Enhancing speed of Gaussian processes

Indian Statistical Institute, Kolkata

Sanchari Sil
Supervisor: Sourav Chakraborty

July, 2020

Abstract

Gaussian Processes are used in supervised learning. They have been in the world of machine learning for quite some time, dealing with complex data sets where parametric methods fail. While calculating the gaussian distribution function for a large feature vector, we need a matrix inversion algorithm which has high run time complexity $O(n^3)$ and space complexity $O(n^2)$. To increase its performance, subset sampling is an important technique used, one method was described in the paper Fast Gaussian Process Regression for Big Data by Sourish Das, Sasanka Roy, Rajiv Sambasivan. It described an algorithm involving combined estimates from models developed using subsets sampled uniformly, much similar to bootstrap sampling. But as a drawback it has been found that the method doesn't work well for all kinds of data. The results developed were based on synthetic data only. In our work we shall provide a different sampling technique. We put weights on the points and sample accordingly. This is thought to be a better approach if the weights are chosen wisely. Empirical results to establish our idea have been provided.

Acknowledgement

I sincerely express my gratitude towards my supervisor, Sourav Chakraborty for providing me with the necessary materials and for all his advices and support.

Contents

Chapter 1

Introduction

When traditional parametric methods are a failure, Gaussian Processes (GP) would be helpful to perform supervised learning tasks on complex datasets. They are convinient to use in comparison to alternatives like neural networks (Rasmussen [2006]). They can be used for both regression and classification. For classification, there are various applications, recognition of handwritten digits for an instance. In case of regression, predictive soil mapping, or to learn inverse dynamics of a robot arm may be some fields where GP can be used.

The main advantage of a Gaussian Process over other alternative models is that, being non parametric, it can model arbitrary functions of the input points. They may be complex to understand but nonethless, lead to simple linear algebric implementations. Applying Gaussian Processes to Regression problems, one can get exact inference. But the computation of the solution requires a matrix inversion. A dataset of size n has $O(n^3)$ as the time complexity of matrix inversion. The space complexity of storing a matrix of size n is $O(n^2)$. This makes the applicability of the technique restricted to small or moderate sized datasets. To apply it for large data sets, different approaches have been tried. *Gaussian Processes for Machine Learning by Rasmussen and Williams*[Chapter 8][1] provides some approximate methods to speed up Gaussian Processes. Here, we shall deal with subset sampling. It may mean to be wasteful of data, but it is sane if the predictions obtained are quite accurate according to our requirement.

Let's get some notions about what a Gaussian Process exactly is.

1.1 Introduction to Gaussian Processes

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.

Let D be a data set of n observations comprising of $(\mathbf{x}_i, y_i)$ where, $i = 1, \ldots, n$. $\mathbf{x}_i$ is the input vector of independent variables of size p, and y_i is the output. A Gaussian process, $f(\mathbf{x})$ is completely specified by its mean function $\mu(\mathbf{x})$ and covariance function, $k(\mathbf{x}, \mathbf{x'})$ where,

$$\mu(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$
$$k(\mathbf{x},\mathbf{x'}) = \mathbb{E}[(f(\mathbf{x}) - \mu(\mathbf{x}))(f(\mathbf{x'}) - \mu(\mathbf{x'})]$$

Then $f(\mathbf{x}) \sim GP(\mu(\mathbf{x}), k(\mathbf{x},\mathbf{x'}))$
For convinience we assume $\mu(\mathbf{x})$ to be $\mathbf{0}$.

1.1.1 Prediction in absence of noise

Let us divide the complete dataset into two subsets, the training set of size N and the test set of size N_*. In the absence of any random noise component, the joint distribution of the output variables $\mathbf{f}$ and $\mathbf{f}_*$ of the training set and the test set respectively, would be:

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

where,

- X and X_* respectively represent the $N \times p$ and $N_* \times p$ covariate matrices of the training and the test set.
- $K(X,X)$ denotes the $N \times N$ matrix representing the variance-covariance matrix of the training set. Similarly, $K(X_*,X_*)$ of dimension $N_* \times N_*$ is the variance-covariance matrix of the test set
- $K(X,X_*)$ is the $N \times N_*$ matrix of covariances evaluated at each pair of the training points and the test points.

The predictive equations for $\mathbf{f}_*$ are:

$$\mathbf{f}_* \mid X, X_*, \mathbf{f} \sim N(\hat{\mathbf{f}}_*, cov(\mathbf{f}_*))$$
$$\hat{\mathbf{f}}_* = \mathbb{E}[\mathbf{f}_* \mid X, \mathbf{f}, X_*] = K(X_*,X)K(X,X)^{-1}\mathbf{f} \tag{1.1}$$
$$cov(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*) \tag{1.2}$$

1.1.2 Prediction in presence of noise

In real world, we may not have access to functional values but their noisy versions. Given a random noise component η, where $\eta \sim N(0, \sigma_n^2)$, our model can be written as

$$y = f(\mathbf{x}) + \eta \tag{1.3}$$

Here:

- y represents the target variable to be predicted
- $\mathbf{x}$ represents the input vector of co-variates
- The noise terms η are assumed to be *iid* random variables drawn from $N(0, \sigma_n^2)$

Then,

$$cov(\mathbf{y}) = K(X, X) + \sigma_n^2 I \tag{1.4}$$

The joint distribution of the observed values of the target variable y and the function value of the test set f_* can be written as:

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

We want to predict the the response value $\mathbf{f}_*$, at test point X_* and the predictive equations are:

$$\mathbf{f}_* \mid X, \mathbf{y}, X_* \sim N(\bar{\mathbf{f}}_*, cov(f_*))$$

$$\bar{\mathbf{f}}_* = \mathbb{E}[\mathbf{f}_* \mid X, \mathbf{y}, X_*] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y} \tag{1.5}$$

$$cov(\mathbf{f}_*) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*) \tag{1.6}$$

where I is the $N \times N$ identity matrix.

Equation (1.5) requires the computation of an inverse which has $O(N^3)$ time complexity and $O(N^2)$ space complexity. These become a hurdle in the use of GPs. Use of the entire training set becomes difficult when the data size is very large. In this era of big data we often come across such data sets. Since GP regression is both effective and versatile on complex datasets in comparison to parametric methods, a solution to the hurdles mentioned above will enable us to apply GP regression to large datasets that are complex.

1.2 The Problem

When the matrix K(X,X) is quite large, the GP regression becomes slow and we need an alternate method to speed up the regression. The goal of our thesis is to achieve the same. We give an algorithm that can be proved to be correct. A good method can be to cleverly choose a submatrix of the large matrix $K(X, X)$ and use the same to predict the response function $\mathbf{f}_*$. We want to find an estimate, $\hat{\bar{\mathbf{f}}}_*$ that is quite close to the predicted response $\bar{\mathbf{f}}_*$, obtained from original Gaussian regression.

The sketch of the algorithm is as follows:

We know that there may be clusters in our data which leads to bad predictions when SRSWR procedure is followed during sampling a submatrix from K(X,X). Hence we get the clusters, find their sizes and attach weights to each data points as the inverse of the relative frequency of the cluster to which they belong. This seems to give a uniform covarage of all data points when sampling is done.

In a previous work by *Fast Gaussian Process Regression for Big Data Sourish Das, Sasanka Roy and Rajiv Sambasivan*, similar method was used, that is to be discussed in the next chapter. We are intended to get another sampling method which we claim to be better than that of Das, Roy and Sambasivan. Chapter 3 shall deal with the same. Chapter 4 shall be the analytical results of the new algorithm followed by conclusions in Chapter 5.

Chapter 2

Previous Work

2.1 Sketch of the work and the Algorithm

The previuos work on making the Gaussian Process regression faster has been given in *Fast Gaussian Process Regression for Big Data by Sourish Das, Sasanka Roy, Rajiv Sambasivan*[2]. Estimators were developed on small subsets of the data and applied to the big data set. k small subsets of size N_S were taken from the original datasets using Simple Random Sampling With Replacement. Then a Gaussian Process estimator was developed for each such sample. To combine the estimators, the predictions were averaged over all k estimators. The method was similar to bootstrap estimation. The required size of the sample was estimated empirically.

The algorithm thus used is as follows:

Algorithm 1

Input: A dataset D of size n, no. of iterations k

Output: An estimator $\tilde{f}$ that combines the estimators fitted from resampling

1: Select hyperparameters k and N_S
2: **for** $i = 1, 2, \ldots, k$ **do**
3: Choose a sample of size N_S uniformly from 1,2,. . . ,N
4: Uniformly sample from $K(X, X)$, the $N \times N$ submatrix S using the corresponding indices of the sample indices chosen in the previous step.
5: $\tilde{y} \leftarrow$ subset of y with features corresponding to the indices in sampx
6: $Cov\_S \leftarrow$ Subvector of the Covariance vector between training features and the test feature chosen according to the sample of indices.
7: $\tilde{f} \leftarrow \tilde{f} + Cov\_S(X_*, X)[S + \sigma_n^2 I]^{-1}\tilde{y}$
8: **end for**
9: $\tilde{f} \leftarrow \tilde{f}/k$
10: return $\tilde{f}$

The time complexity is reduced from $O(N^3)$ to $O(kN_S^3)$ as now we have to perform matrix inversion on the submatrix of size N_S and the same repeated for k iterations.

The method is based on model averaging. Given, $i = 1, 2, ...k$ models, and given a point X, the conditional distribution of the target variable Y given X is:

$$P(Y \mid X) = \sum_{i=1}^{k} P(Y \mid i, X)P(i)$$

Here it is assumed that each of the model is equally probable, then $p(i) = \frac{1}{k}$.
$P(Y \mid X) \sim N(\mu(x), \sigma^2(x))$,
$\mu(x) = \frac{1}{k}\sum_{i=1}^{k} \mu_i(x)$
$\sigma^2(X) = \frac{1}{k^2}\sum_{i=1}^{k} \sigma_i^2(X)$

The correctness of the algorithm was not proved but analytical results showed that it is empirically true. The results reported were consistent with the results from Minimax theory for non-parametric regression. Simplicity of the algorithm was the main advantage. In general the methods used for applying Gaussian Process regression to large data sets required use of large number of hyper parameters unlike that we are talking about.

2.2 Parameters involved and their estimation

The key parameters of the algorithm were size of the subset and accuracy(specified as a error threshold, ϵ). Two methods were used for estimating the size of the sample.

- **Using Statistical Inference**: Given a dataset of size N, the subset size N_S was represented as

  $$N_S = N^d, \ where \ 0 \leq d \leq 1$$

  .
  d was determined by inference from a proportion using a small sample. Using Simple Random Sampling, a small sample of the original data set was picked. Starting with a small value of d, algorithm 1 was run iterativel incrementing the value of d by a certain fraction at each iteration. The iterative procedure ended when the prediction error became acceptatble that is, lied within a pre-determined threshold value. The size of the data set used being small, the procedure was quite fast. It resulted in a reliable estimate of d.

- **Using Empirical Estimation**: There were two propositions involved that said:

  1. Subset size, N_S is proportional to data size N, that is, $N_S \propto N$
  2. N_S is a decreasing function of the error rate, ϵ.

  These two propositions led to the following empirical estimator of the sample size:

  $$N_S = \frac{N^{\delta(N)}}{g(\epsilon)}$$

  where,
  $\delta(N)$ defines the fact that N_S is proportional to data size N
  $g(\epsilon)$ define the fact that N_S should increase with the decrement in ϵ, the desired error rate.

2.2.1 Effect of the parameters

Algorithm 1 involves 3 parameters, Data size, Subset size, Number of estimators. Based on performed experiments, they got some relation of these parameters on accuracy of the predicted value. The effect of the three hyperparameters used, as observed from experiments, were:

1. **Dataset size**: For each data set of size N, a fraction N^d was picked, and a good estimate of d was obtained to provide a preset target accuracy. It was inferred from the performed that the d required to maintain the given accuracy decreased very slowly with increase in N.

2. **Subset Size**: The subset of size, N_S , used for Gaussian Process model development was N^d. The RMSE, Root Mean Square Error(ϵ), was recorded for a range of values of d. Because of the effect reported in Section 5.2 of [2], d must decrease with increase in Subset Size For complex data sets like Sinc function, the subset size required to get a required level of accuracy was large.

3. **Number of Estimators**: The key observation from the performed experiments was that given a subset size, N_S, the error drops with increase in number of estimators k upto a point after which increasing k has not much effect on the error. Plausibly the explanation for this behavior could be that increase in k reduces the variance component of the error in the bias-variance decomposition of the error(ϵ).

The target variable of each data used, depended on a small number of features, but data size ranged from few thousand to two million. Mainly synthetic data was used and the characteristic of the data provided a good reason for the attractive rate of convergence

as indicated by the results from Mini-max theory for non-parametric regression.

According to the experimental results used, it was inferred that we could get reasonable models with a small subset of data, when the no. of predictor variables were small. The flaw of the above algorithm is that the method doesn't work well with data of varying density. Moreover, the sample was taken independently and the results obtained were based on synthetic data sets. Hence we provide a better (by our claim) algorithm, that would decrease the error as well as speed up our GP Regression.

Chapter 3

Our Algorithm

This Chapter deals with an algorithm that uses certain weights during sampling procedure. We know that density of a data may vary. In fact, there may be clusters in the data set. Use of Algorithm 1 for such data sets may not provide a good estimate of the target variable. This is because when we are randomly selecting from N data points, many points of the same cluster may get into our subset, while if the size of some cluster is very small, there may be no representative point from that cluster in our subset. This is a major problem during estimation. After missing value treatment and outlier removal of the given data, when we are sampling, we need a good subset of sample points so as to correctly predict the target variable of the test point. Hence we use the approach given in the following section.

3.1 An overview

Given $C_1, C_2, \ldots, C_t$, are clusters obtained from the training part of the data using a clustering algorithm like k-means. Each cluster C_i is of size n_i, $i = 1, 2, \ldots, t$. We define $S_n = \sum_{i=1}^{t} n_i$
We associate with each training point $j \in [N]$, a factor a_j such that,

$$a_j = \frac{S_n}{n_i} \quad if\ j \in C_i$$

This a_j is the associated weight with the j^{th} training point.
$A = \{a_1, a_2, \ldots, a_N\}$ is the vector of weights for the training set. Weights of data points within the same cluster will be same and for different clusters the weights would be inversely proportional to the cluster size, n_i. Thus, intuitively, when sampling is done each of the clusters shall be given attention uniformly.

Now we give the algorithm to obtain the subset using weighted sampling as described

above. We first have split the data into training and test set. Then using k-means clustering algorithm divided our data into an optimum number of clusters. The optimum number can be determined by several procedures. Elbow, Silhouette and Gap Statistic methods are some of those. We have used the Silhoutte index.
The Silhoutte value measures how similar a point is to its own cluster compared to other clusters.
Define,

$$a(i) = \frac{1}{n_l - 1} \sum_{j \in C_l, i \neq j} d(i,j) \quad \forall i \in C_k$$

$$b(i) = \min_{k \neq l} \frac{1}{n_k} \sum_{j \in C_l} d(i,j) \quad \forall i \in C_l$$

Silhoutte value for i^{th} data point is

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & if\ a(i) < b(i) \\ 0 & if\ a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1 & if\ a(i) > b(i) \end{cases}$$

The average silhoutte value over all t clusters is max for a specific no. of clusters T. Then T is the required optimum.
We then divide the data set into T clusters and run Algorithm 2.

3.2 The Algorithm

The theory described in the previous section is the backbone of Algorithm 2. The convergence of the clustering algorithm involved ensures the formation of weight vector corresponding to the data points.
Here too, the time complexity of the algorithm 2 is reduced to $O(k * N_S^3)$, though the pre-processing involving clustering algorithm takes $O(N^2)$.

Both the algorithms are fast but the advantage with the second is that it gives a better estimate of the target variable as obtained from experiments. While we believe that we can prove the correctness of the algorithm, we couldn't do it due to lack of time. Instead we have run algorithm 2 on real life data and concluded the results to be discussed in the next Chapter.

Algorithm 2

Input: A dataset D of size n

Output: An estimator $\hat{\tilde{f}}_*$ that combines the estimators fitted from resampling

Using Silhoutte Averaging, obtain optimim number of clusters, T
Perform k-means clustering algorithm to obtain T clusters.
Find the cumulative frequencies of each cluster.
For each $j \in C_i$, $a_j = \frac{S_n}{n_i}$
Define the weight vector A, where $A = (a_1, a_2, ..., a_N)$ each a_j being associated weight of j^{th} training point.
for i=1,2,...k **do**
    $\widetilde{y} \leftarrow$ subset of y sampled using A as weights
    $S \leftarrow$ Submatrix of K formed by features in $\widetilde{y}$.
    $Cov_S \leftarrow$Cov vector between training features and the test feature
    $\hat{\tilde{f}}_* \leftarrow \hat{\tilde{f}}_* + Cov\_S(X_*, X)[S + \sigma_n^2 I]^{-1}\hat{y}$
end for
$\hat{\tilde{f}}_* \leftarrow \hat{\tilde{f}}_*/k$
return $\hat{\tilde{f}}_*$

3.3 Algorithm for Gaussian Regression

The original algorithm for a Gaussian process regression to estimate the target variable, is given below:

Algorithm 3

Input: A dataset D of size N

Output: $\bar{f}$

$\bar{f} \leftarrow K(X_*, X)[K(X, X) + \sigma_n^2 \boldsymbol{I_n}]^{-1} y$
return $\bar{f}$

Consistency of Gaussian Process regression has been discussed in [1], section 7.1, when considering square error loss.

We can compare the Error variances and Time compexity associated with $\bar{f}, \widetilde{f}$ and $\hat{\tilde{f}}_*$ empirically. The next chapter uses real life data to support our ideas.

Chapter 4

Analytical Results

Given a data set D of size n, we have first split D into a training set and test set. The train-test split used is 80-20. We first pre-process our data removing missing values and outliers. After that we run the three algorithms on each of the three data sets to obtain desired results. Lets first see the data used:

4.1 Datasets

The following datasets were used:

1. **data 1**: Concrete_Data.csv from https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength. Concrete is the most important material in civil engineering. The concrete compressive strength is a highly nonlinear function of age and ingredients. These ingredients include cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. There are 1030 instances, number of attributes 9 (8 quantitative input variables and 1 quantitative output variable)

2. **data 2**: House price from https://www.kaggle.com/shree1992/housedata. It has 4600 rows and 10 columns. The attribute, 'price' being our target variable. The independent attributes are, bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors", [6] "waterfront", "view", "condition", "sqft_above" and "sqft_basement"

3. **data 3**: Airfoil_self_noise from https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise. The attributes are angle of attack, in degrees, chord length, in meters free-stream velocity, in meters per second, suction side displacement thickness, in meters. The only output is scaled sound pressure level, in decibels. There are 1503 data points.

data	Algorithms	No.of training points	No. of test points	Time of execution	average absolute error	variance of the estimate	Maximum absolute error (amongst all test points)
data1	Algorithm1	824	206	1.140631e-01	7.354844	88.01718	27.90448
	Algorithm2			1.728981e-01	7.330808	86.21974	27.31775
	Algorithm3			2.413	7.308073	85.69483	27.3466
data2	Algorithm1	3680	920	0.114955	9.874916e-06	1.884632e-12	7.003297e-06
	Algorithm2			0.2470450	1.016547e-07	1.799822e-12	6.988388e-06
	Algorithm3			4.12390	9.639035e-07	1.633238e-12	6.462059e-06
data3	Algorithm1	1203	300	0.21390	26.83056	1060.556	79.77511
	Algorithm2			0.207	26.46328	1058.624	71.91087
	Algorithm3			2.82	26.05414	1001.452	71.07715

Table 4.1: Sample size= 250

data	Algorithms	No.of training points	No. of test points	Time of execution	average absolute error	variance of the estimate	Maximum absolute error (amongst all test points)
data1	Algorithm1	824	206	0.90806	7.901368	98.87474	29.5766
	Algorithm2			0.01810680	7.351767	86.9728	27.02792
	Algorithm3			2.413	7.308073	85.69483	27.3466
data2	Algorithm1	3680	920	0.1008700	1.05469e-06	1.948816e-12	7.111638e-06
	Algorithm2			0.1082000	1.042754e-06	1.900779e-12	6.997945e-06
	Algorithm3			4.12390	9.639035e-07	1.633238e-12	6.462059e-06
data3	Algorithm1	1203	300	1.208033e-01	36.5032	1460.941	73.85941
	Algorithm2			0.11948	32.56631	1448.658	71.40905
	Algorithm3			2.82	26.05414	1001.452	71.07715

Table 4.2: Sample size= 100

4.2 Application of the three algorithms on data sets

All the three algorithms were implemented in R-programs. Clustering was done using k-means() function provided in the package 'cluster'. Other functions used involved sample() and int.sample() in algorithms 2 and 1 respectively to draw random samples under the given conditions. Other functions were written as per the requirement of the programs.

The tables in this chapter compare the mean absolute error, variance of the estimate and time of execution for the three algorithms corresponding to the above data

Description of the tables

The columns of the tables are :

1. data: The different data sets
2. Algorithms: The three Algorithms, 1,2 and 3 are compared for each data set

3. Time of execution: The average time of execution for any algorithm.

4. Average absolute error: Average over the absolute error that is the absolute difference between the predicted value and the observed response, the average being taken over all k iterations.

5. Variance of the estimator: If $\bar{f}_{*i}$ be the estimate of i^{th} test point, Variance of the estimator is

$$Var = \frac{1}{N_*} \sum_{i=1}^{N_*} (\bar{f}_{*i} - y_{*i})^2$$

   where N_* is the size of test set and y_{*i} is the value of the i^{th} target variable in the test set.

6. Maximum absolute error: The maximum absolute difference between the predicted value and the observed response among all the test points.

The tables show how each of the algorithms are related. We find that the Time complexity of ALgorithm 1 and Algorithm 2 are much less than that of Algorithm 3. Also the average absolute error and the variance of the estimates are less for the second algorithm in any of the given data. That is, the second algorithm gives better prediction of the test set target variable than the first. It predicts value closer to the one obtained from Algorithm 3, the Gaussian Process regression. Analytically we see our claim that algorithm 2 is better than algorithm 1, holds.

Chapter 5

Conclusion

The main problem with GP regression is the infinite dimensional parameters of the priors which makes the proof for consistency of posterior distribution a fine challenge. In [4], almost sure consistency for posterior distributions is verified, when the prior distribution on the regression function is a GP, for a single dimensional covariate. We can extend it to multidimensional covariates, to prove almost sure consistency of Algorithm 1 and 2. It woud be a future work. Here though the proofs could not be done, we have used a few data sets to compare the 3 different algorithms and check their behaviour empirically. The datasets obtained were from various fields, with varying feature sizes. We have obtained the given results for fixed hyperparameters. The time complexity is reduced from order $O(n^3)$ to $O(kN_s{}^3)$ where N_s is the sample size and K, the number of iterations. Empirically, Algorithm 2 gives a better prediction than Algorithm1 the error variance being smaller. Also we see that as sample size increases, the error variance decreases towards that of Algorithm 3. Due to lack of time though the theoritical proof of correctness have not been done, we have established a few results to stand by our notion. Although the clustering algorithm takes much time, the advantage of Algorithm 2 is that it is simple and more accurate.

Chapter 6

Bibliography

[1] Gaussian Processes for Machine Learning Carl Edward Rasmussen and Christopher K. I. Williams MIT Press, 2006.

[2] Fast Gaussian Process Regression for Big Data Sourish Das 1 , Sasanka Roy 2 , and Rajiv Sambasivan 1 1 Chennai Mathematical Institute 2 Indian Statistical Institute

[3] Posterior Consistency in Non Parametric Regression Problems under Gaussian Process Priors By Taeryon Choi and Mark J. Schervish Carnegie Mellon University

[4] Posterior Consistency of GP Regression by Choi and Schervish(2007)

[5] Gaussian processes by Chuong B. Do, 2007