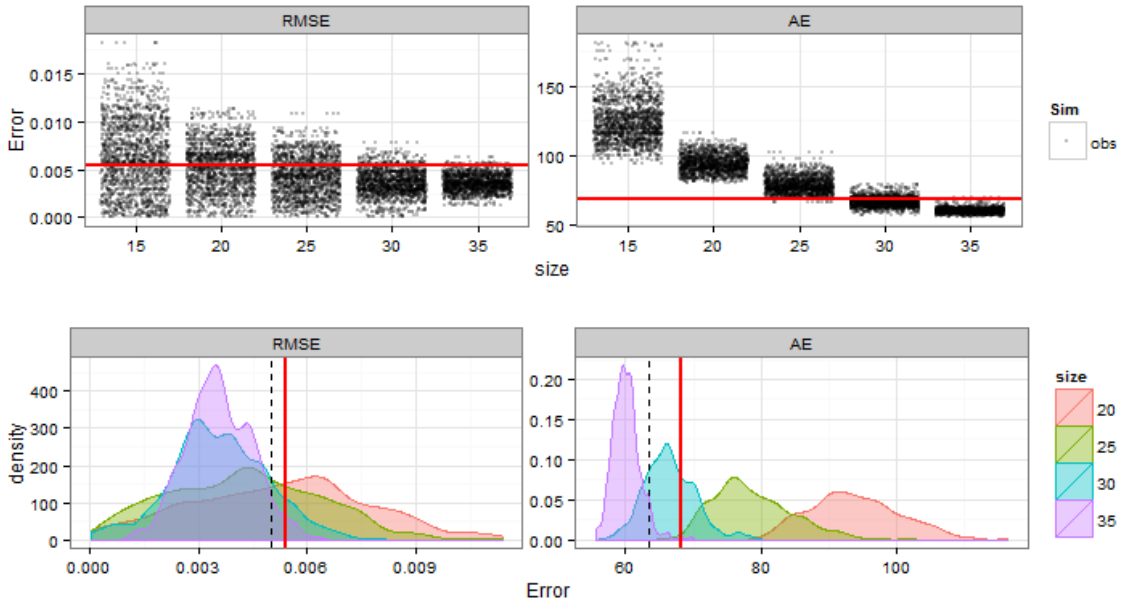


Figure 2: *RMSE and AE simulated 500 times over bootstrap sample sizes of 15 to 35. Top-Left:* RMSE becomes more dense and falls below the standard error of the single GP fit. *Top-Right:* AE progressively falls and becomes concentrated at a size of 35 below the AE of the single GP. *Bottom-Left:* Density of RMSE appears to converge at size 35, below the RMSE of a single model (red line) - single model error falls above the 95th percentile. *Bottom-Right:* Density appears to converge at size 35, below the AE of a single model (red line) - single model error falls above the 95th percentile.

Figure 2 presents four plots, each demonstrating the effect that bagging plays on the prediction error. 500 bagged fits were conducted over each of the bootstrap sub-sample sizes and compared to the prediction error of the single GP fit, denoted as the red line. The results indicate that as we increase our bootstrap re-sample size, the more concentrated our error distributions become. At a re-sample size much lower than the size of the training set, our fits are sporadic and cause high variability in our predictions. At a re-sample size of 35, we are closer to the true size of the training set, and our fits to these sub-samples become more accurate. By averaging them, we can see an improvement in our prediction error beyond that of a single GP fit both in terms of RMSE and AE. The dashed black line is the upper 95th percentile of a bootstrap re-sample size of 35. We notice that the RMSE and AE of a single GP fit fall below this threshold, noting a significantly lower mean RMSE and AE of the bagged GP. On average, with 15 aggregated GP models and a bootstrap sub-sample size of 35, the

bagged GP fit outperforms the single GP in terms of prediction error.

Comparing GP Model Fits

We have experimented with results from the G-Protein data set and identified that a bagging routine involving 15 bootstrapped re-sampled GP fits of size 35 outperforms that of a single GP fit. One such experiment has been conducted with the parameter estimates for a single GP fit and 4 out of the 15 re-sampled fits presented in *Table 2*.

	single GP	ensemble 1	ensemble 2	ensemble 3	ensemble 4
$\hat{\theta_1}$	1.97	1.89	2.00	1.95	1.57
$\hat{\theta_2}$	1.19	1.30	1.19	1.09	1.14
$\hat{\theta_3}$	1.05	1.20	0.97	1.19	1.36
$\hat{\theta_4}$	0.55	0.73	0.72	0.72	0.64
$\hat{\mu}$	0.33	0.40	0.28	0.34	0.36
$\hat{\sigma}^2$	0.14	0.15	0.11	0.11	0.17

Table 2: Power-exponential kernel parameter estimates compared across a single GP fit and a random selection of four GP ensemble fits. Conducted over 15 iterations with a bootstrap sub-sampling size of 35 without replacement.

The parameter estimates appear to be consistent across the single and re-sampled fits. If we were to choose a smaller re-sample size, we would unlikely see such consistency in the model parameters. The *DiceKriging* package has a lower bound of $1e^{-10}$ and an upper -bound of 2 for its parameter estimates. Forcing the upper-bound beyond 2 for the power-exponential results in stability issues and the matrix is no longer positive definite. Fortunately, all parameters appear quite stable and fall within their respective ranges. The concentration of ligand parameter $\hat{\theta_1}$ in Ensemble 2 seems that it could be sitting on its upper-bound, but remains consistent with the other estimates - we would not expect it to be much higher or dramatically affect the results.

Figure 3 demonstrates the RMSE of each of the 15 bootstrap re-sample fits according to the model above. We notice that some sub-sample fits predict poorly, while others predict well. The blue line shows us the combined average of the fits - the RMSE of our bagged GP model. The red line is the RMSE of a single model fit to the 40 training observations. The bagged model outperforms the single fit by approximately 0.0025.

Diagnostics

The diagnostic plots are presented in *Figure 4* and *Figure 5* with the blue triangles corresponding to the single GP fit and the black points corresponding to the bagged GP fit. The standardised residuals over predicted values in *Figure 4* appear to stay within two standard deviations. There is a slight clump of points extending beyond three, but

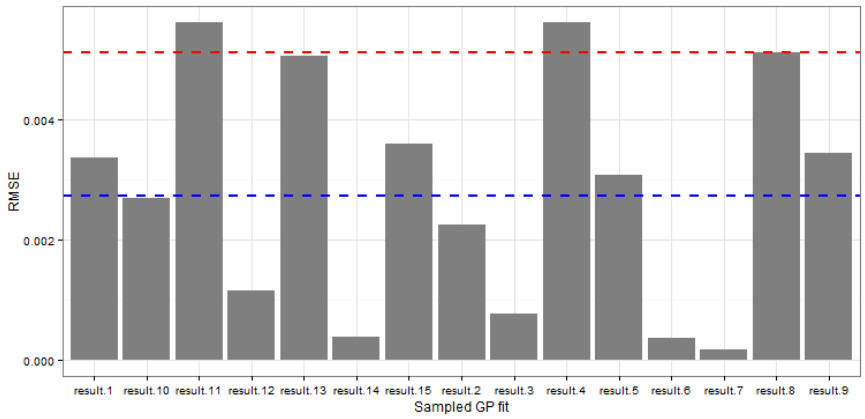


Figure 3: Prediction error of a single GP fit(red) vs. the ensemble GP fit (blue) over 15 iterations. Each bar represents a fitted GP model on a single bootstrap sample. The ensemble GP prediction error improves from the single GP prediction error by approximately 0.0025

due to the size of the test set, this is not overly threatening. Standard errors for both the single and bagged models are fairly valid. We notice less scatter amongst the bagged GP residuals, due to the improvements through aggregation.

Figure 5 plots the observed values with the predicted values of each model, where the red line indicates perfect accuracy. With little deviation, there appears to be good accuracy with the models, even near the ends, which are prone to instability. It is difficult to notice a difference in the accuracy of the single vs. bagged predictions, but the bagged model seems to be slightly more concentrated.

Concluding Remarks

The sub-bagging model outlined in this paper has been proposed to improve the prediction accuracy of Gaussian process models to deterministic outputs. The G-Protein experiment has demonstrated that using a sub-bagging routine with a bootstrap resample size proportional to 5 points less than the training set can improve the accuracy of predictions and provide more robust estimates.

Despite the positive outcome, there are a few conditions that should be emphasised when contemplating the sub-bagging method. Firstly, a training set already providing high accuracy for a single GP model will unlikely benefit from such a sub-bagging procedure, as we would be merely aggregating sub-models of an already strong predictor. This can be demonstrated in *figure 6*, where we see that increasing the training size to 100 results in a more accurate reading, thereby making the sub-bagging procedure less effective (if not worse). It is also important to note that the sub-bagging procedure is dependent

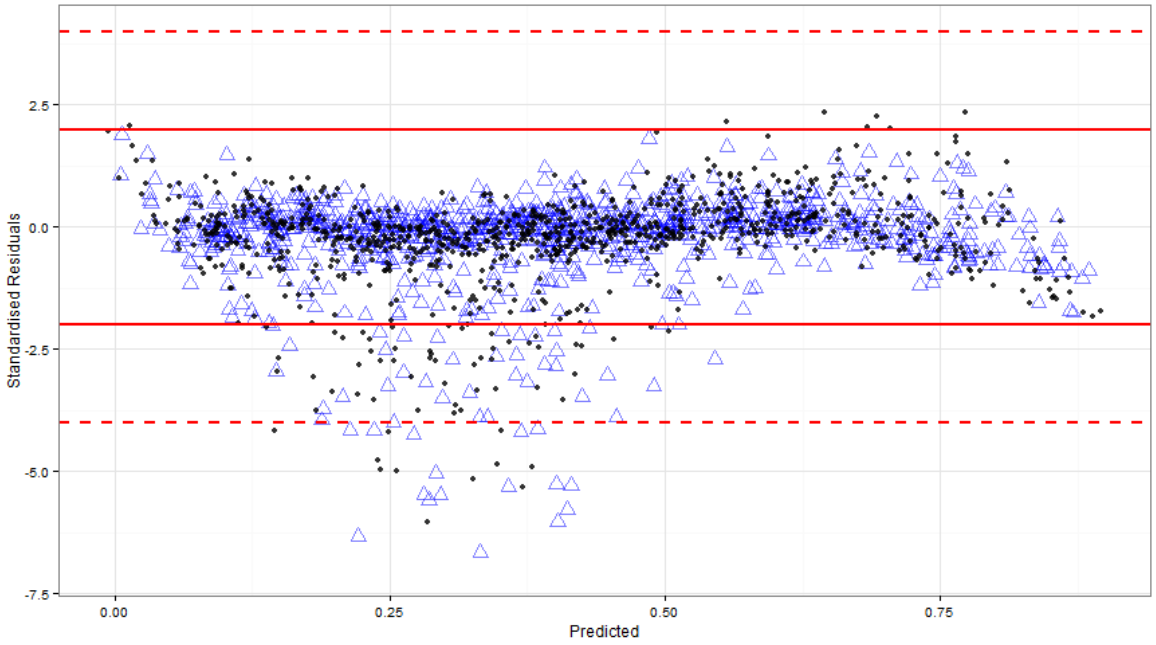


Figure 4: Standardised residuals vs. predictions. Single GP model fit is denoted with blue triangles; Ensemble GP fit is denoted in black.

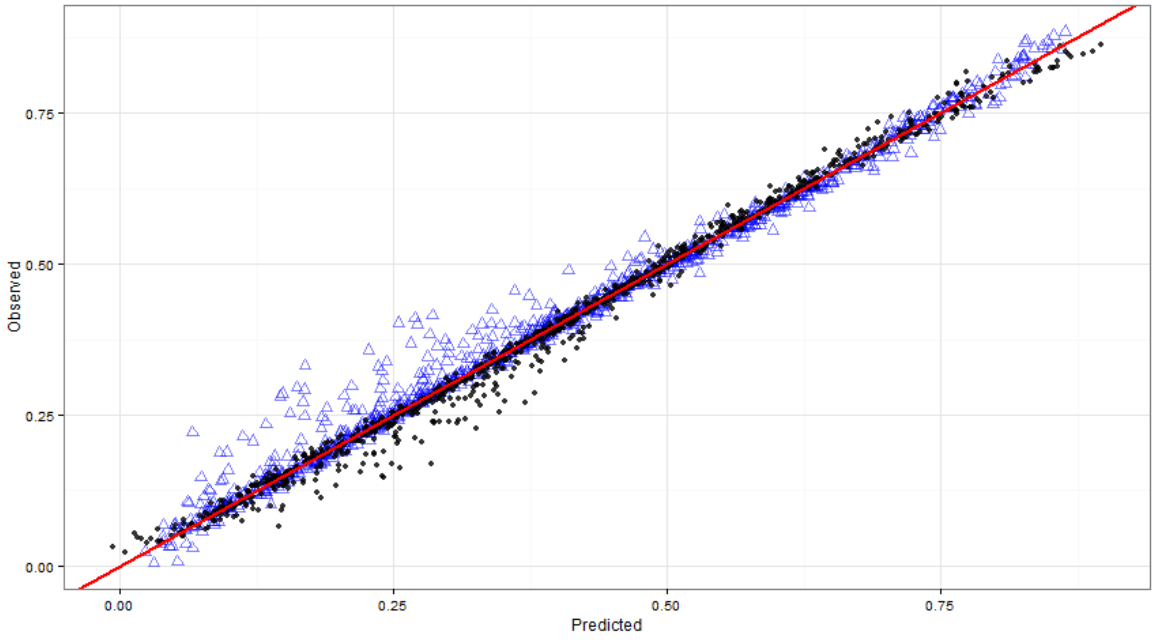
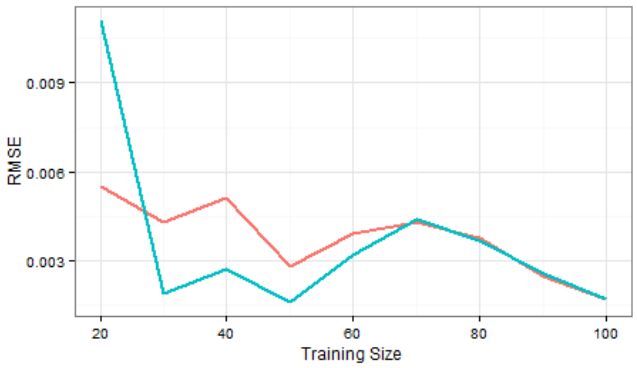


Figure 5: Observations vs. predicted values - evaluating prediction accuracy.



Training Size	single	bagged
20	0.0055	0.0111
30	0.0043	0.0019
40	0.0051	0.0027
50	0.0028	0.0016
60	0.0039	0.0032
70	0.0043	0.0044
80	0.0038	0.0037
90	0.0025	0.0026
100	0.0017	0.0017

Figure 6: *Left*: RMSE of single (red) and bagged (blue) GP model fits over different training data sizes. *Right*: Table of the RMSE for a single and bagged GP fit over different training data sizes.

on the quality of the training set - a poor (or wrong) single GP fit will lead to a poor sub-bagging fit.

Other limitations include the computational cost associated with the aggregation of multiple models. As the bootstrap sample size required of the sub-bagging routine is nearly the size of the training set itself, it may be infeasible to capture the benefit of such a method. Additionally, the bagging procedure does assume that the training data is independent and identically distributed (*iid*), which runs contrary to a Gaussian process (T. Chen, 2009). That being said, if instability is a major concern with the model, the benefit from improving it may be more important than the violation of the *iid* assumption.

It would be interesting to further investigate the impact of using multiple different re-sample sizes for the sub-bagging method, as opposed to a fixed sub-sample size to improve predictive performance. Further investigation may identify other limitations of the sub-bagging procedure, but the G-Protein experiment outlined a possible improvement in prediction accuracy using the sub-bagging model.

References

1. C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, 2006,ISBN 026218253X. Massachusetts Institute of Technology. www.GaussianProcess.org/gpml

2. D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," Journal of Global Optimization, vol. 13, pp. 455-492, 1998.

3. L. Breiman, Bagging predictors, Machine Learning 26 (1996) 123-140.

4. M. Schonlau and W. J. Welch, "Screening the input variables to a computer model via analysis of variance and visualization," in Screening Methods for Experimentation in Industry, Drug Discovery and Genetics (A. M. Dean and S. M. Lewis, eds.), pp. 308-327, New York: Springer-Verlag, 2006.

5. Nicolas Durrande, David Ginsbourger, Olivier Roustant. Additive Kernels for High-dimensional Gaussian Process Modeling. Technical report. 2010, pp.10.

6. P. Buchlmann, B. Yu, Analyzing Bagging, Annals of Statistics 30 (2002) 927-961.

7. T. Chen and J. Ren, "Bagging for Gaussian process regression," Neurocomputing, vol. 72, pp. 1605-1610, 2009

R Source

GP_bag function and Dependencies

```
# Dependencies
suppressPackageStartupMessages({
   library(plyr)
   library(dplyr)
   library(DiceKriging)
   library(foreach)
   library(reshape2)
   library(ggplot2)
   library(knitr)
   library(doMC) # initiate with registerDoMC()
   library(gridExtra)
})

# GP_bag function
GP_bag <- function(x, y, x.test, iterations, size, replace = TRUE, covtype =
    "powexp") {

   # load required packages
   require(plyr);
   require(dplyr);
   require(foreach);
   require(DiceKriging);

   # coerce into data.frame
   df <- data.frame(x, y)

   # iterate GP fit and predictions
   predictions <- foreach(m = 1:iterations, .combine = rbind) %do% {

      # randomly sample from data.frame
      sample <- sample_n(df, size, replace = replace)

      # gaussian process model fit
      GP_mod <- km(design = sample[,-ncol(sample)],
                   response = sample$y,
                   covtype = covtype,
                   control = list(maxit = 500, trace = FALSE)
      )

      # Prediction of GP model to test set
```

```
      pred <- predict(GP_mod, newdata = x.test, type = "UK")

      # Return required results into list
      return(list(pred$mean, pred$sd, pred$lower95, pred$upper95,
                      GP_mod@covariance@range.val, GP_mod@covariance@sd2,
                      GP_mod@trend.coef))
   }

   return(predictions)
}
```

Import Data

```
# Import G-protein data
gpro <- read.csv("gpro.txt", header = FALSE, col.names = c("x1", "x2", "x3",
    "x4", "y"))

# training set
set.seed(99)
train <- data.frame(sample_n(data.frame(gpro[1:4000,]), 40))
x.train <- train[,1:4]
y.train <- train$y

# test set
test <- data.frame(gpro[4001:dim(gpro)[1],])
x.test <- test[,1:4]
y.test <- test$y
```

Single GP Model and Diagnostics

```
# Fit single GP model
set.seed(1)

# fit model
GP_gPro <- km(design = x.train, response = y.train,
                            covtype = "powexp",
                            control = list(maxit = 300,
                            trace = FALSE))

# prediction
pred_gPro <- predict(GP_gPro, newdata = x.test, type = "UK")

# error
```

```
RMSE_gPro <- sqrt(mean(pred_gPro$mean - y.test)^2)
AE_gPro <- sum(abs(pred_gPro$mean - y.test))

error_gPro <- data.frame(RMSE_gPro, AE_gPro)
colnames(error_gPro) <- c("RMSE", "AE")
kable(error_gPro)


# Single GP diagnostics
predictions_gPro <- pred_gPro$mean
residuals_gPro <- sd(y.test - pred_gPro$mean)
std_residuals <- data.frame(pred = predictions_gPro,
                                      std_res = (y.test -
                                         predictions_gPro)/residuals_gPro)

# plot standardised residuals vs predictions
ggplot(std_residuals, aes(pred, std_res)) +
   geom_point(alpha = 0.5) +
   geom_abline(intercept = -4, slope = 0, colour = "red", size = 1, linetype
       = "dashed") +
   geom_abline(intercept = 4, slope = 0, colour = "red", size = 1, linetype =
       "dashed") +
   geom_abline(intercept = -2, slope = 0, colour = "red", size = 1) +
   geom_abline(intercept = 2, slope = 0, colour = "red", size = 1) +
   theme_bw()

# observed vs predicted plot
obs_vs_pred <- data.frame(y.test, predictions_gPro)
ggplot(obs_vs_pred, aes(y.test, predictions_gPro)) +
   geom_jitter(alpha = 0.5) +
   geom_abline(intercept = 0, slope = 1, colour = "red", size = 1) +
   theme_bw()
```

Ensemble GP Model and Diagnostics

```
# GP Ensemble model - Bagging ****************************

set.seed(4)

# fit model using GP_bag function
GP_gPro.bagged <- GP_bag(x.train, y.train, x.test,
                         iterations = 15, size = 35, replace = FALSE)

# prediction error of each sample model
```

```
error <- ldply(GP_gPro.bagged[,1]) %>%
   melt(id.vars = ".id", value.name = "Y_hat") %>%
   group_by(.id) %>%
   summarise(RMSE = sqrt(mean(Y_hat - y.test)^2),
                 AE = sum(abs(Y_hat - y.test)))
kable(error)

# prediction error of bagged model
error_bagged <- ldply(GP_gPro.bagged[,1]) %>%
   melt(id.vars = ".id", value.name = "Y_hat") %>%
   group_by(variable) %>%
   summarise(bagged = mean(Y_hat)) %>%
   summarise(RMSE = sqrt(mean(bagged - y.test)^2),
                 AE = sum(abs(bagged - y.test)))
kable(error_bagged)

# plot predictions of single vs ensemble models
ggplot(error, aes(.id, RMSE)) + geom_bar(stat="identity", fill =
    I("grey50")) +
   geom_hline(data = error_gPro, aes(yintercept = RMSE), colour = "red") +
   geom_hline(data = error_bagged, aes(yintercept = RMSE), colour = "blue") +
   xlab("Sampled GP fit") + ylab("RMSE") + theme_bw()


# GP Ensemble model diagnostics ****************************

# conform data to desired form - ready for diag plots
bag_diag <- ldply(GP_gPro.bagged[,1]) %>%
   melt(id.vars = ".id", value.name = "Y_hat") %>%
   group_by(variable) %>%
   summarise(bagged = mean(Y_hat)) %>%
   mutate(res = y.test - bagged, se = sd(y.test - bagged),
            std_res = res/se, obs = y.test)

# plot observed vs predictions for single and ensemble
ggplot(bag_diag, aes(bagged, obs)) +
  geom_jitter(data = obs_vs_pred,
            aes(y.test, predictions_gPro), colour = "blue",
            size = 3, shape = 2, alpha = 0.5) +
  geom_jitter(alpha = 0.8, size = 2) +
   geom_abline(intercept = 0, slope = 1, colour = "red", size = 1) +
  xlab("Predicted") + ylab("Observed") +
   theme_bw()

# plot standard residuals vs predictions
```

```
ggplot(bag_diag, aes(bagged, std_res)) +
  ylim(-7, 4) +
  geom_point(data = std_residuals, aes(pred, std_res),
             colour = "blue", shape = 2, size = 4, alpha = 0.5) +
   geom_point(size = 2, alpha = 0.8) +
   geom_abline(intercept = -4, slope = 0, colour = "red", size = 1, linetype
       = "dashed") +
   geom_abline(intercept = 4, slope = 0, colour = "red", size = 1, linetype =
       "dashed") +
   geom_abline(intercept = -2, slope = 0, colour = "red", size = 1) +
   geom_abline(intercept = 2, slope = 0, colour = "red", size = 1) +
  xlab("Predicted") + ylab("Standardised Residuals") +
   theme_bw()
```

Iteration Stability

```
set.seed(12)

# number of bootstrap fits
n_iterate <- 30

# predictions on seperate bootstrap fits
iteration_stable <- foreach(i = 1:n_iterate, .combine = cbind) %do% {

   bag <- GP_bag(x.train, y.train, x.test, iterations = 1,
                       size = 35, replace = FALSE)[1]


   return(list(bag))
}

# melt error_gPro into long form
error_gPro.melt <- melt(error_gPro, value.name = "Error")

# unlist predictions into data frame
errors <- data.frame(matrix(unlist(iteration_stable), nrow = dim(x.test)[1]))
# cumulative average of predictions
error_pred <- data.frame(t(apply(errors,1,cumsum)/(seq_len(ncol(errors)))))
# error
RMSE_stabalise <- apply(error_pred, 2, function(x) sqrt(mean(x - y.test)^2))
AE_stabalise <- aaply(error_pred, 2, function(x) sum(abs(x - y.test)))
error.df <- data.frame(iterations = seq(1,length(RMSE_stabalise),1),
                               RMSE = RMSE_stabalise, AE = AE_stabalise)
```

```
# plot errors over iterations
error.df %>%
   melt(id.vars = "iterations", value.name = "Error") %>%
   group_by(variable) %>%
   ggplot(aes(iterations, Error)) +
   geom_bar(stat="identity", fill = "grey50") +
   facet_wrap(~ variable, scales = "free") +
   geom_hline(data = error_gPro.melt, aes(yintercept = Error),
                colour = "red", linetype = "dashed", size = 1) +
   theme_bw()
```

Optimal Bootstrap Sample Size (considerable computing time)

```
# size of bootstrapped samples to iterate over
iterate <- 500

# Sample sizes to simulate over
sample_size <- seq(15, 35, by = 5)

# Utilise parallel computation to simulate 500
# iterations over each size to obtain distribution
bootstrap_size <- foreach(m = 1:iterate, .combine = cbind,
    .packages='reshape2') %:%
   foreach(i = sample_size, .combine = 'c') %dopar% {

      # Fit ensemble GP model
      mod <- GP_bag(x.train, y.train, x.test,
                        iterations = 10, size = i, replace = FALSE)

      # compute error (RMSE & AE)
      RMSE_mod <- ldply(mod[,1]) %>%
         melt(id.vars = ".id", value.name = "Y_hat") %>%
         group_by(variable) %>%
         summarise(bagged = mean(Y_hat)) %>%
         summarise(RMSE = sqrt(mean(bagged - y.test)^2),
                      AE = sum(abs(bagged - y.test)))

      return(list(RMSE_mod))
   }

# Conform data to desired structure
```

```
size_df <- data.frame(iteration = rep(1:iterate, times =
    dim(bootstrap_size)[1], each = dim(bootstrap_size)[1]),
    ldply(bootstrap_size))

kable(head(size_df))


# Further conform data and plot error according to size
size.sim <- size_df %>%
   data.frame(size = sample_size) %>%
   melt(id.vars = c("size", "iteration"),
         measure.vars = c("RMSE", "AE"), value.name = "Error") %>%
   group_by(iteration) %>%
  filter(size < 40) %>%
  mutate(size = as.factor(size)) %>%
   ggplot(aes(size, Error, group = iteration, fill = "obs")) +
   geom_jitter(alpha = 0.25) +
  facet_wrap(~ variable, scales = "free") +
   geom_hline(data = error_gPro.melt, aes(yintercept = Error),
            colour = "red", size = 1) +
  guides(fill = guide_legend(title = "Sim", title.position = "top")) +
   theme_bw()

# Conform data by size and iteration
prob <- size_df %>%
   data.frame(size = sample_size) %>%
   melt(id.vars = c("size", "iteration"),
         measure.vars = c("RMSE", "AE"), value.name = "Error") %>%
   filter(size %in% seq(20,35,5), Error < 250) %>%
   mutate(size = as.factor(size)) %>%
   arrange(size) %>%
   group_by(size)

# Filter necessary data and identify distribution quantiles
quant.RMSE <- prob %>%
  filter(size == 35, variable == "RMSE") %>%
  select(Error)
quant.AE <- prob %>%
  filter(size == 35, variable == "AE") %>%
  select(Error)
quantiles <- data.frame(rbind(as.numeric(quantile(quant.RMSE$Error, probs =
    0.95)), as.numeric(quantile(quant.AE$Error, probs = 0.95))))
colnames(quantiles) <- "quantiles"
variable <- c("RMSE", "AE")
quantiles <- data.frame(cbind(variable, quantiles))
```

```
# plot simulated distributions over sizes and compare
# to single GP model error
size.dens <- ggplot(prob, aes(Error, group = size, fill = size)) +
   geom_density(aes(colour = size), alpha = 0.4) +
   facet_wrap(~ variable, scales = "free") +
   geom_vline(data = error_gPro.melt, aes(xintercept = Error),
                colour = "red", size = 1) +
  geom_vline(data = quantiles, aes(xintercept = quantiles),
            colour = "black", linetype = "dashed") +
   theme_bw()

# arrange plots in a grid formation
grid.arrange(size.sim, arrangeGrob(size.dens, ncol = 1), nrow = 2)
```