**Gaussian Process Regression**

*An Implementation that trains a GP regressor and predicts the output*

**Contents:**

Introduction......................................................................................................... 1

Algorithm and Procedure.............................................................................. 2

Execution steps................................................................................................ 3

Interesting inferences..................................................................................... 4

Observations and Inference....................................................................... 5

Work split-up....................................................................................................... 6

References......................................................................................................... 7

**By**

Anand Ganesh (1209408847)

Ashwin Murthy (1209365361)

**Introduction:**

The Gaussian Process is a non-parametric estimation technique used in the supervised learning problems like regression and classification (depending on the nature of the data). Supervised learning basically requires a training data set that is used to teach the machine or the system, the type of pattern it needs to absorb(learn) so that it can use this data to predict/classify new data that it has not seen before.

In our simple implementation of a GP Regressor, the training dataset is given along with additional validation data (fine-tuning). The final result of this project is the estimated value of an unknown data, and the certainty or confidence with which the regressor knows that value of the unknown data.

The Regressor is fine tuned to not just fit the training data exactly but to learn the right values of parameters (that tailor make the GP for a particular data set) and hence improve the accuracy of its prediction and its confidence.

This learning of the parameters is the core behind machine learning in that the right value of the parameters are to be computed for each data set and this greatly changes the performance of a machine from making random guesses to highly probable educated guesses.

The improvement in the performance of the regressor can be seen from the comparison of the error rates in predicting values that are known to the user but are hidden from the machine.

**Algorithm & Procedure:**

The major difference in the nature of a Gaussian Distribution and a Gaussian Process is the fact that each point on a Gaussian Process is a random variable which is Gaussian in nature by itself. This therefore means that each point on the plot has a mean and it own covariance. Therefore to find the value of any particular unknown data is to find the function at that point (f\*) which is a Gaussian with mean (**μ**\*) and Covariance (**Σ**\*). To put the whole process into a sequence of conceptually abstracted steps, we can write it as:

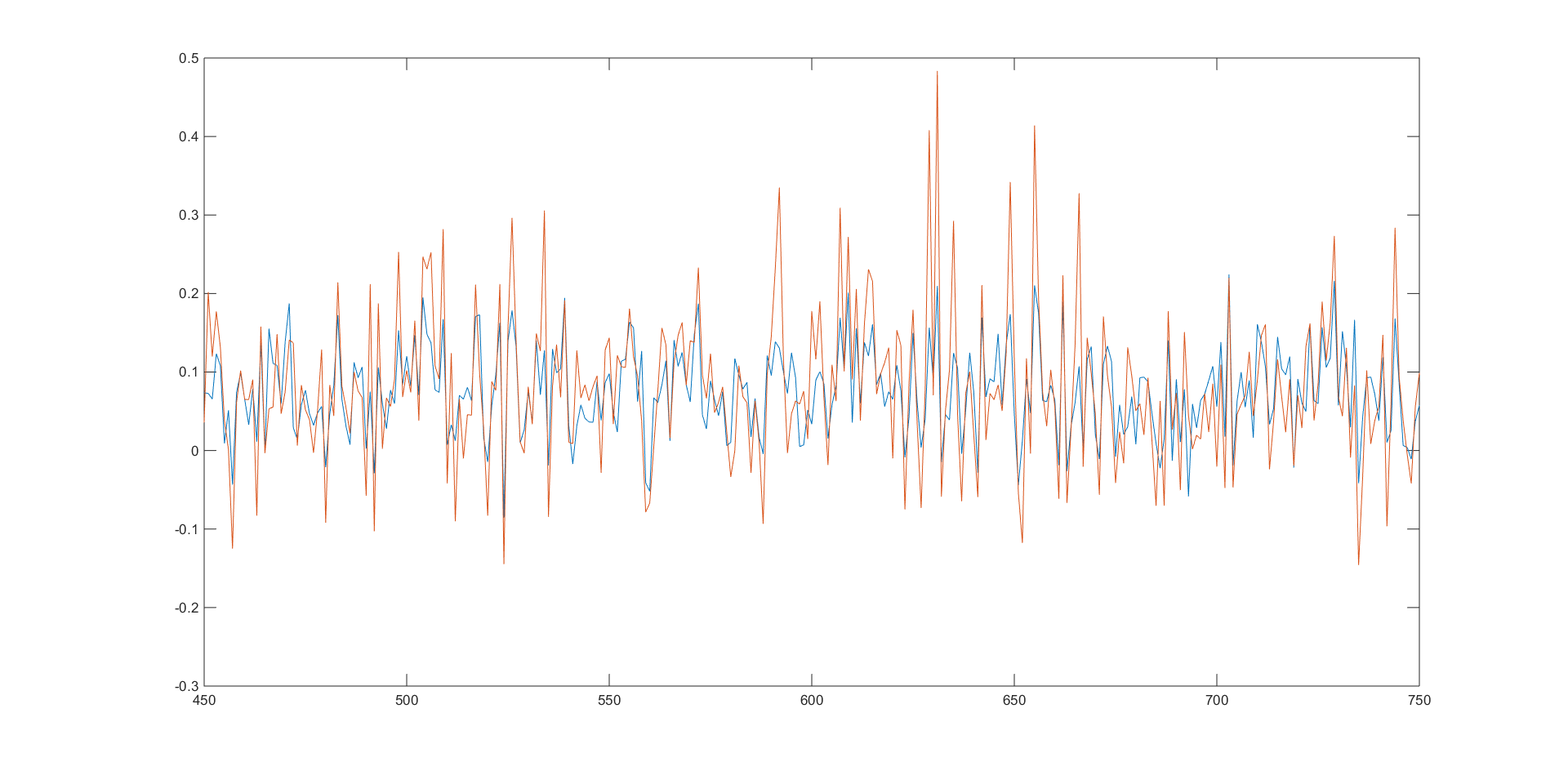
* Use the given training data sets as inputs to the regressor and as the value of the outputs are also known to the user, we can use any value of the hyperparameters that seem fit to find the regressed value of a known test case that is not part of the training data ( this step is done to only compare the error rates of the regressor before and after optimizing values
* Once the regressed values are computed, the best values of the hyperparameters are computed by the minimizing the log marginal likelihood of the dependent variable for a particular value of the independent variable.
* The correctness of these new hyperparameters can be verified by computing the error rates of new data for which the output is known.
* These newly computed hyperparameters are then used to compute the values of unknown test cases and the regressed values are inferred as:

**A Gaussian Random Variable with:**

**(μ\*)** - Most probable value of the unknown test case

**(Σ\*)**  - The possible variation in the value of the predicted value

**Execution steps:**



PLOT(Predicted target\_val, actual target\_val)

The execution of the script “final\_script\_1” present in the zip file will result in the generation of the regressed values, its confidence, the error rates before and after optimization and the various intermediate covariance matrices and terms required to compute the result.

“**Error**” - error rate before optimization

“**New\_error**” – error rate after optimization

“**Optimized regressed target\_test**” – predicted value of the unknown data “input\_test”

“**New\_confidence”** – The confidence of the predicted value of target\_test after optimization

**Interesting Inferences:**

When the output of a regressed value is observed, the fact that the confidence being high isn’t always a bad trait as this finds application in places where the environment has significant changes in values (comprises of many outliers) and hence having a large confidence can make the regressor possibly classify and learn those values as well: This is one of the major uses of GP regression modelling.

The introduction of the signal noise accounts for the fact that the training data set isn’t perfect and prevents what is called “OVERFITTING” where the regressor thinks that the training dataset is perfect and any deviation from the training dataset is undesirable and anything that matches the training dataset means a very high probability of classifying under one type. This is bad practice as this leaves no room for outliers and the real world is full of exceptions. By introducing noise, we prevent the overfitting of the regressor to the training dataset.

The non-parametric nature of the GP model means that it does not try to fit an overall function behind the entire data set but just finds the relation between adjacent data points and their relations with each other to predict the most likely values of an unknown value.

The gradient descent function and other optimizing library functions like fminsearch can find either the local minima or have random jumps that can help it overcome local minima to possibly find the global minima

Other efficient and better ways to do the same process like the evaluation of the inverse and the determinant of a matrix by using Cholesky decomposition of that matrix.

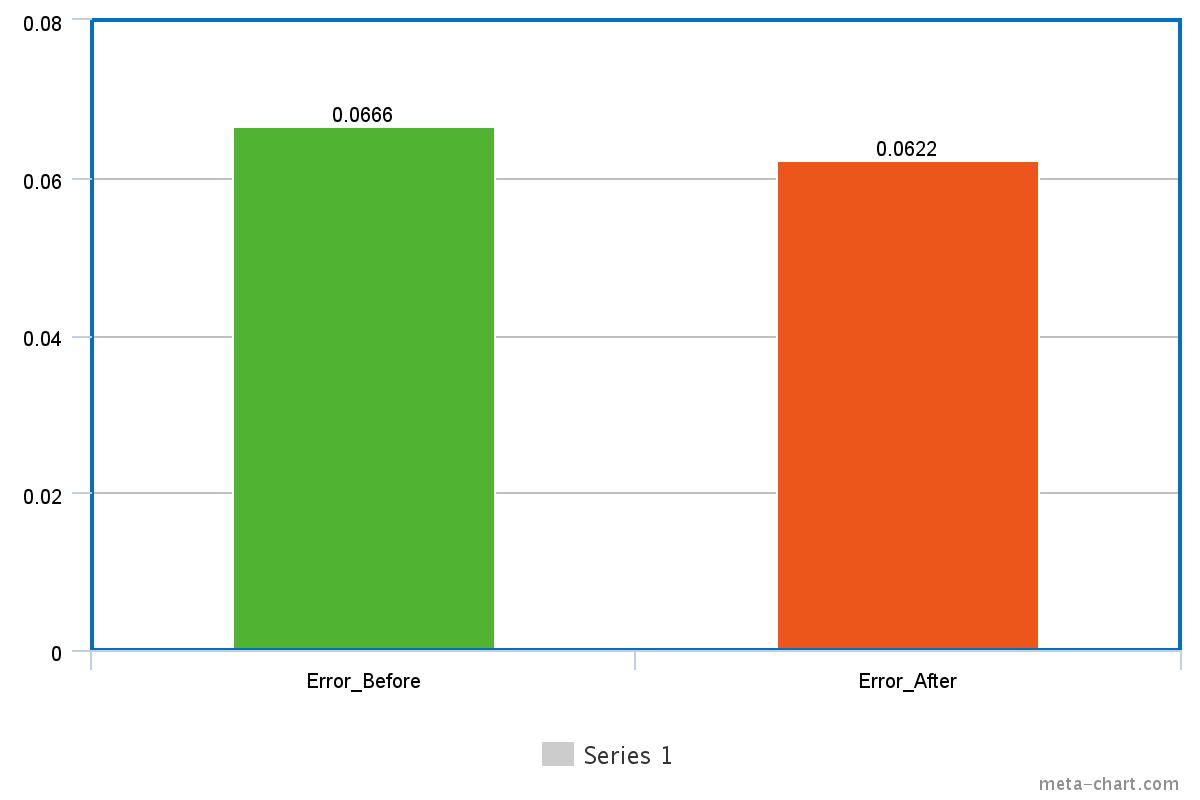
**Observation and Inference:**

When we initially run the regressor, the output value that is predicted is based off of arbitrary values of the hyperparameters.

* X = input\_train Y = target\_train X\* = input\_val Y\* = target\_val
* Initial value of hyperparameters : sigma ‘f’ = **0.25** l = **0.25** sigma ‘n’ = **0.25**
* Error between the predicted value of target\_val and actual value = **0.0666**

After training the value of the hyperparameters, the regression process was repeated again to check the new error rate:

* X = input\_train Y = target\_train X\* = input\_val Y\* = target\_val
* Error between the new predicted value of target\_val and actual value = **0.0622**



**Work-Split and Manual Calculations:**

The work was a joint team effort with both the team members contributing to every aspect of the project, right from the acquisition, reading and understanding of resources to the drafting editing and completion of the poster and the matlab code.

DERIVATION FOR EQUATION OF LOGLIKELIHOOD FUNCTION:

*We use a Gaussian process prior for the latent function:*

*f|X, θ ∼ N (0, K)*

*The likelihood is a factorized Gaussian*

*y|f ∼ N (yi |fi , σ2 n ) ( multiplied from i = 1 to m )*

*The posterior is Gaussian*

*p(f|D, θ) = [p(f|X, θ) \* p(y|f)] / p(D|θ)*

*The value at the test point, f(x ∗ ) is Gaussian given as*

*p(f∗|D, θ, x∗) =* ∫  *p(f∗|f, X, θ, x∗)p(f|D, θ)df,*

*and the predictive class probability is Gaussian*

*p(y∗|D, θ, x∗) =*  ∫  *p(y∗|f∗)p(f∗|D, θ, x∗)df∗.*

*Predictive distribution: can be given as :*

*p(y ∗ |x ∗ , x, y) ∼ N k(x ∗ , x)’ \* ( ( [K + σ 2 noiseI] )^ −1) \* y, k(x ∗ , x∗ ) + ( ( σ noise) ^2 ) − k(x ∗ , x)’ \* ( ( [K + σ 2 noise \* I ] )^ −1 ) \* k(x ∗ , x)*

*To chose between models M1, M2, . . ., compare the posterior for the models*

*p(Mi |D) = [ p(y|x, Mi) \* p(Mi) ] / p(D) .*

*Log marginal likelihood:*

*log p(y|x, Mi) = − 1 / 2 [ ( y ‘) ( (K) ^−1) \* y] − 1 / 2\* ( log |K|) − n 2 log(2π)*

**References:**

[1] Melo, José. "Gaussian Processes for regression: a tutorial."

[2] Mark, Ebden. "Gaussian Processes for regression: A Quick introduction."

[3] Slides by Hanna M Wallach “Introduction to Gaussian Process Regression”.

[4] Gaussian Process Regression Models - < http://www.mathworks.com/help/stats/gaussian-process-regression-models.html >

[5] GPML Matlab functions – sq\_dist – which computes the mean square difference- < http://www.g aussianp rocess.org/gpml/code/matlab/doc/ >

[6] Nando de Freitas – Machine Learning – Introduction to Gaussian Process - < https://www.youtube.com/ watch?v=4vGiHC35j9s >

[7] Nando de Freitas – Machine Learning – Gaussian Process - < https://www.youtube.com/watch?v=M fHKW5z-OOA >

[8] Kernel (Covariance) function options - < http://www.mathworks.com/help/stats/kernel-covariance-function-options.html >

[9] Cholesky Decomposition for estimating the determinant of a matrix which closely singular in nature - < http://www.mathworks.com/matlabcentral/answers/41080-compute-determinant-using-cholesky-decomposition >

[10] John Cunningham – Gaussian Processes - < http://videolectures.net/mlss2012\_cunningham\_gauss ian\_processes/ >

[11] Carl Edward Rasmussen – Derivation for log likelihood for a Gaussian process model – < <http://gpss.cc/gpip>/slides/rasmussen.pdf >