

**A MODIFIED APPROACH TOWARDS GAUSSIAN PROCESS
REGRESSION BASED ON INCREMENTAL LEARNING**

by

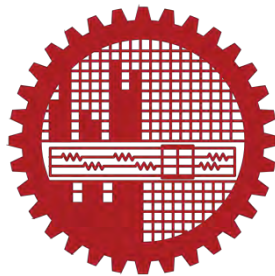
MEHNUMA TABASSUM

A thesis submitted to the
Department of Industrial and Production Engineering,
Bangladesh University of Engineering and Technology,
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

INDUSTRIAL AND PRODUCTION ENGINEERING



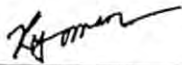
Department of Industrial and Production Engineering (IPE)
Bangladesh University of Engineering and Technology (BUET)

December 28, 2020

CERTIFICATE OF APPROVAL

The thesis titled “A Modified Approach towards Gaussian Process Regression based on Incremental Learning” submitted by Mehnuma Tabassum (Student ID: 1017082027, Session: October 2017) has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Industrial and Production Engineering.

BOARD OF EXAMINERS



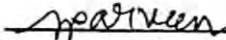
Dr. AKM Kais Bin Zaman
Professor
Department of IPE, BUET, Dhaka

Chairman
(Supervisor)



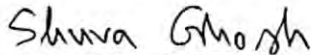
Dr. Nikhil Ranjan Dhar
Professor and Head
Department of IPE, BUET, Dhaka

Member
(Ex – Officio)



Dr. Sultana Parveen
Professor
Department of IPE, BUET, Dhaka

Member



Dr. Shuva Ghosh
Associate Professor
Department of IPE, BUET, Dhaka

Member



Dr. Md. Farhad Hossain
Professor
Department of EEE, BUET, Dhaka

Member
(External)

CANDIDATE'S DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Mehnuma Tabassum

Student ID: 1017082027

ABSTRACT

This research propounds a variant of Gaussian process regression called *Sparse Incremental Gaussian Process Regression (si-GPR)*. Traditional Gaussian process regression, although attractive for smaller datasets, possesses some inherent limitations regarding computation and storage requirements. As a result, traditional or batch-GPR performs poorly when employed for training a large set of data. Again, this algorithm can neither accommodate inputs that become available over time nor the training set that needs the removal of training points. Therefore, appropriate methods are necessitated to deal with these limitations effectively. To accomplish that, a sparse Gaussian process regression algorithm with an incremental learning and a decremental un-learning policy has been proposed. In this formulation, the idea of sparsification and undertaking streaming inputs have been merged with the provision of model forgetting. The rationale behind following this strategy is three-fold: to lessen the number of data instances to fit, to accommodate streaming input, and to minimize the computation and memory requirements as possible. As the prime aspiration was to lower the calculations without losing the accuracy, an economic update of the kernel matrix and the inverted lower Cholesky matrix has been rendered. The outcome of this research manifests promising results as it provides a general reduction in memory consumption and execution time. Moreover, the proposed *si-GPR* algorithm provides better fitting and predictions over the original Gaussian process regression.

ACKNOWLEDGEMENT

The author acknowledges deepest thanks and indebtedness to the honorable supervisor, Dr. AKM Kais Bin Zaman, *Professor, Department of Industrial and Production Engineering, BUET* for his constant guidance and whole-hearted supervision throughout the M.Sc. thesis. His valuable suggestions and encouragements urged the author to complete this thesis on-time. It was an honor for the author to work under his supervision.

The author extends her thanks to Dr. Nikhil Ranjan Dhar, *Professor and Head, Department of Industrial and Production Engineering, BUET*, Dr. Sultana Parveen, *Professor, Department of Industrial and Production Engineering, BUET*, Dr. Shuva Ghosh, *Associate Professor, Department of Industrial and Production Engineering, BUET*, and Dr. Md. Farhad Hossain, *Professor, Department of Electrical and Electronics Engineering, BUET* for the constructive assessment and remarks.

The family has always been a part of the author's journey. She expresses her gratitude for doing this thankless job.

Last but not the least; the author conveys heartfelt regards to the Almighty Allah, the greatest, for every blessing He has bestowed upon her, for nothing would have been possible without that.

LIST OF TABLES

Sl. No.	Title of the Table	Page No.
Chapter-3		
1	Table 3.1: Popular kernel functions	25
Chapter-5		
2	Table 5.1: Notations for <i>si</i> -GPR algorithm	44
3	Table 5.2: Initial sparse algorithm	55
4	Table 5.3: Data clustering algorithm, CURE	57
5	Table 5.4: Algorithm for concatenation of clusters	58
6	Table 5.5: Algorithm for incremental learning	60
7	Table 5.6: Algorithm for decremental un-learning	61
Chapter-6		
8	Table 6.1: Datasets used in experimentations	64
9	Table 6.2: Specification of the system used for experimentation	73
Chapter-7		
10	Table 7.1: Results of preliminary testing	76
11	Table 7.2: Chosen kernels for basic GPR and <i>si</i> -GPR	79
12	Table 7.3: Comparison of the number of training points for basic GPR and <i>si</i> -GPR	80
13	Table 7.4: Results obtained for training sets using basic GPR	81
14	Table 7.5: Results obtained for training sets using <i>si</i> -GPR	82
15	Table 7.6: Results obtained for test sets using basic GPR	86
16	Table 7.7: Results obtained for test sets using <i>si</i> -GPR	86
17	Table 7.8: Comparison of total memory requirement for basic GPR and <i>si</i> -GPR	92
18	Table 7.9: Comparison of average total run time for basic GPR and <i>si</i> -GPR	93

LIST OF FIGURES

Sl. No.	Title of the Figure	Page No.
Chapter-3		
1	Figure 3.1: Classification of covariance functions	20
2	Figure 3.2: Variations of the exponentiated quadratic kernel	22
3	Figure 3.3: Variations of the rational quadratic kernel ($\sigma=1$)	22
4	Figure 3.4: Variations of the Matérn kernel ($\sigma=1$)	23
5	Figure 3.5: Variations of the dot product kernel	24
6	Figure 3.6: Variations of the periodic kernel	25
7	Figure 3.7: Graphical representation of Gaussian process regression	32
Chapter-4		
8	Figure 4.1: The architecture of batch learning	36
9	Figure 4.2: The architecture of incremental learning	39
Chapter-5		
10	Figure 5.1: Flowchart for the proposed <i>si</i> -GPR algorithm	48
11	Figure 5.2: Visualization of incremental updates of the kernel matrix	50
12	Figure 5.3: Visualization of decremental un-learning strategy	53
13	Figure 5.4: Visualization of obtaining the representative dataset	59
Chapter-6		
14	Figure 6.1: Training signal for the Wool dataset	65
15	Figure 6.2: Training signal for the Manaus dataset	66
16	Figure 6.3: Training signal for the Tree Ring dataset	66
17	Figure 6.4: Pair plot for the Istanbul Stock Exchange dataset	67
18	Figure 6.5: Pair plot for the German Healthcare Dataset	68
19	Figure 6.6: Pair plot for the Abalone dataset	69
20	Figure 6.7: Pair plot for the Pumadyn-8nm dataset	70

Chapter-7

21	Figure 7.1:	Comparison of training points for both methods	81
22	Figure 7.2:	Comparison of training accuracy of basic and proposed GPR	83
23	Figure 7.3:	Comparison of estimated memory requirement for both methods	84
24	Figure 7.4:	Comparison of average training time for both methods	85
25	Figure 7.5:	Predictions on the Wool test set using basic GPR	87
26	Figure 7.6:	Predictions on the Wool test set using proposed GPR	87
27	Figure 7.7:	Predictions on the Manaus test set using basic GPR	88
28	Figure 7.8:	Predictions on the Manaus test set using proposed GPR	88
29	Figure 7.9:	Predictions on the Tree Ring test set using basic GPR	88
30	Figure 7.10:	Predictions on the Tree Ring test set using proposed GPR	88
31	Figure 7.11:	Comparison of Mean Absolute Error (MAE) for both methods	89
32	Figure 7.12:	Comparison of Root Mean Squared Error (RMSE) for both methods	90
33	Figure 7.13:	Comparison of estimated testing memory requirement for both methods	91
34	Figure 7.14:	Comparison of average testing time for both methods	91
35	Figure 7.15:	Comparison of estimated total memory requirement for both methods	93
36	Figure 7.16:	Comparison of average total run time for both methods	94
37	Figure 7.17:	Memory and time savings using <i>si</i> -GPR	94

LIST OF ACRONYMS

AI	: Artificial Intelligence
CURE	: Clustering Using Representatives
DITC	: Deterministic Independent Training Conditional
DTC	: Deterministic Training Conditional
EM	: Expectation-Maximization
EP	: Expectation Propagation
EQ	: Exponentiated Quadratic
FITC	: Fully Independent Training Conditional
FTC	: Fully Training Conditional
GP	: Gaussian Process
GPC	: Gaussian Process Classification
GPR	: Gaussian Process Regression
<i>i</i> -VSGPR	: Incremental Variational Sparse Gaussian Process Regression
JIT	: Just in Time
LR	: Linear Regression
LU	: Lower-Upper
MAE	: Mean Absolute Error
MAP	: Maximum a Posteriori
MAPE	: Mean Absolute Percentage Error
MPE	: Mean Percentage Error
MSE	: Mean Squared Error
NaN	: Not a Number
nMLE	: Negative Log Marginal Likelihood
NLR	: Nonlinear Regression
OSMGP	: Online Sparse-Matrix Gaussian Process
PD	: Positive Definite
PDF	: Probability Density Function
PSD	: Positive Semi-definite
PITC	: Partially Independent Training Conditional

PPA : Projected Process Approximation
RBF : Radial Basis Function
RKHS : Reproducing Kernel-Hilbert Space
RMSE : Root Mean Squared Error
RQ : Rational Quadratic
RVM : Relevance Vector Machine
SE : Squared Exponential
si-GPR : Sparse Incremental Gaussian Process Regression
SGPR : Sparse Gaussian Process Regression
SMGP : Sparse Multiscale Gaussian Process
SoR : Subset of Regressors
SSGP : Sparse Spectrum Gaussian Process

TABLE OF CONTENTS

	Page No.
ABSTRACT	iv
ACKNOWLEDGEMENT	v
LIST OF TABLES	vi
LIST OF FIGURES	vii
LIST OF ACRONYMS	ix
CHAPTER1: INTRODUCTION	1
1.1 Framework of the Study	1
1.2 Objectives with Specific Aims	2
1.3 Possible Outcome of the Research	2
1.4 Outline of the Methodology Used	3
1.5 Contribution of the Study	4
1.6 Limitation of the Study	4
1.7 Organization of the Thesis	4
CHAPTER2:BACKGROUND LITERATURE	6
2.1 General Background of Gaussian Process Regression	6
2.2 Sparse Methodologies of Gaussian Process Regression	7
2.3 Incremental Methodologies of Gaussian Process Regression	11
CHAPTER3: GAUSSIAN PROCESS REGRESSION	14
3.1 Multivariate Gaussian Distribution	14
3.1.1 Marginalization	15
3.1.2 Conditioning	16
3.2 Bayesian Philosophy	16
3.3 Gaussian Process	18
3.3.1 Mean Function	19
3.3.2 Covariance Function	19

3.3.2.1	Stationary Covariance Function	20
3.3.2.1.1	Exponentiated Quadratic Kernel	21
3.3.2.1.2	Rational Quadratic Kernel	22
3.3.2.1.3	Matérn Kernel	23
3.3.2.2	Non-stationary Covariance Function	23
3.3.2.2.1	Dot Product Kernel	24
3.3.2.2.2	Periodic Kernel	24
3.3.2.3	Kernel Hyperparameters	26
3.3.2.4	Validity of Kernels	26
3.3.2.5	Significance of Kernels	26
3.4	Gaussian Process Regression (GPR)	27
3.4.1	Defining the Prior	28
3.4.2	Posterior Predictive Distribution	29
3.4.3	Optimizing Kernel Hyperparameters	31
3.4.4	Merits of Gaussian Process Regression	32
3.4.5	Shortcomings of Gaussian Process Regression	33
CHAPTER 4:	LEARNING DYNAMICS IN MACHINE LEARNING	35
4.1	Batch Learning	35
4.1.1	Assumptions of Batch Learning	35
4.1.2	Characteristics of Batch Learning	37
4.1.3	Merits of Batch Learning	37
4.1.4	Limitations of Batch Learning	37
4.2	Incremental Learning	38
4.2.1	Characteristics of Incremental Learning	38
4.2.2	Merits of Incremental Learning	40
4.2.3	Challenges with Incremental Learning	40
4.2.4	Applications of Incremental Learning	42
CHAPTER 5:	PROPOSED SPARSE INCREMENTAL GAUSSIAN PROCESS	44
	REGRESSION	

5.1	Notations used for the Proposed <i>si</i> -GPR Algorithm	44
5.2	General Overview of Sparse Incremental Gaussian Process Regression (<i>si</i> -GPR)	45
5.2.1	Phase-1: Approximation/ Sparsification	45
5.2.2	Phase-2: Incremental Learning	46
5.2.3	Phase-3: Decremental Un-learning	47
5.3	Detailed Breakdown of the Algorithm	49
5.4	Pseudocodes for Algorithms used	55
5.4.1	Pseudocode for Initial Sparse Algorithm	55
5.4.1.1	Pseudocode for the Data Clustering Algorithm, CURE	56
5.4.1.2	Pseudocode for the Concatenation Algorithm	58
5.4.2	Pseudocode for Incremental Learning Algorithm	59
5.4.3	Pseudocode for Decremental Un-learning Algorithm	60
CHAPTER6: DATASETS AND EXPERIMENTATION		62
6.1	Brief Overview of the Datasets	62
6.1.1	Wool Dataset	62
6.1.2	Istanbul Stock Exchange Dataset	63
6.1.3	Manaus Dataset	63
6.1.4	German Healthcare Dataset	63
6.1.5	Abalone Dataset	63
6.1.6	Tree Ring Dataset	64
6.1.7	Pumadyn-8nm Dataset	64
6.2	Visualization of the Datasets	65
6.3	Data Preprocessing	71
6.3.1	Data Scaling	71
6.3.2	Missing Value Imputation	72
6.3.3	Label Encoding	73
6.3.4	One Hot Encoding	73
6.4	Environment for Experimentation	73
6.5	Preliminary Testing for Kernel Selection	74

CHAPTER7: RESULT ANALYSIS	75
7.1 Results of Preliminary Analysis	75
7.2 Selected Kernels for Datasets using Basic GPR and <i>si</i> -GPR	79
7.3 Results from Experimentations on Training Sets	79
7.3.1 Comparison of Training Points used in Basic GPR and <i>si</i> -GPR	80
7.3.2 Results of Basic GPR on Training Sets	81
7.3.3 Results of <i>si</i> -GPR on Training Sets	82
7.3.4 Visual Comparison of Basic GRR and <i>si</i> -GPR on Training Sets	82
7.4 Results from Experimentations on Test Sets	85
7.4.1 Results of Basic GPR on Test Sets	86
7.4.2 Results of <i>si</i> -GPR on Test Sets	86
7.4.3 Visual Comparison of Basic GRR and <i>si</i> -GPR on Test Sets	87
7.5 Overall Memory and Time Requirement	92
CHAPTER8: CONCLUSIONS AND FUTURE WORK	96
8.1 Concluding Remarks	96
8.2 Avenues for Future Research	97
REFERENCES	98

CHAPTER-1

INTRODUCTION

1.1 Framework of the Study

The emergence of machine learning is opening up newer dimensions in the field of engineering to tackle a wide variety of tasks. Learning from data without any prior knowledge of the system has changed the outlook of researchers. The field of Industrial and Production Engineering is no different from this paradigm. Machine learning has found its applications in various aspects of industrial engineering such as handling scheduling problems (Aytug, Bhattacharyya, Koehler, & Snowdon, 1994; Jain & Meeran, 1998), forecasting and supply chains (Carbonneau, Laframboise, & Vahidov, 2008; Lee, Kim, Park, & Kang, 2014), quality control (Lease, 2011), etc., and in production engineering such as managing uncertainties in manufacturing systems (Monostori, 2003), building just-in-time (JIT) production scheme when the system characteristics are stochastic and run on a rolling basis (Markham, Mathieu, & Wray, 2000; Wray, Rakes, & Rees, 1997), the problem of varying qualities of metal etching (Chatterjee, Croley, Ramamurti, & Chang, 1997), tool wear prediction (Dongdong Kong, Chen, & Li, 2018), machining parameter optimization (Gupta, Guntuku, Desu, & Balu, 2015; Jurkovic, Cukor, Brezocnik, & Brajkovic, 2018), and so on.

Regression analysis is a sub-category of supervised machine learning that enables to estimate the relationship that links the input variable(s) to the response variable(s). Gaussian process regression (GPR) is one such practice. GPR is a widely used kernel-based algorithm. However, the algorithm is expensive. Owing to the greater effort regarding computations which is $O(d^3)$, where d signifies the number of data instances (Rasmussen & Williams, 2006), the process takes a great amount of time to perform the fitting and projection. Furthermore, the memory it takes to train a model and carry out predictions is $O(d^2)$ which is also substantial (Smola & Bartlett, 2001).

To mitigate this problem, efficient algorithms have been proposed (Cheng & Boots, 2016; Lütz, Rodner, & Denzler, 2013; Nguyen-Tuong, Seeger, & Peters, 2009). A major proportion of those algorithms proposed sparse solutions to the problem, some preferred online learning,

while the rest provided other measures to improve the fundamental algorithm from its inherent pitfalls.

This thesis addresses the limitation of the basic Gaussian process regression. To solve this issue, an incremental learning strategy based Gaussian process regression approach is proposed in this work that collaborates a sparsification operation, an incremental update, and a decremental un-learning approach to make the procedure more efficient. In the case of a new data point's arrival, the model does not retrain the whole dataset again, rather with its inexpensive updates of kernel matrix and its lower Cholesky factor, the algorithm provides a good fit with much less computational effort. The proposed modifications are expected to serve the following functions:

- (i) The memory usage is reduced by a greater amount.
- (ii) The execution time is far less than the general algorithm.
- (iii) The accuracy is not compromised given the size of the dataset.

1.2 Objectives with Specific Aims

The specific objectives of this research are:

- To modify the existing Gaussian process regression routine with the introduction of incremental learning in a certain way that the resulting algorithm offers reduced computational expense and execution time without loss of accuracy.
- To employ the modified algorithm into real-world problems of industrial engineering (e.g., demand forecasting and trend projection, parameter selection for manufacturing process optimization, etc.)

1.3 Possible Outcome of the Research

This thesis offers a modified methodology for an existing problem, i.e., regression. Mapping the relationship between input variables and the response variable(s) for large datasets using

Gaussian process regression will be memory and time-efficient. Hence, this research will be effective in solving a variety of industrial and manufacturing decision-making problems.

1.4 Outline of the Methodology Used

The methodology used is outlined below:

- i. The training set was segregated equally into two partitions, called the initial training set and the streaming set, respectively. With the help of a data clustering algorithm called CURE, a representative set was obtained from the initial training set and $m \ll n$, where n indicates the instances of the initial training set and the instances of the representative dataset is denoted by m . Then, the kernel matrix for this smaller subset was computed.
- ii. The streaming set was used in such a way that every point is fed sequentially. Using the kernel matrix for the representative set as the base, the kernel matrix was updated as the points were passed on. In the event of the arrival of new data points, the model was not retrained, rather the kernel matrix was updated and the algorithm only fit the updated kernel matrix.
- iii. For making predictions, there needs to be an inversion of the kernel matrix. As a direct inversion of matrices is costly, this method is rarely implied for small datasets, let alone large ones. Rather, another method known as the Cholesky factorization is typically utilized ([Rasmussen & Williams, 2006](#)). In this thesis, Cholesky decomposition was used for the inversion of the covariance matrix. For the streaming input and any new input, an efficient update of the lower Cholesky factor was offered which decreases computation to a great extent.
- iv. During the fitting of the Gaussian process, the negative log marginal likelihood was minimized.
- v. After training was done, the algorithm was tested on the test sets. For evaluating the model performance, two error metrics were deployed: mean absolute error (MAE) and root mean squared error (RMSE). Moreover, the memory and time required for the

execution were calculated. Furthermore, the error scores, memory consumption, and execution time were matched against that of the basic GPR.

1.5 Contribution of the Study

This study proposes an algorithm that:

- Answers to the continuous input problem and variable training size problem
- Requires much less memory for the same computation
- Offers great savings in terms of execution time
- Does not sacrifice accuracy to the least
- Provides even better fitting in terms of training accuracy for training sets and performance (i.e., accuracy, memory requirement, execution time) on test sets

1.6 Limitation of the Study

This research possesses some limitations regarding scope and resources. For the scope limitation, only single-output regression problem was addressed in this thesis where there is multi-output regression problem as well. Additionally, a predetermined set of kernels was used in this research. Also, the performance of the proposed algorithm was matched against that of the basic GPR algorithm only. As for the resource limitation, there was a ceiling on the size of the dataset handled. The current system specification allowed a maximum of somewhat over 8000 training points to be handled. With the proposed algorithm, datasets of greater size than this could have been handled using the current system specification; however, it would not have been possible to compare the performances of these two algorithms.

1.7 Organization of the Thesis

This thesis is organized as follows:

“Chapter1: Introduction” presents the background motivation of the study with defined objectives and also states the possible outcomes. Additionally, the methodology of this research, contribution, and limitation of this study has been addressed in this chapter.

“Chapter2: Background Literature” summarizes the relevant publications regarding Gaussian Process Regression.

“Chapter3: Gaussian Process Regression” states the theory of Gaussian Process Regression and other relevant terminologies.

“Chapter4: Learning Dynamics in Machine Learning” discusses the two prime machine learning strategies, their properties, benefits, and limitations.

“Chapter5: Proposed Sparse Incremental Gaussian Process Regression” talks about the modifications offered in this thesis elaborately.

“Chapter6: Datasets and Experimentation” describes the datasets used in this research, their characteristics, visualization, and the experimentation procedure.

“Chapter7: Result Analysis” presents the results and possible visualizations of the outcomes acquired from the experimentations and provides necessary justifications.

“Chapter8: Conclusions and Future Work” passes a concluding note and directs to the possible future research opportunities.

CHAPTER2

BACKGROUND LITERATURE

Machine learning, a subcategory of artificial intelligence (AI), is a discipline of knowledge where machines are trained to act and make decisions as humans do (Alpaydin, 2004). This learning of machines can be broadly classified among three categories: supervised learning, unsupervised learning, and reinforcement learning (Dunjko, Taylor, & Briegel, 2016). Customarily speaking, for supervised learning, the response is labeled, i.e., it is specified which values of input produces certain output(s), where unsupervised learning has unlabeled output(s) and the usual notion is to infer patterns from the data (Sathya & Abraham, 2013). Reinforcement learning falls somewhere between these two classes, where the output(s) are not needed to be labeled and the process tries to balance between the available knowledge and the unknown area (Kaelbling, Littman, & Moore, 1996). This thesis solely focuses on supervised learning.

The supervised learning area can again be stratified into two sub-domains: regression task and classification task. The main difference between these two sub-classes is that for the regression problem, the output is continuous whereas it is discrete for classification (Rasmussen & Williams, 2006). Gaussian process (GP) is an elegant way to infer function values at unseen points (Bernardo, Berger, Dawid, & Smith, 1998). GPs can be used in the context of both regression and classification tasks. When used for regression, GP is referred to as Gaussian process regression (GPR) and it is called Gaussian process classification (GPC) when employed for classification problems. This thesis deals with Gaussian process associated with regression, e.g., GPR. In section 2.1, a general background on GPR is provided. Section 2.2 provides a summary of sparse methodologies of GPR and section 2.3 presents the synopsis of incremental methodologies.

2.1 General Background of Gaussian Process Regression

Gaussian process regression comes with a rich history. Earlier publications containing the Gaussian process being used for regression date back to the 1940s (Kolmogoroff, 1941; Wiener,

1949). GPR is an ingenious way of fitting functions through observations where it can extract information about some latent functions from the available data. Defined completely by a mean function and a covariance function, this technique is quite effective for smaller datasets (Yuan, Wang, Yu, & Fang, 2008). Additionally, it can quantify the uncertainties of prediction, i.e., it provides error bars on predictions (Cheng & Boots, 2016). GPR has seen applications in a wide selection of problems. For instance, it has been used in multi-variable spectroscopic calibration (Chen, Morris, & Martin, 2007), reliable multi-objective optimization of wire-cut high-speed electrical discharge machining process (Yuan et al., 2008), producing a high-resolution image from a single image with lower resolution (He & Siu, 2011), analyzing the motion trajectory in traffic monitoring (Kim, Lee, & Essa, 2011), predicting the length of the day (Lei, Guo, Cai, Hu, & Zhao, 2015), modeling of sandy soil infiltration (Sihag, Tiwari, & Ranjan, 2017), prediction of tool wear (D. Kong, Chen, & Li, 2018), and numerous such diverse applications.

Although GPR offers excellent performance regarding small datasets, the computational demand is high (Williams & Rasmussen, 1996). The reason for this high expense is due to the inversion of the covariance matrix of the training set which is required for making predictions. It has been noted that for n data points, GPR requires the inversion of an $n \times n$ matrix that takes $O(n^3)$ time, $O(n^2)$ storage for training, and $O(n)$ time for predicting the response on a test point (M. Seeger, Williams, & Lawrence, 2003). As a result, GPR becomes expensive as the data points keep increasing. Due to this poor scalability, GPR is hardly used for big datasets. Another limitation of this model is that it is vulnerable to overfitting (Mohammed & Cawley, 2017), i.e., that the model performs well on the training set, but produces poor results on the test set (Subramanian & Simon, 2013). A host of techniques have been employed by researchers to rectify these limitations. These techniques can be broadly divided into sparse methods and incremental/online/sequential learning methods.

2.2 Sparse Methodologies of Gaussian Process Regression

The earlier modifications of GPR were brought principally by the sparse methods (Lázaro-Gredilla, Quiñonero-Candela, Rasmussen, & Figueiras-Vidal, 2010; Qi, Abdel-Gawad, & Minka, 2010; M. Seeger et al., 2003; Alex J. Smola & Bartlett, 2001; Snelson & Ghahramani, 2006; Tipping, 2001; Titsias, 2009; Tresp, 2000; Walder, Kwang, & Schölkopf, 2008; C. K. I.

Williams & Seeger, 2001; Yoshioka & Ishii, 2001). The sparse methods are the lower-rank approximations of the full-rank Gaussian process that retain the behavior of the original model due to the intractability of GP for larger datasets (McIntire, Ratner, & Ermon, 2016). This is achieved generally through the means of inducing variables or pseudo-inputs, which are a subset of the original training set (Titsias, 2009). The number of inducing variables to work with is dependent on the user; however, the choice is crucial (M. Seeger et al., 2003). Inducing variables can be obtained from the training set by random selection or by optimization, or otherwise (Bauer, van der Wilk, & Rasmussen, 2016). It has been reported that a random selection of the inducing variables might lead to poor model performance (Lawrence, Seeger, & Herbrich, 2002). Anyhow, various sparse schemes have emerged in the literature. The underlying similarity of these methods is that every sparse algorithm used some form of approximations to obtain the inducing variables set/active set/pseudo-inputs. And the difference among these approaches lies in the form of approximation used. For instance, (Tipping, 2001) used a relevance vector machine (RVM) approximation, which is a finite linear model with Gaussian priors used on the weights. The significance of this model is that it can find sparse solutions like support vector machine while providing predictions as Bayesian kernel machines using a Gaussian process prior (Quiñonero-Candela & Winther, 2003). Then, (Williams & Seeger, 2001) proposed a Nyström approximation for faster performance of GPR. Nyström method is a numerical method (Baker, 1977) that is used to approximate the covariance matrix. However, this approach does not conform to a probabilistic model, i.e., the prior covariance between the latent function values and the test function values was taken to be exact, which was inconsistent with the prior covariance of the latent function values (Williams, Rasmussen, Scwaighofer, & Tresp, 2002). Another significant pitfall was that the covariance matrix did not guarantee the positive definiteness, i.e., it produced negative covariance values (Quiñonero-Candela & Rasmussen, 2005). After that, (Smola & Bartlett, 2001) used a subset of regressors (SoR) approximation for their sparse greedy approach. SoR models are finite with linear parameters that take on specific prior weights. The approach is greedy in the sense that this algorithm tries to optimize the maximum a posteriori (MAP) approximation of the response variable(s) at each step. However, this model suffers from a serious limitation. While predicting function values in a region far from the active set, the model provides unreasonably smaller predictive variances which is not explained by the model (Quiñonero-Candela & Rasmussen, 2005), known as the

predictive variance problem. Later, (Yoshioka & Ishii, 2001) employed a dataset reduction scheme such that the derived dataset would depict the original dataset's characteristics. They were able to demonstrate that with a very small number of representative points, their model could match the performance of the basic GPR; however, the prediction accuracy is contingent upon the allocation of the representative points. Afterwards, (M. Seeger et al., 2003) presented another sparse greedy approach where the deterministic training conditional (DTC) approximation or the projected process approximation (PPA) (Rasmussen & Williams, 2006) was utilized. DTC is based on likelihood approximation where the original likelihood of the full model is replaced by an approximation. This method assumes a deterministic relation between the latent function values and the inducing variables. The significance of this model is that it solves the limitation of the SoR model (Quiñonero-Candela & Rasmussen, 2005) and leads to the same results as (Smola & Bartlett, 2001) even though an approximated likelihood was used. However, (Quiñonero-Candela & Rasmussen, 2005) remarked that the DTC approximation does not conform to a Gaussian process exactly. Subsequently, two more sparse methodologies were offered by (Schwaighofer & Tresp, 2003) called fully independent conditional (FIC) approximation and partially independent training conditional (PITC) approximation. FIC is also a likelihood approximation as DTC, only FIC does not put a deterministic relation on the function values and the pseudo-inputs, rather it considers the conditional distribution as an additional independence assumption. For FIC, this independence assumption is valid for both the training set and the test set. PITC comes from another perspective and it differs from FIC in the sense that the conditional independence is now valid for a group of points in the training set. Interestingly, PITC improves upon the DTC approximation (Quiñonero-Candela & Rasmussen, 2005). Fully independent training conditional (FITC) approximation was exercised by (Snelson & Ghahramani, 2006). FITC is different from FIC as the conditional independence exists over the training set only, not the test set. This is also a likelihood-based approximation procedure; however, it results in a richer covariance (Quiñonero-Candela & Rasmussen, 2005). Thereafter, (Walder et al., 2008) proposed a multiscale version of sparse GP, known as sparse multiscale Gaussian process (SMGP), which was a modification of FITC approximation. This method offered performance improvement over FITC; however, it required learning twice as many parameters (Lázaro-Gredilla et al., 2010). Later, (Titsias, 2009) proposed a variational approach to sparse GP, where the full GP was approximated using a large number of basis

functions. Basis function is a way of generalizing the non-linear relationship between the input and the output (Hensman & Lawrence, 2014). Although the proposed method by Titsias proved to be effective for the overfitting problem, the performance of this method was inferior to FITC or SMGP. After that, (Lázaro-Gredilla et al., 2010) presented a sparse spectrum Gaussian process (SSGP) that used a stationary trigonometric Bayesian model which was computationally economical and provided good performance.

The implementation of sparse methods reduced the GPR complexity to some degree, many of whom reached the same conclusion. For example, the computational requirement in terms of training time was brought down to $O(m^2n)$ from $O(n^3)$, where, m is the size of the subset (inducing set) of the original dataset ($m \ll n$) (Lawrence et al., 2002; Quiñonero-Candela & Rasmussen, 2005; M. Seeger et al., 2003; A. J. Smola & Bartlett, 2001; Snelson & Ghahramani, 2006; Walder et al., 2008; C. K. I. Williams & Seeger, 2001; Yoshioka & Ishii, 2001). Memory needed for training was reduced to $O(nm)$ from $O(n^2)$ (M. Seeger et al., 2003; A. J. Smola & Bartlett, 2001), prediction time decreased from $O(n)$ to $O(m)$ (M. Seeger et al., 2003; A. J. Smola & Bartlett, 2001), or $O(m^2)$ (M. Seeger et al., 2003; Snelson & Ghahramani, 2006), and computations for error bound was $O(nm)$ rather than $O(n^2)$ (Smola & Bartlett, 2001).

The methods described so far work in a batch setting. Batch learning refers to model learning where a bunch of data points is used simultaneously to gain information about the model behavior. In this case, the model does not retain the training knowledge, i.e., if a new data point arrives, the model will have to retrain itself (Carbonara & Borrowman, 1998). Batch learning assumes that the training dataset is readily available before modeling (Gepperth & Hammer, 2016). However, this may not always happen. For example, if a particular application has a variable training set or incoming data points over a defined period (for example feedbacks, time-dependent user-inputs), the model would have to train itself over and over for every new training point. To make matters worse, if the dataset is large, the batch mode will not be economical at all (Carbonara & Borrowman, 1998). In fact, limited processing power and storage capabilities of the system might render batch learning prohibitive altogether (Ade & Deshmukh, 2013). As a measure to treat the scalability problem, a certain learning mode is necessitated that does not assume the availability of training points beforehand and can adapt to events such as the arrival

of new data points or variable training set size. This mode is called incremental learning mode which is described next.

2.3 Incremental Methodologies of Gaussian Process Regression

Incremental learning is a learning strategy that allows the model to expand its knowledge when new information of the system becomes available (Geng & Smith-Miles, 2009). It should be noted that incremental learning, also known as sequential learning, is sometimes used interchangeably with online learning, however, there is a subtle difference. In the case of online learning, the knowledge of training is not always retained depending on the modeler, whereas, incremental learning typically maintains this knowledge and updates the model according to the new data points without retraining the model from scratch (Saffari, Leistner, Santner, Godec, & Bischof, 2009). Many publications in the GP literature have treated incremental learning and online learning in the same sense; however, this thesis is using the term “incremental learning” to avoid any confusion.

The incremental learning techniques regarding the Gaussian process try to update the posterior of the functions incrementally when the dataset is large. An interesting find from the literature survey is that all incremental algorithms have utilized sparsification to some extent as well. In almost all of the cases, the posterior probability has been approximated using some approximation routine. Among the pioneers of incremental learning based GPR algorithms, (Tresp, 2000) proposed a sparse approximate of GPR capable of online processing of data. In this method, models were trained on smaller datasets and the predictions were combined afterward. This approach further required an additional query of the data points. To circumvent this additional query, (Csató & Opper, 2002) presented an online version of sparse GPR, which greedily computes the approximate posterior by going through the dataset only once. The advantage of this model is that it can handle non-continuous likelihoods that might cause problems when Kalman filter or variational Gaussian approximations are used. Despite the advances, instances have been reported in this paper where this algorithm performs poorly compared to batch algorithms. Next, (Quiñonero-Candela & Winther, 2003) made use of relevance vector machines (Tipping, 2001) and presented an incremental learning routine as a variant of the expectation-maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977),

known as “Subspace EM”. The limitation of (Tipping, 2001) was that having as many basis functions as possible in the beginning leads to the same complexity as full GP. Therefore, it was suggested by Tipping to start with a single basis function and add basis functions as necessary. To keep computations minimum, basis functions were also to be removed as per demand. Later, (Quiñonero-Candela & Winther, 2003) used the EM algorithm to strategize this method formally and showed that it guaranteed convergence at the local maximum of model parameters. Subsequently, (Nguyen-Tuong, Peters, & Seeger, 2009) presented their approach to fastening the standard GPR using local GP models which were used to approximate the inverse dynamics for model-based robot control. By using distance-based measures to divide their data into regions and train individual GP models for each region, they were able to learn the parameters online and provide real-time predictions. Afterward, (Ranganathan, Yang, & Ho, 2010) proposed an online sparse matrix Gaussian processes (OSMGP). Their approach combines exact inference and online updates of the Cholesky factor of the kernel matrix. A limitation of this approach is that this method works under the presumption that the kernel matrix is inherently sparse. Following this, (De La Cruz, Owen, & Kulic, 2012) presented a refined version of sparse online Gaussian processes originally provided by (Csató & Opper, 2002) which allowed incremental updates on both the training set and the model hyperparameters. The significance of this approach was that it performed well even without initial training data for the learning of inverse dynamics in model-based robot controls. Thereafter, (Cheng & Boots, 2016) presented an incremental variational sparse Gaussian process regression (*i*VSGPR) that combined sparsification with incremental model hyperparameter updates. The stochastic mirror ascent algorithm was used to update the approximate function posterior and the stochastic gradient ascent method was used for hyperparameter updates. This method solved a mini-GP in each iteration, thereby keeping computations tractable.

Other than these approaches, models have been presented to provide the online version of variational batch algorithms (Hensman, Fusi, & Lawrence, 2013; Hoang, Hoang, & Low, 2015; Qi et al., 2010). For instance, (Qi et al., 2010) updated the FITC approximation by using the expectation propagation (EP) algorithm, which allows processing data online. After that, (Hensman et al., 2013) proposed a stochastic approximation of variational sparse GP originally offered by (Titsias, 2009) based on stochastic natural gradient ascent (Hoffman, Blei, Wang, & Paisley, 2013). Later, (Hoang et al., 2015) generalized all the sparse representations of

GP, i.e., PIC, FTC, DTC, etc. (commonly called SGPRs) and by applying the reverse variational method (as opposed to the variational method by (Titsias, 2009)), this algorithm can assure convergence of the predictive distribution of any of the SGPR models.

In light of these preceding research, this thesis delves into Gaussian process regression with a sparse-incremental mindset. Although there have been a handful of incremental learning methods dealing with variable training set size, this thesis attempts to use a simpler and flexible approximation that provides decent results while maintaining accuracy at an acceptable level and keeping computations as low as possible. In the following chapter, the theoretical background on the Gaussian process regression and associated terminologies are presented and a discussion on the major learning strategies for a machine learning model can be found in the next-to-next chapter.

CHAPTER3

GAUSSIAN PROCESS REGRESSION

Gaussian process (GP) is a Bayesian kernel machine following a multivariate Gaussian distribution (Ranganathan et al., 2010). This method is Bayesian in the sense that it is based on Bayesian philosophy, where the model is provided a prior without seeing any data, and updates itself upon given some observations. The Gaussian process is kernelized because it uses kernel functions as a measure of covariance between any two random variables. In this chapter, the theoretical background of Gaussian processes is provided. At first, multivariate Gaussian distribution and two of its important properties called marginalization and conditioning are discussed in section 3.1 along with Bayesian philosophy in section 3.2. Section 3.3 talks about the Gaussian process and its two parameters: the mean function and the covariance function. Finally, these ideas have been brought together to relate to Gaussian process regression (GPR) in section 3.4.

3.1 Multivariate Gaussian Distribution

It is assumable that the founding brick of the Gaussian process is the Gaussian or Laplace-Gauss or normal distribution. Gaussian distribution is a continuous probability distribution (Walpole, R. E., & Myers, 2012). Infamously known for the “bell curve”, this distribution is widely used to represent random variables with unknown distributions (Casella & Berger, 2002). There are two parameters for Gaussian distribution. For a univariate Gaussian, these parameters are the mean and the variance of the random variable, and for a multivariate Gaussian, parameters include mean and the covariance between random variables (Tong, 2012).

The probability density function (PDF) of a univariate Gaussian is given by:

$$f(X; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right) \quad (3.1)$$

where μ is the mean and σ^2 is the variance of the random variable X .

For the multivariate case, the distribution is defined by a mean vector, $\boldsymbol{\mu}$, and a covariance matrix, $\boldsymbol{\Sigma}$. In a multivariate Gaussian setting, each of the random variables is normally distributed and their joint distribution is also normal. The probability density function (PDF) of a multivariate Gaussian is given by:

$$f(\mathbf{X}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = |\mathbf{2}\pi\boldsymbol{\Sigma}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{X}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{X}-\boldsymbol{\mu})\right) \quad (3.2)$$

where \mathbf{X} is a vector of random variables distributed according to a multi-variable Gaussian distribution. The following expression can be written from equation (3.2):

$$\mathbf{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad (3.3)$$

where,

$$\boldsymbol{\mu} = E(\mathbf{X}) \quad (3.4)$$

$$\boldsymbol{\Sigma} = Cov(\mathbf{X}_i, \mathbf{X}_j) \quad (3.5)$$

Two of the properties of multivariate Gaussian distribution are of importance regarding the Gaussian process. These are called marginalization and conditioning.

3.1.1 Marginalization

Marginalization allows extracting information about a random variable given a joint probability distribution of that variable with another random variable.

For instance, if A and B are two random variables having a joint distribution of $P(A, B)$, then information about either A or B can be found out by:

$$P_A(a) = \int_b P_{A,B}(a, b) db = \int_b P_{A|B}(a|b) P_B(b) db \quad (3.6)$$

$$P_B(b) = \int_a P_{A,B}(a, b) da = \int_a P_{A|B}(b|a) P_A(a) da \quad (3.7)$$

The marginal distributions of the jointly Gaussian random variables are also Gaussian (Tong, 2012).

3.1.2 Conditioning

This is another useful property of the multivariate normal distribution. Conditioning simply means finding the probability distribution of a random variable given another.

For example, just as in the abovementioned case, the conditional distribution of A given B or the conditional distribution of B given A can be found by:

$$P(A|B) = \frac{P_{A,B}(a,b)}{\int_B P_{A,B}(a,b)db} \quad (3.8)$$

$$P(B|A) = \frac{P_{A,B}(a,b)}{\int_A P_{A,B}(a,b)da} \quad (3.9)$$

Similar to marginalization, conditioning on a Gaussian distribution yields another Gaussian distribution (Tong, 2012).

3.2 Bayesian Philosophy

Bayesian thinking is quite popular in probability and statistics. Bayes theorem provides the probability of an event given some apriori knowledge about that event and some other conditions upon which the outcome of the event may depend (Stone, 2013). Generally, the rule can be expressed as follows:

$$\text{Posterior Probability} = \frac{\text{Prior Probability} \times \text{Likelihood}}{\text{Marginal Likelihood}} \quad (3.10)$$

In a Bayesian environment, an assumption about the probability of the concerned event is made, known as the prior probability or “belief”, because this is specified at the beginning without any knowledge of the system. The likelihood is a measure of fitness for function parameters used for models of unknown parameters, i.e., this is the probability of function values given some parameters. And, marginal likelihood is a likelihood function for when the parameter values have

been marginalized, i.e., this is the probability of function values given the input when the parameters have constant values. In a Bayesian system, marginal likelihood is also called “evidence”.

For example, let us assume that there are some data points, $\mathbf{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^n$. For a simple one-dimensional linear noisy regression problem, the relation between input vector \mathbf{x} and response vector \mathbf{y} can be expressed as:

$$\mathbf{y} = f(\mathbf{x}) + \varepsilon \quad (3.11)$$

$$f(x) = \mathbf{x}^T \mathbf{w} \quad (3.12)$$

where \mathbf{w} is the weight or parameter vector and ε is the Gaussian noise which is independently and identically distributed with zero-mean and a variance of σ_n^2 .

$$\varepsilon \sim N(0, \sigma_n^2) \quad (3.13)$$

The prior probability is assigned to the weights. A zero-mean Gaussian distribution with covariance matrix Σ_w is specified as the prior distribution for weights:

$$w \sim N(0, \Sigma_w) \quad (3.14)$$

The likelihood or the probability of observations given parameters can be expressed as $P(\mathbf{y} | \mathbf{x}, \mathbf{w})$, that can be calculated easily if the independence assumption is considered:

$$P(\mathbf{y} | \mathbf{x}, \mathbf{w}) = \prod_{i=1}^n P(y_i | x_i, w_i) \quad (3.15)$$

Equation (3.10) can be re-written as:

$$P(\mathbf{w} | \mathbf{y}, \mathbf{x}) = \frac{P(\mathbf{w}) \times P(\mathbf{y} | \mathbf{x}, \mathbf{w})}{P(\mathbf{y} | \mathbf{x})} \quad (3.16)$$

where, $P(\mathbf{w} | \mathbf{y}, \mathbf{x})$ is the posterior probability that is calculated with the help of the prior probability, $P(\mathbf{w})$, the likelihood $P(\mathbf{y} | \mathbf{x}, \mathbf{w})$, and the marginal probability, $P(\mathbf{y} | \mathbf{x})$.

3.3 Gaussian Process

The main task of machine learning is to deduce some idea about the function that fits through some observations that are known beforehand. Now, given these data points, there could be an infinite number of functions that fit through them. In the case of parametric methods, the parameters of the function are defined first and then the data points are observed later. Therefore, these methods try to fit the data through a predefined function. The limitation of this approach is that the data points may not conform to the function characteristics as it was specified arbitrarily (Fattahi, 2011). Therefore, it is not wise to specify the parameter prior to seeing some observations. This is one of the reasons for the poor performance of parametric methods (James, Witten, Hastie, & Tibshirani, 2013).

As opposed to this, the Gaussian process is a non-parametric method that rather than specifying a function, specifies a distribution of functions from a prior belief. Then, upon having some observations, it updates the knowledge of the system. It is to be noted here that the term ‘non-parametric’ does not refer to having any parameter, rather having an infinite number of parameters (Hall, 1989).

Formally, Gaussian processes can be defined as a distribution of functions in an infinite domain, any finite linear combination of which produces a multivariate Gaussian distribution (Rasmussen & Williams, 2006).

A Gaussian process is comprehensively defined using a mean function and a covariance function:

$$f \sim GP(m, k) \tag{3.17}$$

where m is the mean function and k is the covariance function. They are defined as follows:

$$m(\mathbf{x}) = E[f(\mathbf{x})] \tag{3.18}$$

$$k(\mathbf{x}, \mathbf{x}') = E[(f(\mathbf{x}) - m(\mathbf{x})) (f(\mathbf{x}') - m(\mathbf{x}'))] \tag{3.19}$$

The expression in equation (3.17) can be re-written as:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \quad (3.20)$$

For any set of input points $\{\mathbf{x}_i\}_{i=1}^n$, the function vector of these points $\mathbf{f} = [f(\mathbf{x}_1) \dots f(\mathbf{x}_n)]^T$ has a joint Gaussian distribution (Lázaro-Gredilla et al., 2010).

$$P(f | \{\mathbf{x}_i\}_{i=1}^n) = P(f | \boldsymbol{\mu}, \Sigma) \quad (3.21)$$

where $\boldsymbol{\mu}$ is the mean vector and Σ is the covariance matrix.

In the following two subsections, the two parameters of the Gaussian process, i.e., the mean function and the covariance function are discussed.

3.3.1 Mean Function

The mean function is the first parameter of the Gaussian process. This parameter represents the average value of functions. In truth, the mean function is not much interesting, as the mean of the prior distribution does not affect the predictive mean. In fact, it has been shown that for a non-zero mean function, the outcome is the same (Rasmussen & Williams, 2006). There is a convention of taking the mean of the prior distribution to be zero. Following the convention, this thesis considers a zero-mean function for the Gaussian prior.

3.3.2 Covariance Function

The covariance function is the heart and soul of the Gaussian process. GP does not seek to achieve the true form of the output function. Rather, it depends on the nearness of the data points to propose a guess about the corresponding function values. Intuitively speaking, if two data points are close, it is justified to assume that the function values relating to those points should also be close. Based on this intuition, the Gaussian process finds the closeness of function values through another function known as the covariance function or kernel function.

Back to the first assumption of this model, as the prior mean is taken to be zero, it is the covariance matrix that now defines the characteristics of the prior distribution(Quiñonero-

Candela & Rasmussen, 2005). The interesting point regarding this assumption is, as the prior becomes Gaussian and the likelihood is generally presumed to be Gaussian, the posterior distribution is also Gaussian. This simplification leads to an excellent advantage discussed in section 3.4.

This covariance function is used to generate the covariance matrix, which is symmetric in nature. For an $n \times d$ input matrix, where d is the dimension of the input space, the covariance matrix has a size of $n \times n$. The general term for the covariance functions between two points is kernel functions. Figure 3.1 presents a general classification of covariance functions (Rasmussen & Williams, 2006).

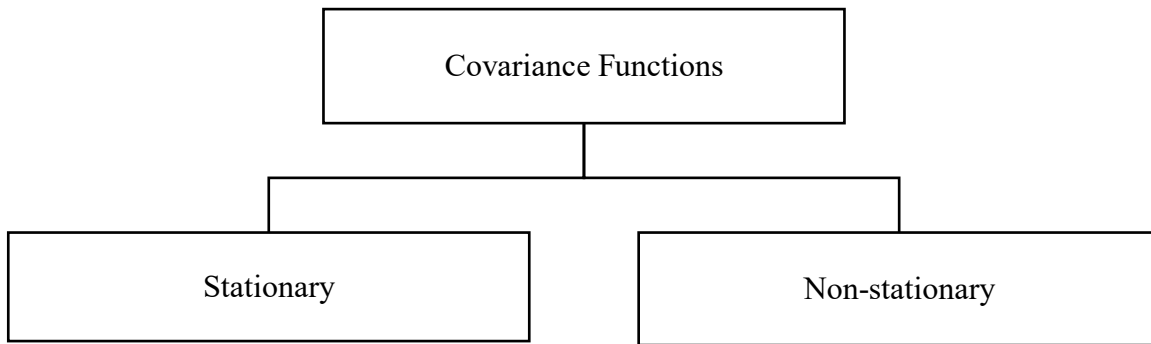


Figure 3.1: Classification of covariance functions

3.3.2.1 Stationary Covariance Function

For two points x_1 and x_2 , a covariance function, k is called stationary if k is a function of $(x_1 - x_2)$. This kind of covariance is unvarying to translation, i.e., the covariance function value depends only on the difference vector, not the original points x_1 and x_2 (Genton, 2001). Popular stationary covariance functions include exponentiated quadratic function, rational quadratic function, etc. It can be noted here that the covariance function, k can also be a function of $\|x_1 - x_2\|$, in which case it will be called an isotropic kernel, where, $\|x_1 - x_2\|$ is the Euclidean distance or L_2 norm (Horn & Johnson, 2012). For any vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, the L_2 norm is:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} \quad (3.22)$$

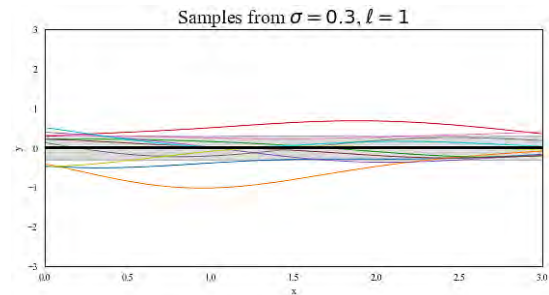
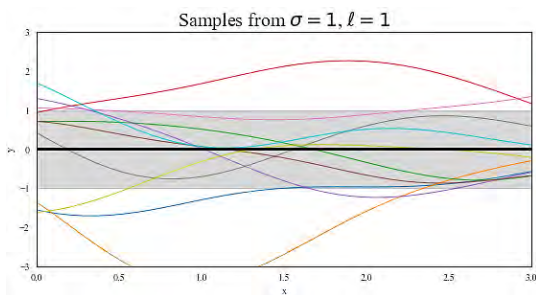
Stationary kernels are known to be used in both forms. Isotropic functions are also known as radial basis functions (RBF).

3.3.2.1.1 Exponentiated Quadratic Kernel

The exponentiated quadratic (EQ) kernel is also known as the squared exponential (SE) kernel. The kernel has the following form:

$$k_{ES}(x_1, x_2) = \sigma^2 \exp\left(-\frac{(x_1 - x_2)^2}{2l^2}\right) \quad (3.23)$$

Here, σ^2 is the variance, and l is known as the length-scale. The variance is a scaling factor whose larger value means the points are far from the mean and a lower value means that the points are closer to the mean. The length-scale determines the smoothness of the function. The lower this value is, the "wigglier" the function is, i.e., the function values can change rapidly, whereas a larger length-scale provides a smoother function. Figure 3.2 shows the variation of output due to different variance and length-scale. It is clear from equation (3.23) that the more any two points x_1 and x_2 are close to each other, the less is the difference $(x_1 - x_2)$, resulting in a higher covariance, and vice-versa. This is how this kernel function provides a sense of similarity or closeness.



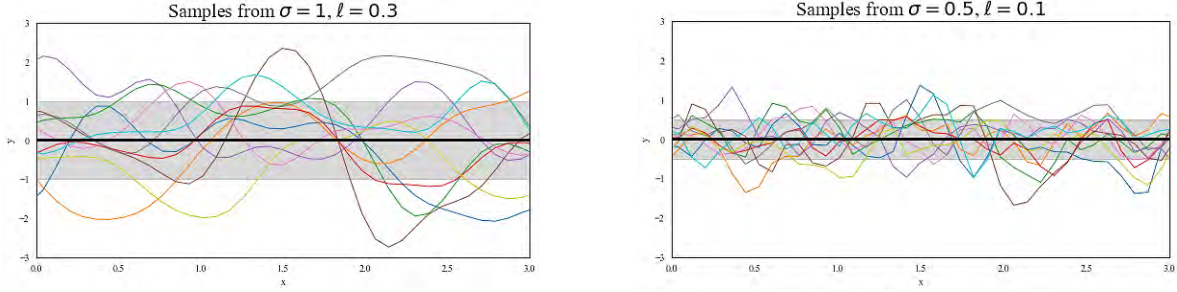


Figure 3.2: Variations of the exponentiated quadratic kernel

3.3.2.1.2 Rational Quadratic Kernel

Rational quadratic (RQ) kernel can be expressed as:

$$k_{\text{RQ}}(x_1, x_2) = \sigma^2 \left(1 + \frac{(x_1 - x_2)^2}{2\alpha l^2} \right)^{-\alpha} \quad (3.24)$$

This kernel has 3 parameters: signal variance, σ^2 , length-scale, l , and scale-mix, α ($\alpha > 0$). The RQ kernel is analogous to having an infinite sum of exponentiated quadratic kernels with different length-scales and α works as a weighting factor. As $\alpha \rightarrow \infty$, the rational quadratic kernel becomes the exponentiated quadratic kernel. Figure 3.3 shows the variations in the RQ kernel for varying l and α for a constant noise variance, $\sigma=1$.

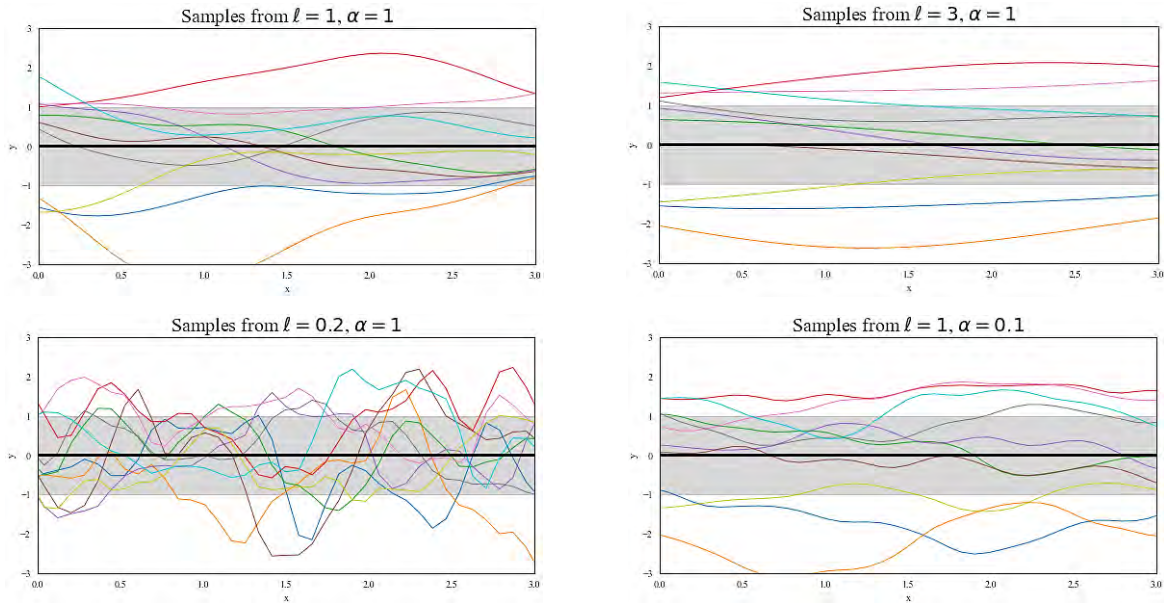


Figure 3.3: Variations of the rational quadratic kernel ($\sigma=1$)

3.3.2.1.3 Matérn Kernel

The Matérn kernel has the following expression:

$$k_\nu(x_1, x_2) = \sigma^2 \frac{2^{1-\nu}}{\Gamma_\nu} \left(\sqrt{2\nu} \frac{\|x_1 - x_2\|}{l} \right)^\nu K_\nu \left(\sqrt{2\nu} \frac{\|x_1 - x_2\|}{l} \right) \quad (3.25)$$

The parameters for the Matérn kernel are σ^2, ν , and l . Here, K_ν is a Bessel function (Abramowitz & Stegun, 1965), ν is the function order, σ^2 is the overall variance, and length-scale, l is similar to the exponentiated quadratic kernel. Figure 3.4 shows variations of this kernel for various l and ν while variance was kept constant at $\sigma=1$.

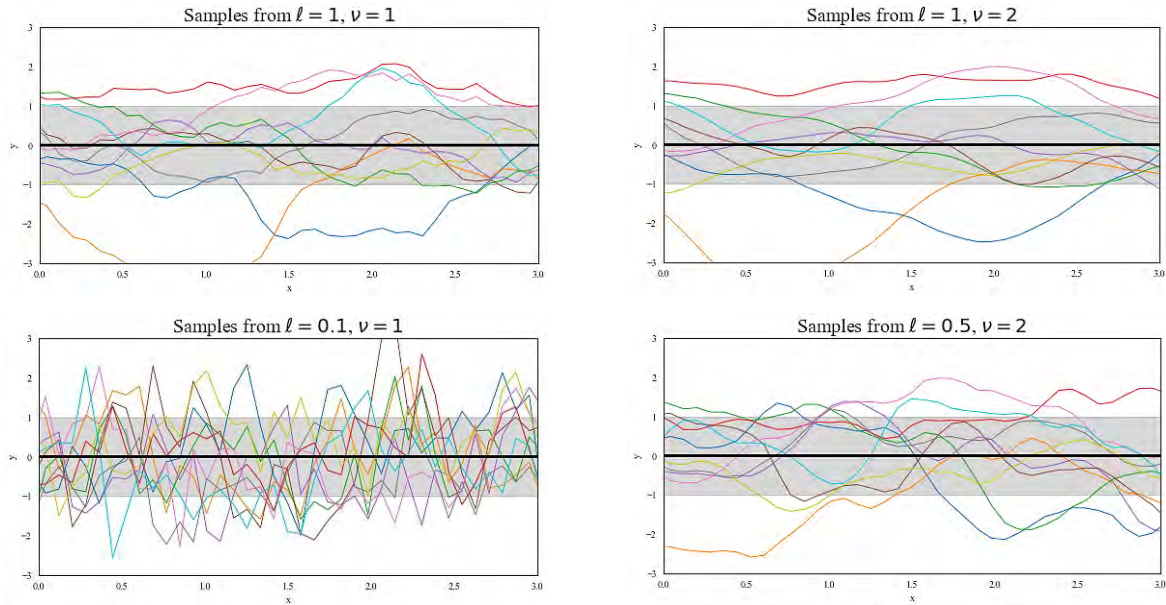


Figure 3.4: Variations of the Matérn kernel ($\sigma=1$)

3.3.2.2 Non-stationary Covariance Function

A covariance function k of two points x_1 and x_2 is called non-stationary if k is not a function of $(x_1 - x_2)$. The functional relationship can vary in this case. Popular non-stationary covariance functions include dot product kernel, periodic kernel, polynomial kernel, etc. (Rasmussen & Williams, 2006).

3.3.2.2.1 Dot Product Kernel

The dot product kernel's expression is:

$$k_{\text{dot}}(x_1, x_2) = \sigma^2 + x_1 \cdot x_2 \quad (3.26)$$

where the k is related to the points x_1 and x_2 through $(x_1 \cdot x_2)$. The only parameter is σ^2 , which is the overall variance. The dot product kernel is not much suited for the regression problem but is highly used in high-dimensional classification task (Schölkopf & Smola, 2002). Figure 3.5 shows the variations of dot product kernel for different values of the signal variance.

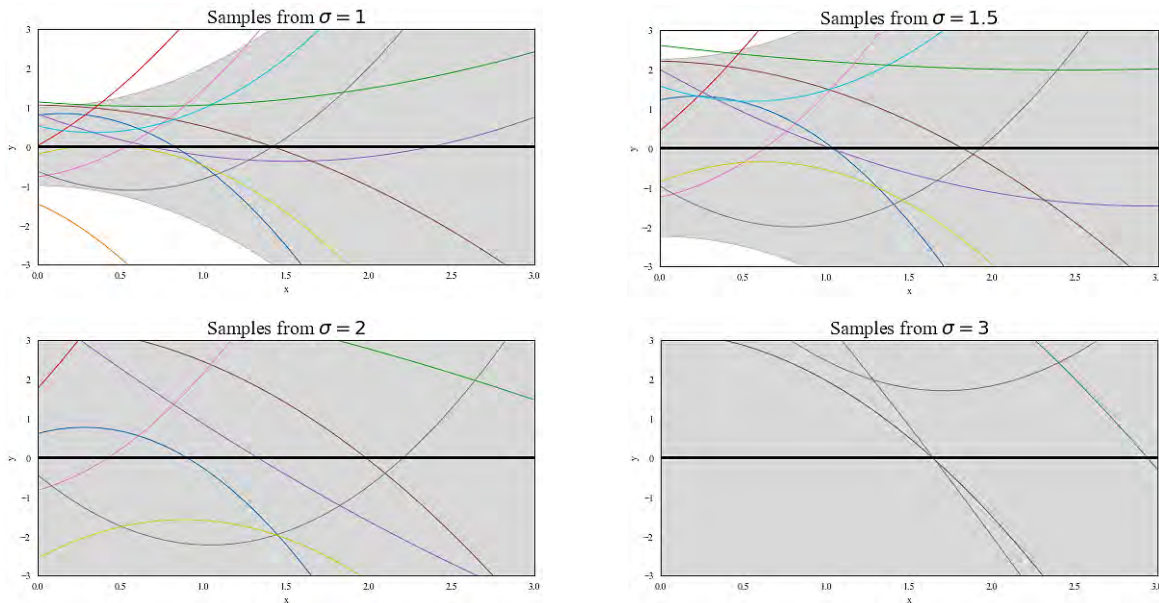


Figure 3.5: Variations of the dot product kernel

3.3.2.2.2 Periodic Kernel

The periodic kernel is a non-stationary kernel that is related to the inputs through a periodic function, specifically a sine function. This kernel was proposed by (MacKay, 1998). It is sometimes known as the exponential sine squared kernel and has the form of:

$$k_{\text{PER}}(x_1, x_2) = \sigma^2 \exp\left(-\frac{2 \sin^2(\pi |x_1 - x_2|/p)}{l^2}\right) \quad (3.27)$$

This kernel allows the modeling of periodic functions. The term p is the period that gives the distance between the repetitive function values and the length-scale, l has the same properties as in the EQ kernel. The variations of this kernel can be seen in Figure 3.6.

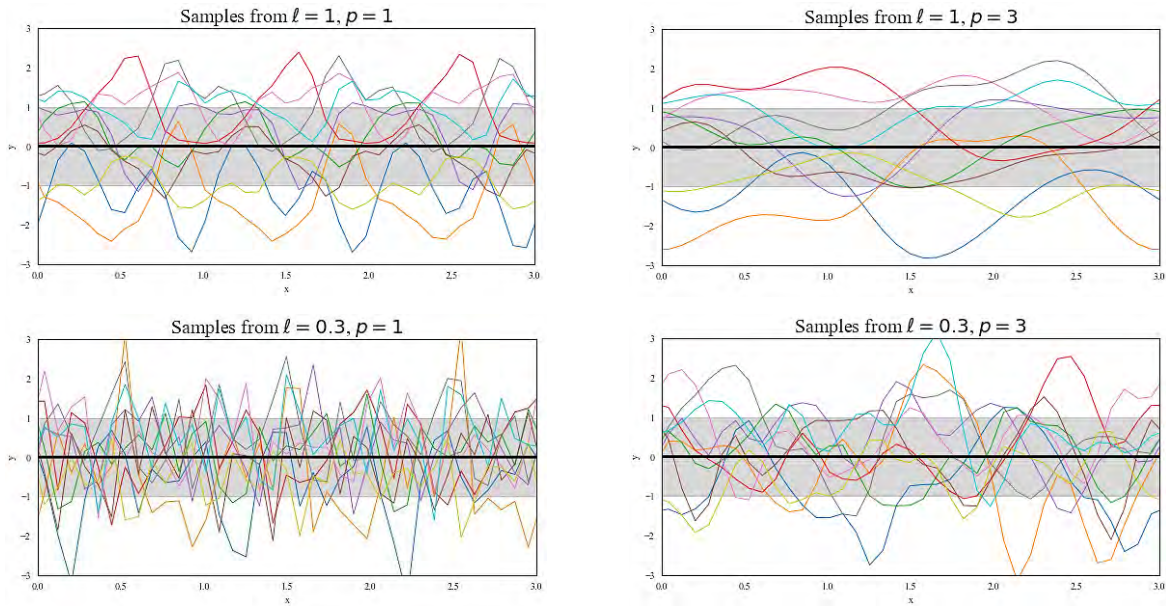


Figure 3.6: Variations of the periodic kernel

In Table 3.1, a summary of the most popular kernels has been presented in terms of their expression, parameters, and their class.

Table 3.1: Popular kernel functions

Name of the Kernel	Expression	Parameters	Class
Exponentiated Quadratic	Equation (3.23)	σ^2, l	Stationary
Rational Quadratic	Equation (3.24)	σ^2, l, α	Stationary
Matérn	Equation (3.25)	σ^2, l, ν	Stationary
Dot Product	Equation (3.26)	σ^2	Non-stationary
Periodic	Equation (3.27)	σ^2, l, p	Non-stationary

There are some other popular kernels such as the constant kernel, white-noise kernel, Gaussian kernel, sigmoid kernel, etc. More about kernels can be found in (Duvenaud, 2014). It is to be noted that two or more kernels can be combined to make a hybrid kernel. The hybrid kernel might be appropriate in situations where the single kernel might not perform well.

3.3.2.3 Kernel Hyperparameters

One important feature of the kernel matrix is the hyperparameters. Generally, model parameters are values that are learned by the algorithm from training (Yuan et al., 2008). For instance, in the case of linear regression, the intercept and slope are the model parameters that are calculated based on some training datapoints. The algorithm does not start with some predefined value for parameters. However, model hyperparameters are values that are primarily defined by the practitioner. Hyperparameters are often used to learn the model parameters (Probst, Bischl, & Boulesteix, 2018). In the context of the Gaussian process, the parameters of the kernel functions are called the kernel hyperparameters because their values are set beforehand.

3.3.2.4 Validity of Kernels

Although there is a wide variety of kernels that may be used as the prior distribution, there is a certain restriction on the kernels to be a valid kernel. The condition is that the kernel needs to be a positive-definite (PD) or positive-semidefinite (PSD) in nature (Jylänki, Vanhatalo, & Vehtari, 2011). This is known as the Mercer theorem (Fuchs & Rogosinski, 1942). A real-valued $n \times n$ matrix, \mathbf{k} is called a positive-definite if satisfies $\mathbf{v}^T \mathbf{k} \mathbf{v} > 0$, for all vectors $\mathbf{v} \in \mathbb{R}^n$, and positive semi-definite if $\mathbf{v}^T \mathbf{k} \mathbf{v} \geq 0$ (Rasmussen & Williams, 2006). A symmetric matrix is positive definite if all the eigenvalues are positive and positive semi-definite if all the eigenvalues are non-negative.

3.3.2.5 Significance of Kernels

As stated earlier, the covariance function is one of the parameters of the Gaussian process. In typical machine learning practice, it is assumed that some form of feature function is available upon which the task (classification, regression, etc.) is performed. However, it is not always

possible to specify this feature function explicitly. For example, text-documents, atomic structures, and similar cases prohibit the use of an explicit feature function. For cases such as these, the logical way to perform the machine learning task is through “similarity functions” (Scholkopf & Smola, 2018). The key idea is to learn the similarity without the knowledge of the feature space. It is relatable that two close points will produce outputs that themselves are close. This is where the kernel functions come into aid. Kernel functions measure the similarity of function values corresponding to given data points. In the GP paradigm, covariance function or kernel function is what determines this closeness or similarity. And given the general notion of taking a zero-mean for the prior distribution, it is the kernel function that specifies the prior distribution entirely (Csató & Opper, 2002). Therefore, the covariance function or the kernel function is what the Gaussian process is built upon.

3.4 Gaussian Process Regression (GPR)

Regression is a machine learning routine to map the functional relation between the input variable(s) or regressor(s) and the response variable(s) (Draper & Smith, 1998). Regression can be broadly classified into two classes based on the characteristics of the relationship relating the input to the output: linear regression (LR) and nonlinear regression (NLR). For linear regression, the input is linked linearly with the response(s) and for nonlinear regression, the relation takes the form of some nonlinear function. Gaussian process regression is specifically suited for nonlinear regression tasks (Snelson & Ghahramani, 2006).

GPR for two cases of nonlinear regression is discussed below: noiseless nonlinear regression and noisy nonlinear regression. For avoiding confusion, the response of a noiseless and noisy NLR is denoted by f and y , respectively.

A typical nonlinear regression (NLR) problem is defined by:

$$\mathbf{f} = \sum_{i=1}^n w_i \phi_i(x_i) = \mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{w} \quad \text{[noiseless NLR]} \quad (3.28)$$

$$\mathbf{y} \sim f(\mathbf{x}) + \boldsymbol{\varepsilon} \quad \text{[noisy NLR]} \quad (3.29)$$

$$\boldsymbol{\varepsilon} \sim N(0, \sigma_n^2) \quad (3.30)$$

Two variants of NLR are considered here. Equation (3.28) represents the noiseless non-linear regression, and equation (3.29) describes a noisy non-linear regression problem, where ε is the Gaussian noise. Here, $\Phi(\mathbf{x}_i) = [\varphi(\mathbf{x}_1), \dots, \varphi(\mathbf{x}_n)]$ is the basis function vector with basis functions represented by $\varphi(x_i)$. The basis function is a way to map the relationship between the input and the response (Hensman & Lawrence, 2014). For linear regression, the basis function would be a linear function of \mathbf{x} and for nonlinear regression, the basis function would be a nonlinear function of \mathbf{x} . For example, the linear relationship can be of $\Phi(\mathbf{x}_i) = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$, where $\Phi(\mathbf{x}_i)$ is linear in terms of \mathbf{x} . In contrast, for non-linear regression, the relationship starts with \mathbf{x} having an exponent of 2 or higher. Also, from equation (3.28), $\mathbf{w} = [w_1, w_2, \dots, w_n]^T$ represents the weight vector.

Based on the Bayesian philosophy, the first task is to define the prior. After that, the posterior is calculated with the help of the evidence.

3.4.1 Defining the Prior

In this regression problem, let the prior on the weight vector be defined by a Gaussian process:

$$\mathbf{w} \sim N(\mathbf{0}, \Sigma_w) \quad (3.31)$$

for the function $f(\mathbf{x}) = \varphi(\mathbf{x})^T \mathbf{w}$, the mean and the covariance is defined by:

$$E[f(\mathbf{x})] = \varphi(\mathbf{x})^T E(\mathbf{w}) = 0 \quad (3.32)$$

$$E[f(\mathbf{x})f(\tilde{\mathbf{x}})] = \varphi(\mathbf{x})^T E(\mathbf{w}\mathbf{w}^T)\varphi(\tilde{\mathbf{x}}) = \varphi(\mathbf{x})^T \Sigma_w \varphi(\tilde{\mathbf{x}}) \quad (3.33)$$

where, $\tilde{\mathbf{x}}$ is another vector and the covariance of the weight vector, \mathbf{w} is given by:

$$\Sigma_w = E(\mathbf{w}\mathbf{w}^T) = K(\mathbf{x}, \tilde{\mathbf{x}}) \quad (3.34)$$

It can be said from equations (3.32) and (3.33) that $f(\mathbf{x})$ and $f(\tilde{\mathbf{x}})$ are jointly Gaussian with zero-mean and covariance of $\varphi(\mathbf{x})^T \Sigma_w \varphi(\tilde{\mathbf{x}})$.

It is noticeable that the specification of the covariance function defines a distribution over functions. In other words, different function values can be achieved for each value of the parameters of the kernel (i.e., hyperparameters) which conforms to a distribution of function values. It is possible to draw samples from this distribution of functions at points of interest. It is important to remember that in GPR, test points are taken into the modeling from the beginning. Let, \mathbf{x}_* be the test points. Using the prior distribution, evaluations can be made at \mathbf{x}_* , which is denoted by \mathbf{f}_* and is defined as:

$$\mathbf{f}_* = N(\mathbf{0}, K(\mathbf{x}_*, \mathbf{x}_*)) \quad [\text{noiseless NLR}] \quad (3.35)$$

$$\mathbf{y}_* = N(\mathbf{0}, K(\mathbf{x}_*, \mathbf{x}_*)) \quad [\text{noisy NLR}] \quad (3.36)$$

3.4.2 Posterior Predictive Distribution

In general, the predictive posterior distribution is:

$$P(\mathbf{f}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{f}) = \int_{\mathbf{w}} P(\mathbf{f}_* | \mathbf{w}, \mathbf{x}, \mathbf{f}, \mathbf{x}_*) P(\mathbf{w} | \mathbf{x}, \mathbf{f}) d\mathbf{w} \quad [\text{noiseless NLR}] \quad (3.37)$$

$$P(\mathbf{y}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{y}) = \int_{\mathbf{w}} P(\mathbf{y}_* | \mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{x}_*) P(\mathbf{w} | \mathbf{x}, \mathbf{y}) d\mathbf{w} \quad [\text{noisy NLR}] \quad (3.38)$$

Here, \mathbf{x} represents the training points, \mathbf{f} and \mathbf{y} are the observations at \mathbf{x} for noiseless regression and noisy regression, respectively, and \mathbf{x}_* denotes the test points. Unfortunately, equations (3.37) and (3.38) do not result in a closed-form solution (Quiñonero-Candela & Rasmussen, 2005). However, there is a solution to this problem. Because of assuming a Gaussian likelihood and a Gaussian prior, the posterior distribution is also Gaussian and the predictive posterior distribution is given by:

$$P(\mathbf{f}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{f}) = N(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \quad [\text{noiseless NLR}] \quad (3.39)$$

$$P(\mathbf{y}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{y}) = N(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \quad [\text{noisy NLR}] \quad (3.40)$$

where, $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ are the mean and covariance of the predictive distribution respectively. The expressions of $\boldsymbol{\mu}_*$ and $\boldsymbol{\Sigma}_*$ can be found easily. By definition of the Gaussian process, the joint distribution of the observed function values, \mathbf{f} , and the predicted function values, \mathbf{f}_* is given by:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{x}, \mathbf{x}) & K(\mathbf{x}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{x}) & K(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right) \quad [\text{noiseless NLR}] \quad (3.41)$$

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{x}, \mathbf{x}) + \sigma_n^2 \mathbf{I} & K(\mathbf{x}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{x}) & K(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right) \quad [\text{noisy NLR}] \quad (3.42)$$

Using short notations,

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_* & \mathbf{k}_* \end{bmatrix}\right) \quad [\text{noiseless NLR}] \quad (3.43)$$

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim N\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_* & \mathbf{k}_* \end{bmatrix}\right) \quad [\text{noisy NLR}] \quad (3.44)$$

Here, $\mathbf{K} = K(\mathbf{x}, \mathbf{x})$ is the covariance matrix of the training points \mathbf{x} , $\mathbf{K}_* = K(\mathbf{x}, \mathbf{x}_*) = K(\mathbf{x}_*, \mathbf{x})$ is the covariance matrix of training points \mathbf{x} and test points \mathbf{x}_* , and $\mathbf{k}_* = K(\mathbf{x}_*, \mathbf{x}_*)$ is the covariance matrix of the test points. If there are n number of training points and n_* test points, \mathbf{K} will be an $n \times n$ matrix, \mathbf{K}_* will be an $n \times n_*$ or an $n_* \times n$ matrix, and \mathbf{k}_* will be an $n_* \times n_*$ matrix.

The main objective here is to incorporate the knowledge about the training data and discard those functions that do not agree with the observations. This is done with the help of conditioning. Conditioning on the joint distribution in equations (3.43) and (3.44) gives:

$$\mathbf{f}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{f} \sim N\left(\mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{x})\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{K}(\mathbf{x}, \mathbf{x}_*), \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{x})\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{K}(\mathbf{x}, \mathbf{x}_*)\right) \quad [\text{noiseless NLR}] \quad (3.45)$$

$$\mathbf{y}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{y} \sim N\left(\mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{x})\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{K}(\mathbf{x}, \mathbf{x}_*), \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{x})\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{K}(\mathbf{x}, \mathbf{x}_*) + \sigma_n^2 \mathbf{I}\right) \quad [\text{noisy NLR}] \quad (3.46)$$

or,

$$\mathbf{f}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{f} \sim N\left(\mathbf{K}_* \mathbf{K}^{-1} \mathbf{f}, \mathbf{k}_* - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*\right) \quad [\text{noiseless NLR}] \quad (3.47)$$

$$\mathbf{y}_* | \mathbf{x}_*, \mathbf{x}, \mathbf{y} \sim N\left(\mathbf{K}_* (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}, \mathbf{k}_* - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_*\right) \quad [\text{noisy NLR}] \quad (3.48)$$

Equations (3.47) and (3.48) give the mean and the covariance of the predictive posterior distribution for the noiseless case and noisy case, respectively. From equations (3.47) and (3.48):

$$\text{Predictive mean, } \boldsymbol{\mu}_* = \mathbf{K}_* \mathbf{K}^{-1} \mathbf{f} \quad [\text{noiseless NLR}] \quad (3.49)$$

$$\text{Predictive Covariance, } \boldsymbol{\Sigma}_* = \mathbf{k}_* - \mathbf{K}_* \mathbf{K}^{-1} \mathbf{K}_* \quad [\text{noiseless NLR}] \quad (3.50)$$

$$\text{Predictive mean, } \boldsymbol{\mu}_* = \mathbf{K}_* (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \quad [\text{noisy NLR}] \quad (3.51)$$

$$\text{Predictive Covariance, } \boldsymbol{\Sigma}_* = \mathbf{k}_* - \mathbf{K}_* (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{K}_* \quad [\text{noisy NLR}] \quad (3.52)$$

From an earlier point, it was mentioned that mean of the prior distribution does not affect the posterior mean. From equations (3.49) or (3.51), it is evident that, despite using a zero-mean for the prior distribution, the predictive mean is not zero at all. It is also noticeable from equations (3.49) to (3.52) that an inversion of the kernel matrix of the training points, \mathbf{K} is needed to make predictions. The main computational challenge occurs here. With n increasing, \mathbf{K} increases proportionally in size ($n \times n$), leading to unmanageable computations with large n .

3.4.3 Optimizing Kernel Hyperparameters

The kernel hyperparameters are the parameters of the kernel functions that are prespecified and not learned from the model. As the values of the hyperparameters are user-defined, it is important to find the optimum values for the hyperparameters, $\boldsymbol{\theta}$. The marginal likelihood or the evidence is linked to the kernel by the following equations (Rasmussen & Williams, 2006). Hyperparameters $\boldsymbol{\theta}$ are embedded in the kernel function.

$$\log P(\mathbf{f} | \mathbf{X}) = -\frac{1}{2} \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f} - \frac{1}{2} \log |\mathbf{K}| - \frac{n}{2} \log 2\pi \quad [\text{noiseless NLR}] \quad (3.53)$$

$$\log P(\mathbf{y} | \mathbf{X}) = -\frac{1}{2} \mathbf{y}^T (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K} + \sigma_n^2 \mathbf{I}| - \frac{n}{2} \log 2\pi \quad [\text{noisy NLR}] \quad (3.54)$$

In equations (3.53) and (3.54), the first term denotes model fitting, the second term is for the model complexity depending only on the kernel function, and the last term represents a marginalization constant (Rasmussen & Williams, 2006). The optimum values of hyperparameters can be found by maximizing the log marginal likelihood or by minimizing the negative log-marginal likelihood (Lázaro-Gredilla et al., 2010).

In Figure 3.7, GPR has been presented graphically. Figure 3.7(a) shows the prior distribution over functions, where 30 samples have been taken. Figure 3.7(b) plots the observations. In figure 3.7(c), the knowledge of the training data is fed to the prior distribution. Figure 3.7(d) illustrates that the GPR model only retains those functions that go through the observations and discards the rest of the functions. This is how a non-parametric method such as Gaussian process regression ensures that the parameters agree with the training knowledge. This figure also gives the predictive posterior distribution. It can be seen from Figure 3.7(a) that the function mean vector is represented by a bold black line. In Figure 3.7(d), this mean line is not at the zero-value anymore, rather it now represents the predictive posterior mean of the process. This proves a point made earlier in the chapter that the assumption of a zero-mean vector does not have any effect on the posterior mean whatsoever.

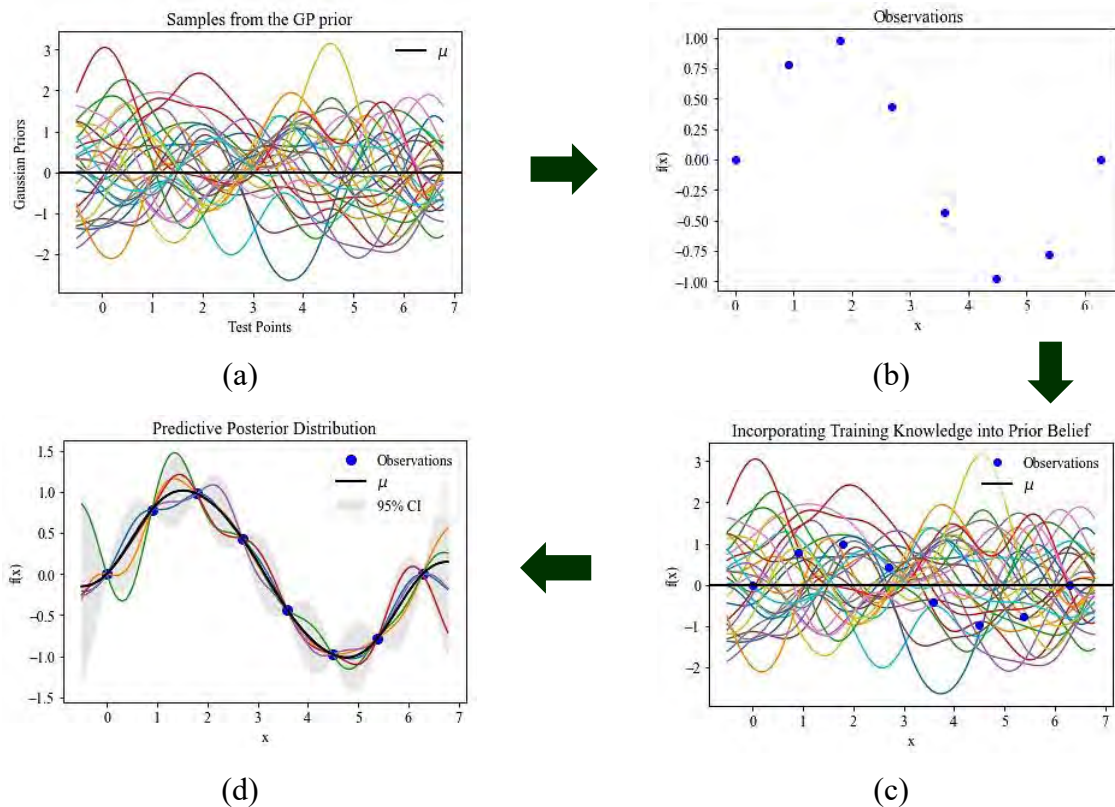


Figure 3.7: Graphical representation of Gaussian process regression

3.4.4 Merits of Gaussian Process Regression

Gaussian process regression is a widely used nonlinear fitting algorithm because it has the following properties:

Exact Inference of Posterior Distribution: GPR allows the analytical calculation of the predictive posterior distribution. From equations(3.37) and (3.38), it can be seen that the expression could not be evaluated in closed-form. However, because of using a Gaussian process prior and a Gaussian likelihood, the predictive posterior distribution can be computed exactly in closed form. This is a rare property for a non-parametric method(Duvenaud, 2014).

Quantification of Prediction Uncertainty: Typical kernel machines such as support vector machine are not probabilistic in nature, i.e., they cannot quantify the error in the prediction. However, Gaussian process regression overcomes this limitation of kernelized methods due to its explicit probabilistic formulation (Ranganathan et al., 2010). As a result, they can provide a confidence interval of their predictions.

Efficiency for Smaller Datasets: Equations (3.49) to (3.52) presents the predictive posterior mean and covariance of the Gaussian process. The main computational burden occurs at the inversion of the kernel matrix. As long as the dataset is small or moderate in size, the basic GPR provides excellent performance in terms of computational time and memory.

Superior Fitting Performance: A useful property of GPR is that it converges to the data distribution very rapidly. For typical parametric methods and even for other non-parametric methods, the number of training points plays a role in the model performance. Generally, these methods require relatively more data points to provide an acceptable model performance. However, it has been reported that GPR can perform at a satisfactory level even with a small set of training data (Yuan et al., 2008).

Flexible Modelling: One of the reasons for the wide usage of GPR as a modeling tool is that it provides impeccable flexibility to the modeler. For instance, there is a wide range of kernel functions available, and the modeler can test the performance of the kernels and choose any kernel according to need. Again, modeling with non-Gaussian likelihood is also possible. Of course, an exact posterior distribution cannot be for this case; however, approximate results can be achieved (Hensman et al., 2013; Jylänki et al., 2011).

3.4.5 Shortcomings of Gaussian Process Regression

Despite having some excellent properties, GPR has limited usage in some cases due to the following limitations:

Inefficacy to Handle Large Dataset: Equations (3.49) to (3.52) give that for making predictions an inversion of the $n \times n$ covariance matrix is mandatory which has $O(n^3)$ computations and $O(n^2)$ memory (Lázaro-Gredilla et al., 2010). As n grows, it becomes computationally infeasible. Although the use of Cholesky decomposition reduces the computation in half, still GPR for big datasets is not time and memory-efficient (Alex J. Smola & Bartlett, 2001).

Overfitting Issue: Gaussian process regression is quite susceptible to overfitting (Mohammed & Cawley, 2017). Overfitting refers to the problem where the model performs very well on the training set but gives poor predictions on the test set (Subramanian & Simon, 2013). As a result, the predictive distribution can amplify very small fluctuations in the training dataset. A possible treatment of overfitting is using some form of regularization such as ridge regression (Sarle, 1996).

Gaussian process regression, despite being an excellent regression method, is prohibitive in some cases, especially involving big datasets. To alleviate the limitations of the original model, researchers first proposed sparse methods. Sparse methods, although lucrative, possess some inherent pitfalls, one of the main reasons behind this is that these methods are implemented in a batch setting. To cope with the issue, incremental learning emerged. Eventually, the collaboration of many existing methods was seen in the literature. In the next chapter, a discussion on the principal learning dynamics of a typical machine learning model is presented. Based on the discussion of the next chapter, the proposed methodology is presented in the next-to-next chapter.

CHAPTER4

LEARNING DYNAMICS IN MACHINE LEARNING

Before presenting the mechanism of the proposed methodology for Gaussian process regression in the next chapter, a discussion on learning schemes in machine learning seems appropriate. In this chapter, two principal learning techniques in machine learning namely batch learning and incremental learning have been discussed. In section 4.1, a description of batch learning, its assumptions, characteristics, merits, and limitations have been provided. Section 4.2 offers a discussion on incremental learning strategy, its properties, benefits, challenges, and applications.

4.1 Batch Learning

Batch learning is the traditional way to perform machine learning tasks. This is also known as *offline learning*. Generally, the dataset is allotted into two groups for this learning: a training set and a validation/test set. This is usually done in a 70%-30% ratio, although this is up to the user. The main reason for dividing the dataset into these two portions is, only training a model based on the dataset is not enough, information on the model performance is also important. Therefore, after training the model using the training set, the validation set is used to predict the model output(s) and compare it to the true output(s). At this point, some error metrics can help understand how competently the model is performing. The batch learning methodology is presented in Figure 4.1.

4.1.1 Assumptions of Batch Learning

Batch learning deems the following assumptions:

Availability of training data: The main assumption of batch learning is, it takes that all the training data is available before modeling(Bishop, 2006).

Consistency in data characteristics: Batch learning considers the dataset and its characteristics to be static (Gepperth & Hammer, 2016),i.e., the underlying distribution of the data does not change over time.

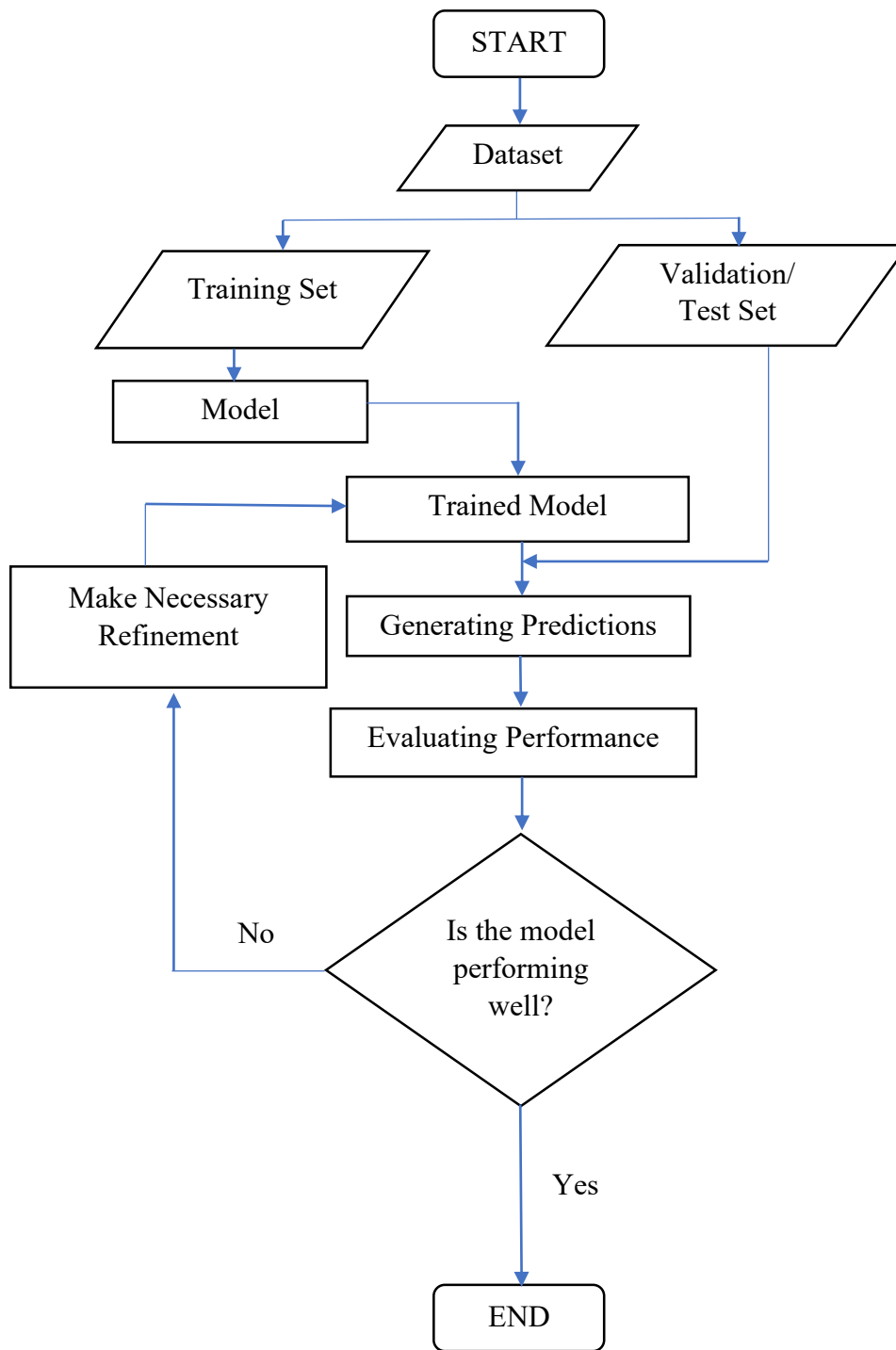


Figure 4.1: The architecture of batch learning

4.1.2 Characteristics of Batch Learning

Batch learning possesses the following characteristics:

Simultaneous use of the full dataset: In batch learning, all training points are used at once (Nilsson, 1996). If there is any change in the training set, the model needs to be updated as a whole if operating in a batch setting.

Memoryless property: Batch learning does not retain training knowledge (Mitchell, 1997). It can generate predictions based on a static training set. For any additional information, the model has no way of updating the existing knowledge other than to be trained again.

4.1.3 Merits of Batch Learning

Batch learning inherits the following advantages:

Reliable estimation of model parameters: As batch learning uses every point in the dataset at the same time, model parameters can be learned with greater accuracy (Bottou & Cun, 2003). Even if this is an approximated measure, it is close to the actual parameters.

Economy for smaller datasets: For small datasets, batch learning is quite effective. It does not take much time or memory to train a model on small datasets with batch learning (Cheng & Boots, 2016).

4.1.4 Limitations of Batch Learning

Some of the major limitations of batch learning are listed below:

The high expense for big datasets: Computational complexity and memory requirement of batch learning generally grows with the data instances. When the size of the dataset grows, batch learning becomes expensive (Cheng & Boots, 2016). Typical time complexity and memory requirement for batch-GPR are $O(n^3)$ and $O(n^2)$, which increases with the dataset size (Smola & Bartlett, 2001).

Incapability to accommodate streaming input: For streaming input, batch learning fails to accommodate the processing because the algorithm will have to retrain every time a new training point becomes available, which is computationally infeasible (Ade & Deshmukh, 2013).

Inability to adjust to variable training set: Batch learning also fails when training data points are removed (Lütz et al., 2013). Retraining is inevitable for this case too.

To overcome these shortcomings, another learning strategy called *incremental learning* has become popular.

4.2 Incremental Learning

Incremental learning is the most recent machine learning strategy. This learning scheme is also known as *sequential learning*. Incremental learning is sometimes used interchangeably with *online learning*; however, online learning might be slightly different in the sense that online learning is not committed to always keep the training knowledge in memory (Saffari et al., 2009). Many researchers have used them to point at the same thing. Anyhow, the basic idea of incremental learning is to not consider the whole training dataset at once, rather taking up training points one-by-one and learning from it. Incremental learning offers certain key advantages because incremental learning does not confine itself to the assumptions of batch learning. The incremental or sequential learning strategy can be perceived as in Figure 4.2.

4.2.1 Characteristics of Incremental Learning

Incremental learning has the following traits:

Sequential use of training points: In contrast to batch learning, this methodology does not entertain the use of training sets in the traditional sense. Instead of the instantaneous learning from the training set, this model learns sequentially from individual points (Mitchell, 1997).

Retention of model knowledge: Incremental learning is ought to hold on to the learning for as long as the modeler wants (Ade & Deshmukh, 2013). As this strategy is based on learning from a sequential feeding, the model will not work if it cannot remember what it has learned so far. That

is why the incremental learning model typically possesses the model knowledge retention property.

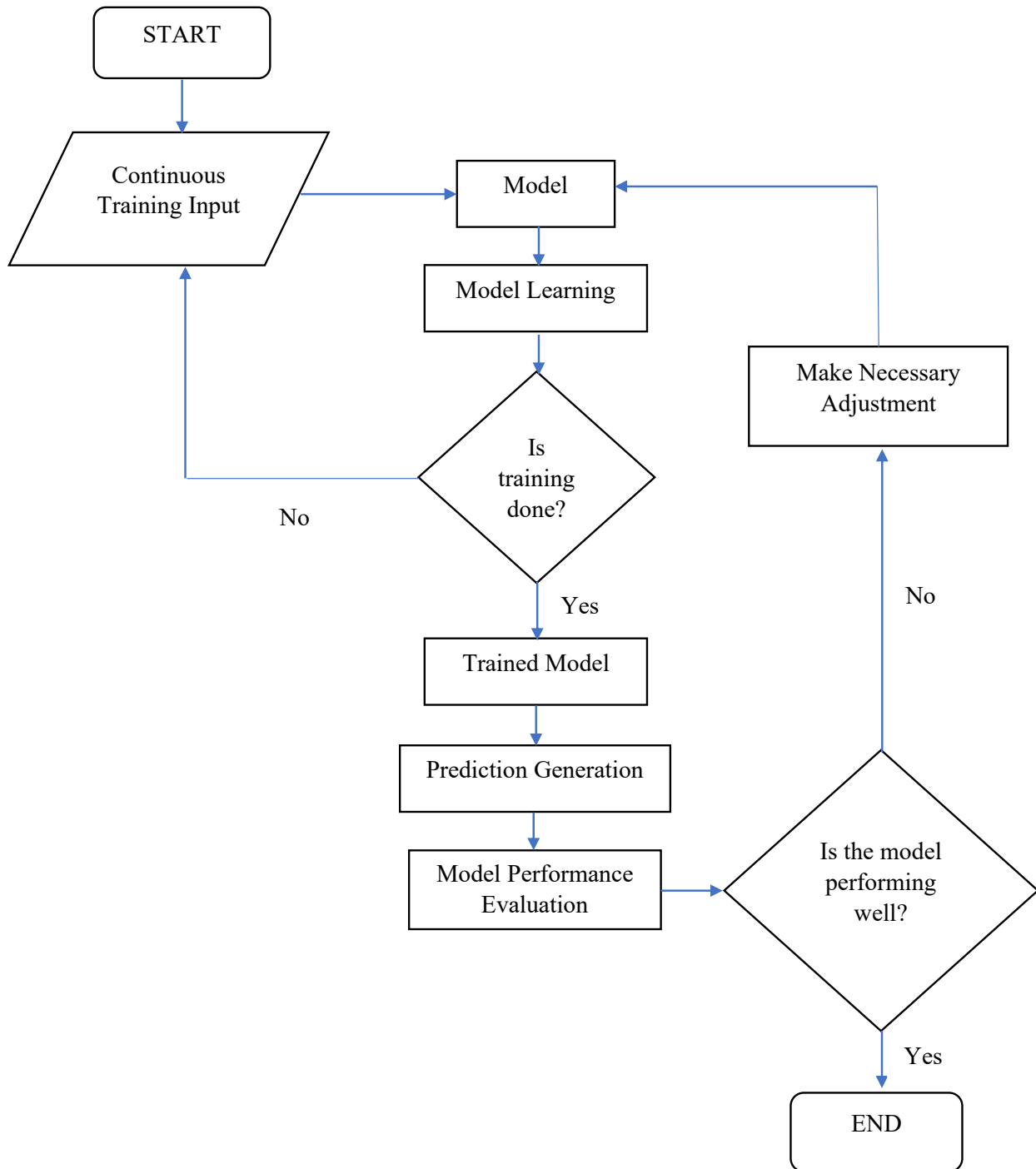


Figure 4.2: The architecture of incremental learning

Alteration in data distribution: Batch learning takes the data characteristics to be static for granted, which may not work for incremental learning. As the application of incremental learning dictates the length of time for which the training points will be collected, there is no guarantee of how long this time is going to be. It could be in months, or even years. In this long period, the underlying data distribution might change (Tsymbal, 2004). Therefore, this assumption does not hold water for incremental learning.

4.2.2 Merits of Incremental Learning

Incremental learning is attractive as it provides the following benefits:

Flexibility in modeling: This process can adjust to the new arrival of datapoints which makes it compatible to work with streaming input (Nguyen-Tuong, Seeger, et al., 2009). As a result, in the cases where the data is provided by human feedback can be handled with this kind of learning strategy. Not only for new arrivals, but the existing points in the training set can also be removed through appropriate “de-learning” techniques (Cao & Yang, 2015).

Efficiency in memory usage: Incremental learning offers efficient memory consumption. As this kind of algorithm is often used in applications with memory or resource limitations, data are kept in a concise way that ensures inexpensive memory usage (Gepperth & Hammer, 2016).

Control over model expense: Incremental learning scheme can control the cost to update the knowledge (Mouchaweh, Devillez, Lecolier, & Billaudel, 2002). Computations and memory requirements can be kept to a minimum (Nguyen-Tuong, Seeger, et al., 2009).

4.2.3 Challenges with Incremental Learning

Some real challenges with incremental learning are listed below:

Online parameter adaptation: Incremental learning tries to provide a reliable model whenever a new data point is trained. In batch learning, all the data points arrive at the same time, so there is no problem in determining the parameters from the data. However, incremental learning does not permit knowing how many points to train in advance. As a result, the parameters need to be

estimated every time the model takes in new samples. This is a major challenge for incremental learning.

Concept drift: One important presumption of batch learning is that it treats the observations as static, i.e., the data characteristics do not change over time. However, incremental learning works with a broader time-range. Therefore, the data characteristics might alter with time. This alteration in the distribution of the data is known as *concept drift* (Kulkarni & Ade, 2014; Polikar & Alippi, 2013; Tsymbal, 2004) Some typical concept drift seen in practice are:

(i) **Virtual concept drift:** This drift refers to the changes in the input distribution. For the input x of a certain system, alterations in $p(x)$ will be known as virtual concept drift (Ditzler, Roveri, Alippi, & Polikar, 2015). This occurs when the training data and the test data does not conform to the same distribution. This is also known as *covariate shift* (Sugiyama, Krauledat, & MÄžller, 2007).

(ii) **Real concept drift:** This drift points to the changes in the marginal distribution of the output. For a set of input x and output y , if the marginal distribution or evidence $p(y|x)$ changes with time, it is known as real concept drift (Gama, Žliobaitė, Bifet, Pechenizkiy, & Bouchachia, 2014). This is a serious problem for machine learning tasks, especially classification (Gepperth & Hammer, 2016).

(iii) **Concept shift:** Concept drift can occur slowly or abruptly. If the change is sudden and vigorous, then this phenomenon is known as *concept shift* (Vorburger & Bernstein, 2006).

(iv) **Local concept drift:** If the drift occurs in a particular region, it is called *local concept drift* (Tsymbal, 2004).

A combination of incremental and decremental learning might be helpful against concept drift (Raducanu & Vitria, 2008).

Stability-plasticity dilemma: When concept drift exists in the system, another challenge presents itself regarding the timing and the means of updating the knowledge of the system. If the system is updated quickly, the new information is updated quickly; however, there is a chance that the old information is forgotten equally quickly. On the other hand, if the system is

slow to update, the old knowledge is kept all right but the new information might be lost due to the lag. This trading off of the knowledge between old and new models is known as the *stability-plasticity dilemma* (Mermillod, Bugaiska, & Bonin, 2013). If the stability-plasticity dilemma gets even worse, it can give rise to *catastrophic forgetting*, which is extremely undesirable for machine learning models as this leads the model to forget everything it has learned so far (McCloskey & Cohen, 1989).

Adaptive model complexity: For batch learning, model complexity is static. However, as the training samples vary from time to time for incremental models and full training knowledge is not available at any given moment, the model complexity is variable in this scenario. Due to this reason, resource allocation for incremental learning is of importance. A typical treatment of this issue is gradual de-learning of the model which allows the model to remove training points after the training set has reached a certain size (Gepperth & Hammer, 2016).

Efficient memory management: Incremental models are needed particularly in the cases where there is a resource limitation (Ade & Deshmukh, 2013). Therefore, these models are required to store information concisely. This challenge limits these algorithms to use expensive approximations. Also, incremental learning methods sometimes require the use of decremental un-learning strategies to ensure optimum memory usage (Raducanu & Vitria, 2008).

4.2.4 Applications of Incremental Learning

Some of the applications of incremental learning are listed below:

Big data analytics: Principal application of incremental learning methodology is in big data processing, unquestionably. Due to the high expense of batch learning in the context of big datasets, incremental learning by default becomes the preferred choice. For example, incremental learning has been popular with big datasets that allow only a single sweep of the training set due to memory limitations (Hammer, He, & Martinetz, 2014). Other than this, big data visualization (Malik, Hussain, & Wu, 2016), extreme learning in big data classification (Xin, Wang, Qu, & Wang, 2015), and network data processing (Dhanjal, Gaudel, & Cl  men  on, 2014) are some of the many examples of the use of incremental learning in the big data processing.

Robotics: Learning of the system dynamics and the human-robot interactions in robotics are inherently incremental. The system learns from a stream of data points arriving over time (Gepperth & Hammer, 2016). This is analogous to “lifelong learning” for humans. Other than robotics, autonomous driving is also using the basic principles of incremental learning (Mozaffari, Vajedi, & Azad, 2015).

Image processing: Incremental learning is beneficial to applications involving training samples in the form of images or videos being available over time. Typical application areas include object identification, video surveillance, facial recognition (Bai, Ren, Zhang, & Zhou, 2015; Dewan, Granger, Marcialis, Sabourin, & Roli, 2016; Lu, Boukharouba, Boonært, Fleury, & Lecoeuche, 2014).

Based on the description above, it suffices to say that batch learning is not a popular way of training machine learning models, especially for big datasets and streaming training points. Even if not for these applications, batch learning has other limitations. As a result, batch algorithms, in this context specifically batch-GPR is not attractive anymore. Naturally, the incremental implementation of Gaussian process regression is necessitated. In the next chapter, the proposed methodology that this thesis is proposing is presented in detail.

CHAPTER 5

PROPOSED SPARSE INCREMENTAL GAUSSIAN PROCESS REGRESSION

Gaussian process regression, despite being useful and efficient in certain situations, is not sparse, *i.e.*, it considers the entire training set. Therefore, the algorithm struggles with additional computational complexity and memory requirements in general for larger datasets. As a result, sparse methodologies have been used massively among researchers to approximate the full-rank GPR. Additionally, some research proposed online learning strategies. In this chapter, a methodology called *sparse-incremental Gaussian process regression (si-GPR)* has been presented that combines sparse representation and incremental learning strategy. The proposed methodology uses a simpler yet effective approximation tool with a flexible model updating routine and an additional de-learning approach so that the proposed algorithm remains computationally feasible and provides better performance. In section 5.1, the notations used to denote the components of the algorithm are provided. Section 5.2 presents a general overview of the proposed algorithm. In section 5.3, a detailed breakdown of the algorithm is given with a flowchart. And finally, the pseudocodes for the modules are presented and explained in section 5.4.

5.1 Notations used for the Proposed *si-GPR* Algorithm

Table 5.1 presents the notations used for explaining the *si-GPR* algorithm:

Table 5.1: Notations for *si-GPR* algorithm

Symbol	Description
n	The original number of data points
\mathbf{D}	Original dataset
\mathbf{D}_{train}	Original training Set
\mathbf{D}_{test}	Test set

\mathbf{D}_1	Initial training set
\mathbf{D}_2	Streaming set
\mathbf{D}_R	Representative set
\mathbf{K}_R	Kernel matrix of the representative set, \mathbf{D}_R
\mathbf{L}_R	Lower Cholesky factor of the kernel matrix, \mathbf{K}_R
\mathbf{L}_R^{-1}	Inverse lower Cholesky factor of the kernel matrix, \mathbf{K}_R
$K_{R\max}$	Maximum size of the kernel matrix
\mathbf{X}_R	Training matrix at any given iteration for streaming input
\mathbf{K}_{new}	Covariance vector of existing training vector with a new point
K_*	The variance of the new point
$\mathbf{K}_{\text{updated}}$	Updated kernel matrix at any iteration for streaming points
\mathbf{L}_{new}	Lower Cholesky factor for the \mathbf{K}_{new} matrix
$\mathbf{L}_{\text{updated}}$	Updated lower Cholesky factor for $\mathbf{K}_{\text{updated}}$ at any iteration
$\mathbf{L}_{\text{updated}}^{-1}$	Inverse lower Cholesky factor for $\mathbf{K}_{\text{updated}}$ matrix

5.2 General Overview of Sparse Incremental Gaussian Process Regression (si-GPR)

The proposed methodology consists of the following three phases:

5.2.1 Phase-1: Approximation/ Sparsification

This is the first phase of the proposed algorithm. The training set is segregated into two classes called the *initial training set* and the *streaming set* in a 50%-50% ratio. Phase-1 deals with only the initial training set. This algorithm does not seek to make predictions with the whole dataset, rather tries to approximate the original dataset without increasing computation. That is why this thesis has used a representative dataset instead of the original set. A representative dataset is a set that can replicate the behavior of the original dataset despite having lesser instances than the original set (Yoshioka & Ishii, 2001). An effective algorithm called clustering using representatives or CURE (Guha, Rastogi, & Shim, 1998) was used for this purpose. The output of this algorithm, i.e., the representative points tries to reproduce the geometry of the original clusters. This particular algorithm was used because it has some desirable properties:

(i) It can identify arbitrarily shaped clusters

(ii) The algorithm is immune to outliers

(iii) The computational requirement is less. The time complexity for this algorithm is $O(n^2)$, and the storage requirement is $O(n)$, where n is the total number of data points.

The operation of obtaining the representative set is known as the *sparsification* operation. The rationale for using sparsification was to gain a computational edge. After obtaining the representative set, the kernel matrix for this dataset, \mathbf{K}_R , and the lower Cholesky factor of the \mathbf{K}_R matrix, \mathbf{L}_R was calculated which were required for a specific reason mentioned in the next phase.

5.2.2 Phase-2: Incremental Learning

This phase allows the proposed algorithm to accommodate streaming input, i.e., input that arrives over time and is fed to the model sequentially. It should be noted here that this thesis did not handle actual streaming input due to resource limitations. To make up for this limitation, the streaming set was loaded in the memory at a time but the points were fed sequentially just as the real feedback input would have been. From the discussion in chapter 3, it is noticeable that an inversion of the covariance matrix needs to be performed to make predictions. In typical GPR, for n data instances, the covariance/kernel matrix is an $n \times n$ matrix, that needs to be computed at once. The inversion of the covariance matrix demands $O(n^3)$ operations. However, the streaming input will arrive over time and there is no guarantee regarding when the training might end. If the computations were to run every time a new point arrives, the incremental learning attempts would have been futile. If the kernel matrix is examined closely, it can be noticed that the interaction of a new point with the existing points can be expressed through a particular row and column of the kernel matrix. Hence, the idea of augmenting the kernel matrix and its lower Cholesky factor emerged rather than computing it in every single iteration. From phase-1, it can be seen that a kernel matrix of the representative set and its lower Cholesky factor was calculated. The basic idea of implementing incremental learning in this thesis was to extend these two matrices for each new point. Although the kernel matrix is being augmented here in every iteration, this will not be directly used for the inversion. Rather, the Cholesky factorization will be used to invert this matrix. The reason for using Cholesky decomposition instead of direct

inversion is due to the lesser computation required by the Cholesky method, almost half of the direct method of matrix inversion (Rasmussen & Williams, 2006). Now, for implementing the incremental learning, the idea is to augment the lower Cholesky factor in every iteration, which means appending the basic lower Cholesky factor by one row and one column. After the end of the training period, the inverse matrix of the augmented lower Cholesky factor was computed. From the equation (5.10), it should be clear that the inverse of the updated kernel matrix can be calculated through the inverse augmented lower Cholesky matrix. It should also be mentioned here that the calculation of the kernel matrix might seem redundant at this stage; however, this is an important step for the de-learning strategy described in the next step.

5.2.3 Phase-3: Decremental Un-learning

Incremental learning algorithms are often used where there is a memory limitation. Therefore, it might be required to keep the computations tractable so that less memory is required. The management of the computations can be done through a sequential de-learning strategy. As the main computational burden comes from the kernel matrix, an operation can be performed that helps the kernel matrix be in a specific size. For instance, if the specified size of the kernel matrix is $p \times p$, then when a new data point arrives, the algorithm deletes the first training instance, i.e., it deletes the first row and first column of the kernel matrix, so that the new point can be accommodated within the preferred size ($p \times p$). This operation ensures the constant size of the kernel matrix, thereby confirming a steady training memory usage.

In summary, this algorithm provides a lower-rank approximation through sparsification (phase-1), an incremental learning strategy that allows sequential input and efficient kernel matrix inversion (phase-2), and a decremental unlearning approach that keeps the algorithm computationally feasible and also addresses the memory limitation issue (phase-3). Figure 5.1 maps out the overall structure of the algorithm and the next section provides a detailed breakdown of the algorithm.

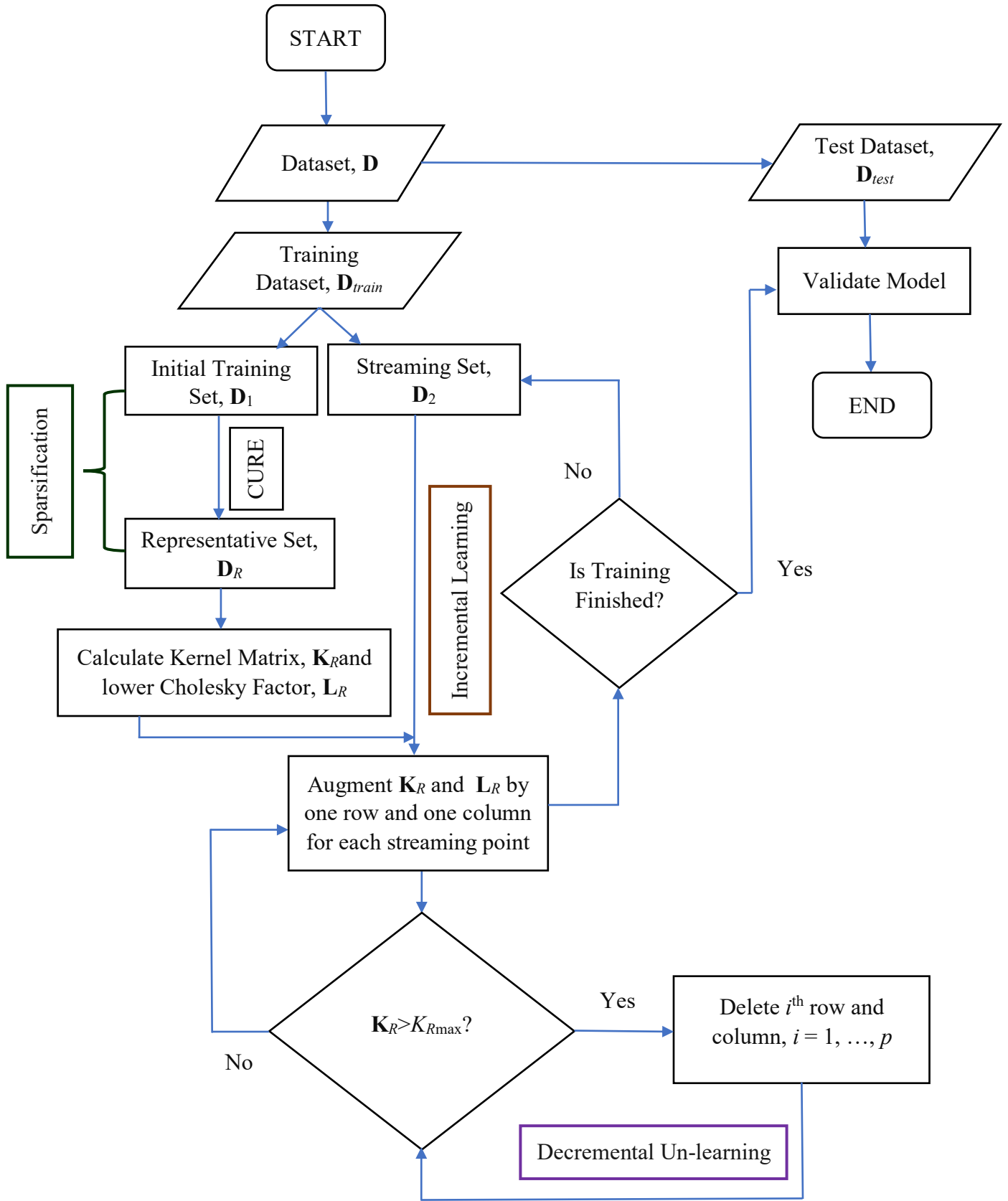


Figure 5.1: Flowchart for the proposed *si*-GPR algorithm

5.3 Detailed Breakdown of the Algorithm

The outline of the proposed algorithm can be found in Figure 5.1. In this section, the proposed modifications have been offered. Steps 1 through 3 of the algorithm fall under phase-1, step 4 is dedicated to phase-2, phase-3 concerns step-5, step-6 denotes the end of the training, and step-7 is related to the model validation.

Step-1: The training set was divided into halves. The first part of the training set is called the initial training set, \mathbf{D}_1 , and the other part is noted as the streaming set, \mathbf{D}_2 .

Step-2: A data clustering algorithm called CURE (Guha et al., 1998) was applied on the initial training set, \mathbf{D}_1 , and a representative set, \mathbf{D}_R was obtained, which is smaller than the initial training set.

Step-3: Using this representative dataset, \mathbf{D}_R , the kernel matrix for this set, \mathbf{K}_R was calculated. Additionally, the lower Cholesky factor of the kernel matrix, \mathbf{L}_R was also computed.

Step-4: This step allows incremental learning. For the streaming set, \mathbf{D}_2 , each data point will be fed successively. In this case, an efficient update of the kernel matrix is needed rather than going through the same calculation in each iteration. The strategy used to update the kernel matrix and the lower Cholesky factor is similar to (Nguyen-Tuong, Seeger, et al., 2009).

Efficient Update of the Kernel Matrix:

The basis for this update is the existing covariance matrix for the representative dataset, \mathbf{K}_R , that was obtained from step-3. Now, for every new training point that is fed one-by-one, only the last row and the last column need to be updated. Two interactions are involved here: the covariance of the existing training vector with the new point, \mathbf{K}_{new} , and the variance of the point itself, K_* . The update is given in equation (5.3). Because the kernel matrix is symmetric, only one of them (either the row or the column) needs to be calculated that can be transposed to get the other one. The last element of the matrix is the variance of the new point. In short, two quantities need to be calculated from equations (5.1) and (5.2), which will be used to extend the kernel matrix found from the representative set by one row and one column at a time for a new data point. The update on the kernel matrix can be seen graphically in Figure 5.2.

At the end of phase-1, \mathbf{K}_R

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_{n/2})$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_{n/2})$
\vdots	\vdots	.	\vdots
$K(\mathbf{x}_{n/2}, \mathbf{x}_1)$	$K(\mathbf{x}_{n/2}, \mathbf{x}_2)$	$K(\mathbf{x}_{n/2}, \mathbf{x}_{n/2})$

After 1st iteration in phase-2, for point $\mathbf{x}_{s_1}, \mathbf{K}_{\text{updated}}$

\mathbf{K}_R	$K(\mathbf{x}_R, \mathbf{x}_{s_1})$
$K(\mathbf{x}_{s_1}, \mathbf{x}_R)$	$K(\mathbf{x}_{s_1}, \mathbf{x}_{s_1})$

After 2nd iteration in phase-2, for point $\mathbf{x}_{s_2}, \mathbf{K}_{\text{updated}}$

\mathbf{K}_R	$K(\mathbf{x}_R, \mathbf{x}_{s_1})$	$K(\mathbf{x}_R, \mathbf{x}_{s_2})$
$K(\mathbf{x}_{s_1}, \mathbf{x}_R)$	$K(\mathbf{x}_{s_1}, \mathbf{x}_{s_1})$	
$K(\mathbf{x}_{s_2}, \mathbf{x}_R)$		$K(\mathbf{x}_{s_2}, \mathbf{x}_{s_2})$

After n^{th} iteration in phase-2, for point $\mathbf{x}_{s_n}, \mathbf{K}_{\text{updated}}$

\mathbf{K}_R	$K(\mathbf{x}_R, \mathbf{x}_{s_1})$	$K(\mathbf{x}_R, \mathbf{x}_{s_2})$	$K(\mathbf{x}_R, \mathbf{x}_{s_n})$
$K(\mathbf{x}_{s_1}, \mathbf{x}_R)$	$K(\mathbf{x}_{s_1}, \mathbf{x}_{s_1})$			
$K(\mathbf{x}_{s_2}, \mathbf{x}_R)$		$K(\mathbf{x}_{s_2}, \mathbf{x}_{s_2})$		
\vdots				
$K(\mathbf{x}_{s_n}, \mathbf{x}_R)$				$K(\mathbf{x}_{s_n}, \mathbf{x}_{s_n})$

Figure 5.2: Visualization of incremental updates of the kernel matrix

Let, the existing training vector now be \mathbf{X}_R and the new point \mathbf{x} . The covariance of the existing training vector with the new point is denoted by \mathbf{K}_{new} , and the variance of the new point is denoted by K_* , that can be calculated from equations (5.1) and (5.2).

$$\mathbf{K}_{\text{new}} = \mathbf{K}(\mathbf{X}_R, \mathbf{x}) \quad (5.1)$$

$$K_* = \mathbf{K}(\mathbf{x}, \mathbf{x}) \quad (5.2)$$

Given the current covariance matrix \mathbf{K}_R , the updated kernel matrix will be:

$$\mathbf{K}_{\text{updated}} = \begin{bmatrix} \mathbf{K}_R & \mathbf{K}_{\text{new}} \\ \mathbf{K}_{\text{new}}^T & K_* \end{bmatrix} \quad (5.3)$$

Efficient Update of the Lower Cholesky Factor:

For inversion of square matrices, popular methods such as the lower-upper (LU) decomposition (Rasmussen & Williams, 2006) or the Woodbury formula (M. W. Seeger, 2004) are generally incorporated.

LU decomposition decomposes the original matrix to be inverted into two matrices: an upper triangular matrix, \mathbf{U} , and a lower triangular matrix, \mathbf{L} (Chapra & Canale, 2010). For a typical square matrix \mathbf{B} , if the lower and upper factors are \mathbf{L} and \mathbf{U} respectively, then the following expression can be written:

$$\mathbf{B} = \mathbf{LU} \quad (5.4)$$

$$\mathbf{B}^{-1} = (\mathbf{LU})^{-1} \quad (5.5)$$

$$= \mathbf{U}^{-1}\mathbf{L}^{-1} \quad (5.6)$$

The inverse matrices of \mathbf{L} and \mathbf{U} can be obtained from equations (5.7) and (5.8).

$$\mathbf{LL}^{-1} = \mathbf{I} \quad (5.7)$$

$$\mathbf{UU}^{-1} = \mathbf{I} \quad (5.8)$$

When the matrix is symmetric, another technique similar to the LU decomposition known as the Cholesky decomposition is used. As the kernel matrix is symmetric, Cholesky decomposition can be incorporated for the inversion operation.

A symmetric matrix \mathbf{A} , it can be written as equation (5.4), where \mathbf{L} is the lower Cholesky factor:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T \quad (5.9)$$

$$\begin{aligned} \mathbf{A}^{-1} &= (\mathbf{L}\mathbf{L}^T)^{-1} \\ &= (\mathbf{L}^T)^{-1}\mathbf{L}^{-1} \\ &= (\mathbf{L}^{-1})^T\mathbf{L}^{-1} \end{aligned} \quad (5.10)$$

It can be understood from equation (5.10) that only the inverse of the lower Cholesky factor needed to be calculated for finding the inverse of the original matrix. This simple operation reduces computations to a good extent.

Now, for streaming input, the lower Cholesky of the kernel matrix, \mathbf{L}_R will be updated as follows:

$$\mathbf{L}_{\text{updated}} = \begin{bmatrix} \mathbf{L}_R & \mathbf{0} \\ \mathbf{L}_{\text{new}}^T & L_* \end{bmatrix} \quad (5.11)$$

$$\mathbf{L}_R\mathbf{L}_{\text{new}} = \mathbf{K}_{\text{new}} \quad (5.12)$$

$$L_* = \sqrt{K_* - \|\mathbf{L}_{\text{new}}\|^2} \quad (5.13)$$

After the training, only the inverse of $\mathbf{L}_{\text{updated}}$, denoted by $\mathbf{L}_{\text{updated}}^{-1}$, is needed to find out the inverse of the kernel matrix, $\mathbf{K}_{\text{updated}}$.

Step-5: To keep the computations tractable, a decremental un-learning strategy is also offered. When the kernel matrix reaches a certain size, for every data point, one row and one column will be deleted from the top to keep the matrix in a prespecified shape.

For example, when the size of $\mathbf{K}_{\text{updated}} > K_{Rmax}$, the algorithm starts deleting the first row and column from the current $\mathbf{K}_{\text{updated}}$ matrix. The procedure is illustrated in Figure 5.3.

After p^{th} iteration, $\mathbf{K}_{\text{updated}}$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_p)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_p)$
\vdots	\vdots	.	\vdots
$K(\mathbf{x}_p, \mathbf{x}_p)$	$K(\mathbf{x}_p, \mathbf{x}_2)$	$K(\mathbf{x}_p, \mathbf{x}_p)$

At $(p+1)^{\text{th}}$ iteration, $\mathbf{K}_{\text{updated}}$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_p)$	
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_p)$	$K(\mathbf{x}_2, \mathbf{x}_{p+1})$
\vdots	\vdots	.	\vdots	\vdots
$K(\mathbf{x}_p, \mathbf{x}_1)$	$K(\mathbf{x}_p, \mathbf{x}_2)$	$K(\mathbf{x}_p, \mathbf{x}_p)$	$K(\mathbf{x}_p, \mathbf{x}_{p+1})$
	$K(\mathbf{x}_{p+1}, \mathbf{x}_2)$	$K(\mathbf{x}_{p+1}, \mathbf{x}_3)$	$K(\mathbf{x}_{p+1}, \mathbf{x}_{p+1})$

At $(p+2)^{\text{th}}$ iteration, $\mathbf{K}_{\text{updated}}$

$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$	$K(\mathbf{x}_2, \mathbf{x}_{p+1})$	
$K(\mathbf{x}_3, \mathbf{x}_2)$	$K(\mathbf{x}_3, \mathbf{x}_3)$	$K(\mathbf{x}_3, \mathbf{x}_{p+1})$	$K(\mathbf{x}_3, \mathbf{x}_{p+2})$
\vdots	\vdots	.	\vdots	\vdots
$K(\mathbf{x}_{p+1}, \mathbf{x}_2)$	$K(\mathbf{x}_{p+1}, \mathbf{x}_3)$	$K(\mathbf{x}_{p+1}, \mathbf{x}_{p+1})$	$K(\mathbf{x}_{p+1}, \mathbf{x}_{p+2})$
	$K(\mathbf{x}_{p+2}, \mathbf{x}_3)$	$K(\mathbf{x}_{p+2}, \mathbf{x}_4)$	$K(\mathbf{x}_{p+2}, \mathbf{x}_{p+2})$

Figure 5.3: Visualization of decremental un-learning strategy

If the maximum size of the kernel matrix $K_{Rmax} = p \times p$, then the matrix can follow the operations in Figure 5.3 to keep the computations in check.

Step-6: When the program reached the end of the training, using the currently updated kernel matrix in the last iteration, the algorithm fitted a Gaussian process through it.

Step-7: Based on this fitting, further evaluations were made on the validation or test dataset. There are some popular error metrics such as mean absolute error (MAE), mean squared error (MSE), mean percentage error (MPE), mean absolute percentage error (MAPE), etc. for validating the model performance. For testing the proposed algorithm, mean absolute error (MAE) and root mean squared error (RMSE) metrics were used from equations (5.14) and (5.15) as a basis of error evaluation.

$$MAE = \frac{\sum_{i=1}^n |\hat{y}_i - y_i|}{n} \quad (5.14)$$

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}} \quad (5.15)$$

where,

$y_i = i^{\text{th}}$ original response

$\hat{y}_i = i^{\text{th}}$ approximated response

$n =$ Total number of data points

The rationale behind using these two measures is that they are the most used error bars (Chai & Draxler, 2014) and the other metrics can be derived from them. It may be noted here, another popular metric for regression, R^2 was not used. It was due to the reason that the R^2 score is not a recommended performance estimator for the nonlinear regression problem and is often too risky (Cornell & Berger, 1987).

Additionally, some performance benchmarks such as training time, training memory, testing time, testing memory, fitting accuracy, validation accuracy, etc. were evaluated to assess the performance of the proposed algorithm.

5.4 Pseudocodes for Algorithms used

Overall, three algorithms have been combined to build the proposed algorithm: (a) Initial sparse algorithm (b) Incremental learning algorithm (c) Decremental un-learning algorithm. The initial sparse algorithm is presented in section 5.4.1. The incremental learning algorithm and the decremental learning algorithm are presented in sections 5.4.2 and 5.4.3, respectively.

5.4.1 Pseudocode for Initial Sparse Algorithm

This is the initial stage of the algorithm. The algorithm in Table 5.2 is applied in this stage to the initial training set, \mathbf{D}_1 to obtain the representative set, \mathbf{D}_R , the kernel matrix obtained from it, \mathbf{K}_R , and the lower Cholesky factor, \mathbf{L}_R . It is noticeable that the algorithm in Table 5.2 uses the CURE algorithm. The code for the CURE algorithm is given in Tables 5.3 and 5.4.

Input: Initial Training Set, \mathbf{D}_1

Output: Representative set, \mathbf{D}_R , kernel matrix, \mathbf{K}_R , lower Cholesky factor, \mathbf{L}_R

Table 5.2: Initial sparse algorithm

Algorithm-1: Initial Sparse Algorithm

START

INPUT $\{x_i, y_i\}_{i=1}^n$

APPLY the data clustering algorithm (**Table 5.3**)

FIND the representative dataset, \mathbf{D}_R

COMPUTE covariance matrix, \mathbf{K}_R for \mathbf{D}_R

COMPUTE the lower Cholesky factor, \mathbf{L}_R

END

5.4.1.1 Pseudocode for the Data Clustering Algorithm, CURE

This data clustering algorithm treats every point as a cluster and merges them with the closest cluster in subsequent steps (Guha et al., 1998). Two data structures are used by CURE: a heap data structure (Cormen, Leiserson, & Rivest, 1990), and a k - d tree data structure (Samet, 1990). The heap structure arranges the distances of the clusters to the closest clusters in ascending order and the k - d tree structure stores the representative points for each cluster. Also, when two clusters are merged, the k - d tree structure computes the cluster which is currently closest to the newly merged cluster.

This algorithm is provided in Table 5.3. It is to be noted that this procedure uses a concatenation module for merging the clusters which is given in Table 5.4. The notations, input parameters, and the output of the algorithm are provided below:

Notations for CURE:

cluster.mean = Mean of any cluster

cluster.rep = Representative points of any cluster

cluster.closest = Closest cluster to any cluster

dist (p, q) = Distance between any two points p and q (Any L_p distance/ other kernels)

extract_min = Routine for deleting the top element from the heap

r = Number of well-scattered points

Input: Initial training dataset, \mathbf{D}_1 , and desired number of cluster, k

Output: Representative cluster, \mathbf{c} (Denoted as the representative dataset, \mathbf{D}_R in this thesis)

Table 5.3: Data clustering algorithm, CURE

Algorithm-2: Data Clustering Algorithm, CURE

```
START

T := construct_kd_tree (D)
Q := construct_heap (D)

WHILE size (Q) > k DO {
a := extract_min (Q)
b := a.closest
delete (Q, b)

c := concatenate (a, b) (Table 5.4)

delete_rep (T, a); delete_rep (T, b); insert_rep (T, c)
    c.closest := x /* x is an arbitrary cluster in Q */

FOR each x ∈ Q DO {
IF dist (c, x) < dist (c, c.closest)
c.closest := x
IF x.closest is either a or b {
    IF dist (x, x.closest) < dist (x, c)
        x.closest := closest_cluster (T, x, dist (x, c))
    ELSE
        x.closest := c
        relocate (Q, x)
    }
ELSE IF dist (x, x.closest) > dist (x, c) {
    x.closest := c
    relocate (Q, x)
}
}
insert (Q, c)
}
END
```

5.4.1.2 Pseudocode for the Concatenation Algorithm

This function merges two clusters. The CURE algorithm proceeds by merging any cluster with its closest cluster. The algorithm in Table 5.4 is used for this operation. The merged cluster is used again in the algorithm in Table 5.3.

Input: Any two clusters a and b

Output: Merged cluster, c

Table 5.4: Algorithm for concatenation of clusters

Algorithm-3: Concatenate (a, b)
START
$c := a \cup b$
$c.\text{mean} = \frac{ a a.\text{mean} + b b.\text{mean}}{ a + b }$
temp := \emptyset
FOR $i := 1$ to r DO {
maxDist := 0
FOReach point p in cluster c DO {
IF $i = 1$
minDist := dist ($p, c.\text{mean}$)
ELSE
minDist := min {dist (p, q) : $q \in \text{temp}$ }
IF (minDist \geq maxDist){
maxDist := minDist
maxPoint := p
}
}
temp := temp \cup {maxPoint}
}
FOReach point p in temp DO
$c.\text{rep} := c.\text{rep} \cup \{p + \alpha*(c.\text{mean} - p)\}$
return c
END

Figure 5.4 presents the initial sparse algorithm graphically. The left side of the figure represents the training signal of the *Manaus* dataset (details are in chapter-6) which is used as the input for the initial sparse algorithm, known as the initial training set. The right-side figure presents the representative set obtained from the initial training set, i.e., the output from the initial sparse algorithm. Based on this set, the incremental learning algorithm proceeds.

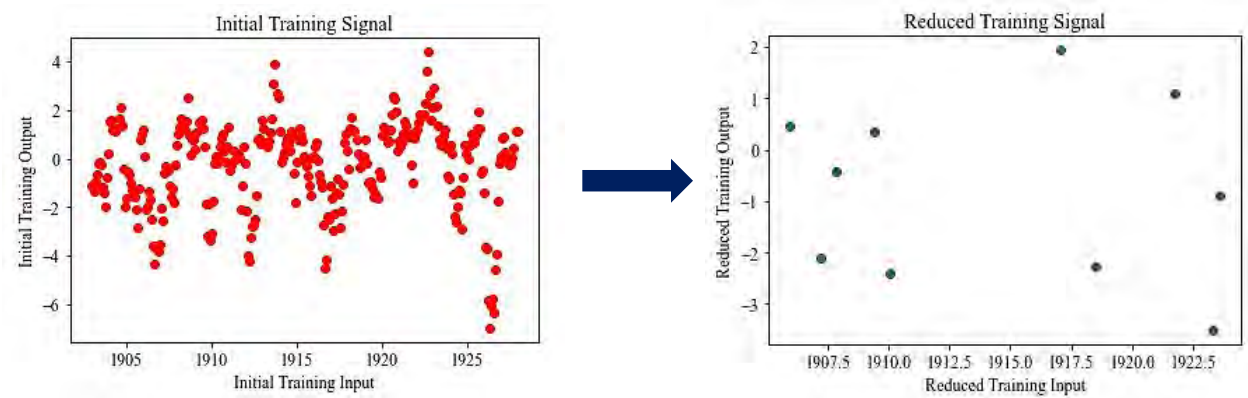


Figure 5.4: Visualization of obtaining the representative dataset, \mathbf{D}_R

5.4.2 Pseudocode for Incremental Learning Algorithm

This procedure is used for cases when the size of the training set is not fixed, or when a certain number of points become available at some time gap. Given this situation, it is important to update the kernel matrix efficiently. From earlier discussion, it is apparent that when a new data point becomes available, the only update in the previous matrix is the addition of a row and a column, which is just the interaction between the new point and the previous input vector. The update rule of the kernel matrix is given in equation (5.3). Table 5.5 provides the code for incremental learning.

Input: Kernel matrix of the representative set \mathbf{K}_R , its lower Cholesky Factor \mathbf{L}_R , and the streaming set, \mathbf{D}_2

Output: The updated kernel matrix, $\mathbf{K}_{\text{updated}}$ and the updated lower Cholesky factor $\mathbf{L}_{\text{updated}}$

Table 5.5: Algorithm for incremental learning

Algorithm-4: Incremental Learning Algorithm

START

Given $\mathbf{K} = \mathbf{K}_R$:

For streaming data $(\mathbf{x}_s, \mathbf{y}_s)$:

COMPUTE $\mathbf{K}_{\text{new}} = \mathbf{K}(\mathbf{X}_R, \mathbf{x})$

COMPUTE $K_* = \mathbf{K}(\mathbf{x}, \mathbf{x})$

COMPUTE $\mathbf{K}_{\text{Updated}}$

For the lower factor of Cholesky decomposition of $\mathbf{K}_R, \mathbf{L}_R$:

$\mathbf{L}_R \mathbf{L}_{\text{new}} = \mathbf{K}_{\text{new}}$

$L_* = \sqrt{K_* - \|\mathbf{L}_{\text{new}}\|^2}$

COMPUTE $\mathbf{L}_{\text{Updated}}$

FIT GPR using $\mathbf{K}_{\text{Updated}}$

END

5.4.3 Pseudocode for Decremental Un-learning Algorithm

When there is a larger stream of data points, the updated covariance matrix can grow much bigger. To keep the kernel matrix in check, this algorithm starts deleting rows and columns of the kernel matrix from the very beginning. For instance, if the maximum specified size of the kernel matrix, K_{Rmax} is 200×200 , for the 201st data point, the algorithm will delete the first row and column of the matrix and then attach a row and column at the end of the matrix to keep it in a size of 200×200 . Table 5.6 presents the code for the un-learning algorithm.

Input: Kernel matrix, $\mathbf{K}_{\text{updated}}$ at p^{th} iteration

Output: Updated kernel matrix, $\mathbf{K}_{\text{updated}}$ at the end of the training

Table 5.6: Algorithm for decremental un-learning

Algorithm-5: Decremental Un-learning Algorithm

START

Given $\mathbf{K}_{\text{updated}}$ at p^{th} iteration:

When $\mathbf{K}_{\text{updated}} \geq K_{Rmax}$

DELETE row[i] and column[i] for i th streaming point $\{i = 1, 2, \dots, m\}$

APPEND \mathbf{K}_{new} and K_* for new point

UPDATE $\mathbf{K}_{\text{updated}}$

FIT GPR using $\mathbf{K}_{\text{updated}}$

END

This chapter was dedicated to the mechanics of the proposed algorithm for implementing Gaussian process regression incrementally and economically. Given the modifications, the algorithm was tested for its efficacy. Some well-known regression datasets were used for this case. In the next chapter, a description of those datasets, their properties, visual representations, preprocessing, etc. are provided. The next-to-next chapter presents the results of these experiments on the selected datasets.

CHAPTER 6

DATASETS AND EXPERIMENTATION

The proposed algorithm in chapter-5 needs to be validated in terms of fitting and performance. Thus, the use of machine learning datasets is warranted to test the proposed model. In this chapter, a discussion on such datasets, their features, preprocessing of the datasets, environment for testing, etc. are presented. First, brief descriptions and visualization of the datasets used are provided in sections 6.1 and 6.2, respectively. Additionally, a discussion on preprocessing machine learning datasets is presented in section 6.3. Moreover, a brief look at the environment for experimentation is given in section 6.4. Finally, the conclusion of this chapter is drawn by a short note on preliminary testing for kernel selection in section 6.5.

6.1 Brief Overview of the Datasets

To test the validity of the proposed algorithms, some experimentations were performed. For this analysis, 7 datasets were used. These are fairly well-known datasets in the machine learning community. The general objective of selecting the datasets was to ensure variety in total data instances as well as in features. Overall, the total points range from 309 to 8192, and the datasets expand from a single feature to 12 features. Data characteristics of selected datasets also vary to some extent. In addition to having numerical features, categorical features were also handled. A brief description of the datasets is given below:

6.1.1 Wool Dataset

This is a time-series dataset containing the logarithmic ratio of fine-grade wool prices and the floor price set by the Australian Wool Corporation. The data values were taken for each week from July 1976 to June 1984 (Diggle, 1990). This is a two-dimensional dataset, where the data values are values of the price of wool against time. The task is to predict the price of the wool for an unseen period.

6.1.2 Istanbul Stock Exchange Dataset

In this data set, returns of the Istanbul stock exchange with other international indices are collected. The duration of data collection was from January 5, 2009 to February 22, 2011 (Akbulgic, Bozdogan, & Balaban, 2014). There are 8 features in this dataset including Istanbul stock exchange national 100 indexes, standard & poor's 500 return index, stock market return index of Germany, stock market return index of UK, stock market return index of Japan, stock market return index of Brazil, MSCI European index, and MSCI emerging markets index. The task is to predict the stock index of Istanbul.

6.1.3 Manaus Dataset

This is a time-series dataset where data values represent the monthly averages of the daily stages (heights) of the Rio Negro river at Manaus, Brazil. The data covers 90 years from January 1903 to December 1992. (Sternberg, 1987). This is also a two-dimensional dataset.

6.1.4 German Healthcare Dataset

Under the German healthcare reform, this dataset contains survey results from one year before and after the reform to assess if the number of visits to doctors has declined. This dataset is a sub-collection of the German Socio-Economic Panel (SOEP) (Rabe-Hesketh & Skrondal, 2008). Of the 12 features, there are age group, income, education, etc. and the target variable is the total number of visits to the doctor.

6.1.5 Abalone Dataset

This dataset contains the features such as length, diameter, heights, shell weights, etc. of the abalone shellfish. The task is to determine its age based on the rings it has. (Nash, Sellers, Talbot, Cawthorn, & Ford, 1994). This is a very popular dataset and the task associated with it can be formulated as both a classification and a regression problem. This thesis treated this task as a regression problem.

6.1.6 Tree Ring Dataset

This is a univariate time-series dataset bearing the normalized tree-ring widths for each year. Data collection was performed by Donald A. Graybill from Gt. Basin Bristlecone Pine 2805M, 3726-11810 in Methuselah Walk, California in 1980 (Graybill, 1985). The task is to predict the number of rings at a later period in time.

6.1.7 Pumadyn-8nm Dataset

This dataset is derived from a family of datasets concerned with a practical simulation of the kinematics of a Puma 560 robot arm. The angular acceleration of the links of the robot arm is to be predicted from information such as angular location, velocity, and torques (Ghahramani, 1996).

Table 6.1 presents a summary of the datasets used:

Table 6.1: Datasets used in experimentations

Title of the Dataset	Total Data Points	Number of Training Points	Number of Test Points	Number of Regressors	Categorical Features
1. Wool	309	200	109	1	N/A
2. Istanbul Stock Exchange	537	300	237	8	N/A
3. Manaus	1081	700	381	1	N/A
4. German Healthcare	2228	1500	728	12	Yes
5. Abalone	4177	3000	1177	8	Yes
6. Tree Ring	7981	5000	2981	1	N/A
7. Pumadyn-8nm	8192	5000	3192	8	N/A

6.2 Visualization of the Datasets

This section presents the full or partial visualization of the training signal of the datasets used. The visualization remains tractable as long as the dataset is two-dimensional or three-dimensional. However, visualization becomes difficult when the feature size increases. There is no direct way of visualizing high-dimensional datasets. In such cases, pair plots were utilized. A pair plot is a graphical way to represent the interactions of datasets having higher dimensions. This is essentially a scatter plot that offers a group of displays of paired combinations of the variables concerned (Emerson et al., 2013). Python's *matplotlib* and *seaborn* library were used for visualization purposes.

Among the datasets used, three datasets were two-dimensional (*Wool*, *Manaus*, *Tree Ring*). Training signals for these datasets are plotted in Figures 6.1 through 6.3, respectively. Pair plots have been used for the rest of the datasets, shown in Figures 6.4 through 6.7. Below are the representations of the datasets:

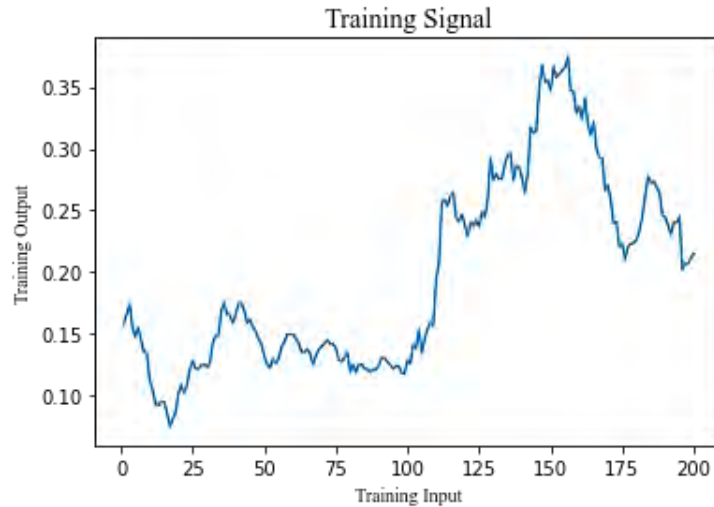


Figure 6.1: Training signal for the Wool dataset

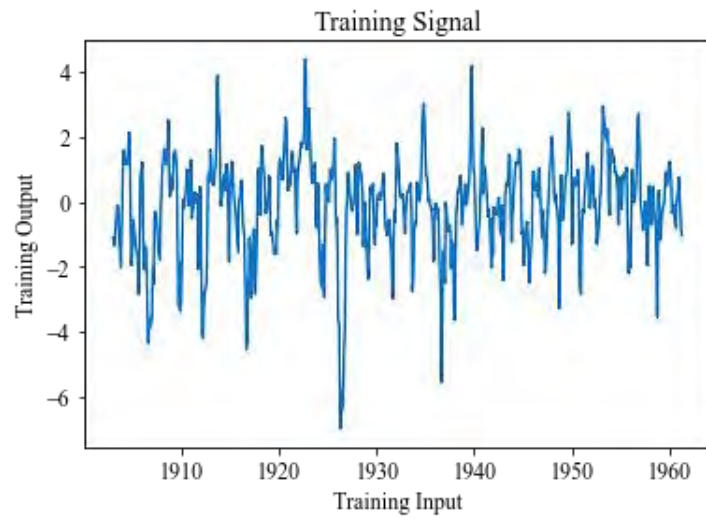


Figure 6.2: Training signal for the Manaus dataset

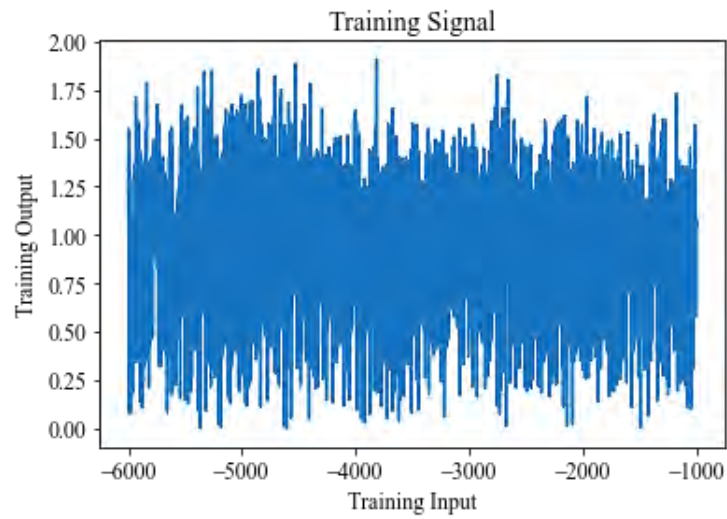


Figure 6.3: Training signal for the Tree Ring dataset

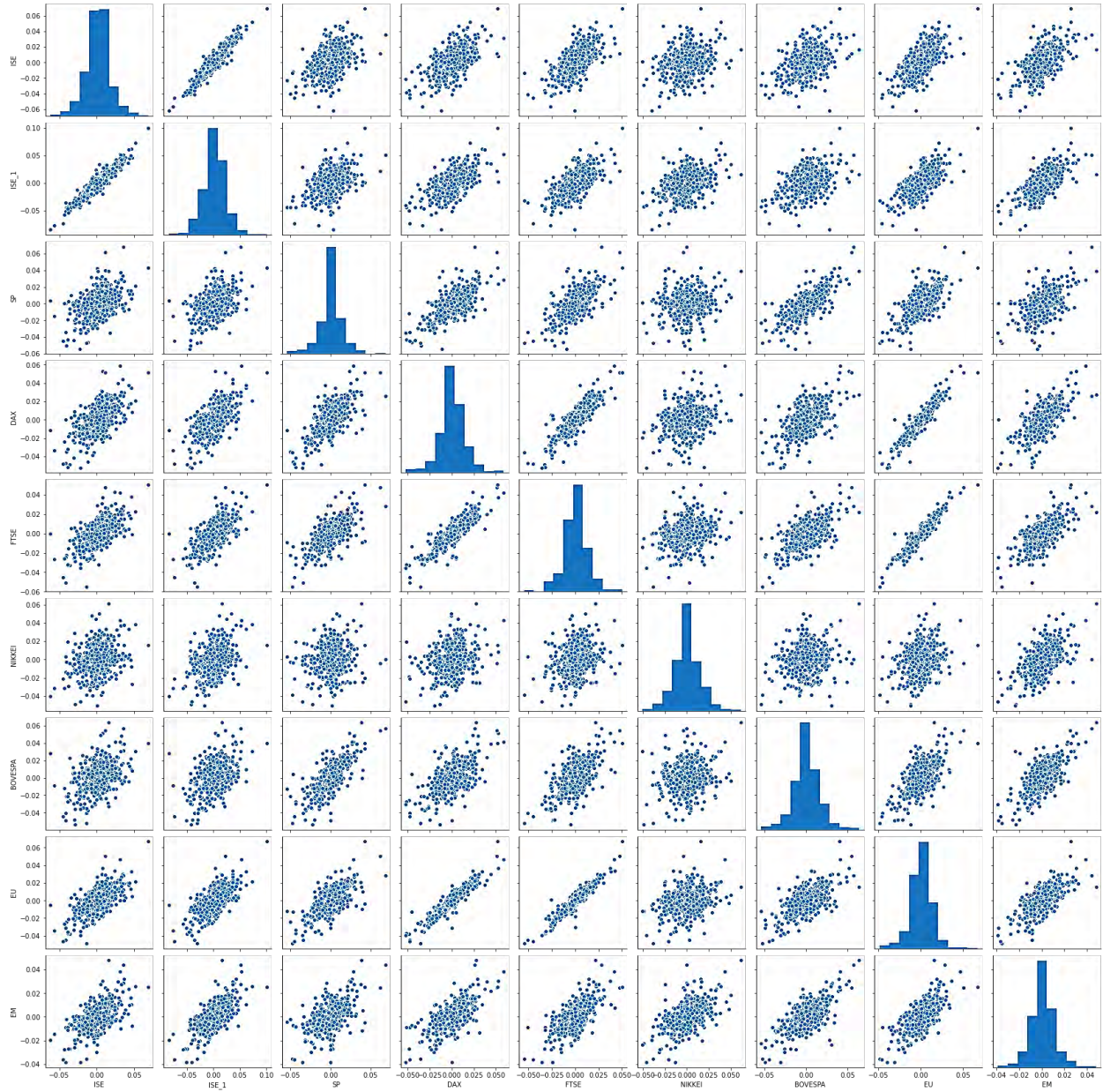


Figure 6.4: Pair plot for the Istanbul Stock Exchange dataset

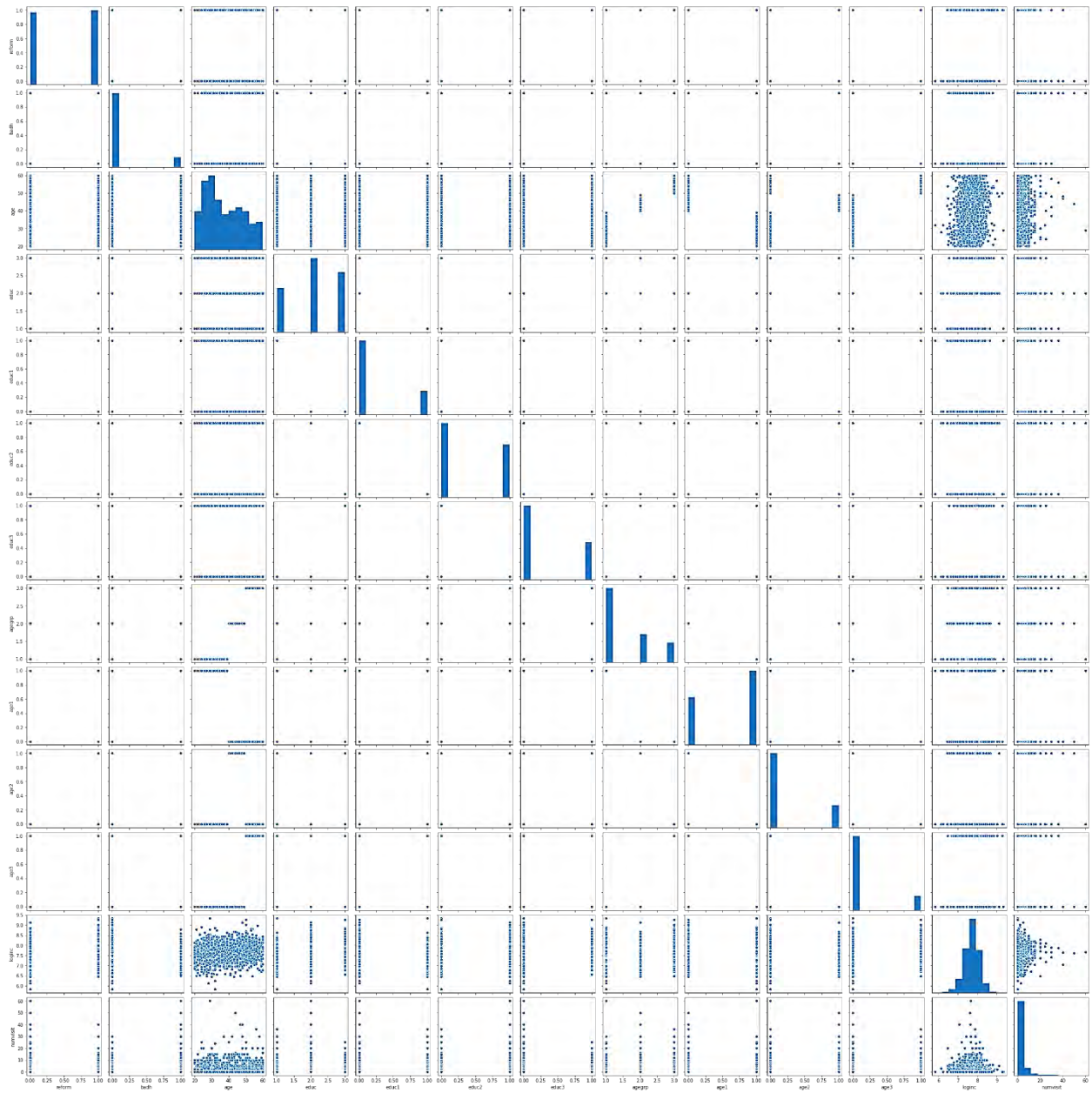


Figure 6.5: Pair plot for the German Healthcare Dataset

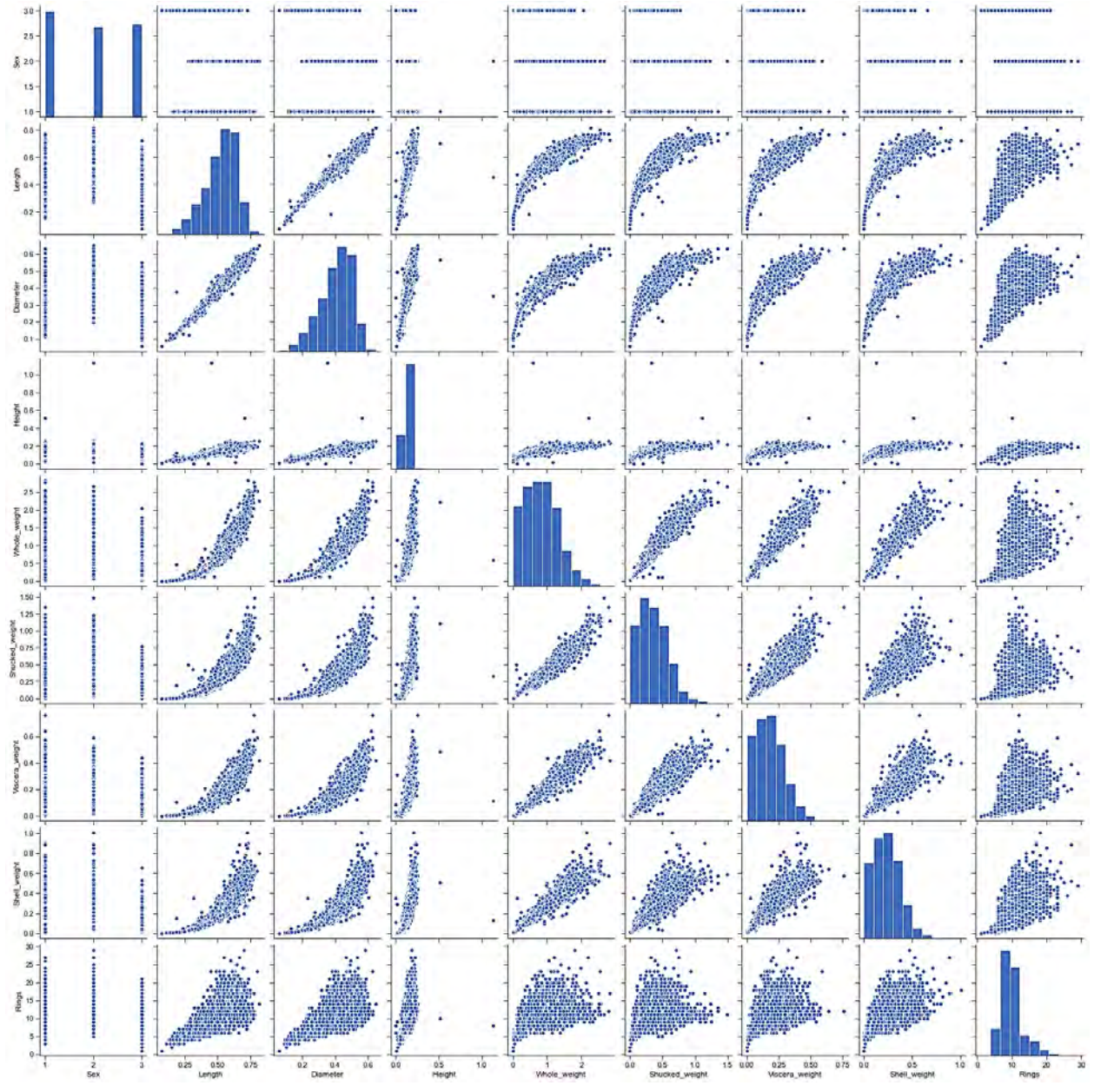


Figure 6.6: Pair plot for the Abalone dataset

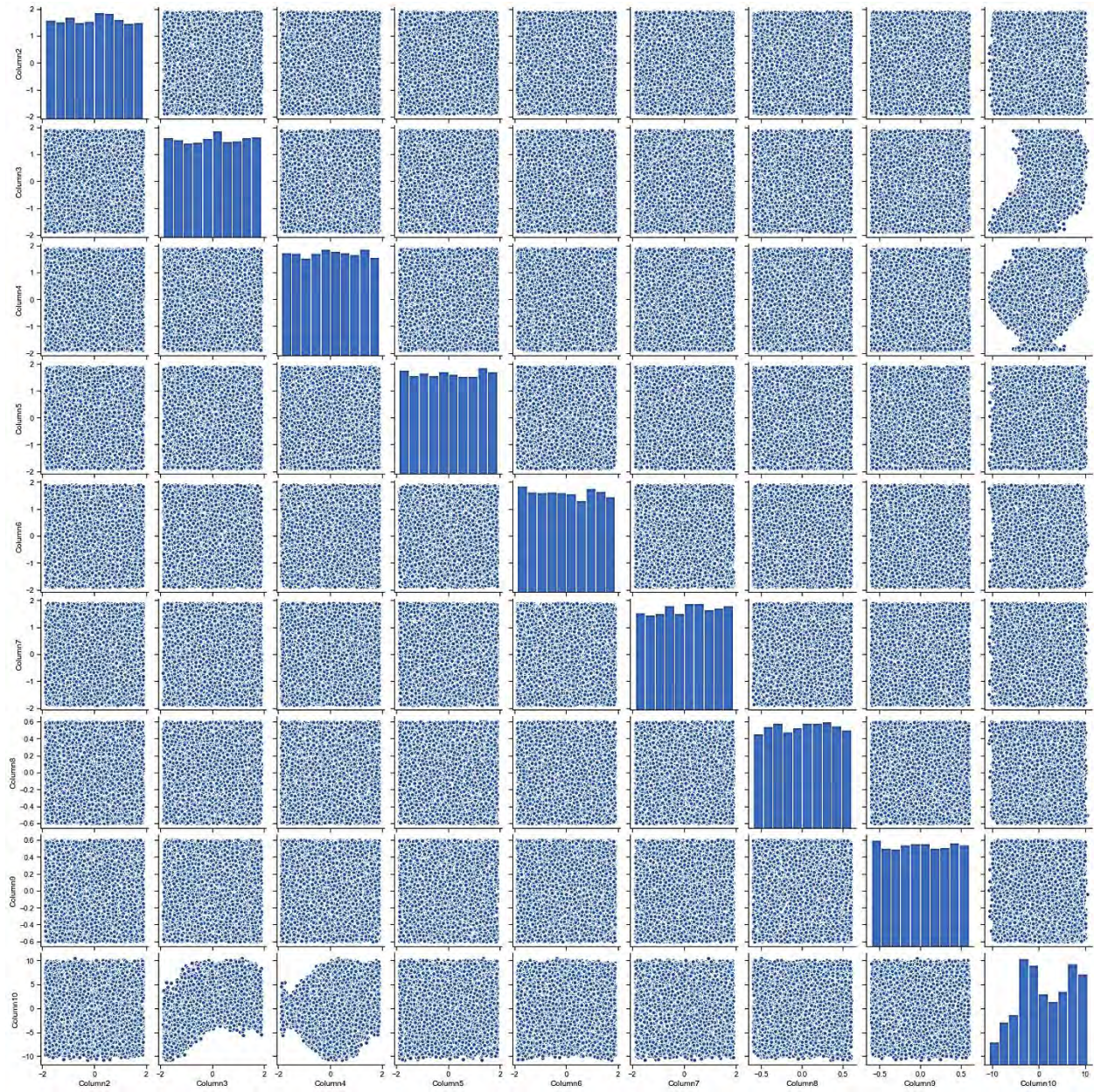


Figure 6.7: Pair plot for the Pumadyn-8nm dataset

6.3 Data Preprocessing

Data preprocessing is a crucial step in machine learning. Before proceeding to experimentations, some polishing of the datasets is often needed. Typical data preprocessing operations include data scaling or transformation, normalization, imputation of missing instances in the dataset, data reduction, extracting feature information, managing dissimilarities in feature characteristics, etc.(Kotsiantis, Kanellopoulos, & Pintelas, 2006). Despite being a major step for any machine learning routine, preprocessing is often ignored (García, Luengo, & Herrera, 2015). Lack of proper data-polishing can produce disastrous predictions (Oliveri, Malegori, Simonetti, & Casale, 2019). This processing takes various forms for various purposes. The most common preprocessing terms are discussed below:

6.3.1 Data Scaling

Data scaling or *feature scaling* is the most common form of preprocessing. It involves transforming data to fit into a certain range. The most common form of data scaling is rescaling and standardization. Rescaling, also known as *min-max normalization*, transforms feature values to have a range of [0, 1]. The following equation can be used to obtain min-max normalization:

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \quad (6.1)$$

where \mathbf{x} , \mathbf{x}' , $\min(\mathbf{x})$, and $\max(\mathbf{x})$ are the original feature vector, transformed feature vector, minimum value of \mathbf{x} , and the maximum value of \mathbf{x} , respectively.

Standardization, also called *z-score normalization*, transforms a feature such that it follows a standard normal distribution. Therefore, the transformed vector will have a zero-mean and unit variance and the range of the data points is [-1, 1]. Standardization can be achieved through the following operation:

$$\mathbf{x}' = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma} \quad (6.2)$$

where \mathbf{x} , \mathbf{x}' , $\bar{\mathbf{x}}$, and σ are the original feature vector, transformed feature vector, mean value of \mathbf{x} , and standard deviation of \mathbf{x} , respectively.

Python's *scikit-learn* library has various data scaling methods. For rescaling, the *MinMaxScaler* can be used and the *StandardScaler* can be used for standardization (Pedregosa et al., 2011). For this thesis, the *StandardScaler* module was used.

6.3.2 Missing Value Imputation

The datasets can have missing values in various instances. Missing values may appear due to mainly two reasons: either the data-point does not exist or it was not recorded. For the first reason, there is no point in trying to guess it, so it should be left as NaN (Not a Number). The second case can be handled in two ways. First, if the missing data occurrences are not substantial in number and they do not contribute much to the dataset, then these points can be dropped. The second way to handle this situation is to make intuitions about the data points. There are different ways to make these assumptions such as substituting the missing values by a constant, using the column mean (Schneider, 2001) or median, or even using regression techniques (Zhang, Qin, Zhu, Zhang, & Zhang, 2006), etc.

6.3.3 Label Encoding

Most often, a dataset can have categorical features. In regression tasks, these features cause problems. Therefore, they need to be managed first before proceeding further. Label encoding is a technique to convert these categorical features into machine-understandable numerical features. There exists an inherent problem with label encoding. When encoded, the qualitative features are given numeric values, which are generally in order (i.e., 0, 1, 2, etc.). Typically, there is no relation between the categorical features; however, after being encoded, the model can now wrongly assume a logical order among these classes. Nonetheless, it can be useful sometimes, especially when the classes of the features are in a logical order. Scikit-learn's *LabelEncoder* module can be used for label encoding (Pedregosa et al., 2011).

6.3.4 One Hot Encoding

To overcome the limitation of the label encoding method, a different technique called *one hot encoding* can be used. This is especially suitable for the case where there are several labels to a categorical feature with no logical order among them. What this method provides is, it adds a dummy column for each of the labels of the categorical features. For a certain data point, only one label from the feature will have the value of 1 and the remaining labels will be assigned a 0 value. This is analogous to “turning a switch on” whenever it falls under that particular label. In this way, the problem with ordering is resolved. However, too much incorporation of dummy columns can increase the model’s complexity. Scikit-learn has a module names *OneHotEncoder* for this purpose (Pedregosa et al., 2011).

6.4 Environment for Experimentation

For executing these experiments, a popular Python distribution called Anaconda was used. The whole programming task was performed on Jupyter Notebook. Python needs to fetch some dependencies which are known as libraries as it is a general programming language. In this task, some popular Python libraries such as *numpy*, *pandas*, *scikit-learn*, *pyclustering*, *scipy*, *matplotlib*, and *seaborn* were utilized. *Scikit-learn* has a built-in module for Gaussian process regression that implements the algorithm 2.1 from Rasmussen and Williams’s book (Rasmussen & Williams, 2006). For evaluating the datasets using the regular approach, the GPR module from this library was used. All of the experiments were carried out on a personal computer bearing the following specifications:

Table 6.2: Specification of the system used for experimentation

Processor	Intel® Core™ i5
Clock Speed	2.71 GHz
RAM	8 GB
Operating System	Microsoft Windows 10, 64-bit

6.5 Preliminary Testing for Kernel Selection

Kernels are the heart and soul of any Gaussian process. As GP is defined by a mean and a covariance or kernel function, the significance of kernels thus is self-explanatory. Also, when a zero mean function is assumed, the kernel solely defines the prior distribution (Csató & Opper, 2002). In this thesis, every dataset was scaled beforehand to fit a standard normal distribution, hence the zero-mean assumption is in force. Therefore, kernels were the sole deciding factor of the performance of the algorithm. Now, not every dataset performs well under the same kernel. Therefore, choosing the appropriate kernel for the datasets is of importance. A preliminary examination for the choice of kernels has been performed for this reason.

The following chapter presents the results of this examination. Additionally, the results of the final experimentations are also provided. The performance of the proposed algorithm has been evaluated and compared against that of the original algorithm in the next chapter.

CHAPTER-7

RESULT ANALYSIS

In this chapter, results from preliminary and final experimentation have been presented and analyzed. First, the results from the preliminary analysis have been presented in section 7.1, and based on this experiment, appropriate kernels have been chosen which is shown in section 7.2. In section 7.3, results from training using both of these algorithms have been provided in terms of training accuracy, estimated training memory requirement, and average training time. Section 7.4 yields the results on the validation sets for both these algorithms. In this section, error analysis in terms of two popular error metrics has been provided. Additionally, results on average testing memory requirement and average test time for these methods have been shown. Finally, the chapter comes to an end with an overall comparison of the two algorithms regarding memory and execution time in section 7.5.

7.1 Results of Preliminary Analysis

Preliminary testing to select the appropriate kernel function was performed. A set of five popular covariance functions were used for this initial experimentation. These kernels are exponentiated quadratic kernel, rational quadratic kernel, periodic/exponential sine squared kernel, a product of the dot product and constant kernel, and Matérn kernel. (Duvenaud, 2014) pointed out that the marginal likelihood of any kernel on the training set should determine its appropriateness. The basis of the evaluation was the training accuracy and the negative log-marginal likelihood (nMLE) of the kernel. Comparatively higher accuracy of training and a comparatively lower negative log-marginal likelihood is attractive for a kernel function to be selected. The weightage is more on the training accuracy for this selection. Table 7.1 presents detailed information on this preliminary examination of all the datasets in respect of the training accuracy and nMLE value of each kernel.

Table 7.1: Results of preliminary testing

Dataset	Method	Kernel	Training Accuracy	nMLE
Wool	Regular	Exponentiated Quadratic	0.9947261223098804	-260173295.51791286
		Rational Quadratic	0.9999999999999697	-343835230.48067156
		Exponential Sine Squared	0.9958372243131	-113242811.03245380
		Constant* DotProduct	0.5743959296394401	-2594487150.3695316
		Matérn	0.999999999999275	240.9177060911803
	Proposed	Exponentiated Quadratic	0.996839650983412	-126068167.27726403
		Rational Quadratic	0.999999999999697	-224741590.07447702
		Exponential Sine Squared	0.999999999544017	-95444590.613737055
		Constant* DotProduct	0.3929351036656833	-15113911.556702454
		Matérn	0.999999999999419	179.2544338015088
Istanbul Stock Exchange	Regular	Exponentiated Quadratic	0.999999999940168	-95.75415258597832
		Rational Quadratic	0.999999999970566	-227.00254285753056
		Exponential Sine Squared	-	-
		Constant* DotProduct	0.8488243204387557	-3.314455175570295
		Matérn	0.999999999947695	-189.7692644897769
	Proposed	Exponentiated Quadratic	0.999999999898639	203.0380248703991
		Rational Quadratic	0.999999999919486	76.62507127743311
		Exponential Sine Squared	-	-
		Constant* DotProduct	0.8675436205575137	-7806939.8534250455
		Matérn	0.999999999876777	90.834474066509841
Manaus	Regular	Exponentiated Quadratic	0.39671439574630735	-7303956474549.592
		Rational Quadratic	1.0	-3560458019785.658
		Exponential Sine Squared	0.44473451939264236	-5995321653403.729
		Constant* DotProduct	0.011722420176489723	-7752202908299.67

		Matérn	1.0	-284183643.3794945
		Exponentiated Quadratic	0.6960155371789778	-2888173340815.844
		Rational Quadratic	1.0	-1238779504719.6501
Proposed		Exponential Sine Squared	0.8242019406095086	-2390049874383.461
		Constant* DotProduct	0.02273618573122771	-3118724384497.8506
		Matérn	1.0	-39308967.95843716
		Exponentiated Quadratic	0.9809116538940152	-19634200113046.93
		Rational Quadratic	0.9826274104535657	-2534609851069.555
Regular		Exponential Sine Squared	-	-
		Constant* DotProduct	0.19851528889549797	-108511546373820.05
		Matérn	0.9810382794859024	-2522092842215.4272
German Healthcare		Exponentiated Quadratic	0.9993215409842112	-128881439757.23404
		Rational Quadratic	0.9389063484328224	-15838783721.572977
Proposed		Exponential Sine Squared	-	-
		Constant* DotProduct	0.42596712461376035	-15118057710720.252
		Matérn	0.9989296231966953	-15833399411.73557
		Exponentiated Quadratic	1.0	-16559995479.24172
		Rational Quadratic	1.0	-175874.6957
Regular		Exponential Sine Squared	-	-
		Constant* DotProduct	0.5232322008325582	-1545976207660.5745
		Matérn	1.0	-18337.88
Abalone		Exponentiated Quadratic	1.0	-433516761.1758648
		Rational Quadratic	1.0	-210456.83840256068
Proposed		Exponential Sine Squared	-	-
		Constant* DotProduct	0.624889454329584	-25422867355911.457

	Matérn	1.0	-64523.73221023212
	Exponentiated Quadratic	0.04055850104392911	-2411205152383.373
	Rational Quadratic	1.0	-2335117005120.7417
Regular	Exponential Sine Squared	0.03994536322066167	-2400698008094.963
	Constant* DotProduct	0.0010549904101812002	-2427775021096.2285
	Matérn	0.9999999999999981	-256285016571.1286
Tree Ring	Exponentiated Quadratic	0.08834587671169558	-1087830373675.23
	Rational Quadratic	1.0	-1030742240704.2623
Proposed	Exponential Sine Squared	0.0520059047482767	-1073255303213.8119
	Constant* DotProduct	0.0025867544671799303	-1098826219129.2125
	Matérn	1.0	-20505844013.088062
	Exponentiated Quadratic	1.0	-25482.375788
	Rational Quadratic	1.0	-47245.225491965924
Regular	Exponential Sine Squared	-	-
	Constant* DotProduct	0.5787774785820672	-316300472119932.2
	Matérn	1.0	-18794.66264085393
Pumadyn- 8nm	Exponentiated Quadratic	1.0	-13572.801468487683
	Rational Quadratic	1.0	-31414.497920440943
Proposed	Exponential Sine Squared	-	-
	Constant* DotProduct	0.5716168735526539	-161238631976435.38
	Matérn	1.0	-11969.736093931422

**Note: The blank space means the kernel did not return a positive semi-definite kernel matrix.

7.2 Selected Kernels for Datasets using Basic GPR and *si*-GPR

Based on the training accuracy and the negative log marginal likelihood, the appropriate kernel for each dataset has been chosen. The results are displayed in Table 7.2.

Table 7.2: Chosen kernels for basic GPR and *si*-GPR

Dataset	Chosen Kernel	
	Basic GPR	<i>si</i> -GPR
Wool	Rational Quadratic	Rational Quadratic
Istanbul Stock Exchange	Rational Quadratic	Rational Quadratic
Manaus	Rational Quadratic	Rational Quadratic
German Healthcare	Rational Quadratic	Exponentiated Quadratic
Abalone	Exponentiated Quadratic	Exponentiated Quadratic
Tree Ring	Rational Quadratic	Matérn
Pumadyn-8nm	Rational Quadratic	Rational Quadratic

7.3 Results from Experimentations on Training Sets

In this section, the experimental outcomes for the training sets using both methodologies have been provided. First, section 7.3.1 shows a comparison of training points for both of the methodologies used. Section 7.3.2 summarizes the result for the basic GPR on training sets. The results are presented in terms of training accuracy, estimated memory requirement, and average training time. Section 7.3.3 also presents the same findings for the proposed algorithm. Section 7.3.4 presents a visual comparison of these two methods based on training accuracy, anticipated memory requirement, and mean training time.

7.3.1 Comparison of Training Points used in Basic GPR and *si*-GPR

The total number of training points used in the original and proposed GPR does not match due to using a sparsification operation. There is a significant reduction in the number of training points for the *si*-GPR algorithm. Table 7.3 provides that change concerning the training points used in both methods and the percentage reduction of training points for each dataset.

Table 7.3: Comparison of the number of training points for basic GPR and *si*-GPR

Dataset	Basic GPR	<i>si</i> -GPR	%Reduction in Data Points
Wool	200	110	45.00%
Istanbul Stock Exchange	300	156	48.00%
Manaus	700	360	48.57%
German Healthcare	1500	756	49.60%
Abalone	3000	1507	49.77%
Tree Ring	5000	2510	49.80%
Pumadyn-8nm	5000	2506	49.88%

Figure 7.1 shows a graphical comparison of the training points for both methods. In this figure, the number of training points has been plotted against the datasets. The datasets were arranged in the ascending order of their training points. The last two datasets, the *Tree Ring* dataset, and the *Pumadyn-8nm* dataset, both have the same number of training points. When the data clustering algorithm was applied to all these datasets as part of the proposed algorithm, it could offer up to about a 50% reduction in the number of training points. This helps to bring down the memory usage or the training time by a good amount, which is shown later.

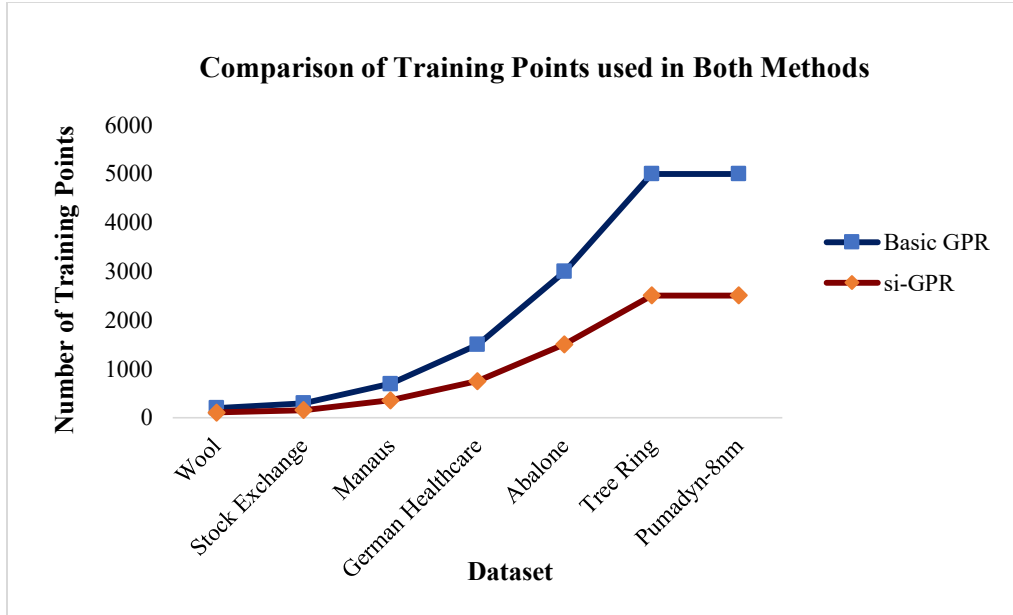


Figure 7.1: Comparison of training points for both methods

7.3.2 Results of Basic GPR on Training Sets

In Table 7.4, the results of the training sets for the basic GPR are presented.

Table 7.4: Results obtained for training sets using basic GPR

Dataset	Training Accuracy	Estimated Memory Requirement (MB)	Average Training Time (s)
Wool	0.99999999999999697	2.44140625	5.3
Istanbul Stock Exchange	0.9999999999970566	5.493164063	11.3
Manaus	1.0	29.90722656	93
German Healthcare	0.982627410453565	137.3291016	804
Abalone	1.0	549.3164063	2589
Tree Ring	1.0	1525.878906	6576
Pumadyn-8nm	1.0	1525.878906	8125

7.3.3 Results of *si*-GPR on Training Sets

Table 7.5 provides the results of the training sets for *si*-GPR.

Table 7.5: Results obtained for training sets using *si*-GPR

Dataset	Training Accuracy	Estimated Memory Requirement (MB)	Average Training Time (s)
Wool	0.99999999999999697	0.738525391	2.71
Istanbul Stock Exchange	0.99999999999919486	1.485351563	3.5
Manaus	1.0	7.91015625	32.6
German Healthcare	0.999321540984211	34.88378906	117
Abalone	1.0	138.6138306	492
Tree Ring	1.0	384.5275879	1146
Pumadyn-8nm	1.0	383.3029785	1729

In the following section, a comparison of the two approaches regarding the results of training sets is shown graphically.

7.3.4 Visual Comparison of Basic GRR and *si*-GPR on Training Sets

Figure 7.2 presents the comparison of the training accuracy of both original and proposed GPR, where the training accuracy has been plotted against the datasets. Generally, the performance of regression algorithms depends on the type of features, presence of missing values in the dataset, amount of noise in the system, etc.(Li et al., 2019). Some datasets even require specific kinds of scaling to offer good performance. Therefore, it is justified to say that fitting accuracy is heavily influenced by the characteristics of the datasets. It can be seen from Figure 7.2 that the training accuracy is almost perfect for the basic algorithm for every dataset except the *German Healthcare* dataset. The reason for the slightly lesser accuracy can be attributed to having more than one categorical feature in the set which affects the training accuracy directly. However, the proposed algorithm solves this issue for this dataset and provides consistent behavior for all the

datasets. This further proves that the proposed *si*-GPR algorithm handles categorical features better than the original algorithm.

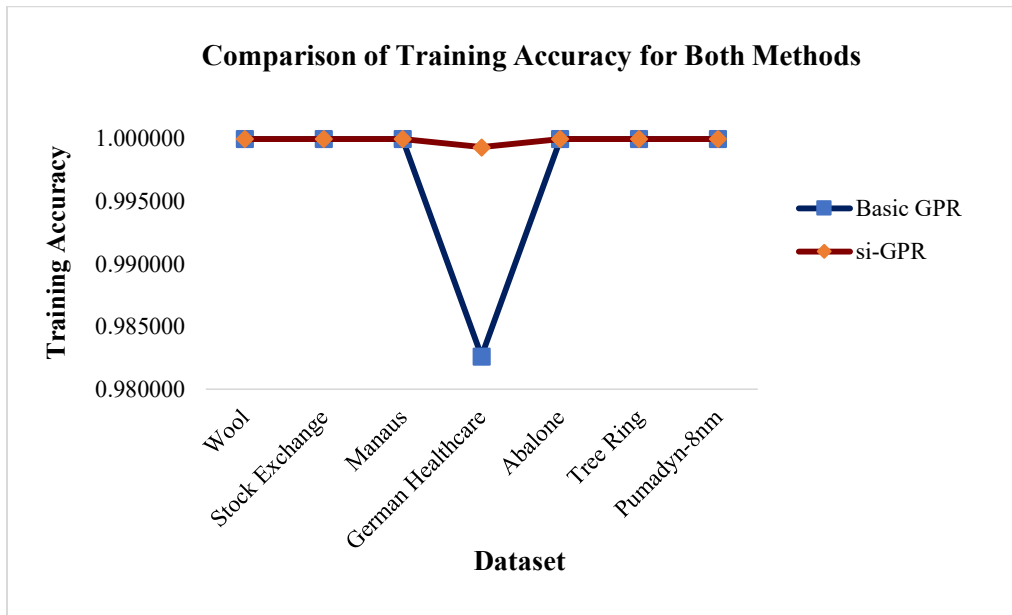


Figure 7.2: Comparison of training accuracy of basic and proposed GPR

Figure 7.3 provides a comparison of memory consumption for training. It needs to be justified that the memory calculation was approximated and not exact because it was not possible to extract the true memory usage information from the system. The memory usage by the system is not static, i.e., the computer allocates variable memory as many times as the experiments are executed. Intuitively it makes sense that the more training points there are, the more system memory will be needed. That is why the memory requirement was calculated based on the operations and the object sizes to store the information in Python. Again, the datasets have been arranged in the ascending order of their training points. As a result, the plot has an increasing trend as it goes forward in the right direction of the horizontal axis. The last two datasets, i.e., *Tree Ring* and *Pumadyn-8mm* datasets also require the same amount of memory to train due to having the same number of training points. As for the *si*-GPR algorithm, the memory requirement also shows an increasing trend due to the order of the datasets; however, the memory requirements are way less than the basic GPR. One possible reason for the lesser

memory requirement can be attributed to the reduction in the number of training points. Additionally, the incremental update on the kernel matrix speeds up the proposed algorithm and requires much less computational power. The last two datasets, again, are reduced to the same number of data points and hence require the same memory for the *si*-GPR algorithm as well.

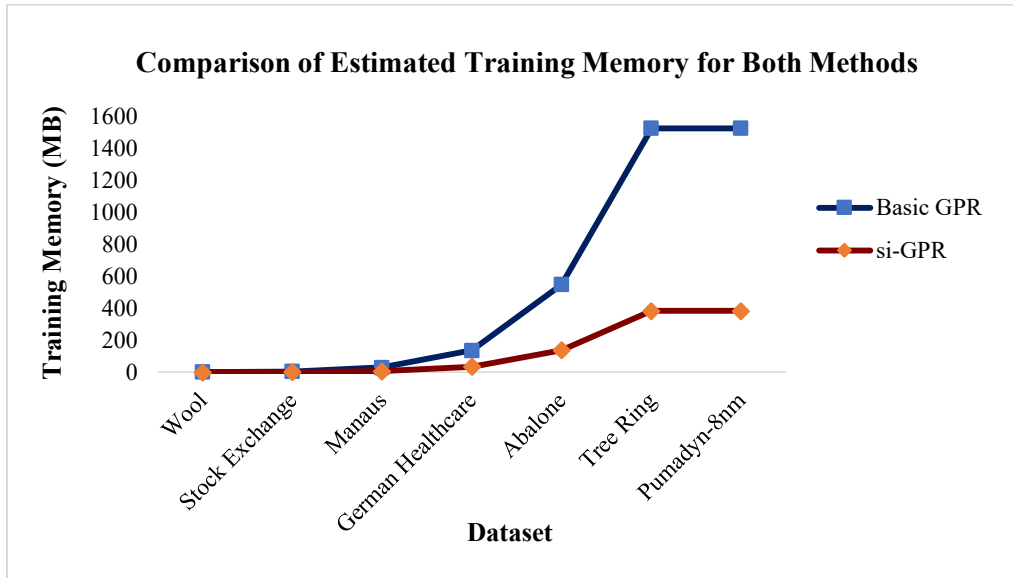


Figure 7.3: Comparison of estimated memory requirement for both methods

In Figure 7.4, the comparison of training time of the basic and proposed GPR has been shown. In this plot, the mean training time is plotted against the datasets. Training time is directly related to the number of training points, the complexity in data characteristics, and the presence of categorical features. As the datasets increase in the number of training points from left to right, the training time also increases accordingly. It is to be noted that the *Tree Ring* and the *Pumadyn-8nm* datasets have the same number of training points but they differ in the amount of time needed to train the model. The reason is that the *Pumadyn-8nm* dataset is a nine-dimensional dataset and the *Tree Ring* dataset is a two-dimensional dataset. Due to the additional features, the complexity of the *Pumadyn-8nm* dataset increases, which directly attributes to an increase in the training time. A similar trend can be seen for *si*-GPR; however, the training time is reduced by a good amount. For example, to train a model of 5000 training points and 9

features, the basic GPR takes about 9000 seconds, whereas this is about one-ninth of that time for the *si*-GPR algorithm. This reduction in training time was due to the reduction of data points and incremental learning methodology.

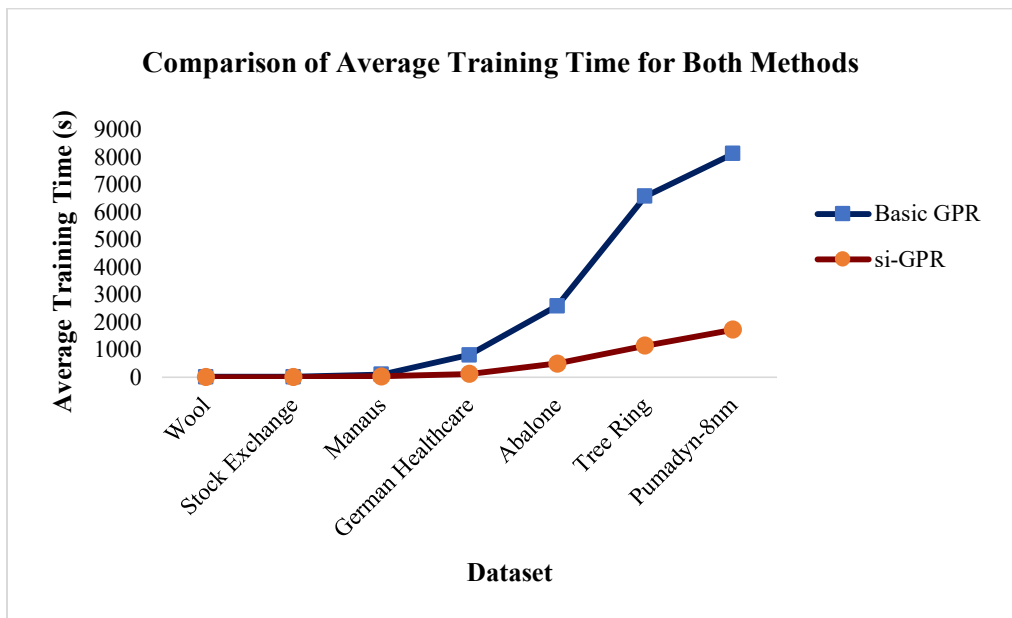


Figure 7.4: Comparison of average training time for both methods

It is evident from Figures 7.2, 7.3, and 7.4 that the performance of the proposed algorithm over training datasets is superior to that of the basic algorithm whether it is training accuracy, memory requirement (estimated), or the execution time.

7.4 Results from Experimentation on Test Sets

For validating the test sets, the mean absolute error (MAE) and the root mean squared error (RMSE) metrics were used. In section 7.4.1, the results of test sets using basic GPR is presented. Similarly, section 7.4.2 presents the results for the proposed algorithm. Section 7.4.3 offers a graphical comparison of the predictions made using both algorithms. Additionally, the

comparison regarding MAE score, RMSE score, estimated testing memory, and average testing time can also be found.

7.4.1 Results of Basic GPR on Test Sets

Table 7.6 gives the results of the test datasets for basic GPR. In this table, the error scores, the required memory, and the mean time for validation can be observed.

Table 7.6: Results obtained for test sets using basic GPR

Dataset	MAE Score	RMSE Score	Estimated Memory Requirement (MB)	Average Run Time (s)
Wool	0.1141	0.1281	3.784179688	0.82
Istanbul Stock Exchange	0.0045	0.0059	9.851074219	1.2
Manaus	1.6550	2.1433	46.22802734	3
German Healthcare	0.1403	0.5323	204.0710449	43
Abalone	2.0078	2.7839	765.0146484	253
Tree Ring	0.2427	0.3136	2435.913086	1096
Pumadyn-8nm	0.9503	1.2258	2500.305176	2102

7.4.2 Results of *si*-GPR on Test Sets

The experimental outcome on the test datasets using the *si*-GPR algorithm can be seen in Table 7.7.

Table 7.7: Results obtained for test sets using *si*-GPR

Dataset	MAE Score	RMSE Score	Estimated Memory Requirement (MB)	Average Run Time (s)
Wool	0.0819	0.1093	1.477050781	0.082
Istanbul Stock Exchange	0.0035	0.0047	3.751464844	0.19
Manaus	1.5215	1.9356	16.30371094	1.8
German Healthcare	0.1577	0.5396	68.52172852	30
Abalone	1.8191	2.5136	246.9662476	93

Tree Ring	0.2355	0.3073	841.3647461	133
Pumadyn-8nm	0.0853	0.6219	871.6854248	151

The following section provides some visualization of the result on some of the test sets. As mentioned in Chapter 6, the visualization is not always possible due to high data dimensionality, the results from only the two-dimensional datasets could be represented. It should be noted that the prediction accuracy depends on the factors such as fitting accuracy, data complexity, choice of kernel functions, presence of class variables, etc. That is why the %reduction of error for the *si*-GPR varies from dataset to dataset. In addition to providing the visualization of predictions, the MAE and RMSE scores have been plotted against the test sets. Additionally, memory and time comparisons are also shown.

7.4.3 Visual Comparison of Basic GRR and *si*-GPR on Test Sets

At the beginning of this section, side-by-side comparisons of the results from the two-dimensional datasets (i.e., *Wool* dataset, *Manaus* dataset, and *Tree Ring* dataset) have been presented graphically in Figures 7.5 through 7.10.

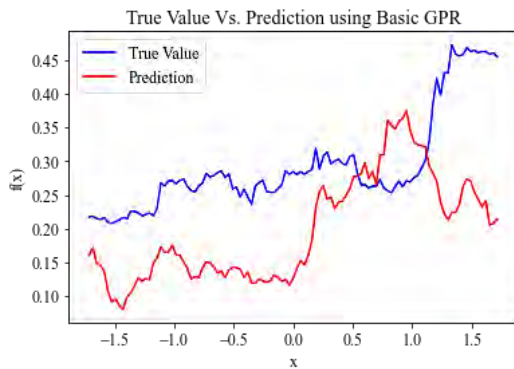


Figure 7.5: Predictions on the Wool test set using basic GPR

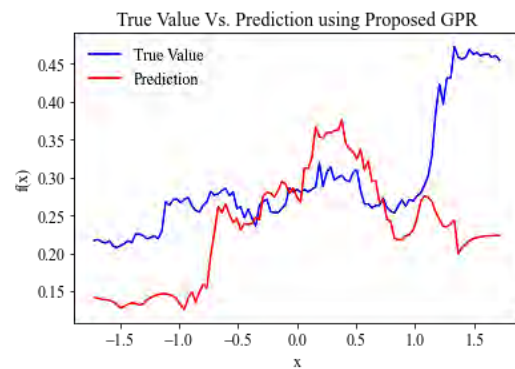


Figure 7.6: Predictions on the Wool test set using proposed GPR

Figures 7.5 and 7.6 present the predictions on the *Wool* dataset for basic GPR and *si*-GPR, respectively. From Tables 7.6 and 7.7, it can be seen that *si*-GPR offers 28.22% less MAE and 14.68% less RMSE than the original algorithm. Although the error is reduced, deviations in certain regions still appear to be large. The reason for this deviation could be model overfitting. As mentioned in Chapter 2, GPR tends to overfit the training data. The proposed algorithm might not have solved the overfitting issue fully for this dataset; however, it treats the overfitting better than the original algorithm.

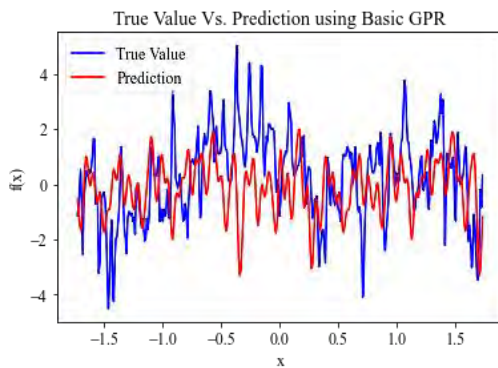


Figure 7.7: Predictions on the Manaus test set using basic GPR

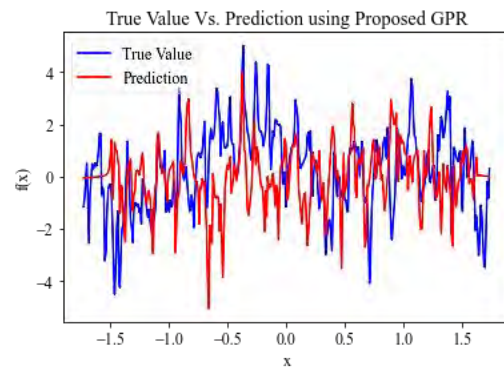


Figure 7.8: Predictions on the Manaus test set using proposed GPR

Figures 7.7 and 7.8 provide the prediction visualizations on the Manaus dataset. These models were not overfitted, and the *si*-GPR again exceeds the basic GPR in terms of prediction accuracy. The MAE score for this dataset has been reduced by 8.1% and the RMSE score has been decreased by 9.7% due to using *si*-GPR.

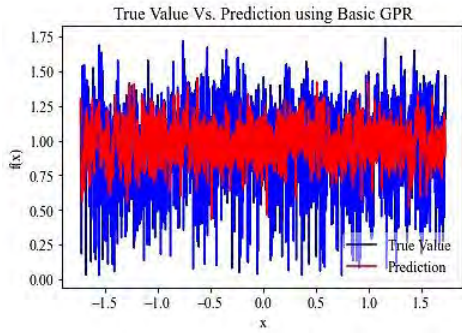


Figure 7.9: Predictions on the Tree Ring test set using basic GPR

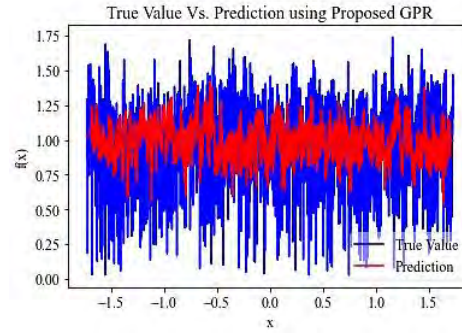


Figure 7.10: Predictions on the Tree Ring test set using proposed GPR

Figures 7.9 and 7.10 present the predictions on the *Tree Ring* dataset. In this dataset, the variations for the predictions of *si-GPR* are lesser than that of the basic GPR. The *si-GPR* offers a 3% reduction in MAE score and a 2% reduction in the RMSE score. For this dataset, the performance of the basic algorithm is close to the *si-GPR* algorithm. The reason is that this is a two-dimensional dataset with no major complexity. There is no missing value or any class variable in this dataset. Still, the *si-GPR* can offer predictions with lesser error.

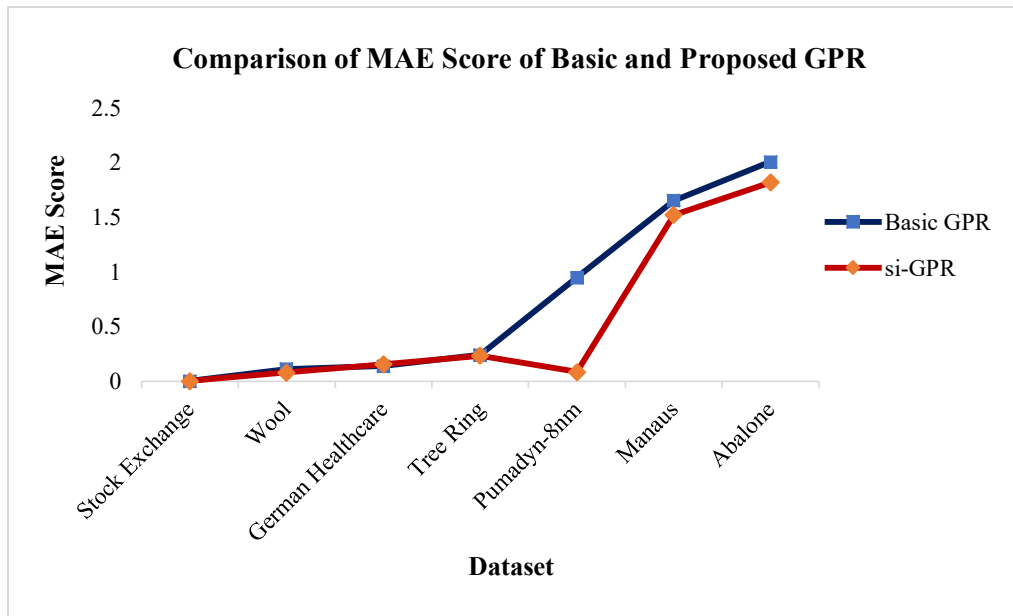


Figure 7.11: Comparison of Mean Absolute Error (MAE) for both methods

Figure 7.11 presents the contrast of the algorithms concerning the MAE metric. From this figure, it can be seen for the first four datasets, the performances of the two algorithms are almost the same. For the *Pumadyn-8nm* dataset, this difference becomes quite apparent. This dataset is nine-dimensional and basic GPR suffers in performance from higher data dimensionality. On the other hand, *si*-GPR could fit this dataset well in training as well as in testing. For the rest of the datasets, there is a smaller difference in the performance of the algorithms.

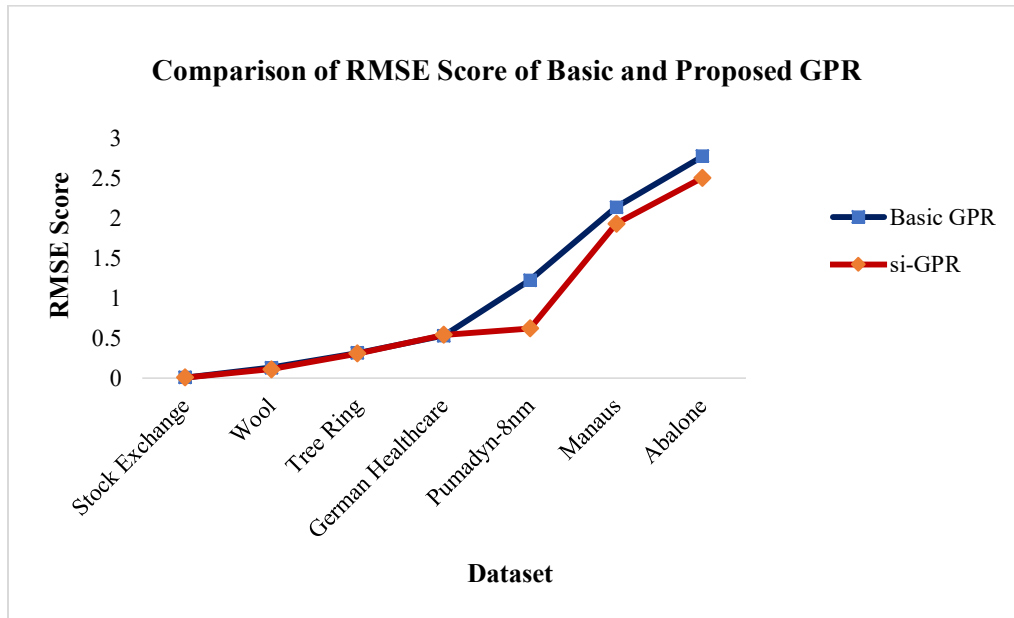


Figure 7.12: Comparison of Root Mean Squared Error (RMSE) for both methods

The RMSE score comparison can be seen in Figure 7.12. Generally, RMSE scores will always be equal to or greater than the MAE scores due to the triangular inequality (Chai & Draxler, 2014), which is reflected in the results. In this case, the same trend can be seen as the MAE scores, except that the difference between the RMSE scores regarding the *Pumadyn-8nm* dataset is slightly less than the MAE scores.

Figures 7.13 and 7.14 provide visualization on the estimated testing memory and mean test time for basic GPR and *si*-GPR. In equation (3.49) in Chapter 3, it was stated that the computation calls for inverting the kernel matrix acquired from the training points while making predictions.

For a larger set of training points, the shape of the covariance matrix and the inverted covariance matrix is also large, hence requiring more memory consumption and more time for execution. As the new version allows lesser training points, all computational efforts are reduced in addition to providing better performance. Additionally, due to employing the incrementally learning and decrementally unlearning ranked data points, the joint covariance matrix of the training points and the test points for the *si*-GPR is always smaller in size than the original GPR algorithm.

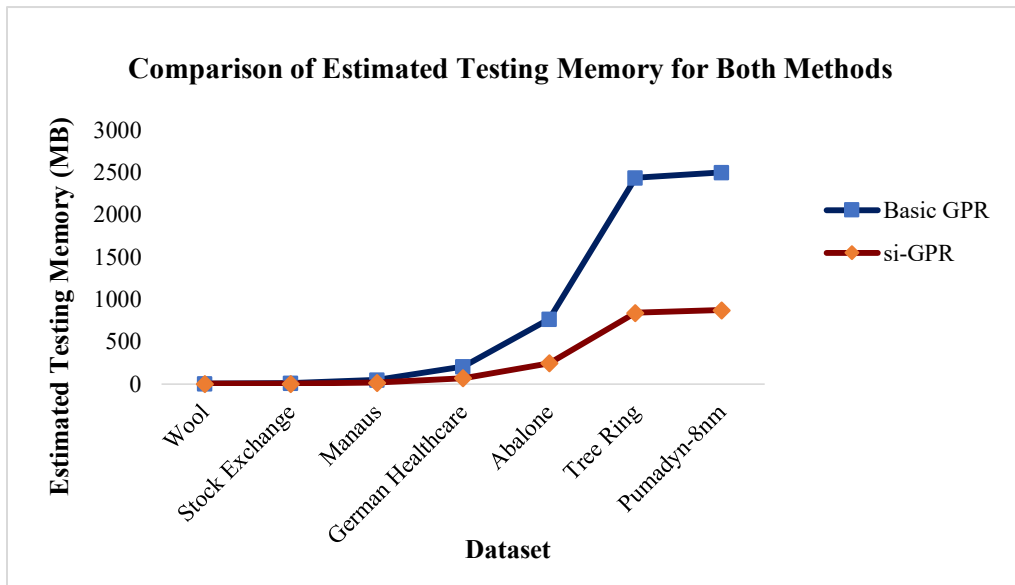


Figure 7.13: Comparison of estimated testing memory requirement for both methods

Figure 7.13 shows the same trend as the training set due to similar reasons. The testing also requires considering the training set as the joint distribution is needed for making predictions. That is why the incorporation of the training set makes the trend remain the same.

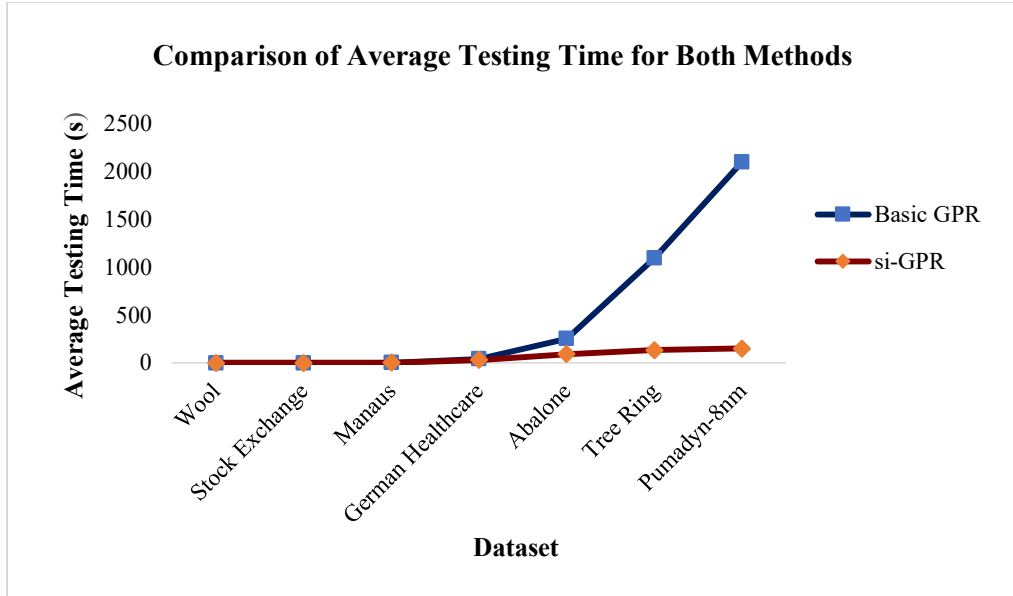


Figure 7.14: Comparison of average testing time for both methods

Figure 7.14 compares the mean time needed to validate the models. Basic GPR in this case works with the whole training set plus the test set, whereas *si-GPR* deals with the sparse training set plus the test set. There is a noticeable reduction in the total number of points these two algorithms are handling. It is evident from the figure that the proposed algorithm offers significant savings in test times. For instance, for the same test set of about 3000 points for the *Pumadyn-8nm* dataset, basic GPR takes 2102 seconds on average, where *si-GPR* takes only 151 seconds, promising about 93% savings.

7.5 Overall Memory and Time Requirements

Results on training datasets and test datasets have been shown separately in the previous sections. The proposed modifications have offered promising results over each of the sections. In this segment, a general overview has been presented. At first, an overall comparison regarding total memory consumption has been shown in Table 7.8.

Table 7.8: Comparison of total memory requirement for basic GPR and *si*-GPR

Dataset	Total Memory Requirement (MB)		% Memory Savings
	Basic GPR	<i>si</i> -GPR	
Wool	6.225585938	2.215576172	64.41%
Istanbul Stock Exchange	15.34423828	5.236816407	65.87%
Manaus	76.13525390	24.21386719	68.20%
German Healthcare	341.4001465	103.4055176	69.71%
Abalone	1314.331055	385.5800782	70.66%
Tree Ring	3961.791992	1119.545465	71.74%
Pumadyn-8nm	4325.155456	1175.156216	72.83%

The results of overall memory requirements have been shown graphically in Figure 7.15. In this case, both training and testing memory requirements have been taken into consideration. The *si*-GPR algorithm has been efficient in both training memory and testing memory usage. Hence, the trend also continues for the total memory requirements.

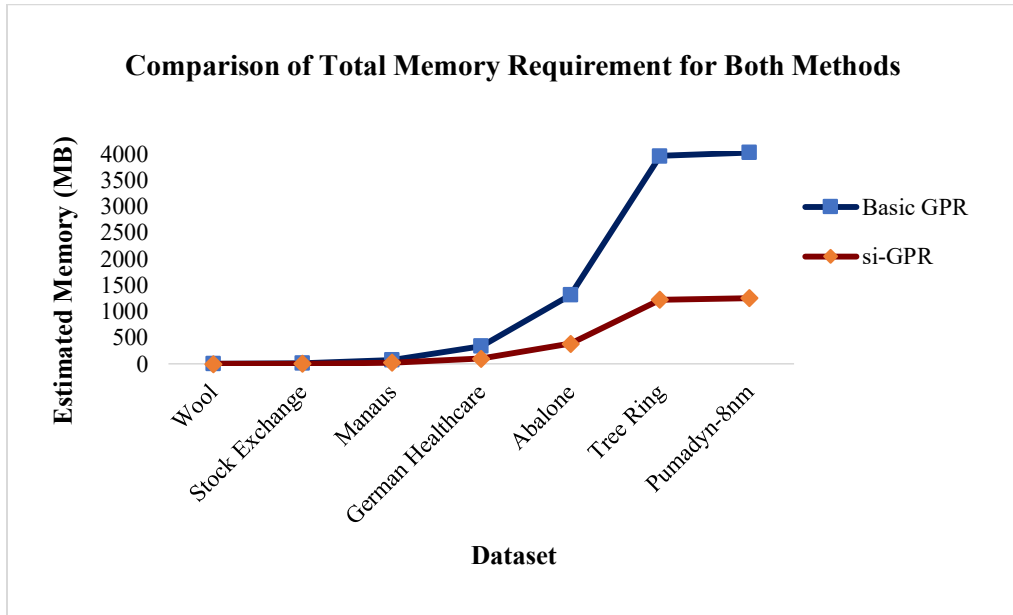


Figure 7.15: Comparison of estimated total memory requirement for both methods

Similarly, the average total execution time information is provided in Table 7.9, which is plotted in Figure 7.16.

Table 7.9: Comparison of average total run time for basic GPR and *si*-GPR

Dataset	Average Total Run Time (s)		%Time Savings
	Basic GPR	<i>si</i> -GPR	
Wool	6.12	2.792	54.38%
Istanbul Stock Exchange	12.5	4.5	64.00%
Manaus	114	34.4	69.82%
German Healthcare	769	163	78.80%
Abalone	2842	585	79.42%
Tree Ring	7672	1396	81.80%
Pumadyn-8nm	10227	1760	82.79%

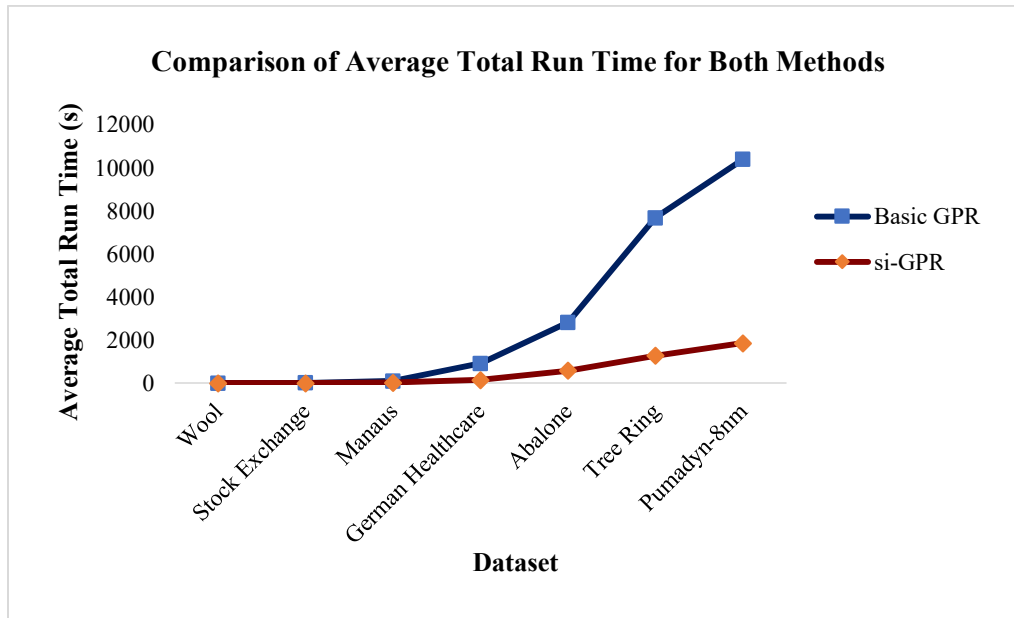


Figure 7.16: Comparison of average total run time for both methods

Figure 7.16 gives a comparison of the mean total run time of both algorithms. Similar to memory usage, *si*-GPR has used less training time and testing time than the basic GPR algorithm. As a result, the trend of time complexity also follows the same as the training and testing times.

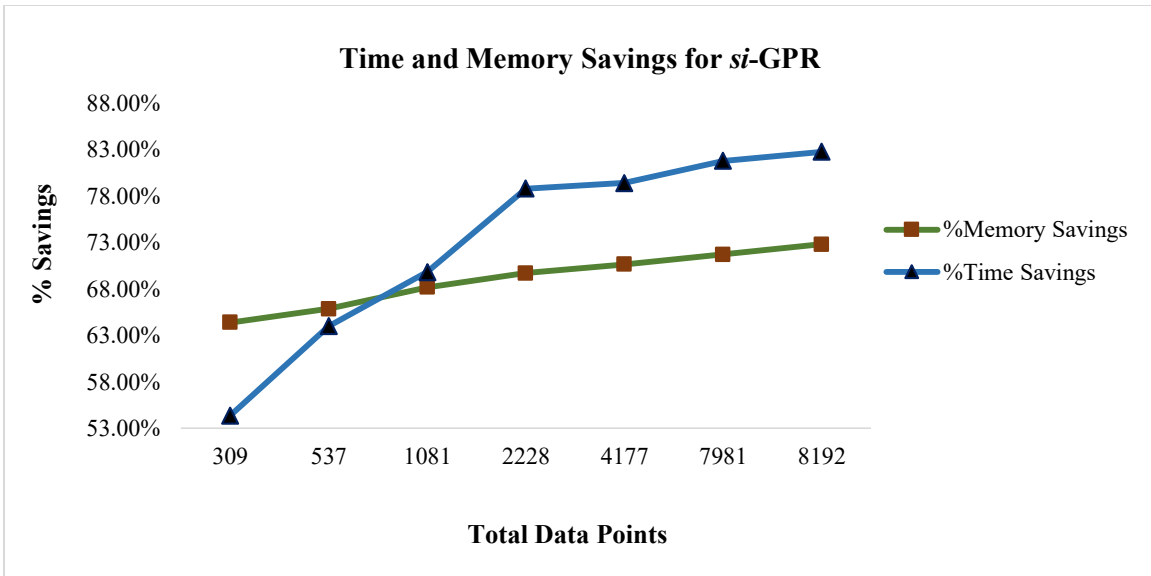


Figure 7.17: Memory and time savings using *si-GPR*

Figure 7.17 presents the savings in terms of memory consumption and total run time. It can be observed that the *si-GPR* offers about 64% to 73% lesser memory usage for the datasets used. Also, the savings regarding the training and testing ranges from about 54% to 83%. It is also clear from the plot that as the data points increase, the *si-GPR* becomes more efficient and the performance difference with the basic GPR becomes more apparent.

In this chapter, the results from preliminary and final examination on the datasets using the basic GPR and *si-GPR* were presented. The results suffice that the *si-GPR* algorithm is better than the original algorithm in all aspects. The next chapter concludes the thesis.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

8.1 Concluding Remarks

The objective of this thesis was to present a Gaussian process regression methodology based on incremental learning. The problem of managing big datasets is ever increasing. The original GPR, being not economic with big datasets, called for a remedy to this problem. Also, traditional GPR is incapable of managing feedback-based datasets or training sets with variable sizes. Furthermore, the basic GPR fails to adjust to the resource limitation issue. Therefore, these problems were the focusing points of this thesis. Up to this point, a considerable quantity of research involving the sparse approximation of the problem has surfaced. Some other works have reported the use of online learning as well. This thesis offered an algorithm called sparse-incremental Gaussian process regression (*si*-GPR) that merged the sparse implementation as well as incremental learning dynamic. The goal of *si*-GPR was to address the limitations of basic GPR in the context of big datasets, streaming training sets, and resource limitations.

When the motivation was decreasing the calculational burden, a natural solution came to mind that involves lowering the number of data instances that the algorithm should fit. It is relatable that if every datapoint is considered, the computations are going to be lengthy and strenuous. Hence, the idea of a sparse representation emerged and implementing that vision together with the idea of sequentially augmenting the kernel matrix rather than computing it at once provided a computational advantage. It is mention-worthy that even after employing a sparse presentation using the representative dataset, the accuracy was not the least compromised, rather the modified version of the algorithm provided better results overall, which had also been a prime concern. In addition to a static training set, the thesis also outlined the method of handling variable training set size problems (streaming input and removing data points from the training set). To keep computations further on the leash, another scheme for un-learning was offered. The principal idea was to keep the kernel matrix fixed in a certain size so that it does not become costly to invert the matrix. To implement this, the algorithm began to ‘un-learn’ starting from the first

point when the kernel matrix surpassed the prespecified size. Keeping the kernel matrix at a certain size was also helpful to keep the computations tractable.

The basic GPR and the proposed *si*-GPR were put to test using seven popular machine-learning datasets. Before that, a short preprocessing and experimentations for kernel selection were performed. The results of final testing showed that the proposed algorithm provides better performance in every aspect relative to the original GPR algorithm.

8.2 Avenues for Future Research

There have been some limitations regarding the scope and resources as stated earlier in Chapter 1. Hence, some directions to venture in the future have been proposed. For example:

- The data clustering method that was used could be changed to assess if any other clustering algorithm provides better performance.
- It would be interesting to vary the selection of kernels. In the experiments, a set of fixed kernels were used. This selection can be augmented and their effect can be evaluated.
- This thesis worked on the single-output regression problem. It would be compelling to extend this algorithm for the multivariate output problem.
- The research can be extended to even bigger datasets, which could not be implemented due to resource limitations.
- GPR has previously been used for classification problems as well. It would be nice to extend the *si*-GPR algorithm for the classification problem as well.
- A comparison with other variants of sparse Gaussian process regression would be of significance.

Hopefully, these ideas will be considered in subsequent research.

REFERENCES

- Abramowitz, M., & Stegun, I. A. (1965). *Handbook of Mathematical Functions*. Dover Publications, New York, USA.
- Ade, R. R., & Deshmukh, P. R. (2013). Methods for Incremental Learning: A Survey. *International Journal of Data Mining & Knowledge Management Process*, 3(4), 119.
- Akbilgic, O., Bozdogan, H., & Balaban, M. E. (2014). A novel Hybrid RBF Neural Networks model as a forecaster. *Statistics and Computing*, 24(3), 365–375.
- Alpaydin, E. (2004). *Introduction to Machine Learning*. MIT Press.
- Aytug, H., Bhattacharyya, S., Koehler, G. J., & Snowdon, J. L. (1994). A review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, 41(2), 165–171.
- Bai, X., Ren, P., Zhang, H., & Zhou, J. (2015). An incremental structured part model for object recognition. *Neurocomputing*, 154, 189–199.
- Baker, C. T. (1977). *The Numerical Treatment of Integral Equations*.
- Bauer, M., van der Wilk, M., & Rasmussen, C. E. (2016). Understanding Probabilistic Sparse Gaussian Process Approximations. *Advances in Neural Information Processing Systems*, 1533–1541.
- Bernardo, J., Berger, J., Dawid, A., & Smith, A. (1998). Regression and Classification using Gaussian Process Priors. *Bayesian Statistics 6*, 475.
- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Bottou, L., & Cun, Y. (2003). Large scale online learning. *Advances in Neural Information Processing Systems*, 16, 217–224.
- Cao, Y., & Yang, J. (2015). Towards making systems forget with machine unlearning. *2015 IEEE Symposium on Security and Privacy*, 463–480. IEEE.
- Carbonara, L., & Borrowman, A. (1998). A comparison of batch and incremental supervised learning algorithms. *European Symposium on Principles of Data Mining and Knowledge Discovery*, 264–272.
- Carbonneau, R., Laframboise, K., & Vahidov, R. (2008). Application of machine learning techniques for supply chain demand forecasting. *European Journal of Operational Research*, 184(3), 1140–1154.
- Casella, G., & Berger, R. L. (2002). *Statistical inference*. Pacific Grove, CA: Duxbury.
- Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, 7(3), 1247–1250.
- Chapra, S. C., & Canale, R. P. (2010). *Numerical methods for engineers*. Boston: McGraw-Hill

Higher Education.

- Chatterjee, A., Croley, D., Ramamurti, V., & Chang, K. Y. (1997). Application of machine learning to manufacturing: results from metal etch. *Nineteenth IEEE/CPMT International Electronics Manufacturing Technology Symposium*, 372–377. IEEE.
- Chen, T., Morris, J., & Martin, E. (2007). Gaussian process regression for multivariate spectroscopic calibration. *Chemometrics and Intelligent Laboratory Systems*, 87, 59–71. <https://doi.org/10.1016/j.chemolab.2006.09.004>
- Cheng, C.-A., & Boots, B. (2016). Incremental Variational Sparse Gaussian Process Regression. *Advances in Neural Information Processing Systems*, 4410–4418.
- Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (1990). *Introduction to Algorithms*. MIT Press.
- Cornell, J. A., & Berger, R. D. (1987). Factors that influence the value of the coefficient of determination in simple linear and nonlinear regression models. *Phytopathology*, 77(1), 63–70.
- Csató, L., & Opper, M. (2002). Sparse On-Line Gaussian Processes. *Neural Computation*, 14(3), 641–668.
- De La Cruz, J. S., Owen, W., & Kulic, D. (2012). Online Learning of Inverse Dynamics via Gaussian Process Regression. *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.
- Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1), 1–22.
- Dewan, M. A. A., Granger, E., Marcialis, G. L., Sabourin, R., & Roli, F. (2016). Adaptive appearance model tracking for still-to-video face recognition. *Pattern Recognition*, 49, 129–151.
- Dhanjal, C., Gaudel, R., & Cléménçon, S. (2014). Efficient eigen-updating for spectral graph clustering. *Neurocomputing*, 131, 440–452.
- Diggle, P. J. (1990). *Time Series: A Biostatistical Introduction (No. 04; QA280, D5.)*.
- Ditzler, G., Roveri, M., Alippi, C., & Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4), 12–25.
- Draper, N. R., & Smith, H. (1998). *Applied regression analysis*. John Wiley & Sons.
- Dunjko, V., Taylor, J. M., & Briegel, H. J. (2016). Quantum-enhanced Machine Learning. *Physical Review Letters*, 117(13).
- Duvenaud, D. (2014). *Automatic Model Construction with Gaussian Processes*. University of Cambridge.
- Emerson, J. W., Green, W. A., Schloerke, B., Crowley, J., Cook, D., Hofmann, H., & Wickham, H. (2013). The generalized pairs plot. *Journal of Computational and Graphical Statistics*, 22(1), 79–91.

- Fattahi, S. (2011). A Comparative Study of Parametric and Nonparametric Regressions. *Iranian Economic Review*, 16(30), 19–43.
- Fuchs, W. H. J., & Rogosinski, W. W. (1942). A note on Mercer's theorem. *Journal of the London Mathematical Society*, 1(4), 204–210.
- Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4), 1–37.
- García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining*. Cham, Switzerland: Springer International Publishing.
- Geng, X., & Smith-Miles, K. (2009). *Incremental Learning*.
- Genton, M. G. (2001). Classes of Kernels for Machine Learning: a Statistics Perspective. *Journal of Machine Learning Research*, 2(Dec), 299–312.
- Gepperth, A., & Hammer, B. (2016). Incremental learning algorithms and applications. *European Symposium on Artificial Neural Networks (ESANN)*.
- Ghahramani, Z. (1996). *The pumadyn datasets*.
- Graybill, D. A. (1985). *Western US tree-ring index chronology data for detection of arboreal response to increasing carbon dioxide*.
- Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An Efficient Clustering Algorithm for Large Databases. *ACM Sigmod Record*, 27(2), 73–84.
- Gupta, A. K., Guntuku, S. C., Desu, R. K., & Balu, A. (2015). Optimisation of turning parameters by integrating genetic algorithm with support vector regression and artificial neural networks. *The International Journal of Advanced Manufacturing Technology*, 77(1–4), 331–339.
- Hall, P. (1989). On convergence rates in nonparametric problems. *International Statistical Review*, 57(1), 45–58.
- Hammer, B., He, H., & Martinetz, T. (2014). Learning and modeling big data. *ESANN*, 343–352.
- He, H., & Siu, W. (2011). Single Image Super-Resolution using Gaussian Process Regression. *CVPR*, 449–456.
- Hensman, J., Fusi, N., & Lawrence, N. D. (2013). Gaussian Processes for Big Data. *ArXiv Preprint ArXiv:1309.6835*. [https://doi.org/10.1016/S0074-7696\(01\)08005-6](https://doi.org/10.1016/S0074-7696(01)08005-6)
- Hensman, J., & Lawrence, N. D. (2014). Nested variational compression in deep Gaussian processes. *ArXiv Preprint ArXiv:1412.1370*.
- Hoang, T. N., Hoang, Q. M., & Low, K. H. (2015). A Unifying Framework of Anytime Sparse Gaussian Process Regression Models with Stochastic Variational Inference for Big Data. *32nd International Conference on Machine Learning*, 569–578.
- Hoffman, M. D., Blei, D. M., Wang, C., & Paisley, J. (2013). Stochastic Variational Inference. *The Journal of Machine Learning Research*, 14(1), 1303–1347.

- Horn, R. A., & Johnson, C. R. (2012). *Matrix analysis*. Cambridge university press.
- Jain, A. S., & Meeran, S. (1998). Job-shop scheduling using neural networks. *International Journal of Production Research*, 36(5), 1249–1272.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning*. New York: Springer.
- Jurkovic, Z., Cukor, G., Brezocnik, M., & Brajkovic, T. (2018). A comparison of machine learning methods for cutting parameters prediction in high speed turning process. *Journal of Intelligent Manufacturing*, 29(8), 1683–1693.
- Jylänki, P., Vanhatalo, J., & Vehtari, A. (2011). Robust Gaussian Process Regression with a Student- t Likelihood. *Journal of Machine Learning Research*, 12(Nov), 3227–3257.
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4, 237–285.
- Kim, K., Lee, D., & Essa, I. (2011). Gaussian process regression flow for analysis of motion trajectories. *2011 International Conference on Computer Vision*, 1164–1171. IEEE.
- Kolmogoroff, A. (1941). Interpolation und extrapolation von stationären zufälligen folgen. *Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya*, 5(1), 3–14.
- Kong, D., Chen, Y., & Li, N. (2018). Gaussian process regression for tool wear prediction. *Mechanical Systems and Signal Processing*, 104, 556–574.
- Kong, Dongdong, Chen, Y., & Li, N. (2018). Gaussian process regression for tool wear prediction. *Mechanical Systems and Signal Processing*, 104, 556–574.
- Kotsiantis, S. B., Kanellopoulos, D., & Pintelas, P. E. (2006). Data preprocessing for supervised learning. *International Journal of Computer Science*, 1(2), 111–117.
- Kulkarni, P., & Ade, R. (2014). Incremental learning from unbalanced data with concept class, concept drift and missing features: a review. *International Journal of Data Mining & Knowledge Management Process*, 4(6), 15.
- Lawrence, N. D., Seeger, M., & Herbrich, R. (2002). Fast Sparse Gaussian Process Methods: The Informative Vector Machine. *Neural Information Processing Systems*, 13.
- Lázaro-Gredilla, M., Quiñonero-Candela, J., Rasmussen, C. E., & Figueiras-Vidal, A. R. (2010). Sparse Spectrum Gaussian Process Regression. *Journal of Machine Learning Research*, 11, 1865–1881.
- Lease, M. (2011). On quality control and machine learning in crowdsourcing. *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*.
- Lee, H., Kim, S. G., Park, H. W., & Kang, P. (2014). Pre-launch new product demand forecasting using the Bass model: A statistical and machine learning-based approach. *Technological Forecasting and Social Change*, 86, 49–64.
- Lei, Y., Guo, M., Cai, H., Hu, D., & Zhao, D. (2015). Prediction of Length-of-day Using Gaussian Process Regression. *The Journal of Navigation*, 68(3), 563–575.

<https://doi.org/10.1017/S0373463314000927>

- Lu, Y., Boukharouba, K., Boonært, J., Fleury, A., & Lecoeuche, S. (2014). Application of an incremental SVM algorithm for on-line human recognition from video surveillance using texture and color features. *Neurocomputing*, *126*, 132–140.
- Lütz, A., Rodner, E., & Denzler, J. (2013). I Want to Know More-Efficient Multi-Class Incremental Learning Using Gaussian Processes. *Pattern Recognition and Image Analysis*, *23*(3), 402–407.
- MacKay, D. J. (1998). Introduction to Gaussian Processes. *NATO ASI Series F Computer and Systems Sciences*, *168*, 133–166.
- Malik, Z. K., Hussain, A., & Wu, J. (2016). An online generalized eigenvalue version of Laplacian Eigenmaps for visual big data. *Neurocomputing*, *173*, 127–136.
- Markham, I. S., Mathieu, R. G., & Wray, B. A. (2000). Kanban setting through artificial intelligence: A comparative study of artificial neural networks and decision trees. *Integrated Manufacturing Systems*, *11*(4), 239–246.
- McCloskey, M., & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, *24*, 109–165.
- McIntire, M., Ratner, D., & Ermon, S. (2016). Sparse Gaussian Processes for Bayesian Optimization. *UAI*, (June).
- Mermillod, M., Bugaiska, A., & Bonin, P. (2013). The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, *4*, 504.
- Mitchell, T. (1997). Introduction to machine learning. *Machine Learning*, *7*, 2–5.
- Mohammed, R. O., & Cawley, G. C. (2017). Over-Fitting in Model Selection with Gaussian Process Regression. *International Conference on Machine Learning and Data Mining in Pattern Recognition*, 192–205.
- Monostori, L. (2003). AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Engineering Applications of Artificial Intelligence*, *16*(4), 277–291.
- Mouchaweh, M. S., Devillez, A., Lecolier, G. V., & Billaudel, P. (2002). Incremental learning in fuzzy pattern matching. *Fuzzy Sets and Systems*, *132*(1), 49–62.
- Mozaffari, A., Vajedi, M., & Azad, N. L. (2015). A robust safety-oriented autonomous cruise control scheme for electric vehicles based on model predictive control and online sequential extreme learning machine with a hyper-level fault tolerance-based supervisor. *Neurocomputing*, *151*, 845–856.
- Nash, W. J., Sellers, T. L., Talbot, S. R., Cawthorn, A. J., & Ford, W. B. (1994). The Population Biology of Abalone (*Haliotis* species) in Tasmania. I. Blacklip Abalone (*H. rubra*) from the North Coast and the Islands of Bass Strait. *Sea Fisheries Division, Technical Report 48*, 411.

- Nguyen-Tuong, D., Peters, J., & Seeger, M. (2009). Local Gaussian Process Regression for Real Time Online Model Learning and Control. *Advances in Neural Information Processing Systems*, 1193–1200.
- Nguyen-Tuong, D., Seeger, M., & Peters, J. (2009). Model learning with local Gaussian process regression. *Advanced Robotics*, 23(15), 2015–2034.
- Nilsson, N. J. (1996). *Introduction to machine learning: An early draft of a proposed textbook*.
- Oliveri, P., Malegori, C., Simonetti, R., & Casale, M. (2019). The impact of signal pre-processing on the final interpretation of analytical outcomes—A tutorial. *Analytica Chimica Acta*, 1058, 9–17.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.
- Polikar, R., & Alippi, C. (2013). Guest editorial learning in nonstationary and evolving environments. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1), 9–11.
- Probst, P., Bischl, B., & Boulesteix, A. L. (2018). Tunability: Importance of hyperparameters of machine learning algorithms. *ArXiv Preprint ArXiv:1802.09596*.
- Qi, Y., Abdel-Gawad, A. H., & Minka, T. P. (2010). Sparse-posterior Gaussian Processes for general likelihoods. *26th Conference on Uncertainty in Artificial Intelligence*, 450–457.
- Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(Dec), 1939–1959.
- Quiñonero-Candela, J., & Winther, O. (2003). Incremental Gaussian Processes. *Advances in Neural Information Processing Systems*, 1025–1032.
- Rabe-Hesketh, S., & Skrondal, A. (2008). Multilevel and longitudinal modeling using Stata. In *STATA press*.
- Raducanu, B., & Vitria, J. (2008). Face recognition by artificial vision systems: A cognitive perspective. *International Journal of Pattern Recognition and Artificial Intelligence*, 22(5), 899–913.
- Ranganathan, A., Yang, M.-H., & Ho, J. (2010). Online Sparse Gaussian Process Regression and Its Applications. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 20(2), 391–404.
- Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. Cambridge, MA: MIT Press.
- Saffari, A., Leistner, C., Santner, J., Godec, M., & Bischof, H. (2009). On-line Random Forests. *2009 Ieee 12th International Conference on Computer Vision Workshops, Iccv Workshops*, 1393–1400. IEEE.
- Samet, H. (1990). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley Publishing Company Inc., New York.
- Sarle, W. S. (1996). Stopped training and other remedies for overfitting. *Computing Science and*

Statistics, 352–360.

- Sathya, R., & Abraham, A. (2013). Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2), 34–38.
- Schneider, T. (2001). Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of Climate*, 14(5), 853–871.
- Scholkopf, B., & Smola, A. J. (2018). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series.
- Schwaighofer, A., & Tresp, V. (2003). Transductive and inductive methods for approximate Gaussian process regression. *Advances in Neural Information Processing Systems*, 977–984.
- Schölkopf, B., & Smola, A. J. (2002). *Learning with Kernels*. MIT Press.
- Seeger, M. W. (2004). Low rank updates for the Cholesky decomposition. *Univ. of California at Berkeley, Tech. Rep.*
- Seeger, M., Williams, C. K. I., & Lawrence, N. D. (2003). Fast Forward Selection to Speed Up Sparse Gaussian Process Regression. *Artificial Intelligence and Statistics 9, Number EPFL-CONF-161318*.
- Sihag, P., Tiwari, N. K., & Ranjan, S. (2017). Modelling of infiltration of sandy soil using gaussian process regression. *Modeling Earth Systems and Environment*, 3(3), 1091–1110. <https://doi.org/10.1007/s40808-017-0357-1>
- Smola, A. J., & Bartlett, P. (2001). Sparse Greedy Gaussian Process Regression. *Advances in Neural Information Processing Systems*, 619–625.
- Smola, Alex J., & Bartlett, P. (2001). Sparse Greedy Gaussian Process Regression. *Advances in Neural Information Processing Systems*, 619–625. <https://doi.org/10.1255/nirn.1392>
- Snelson, E., & Ghahramani, Z. (2006). Sparse Gaussian Processes using Pseudo-inputs. *Advances in Neural Information Processing Systems*, 1257–1264.
- Sternberg, H. O. R. (1987). Aggravation of Floods in the Amazon River as a Consequence of Deforestation? *Geografiska Annaler: Series A, Physical Geography*, 69(1), 201–219.
- Stone, J. V. (2013). *Bayes' rule: A tutorial introduction to Bayesian analysis*. Sebtel Press.
- Subramanian, J., & Simon, R. (2013). Overfitting in prediction models—is it a problem only in high dimensions? *Contemporary Clinical Trials*, 36(2), 636–641.
- Sugiyama, M., Krauledat, M., & MÃzller, K. R. (2007). Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(May), 985–1005.
- Tipping, M. E. (2001). Sparse Bayesian Learning and the Relevance Vector Machine. *Journal of Machine Learning Research*, 1(Jan), 211–244.

- Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. *Artificial Intelligence and Statistics*, 567–574.
- Tong, Y. L. (2012). *The multivariate normal distribution*. Springer Science & Business Media.
- Tresp, V. (2000). A Bayesian Committee Machine. *Neural Computation*, 12(11), 2719–2741.
- Tsymbal, A. (2004). The problem of concept drift: definitions and related work. In *Computer Science Department, Trinity College Dublin* (Vol. 106).
- Vorburger, P., & Bernstein, A. (2006). Entropy-based concept shift detection. *Sixth International Conference on Data Mining (ICDM'06)*, 1113–1118. IEEE.
- Walder, C., Kwang, I. K., & Schölkopf, B. (2008). Sparse Multiscale Gaussian Process Regression. *Proceedings of the 25th International Conference on Machine Learning*, 1112–1119.
- Walpole, R. E., & Myers, R. H. (2012). *Probability & statistics for engineers & scientists*. Pearson Education Limited.
- Wiener, N. (1949). *Extrapolation, Interpolation and Smoothing of Stationary Time Series*. MIT Press.
- Williams, C. K. I., & Rasmussen, C. E. (1996). Gaussian processes for Regression. *Advances in Neural Information Processing Systems*, 514–520.
- Williams, C. K. I., & Seeger, M. (2001). Using the Nystrom Method to Speed Up Kernel Machines. *Advances in Neural Information Processing Systems*, 682–688.
- Williams, C. K., Rasmussen, C. E., Scwaighofer, A., & Tresp, V. (2002). *Observations on the Nyström method for Gaussian process prediction*.
- Wray, B. A., Rakes, T. R., & Rees, L. P. (1997). Neural network identification of critical factors in a dynamic just-in-time Kanban environment. *Journal of Intelligent Manufacturing*, 8(2), 83–96.
- Xin, J., Wang, Z., Qu, L., & Wang, G. (2015). Elastic extreme learning machine for big data classification. *Neurocomputing*, 149, 464–471.
- Yoshioka, T., & Ishii, S. (2001). Fast Gaussian Process Regression using Representative Data. *International Joint Conference on Neural Networks*, 132–137.
- Yuan, J., Wang, K., Yu, T., & Fang, M. (2008). Reliable multi-objective optimization of high-speed WEDM process based on Gaussian process regression. *International Journal of Machine Tools and Manufacture*, 48, 47–60. <https://doi.org/10.1016/j.ijmachtools.2007.07.011>
- Zhang, C., Qin, Y., Zhu, X., Zhang, J., & Zhang, S. (2006). Clustering-based missing value imputation for data preprocessing. *2006 4th IEEE International Conference on Industrial Informatics*, 1081–1086. IEEE.

