1: Introduction

For this lab I have been tasked with the implementation of multiple regression methods, to be used on the SACROS data set. The SACROS data set contains the data of the seven degrees of freedom of an anthropomorphic robot arm the position, velocity and acceleration (21 features) and a mapping from that data to the torque of one of the joints/motors. The task is to learn regression models that can predict the torque given new data of the degrees of freedom.

2: Methods

To do the regression I will implement 4 different regression methods, K-Nearest Neighbours, Linear Regression, Regression forests and Gaussian Process. The reason I'm running multiple methods to regress the data is to able to compare the accuracy of the methods on the given data. For any of these methods to work we need a training dataset from the SACROS data so that we can work out the relationship between 21 features and the torque value.

K-Nearest Neighbours is the simplest of the regression methods. For new data which we aim to estimate the torque from, we calculate the k nearest data points from the training data using the Euclidean distance between to data points. Then normally you would calculate the mean of torque from these data points. To enhance the quality of the method, I calculate the weighted average of the k torque values weighted by the inverse of their distance. For this method the only hyperparameter we need to optimize for is the value of K

Linear Regression aims to find the linear equation that best fits the given data.

$$\boldsymbol{Y} = \boldsymbol{X}\beta + \varepsilon$$

Where **Y** is the torque values given in the training data and **X** is the other 21 features of the training data, with a new column of 1s concatenated on the front and ε being an error variable. The aim is to find β such that it minimised ε. This can be calculated in iteratively with gradient decent, or it can be calculated analytically. If we estimate ε to be the least squared error, then β can be found to be

$$\beta = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

A Regression forest is a set of regression trees, which all give an estimate of the torque given an exemplar of the 21 features. To build a regression tree, given the training data we try to find a split in the data that minuses the Sum of Squares Error (SSE) of the data once it has been split.

$$SSE = \sum_{left}(y_i - \bar{y_l})^2 + \sum_{right}(y_i - \bar{y_r})^2$$

Where $\bar{y_l}$ and $\bar{y_r}$ correspond to the mean value of the left and right-hand sides of the possible split

The tree continues to split the data until the variance of that subset of data at a node is under a threshold, or no best split can be found, or the maximum depth of the tree has been reached, at which point the estimation at that node is set to the mean of the subset of data.

To enhance the performance of the Regression forest and to minimise overfitting on the training data, each tree is only trained on a random subset of the data, and at each split is only allowed to split on a random number of features of the data. The result of the regression forest is the average of all the results from the trees.

The Hyperparameters that need to be optimises for Regression forests are the N number of trees, D the maximum depth of the trees, P number of features and the S the size of the subset of data used in each tree.

Gaussian Process (GP) is a non-parametric approach to regression as it finds a distribution over the possible functions f that are consistent with the observed data. To do so we need to define a distribution over the observed points $x_1, \dots, x_N$ and their function values $f(x_1), \dots, f(x_N)$, in which we assume the function values are jointly gaussian distributed with mean $\bar{x}$ and covariance matrix $\boldsymbol{K}_x$ given by a positive definite kernel function k(x_i, x_j).

The choice of kernel function is up to the decision of the user wanting to use GP, and it effects the shape and smoothness of the functions. I have chosen to investigate the usage of 2 kernel functions the Squared Exponential kernel

$$k(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right)$$

Where l (lengthscale) determines the length of the "wiggle" in the function and σ^2 determines the average distance of your function away from the mean. The second kernel I implemented was the Rational Quadratic Kernel.

$$k(x_i, x_j) = \sigma^2 \left(1 + \frac{(x_i - x_j)^2}{2\alpha l^2}\right)^{-\alpha}$$

Which is equivalent to many SE kernels added together with different lengthscales. α determines the relative weighting of large and small-scale variations. As $\alpha \to \infty$ the RQ kernel becomes indistinguishable from the SE kernel.

In a noiseless model the joint distribution of **y** observed function values and f(**x'**) unobserved variable can be obtained as

$$\begin{pmatrix} \boldsymbol{y} \\ f(\boldsymbol{x}') \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \boldsymbol{K}_x & \boldsymbol{k} \\ \boldsymbol{k}^T & k(\boldsymbol{x}', \boldsymbol{x}') \end{pmatrix}\right)$$

With $\boldsymbol{k} = [k(\boldsymbol{x}', \boldsymbol{x}^1), \dots, k(\boldsymbol{x}', \boldsymbol{x}^N)]^T$

Then we can then calculate the posterior

$$p(f(\boldsymbol{x}')|\boldsymbol{x}', \boldsymbol{y}, \boldsymbol{X}) = \mathcal{N}(\boldsymbol{k}^T {\boldsymbol{K}_x}^{-1} \boldsymbol{y}, k(\boldsymbol{x}', \boldsymbol{x}') - \boldsymbol{k}^T {\boldsymbol{K}_x}^{-1} \boldsymbol{k})$$

If we were to assume there was gaussian noise present we would swap $\boldsymbol{K}_x$ with $(\boldsymbol{K}_x + n^2\boldsymbol{I})$ where n is the noise parameter.

Overall the Gaussian Process has the hyperparameters n plus the parameters of the kernel function

Why (or why not) you would use these algorithms

3: Validation on toy problem

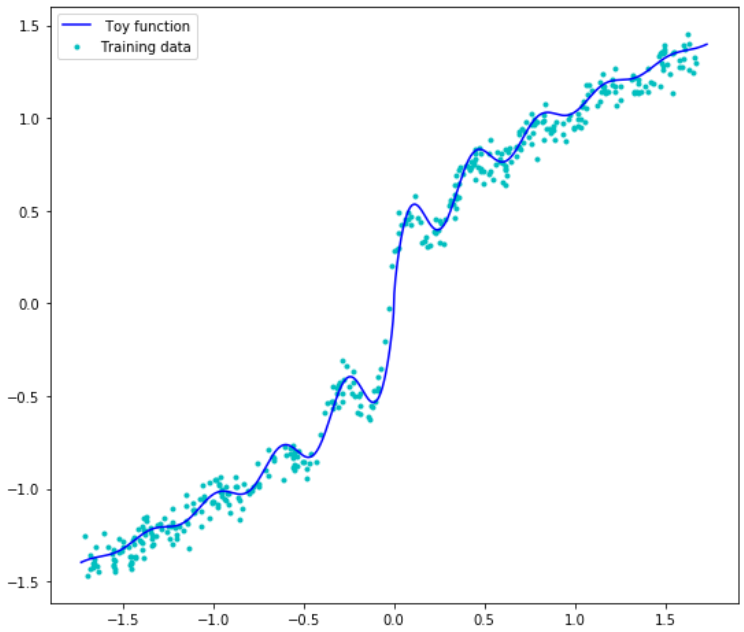
To test and validate my implementations of the regression methods above I will need to design a toy problem to test them on, this can be a simple 2D data set generated from some function. The function I have chosen the data points from is the function

$$y = 0.98^{|x|} * 2\sin\frac{x}{5} + negposSqrt(x)$$

$$negposSqrt(x) = \begin{cases} -\sqrt{|x|}\ if\ x < 0 \\ \sqrt{|x|}\ if\ x > 0 \end{cases}$$

To simulate the noise, you often see when collecting experiment data in the real world we will add some random normally distributed noise to the output the of the function. The rational behind this function is that it is an nonlinear function that oscillates at a decreasing rate as |x| increases, as such this function is not well suited to linear regression. As this function will be hard to regress with linear regression it makes sense that it is a good function to use to test out other regression methods that are better at finding nonlinear functions.

To train and test the regression methods I generated 500 data points from the function within the x range of -200 to 200. With this data I then normalized the data points such that they have a mean of 0 and variance of 1. To generate the difference sets I split the data into an 80% divide between training and testing data.



Before the regression methods can be run on the data we first need to find the optimum hyper-parameters for the data given. I achieve this by running a combination of N fold cross validation and brute force search of the hyperparameters. N fold cross validation is a method used to validate the choices of hyperparameters by running your method on subsets of the training data, In my case I split the data into to 10 equally sized subset and train the methods on 9 of them and then test on the final subset. This is done 10 times on the different combinations of the subsets to calculate an average test result. We can iterate over values of the hyper parameters and compare the results of the nfold cross validation and the hyperparameter that gives us the lowest average error is chosen. The reason for doing optimisation on the training data and validation set instead of the test data is that in machine learning you should know about the final test data, and you shouldn't optimise the methods to give good results for the test data you've got, you should instead optimize your results such that for any test data your methods should get reasonable results.

Instead of using the nfold cross validation method to optimise for the Gaussian Process, I optimise by finding the hyperparameters that maximises the log likelihood of the GP

$$-2\log(\boldsymbol{y}|\boldsymbol{X}) = \boldsymbol{y}^T(\boldsymbol{K}_x + n^2\boldsymbol{I})^{-1}\boldsymbol{y} + \log(\boldsymbol{K}_x + n^2\boldsymbol{I}) + Nlog(2\pi)$$

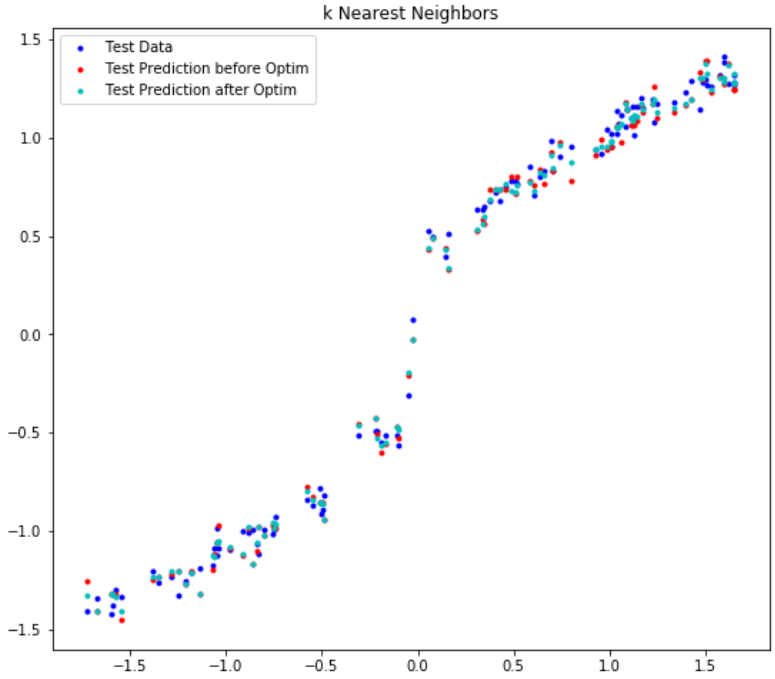
This measures how likely the values of **y** came from a function from the normal distribution with covariance $\boldsymbol{K}_x$. This works with the same methodology of the iteration as the method used before where is find my hyperparameters using a brute force search to find the parameters that maximise the above formula. There are some downsides to this method, for first it is computationally expensive to run as you have to recalculate the covariance matrix for each iteration and then invert it, which is an O($\boldsymbol{N}^3$) operation, which for large N becomes very expensive. The second downfall to this method is that it tries to optimise parameters such that the functions of the distribution evaluate at "common values" eg y = 0 or functions that have a small polynomial order.

3.1 Toy Problem Results

To measure the accuracy of the regression method I have chosen to use the least squares error

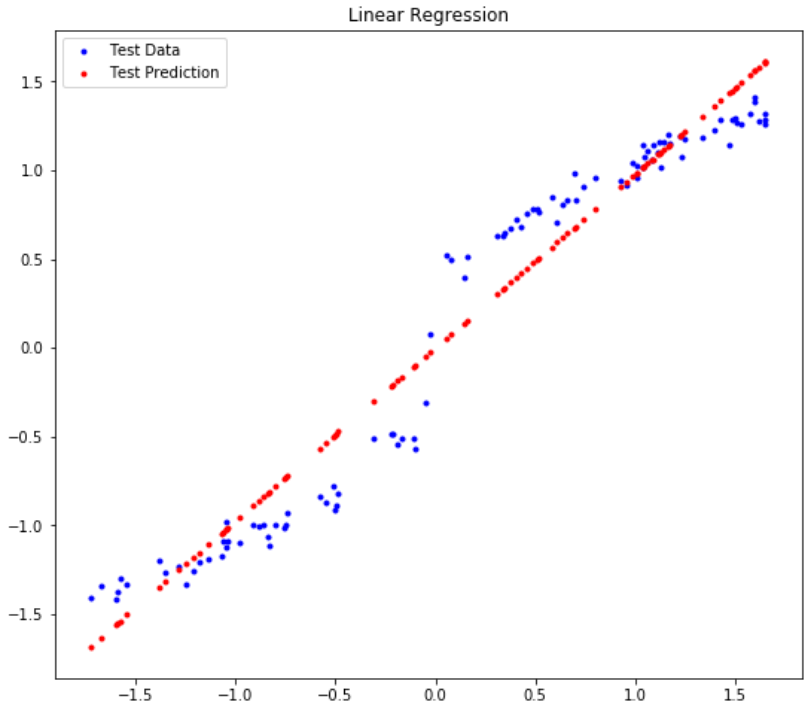
$$LSE(\boldsymbol{y}, \boldsymbol{y}') = \sum_{i=1}^{n} (\boldsymbol{y}'_{\boldsymbol{i}} - \boldsymbol{y}_{\boldsymbol{i}})^2$$

For the K Nearest Neighbour regression method, we got a LSE of 0.641 before optimisation with k = 1 and after optimisation with k = we got the LSE of 0.470.



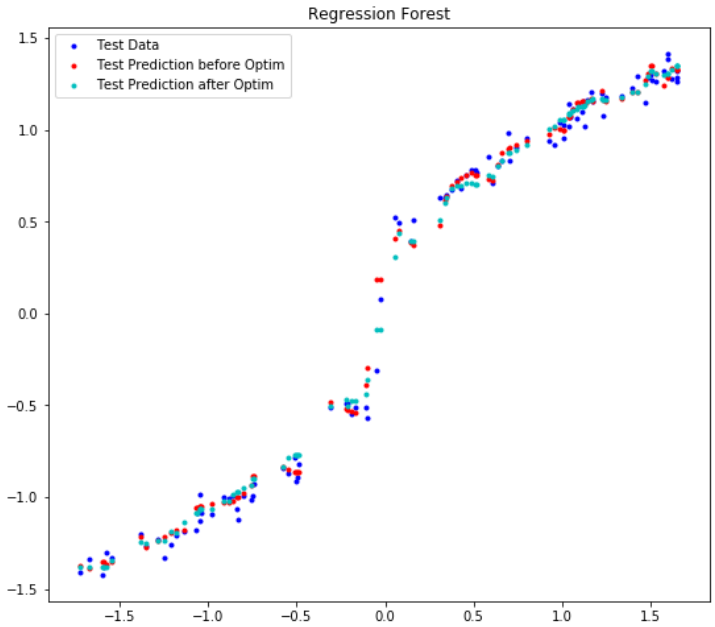
As is evident by the low LSE and the graph of the plotted regressed points shows that this method works quite well for the toy function I have chosen. The only problem with this method is that it seems to not filter out the noise in the data, creating some of the incorrect predictions. But overall this method works quite well for the data presented.

For Linear regression we get a LSE of 5.192, which compared to the results of the other methods is a lot higher, but this was anticipated as I designed the toy function such that it couldn't get a good result for linear regression.

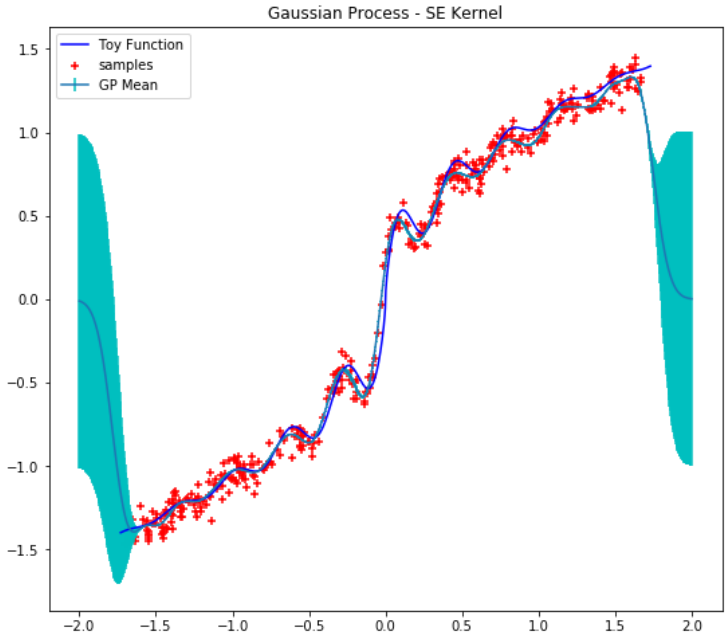


For the regression forest method, we achieved the LSE of 4.346 with the hyperparameters of 20 trees, 1 feature, 125 set size, and a max depth of 20. This was then optimised to the LSE of 4.056, when using the hyperparameters of 10 trees, 60 set size and max depth of 6

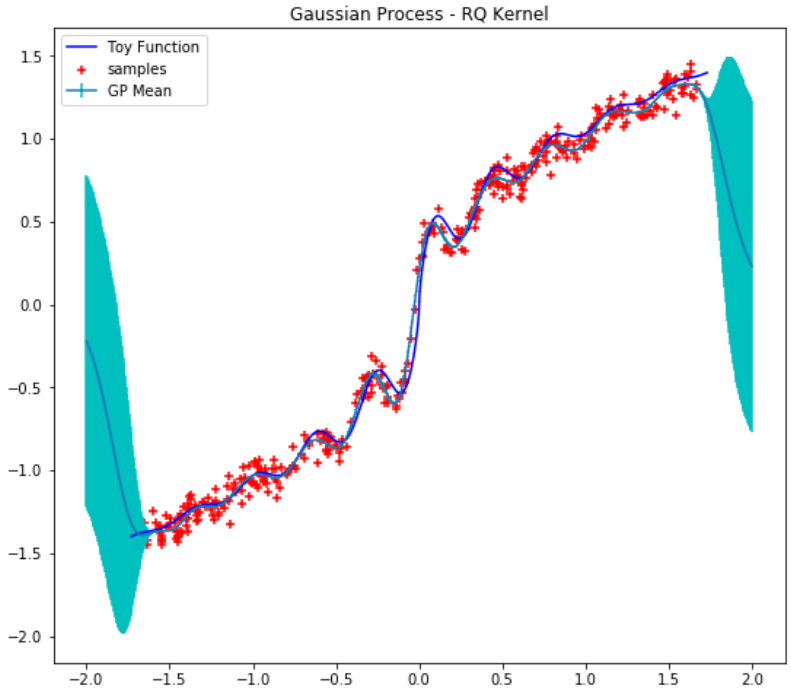
The problem I observed with regression forest is that the results often changed each time I ran the method to generate the forest, this is due to the randomness built into the method, but I made it difficult to optimise and test due to differing results each time I ran the method. This could be because the hypermeters weren't optimised correctly.



For the Gaussian process using the SE kernel we achieved a LSE of 0.3603, with the hyperparameters of sigma equal to 0.5, l being 0.1 and the noise set to 0.05. This result is significantly better than the result results gathered from the KNN method.



The Gaussian process with the RQ kernel performed even better than the SE kernel with a LSE of 0.31804. This was with the hyperparameters of alpha = 5, sigma = 3.2, l = 0.1 and noise = 0.05.

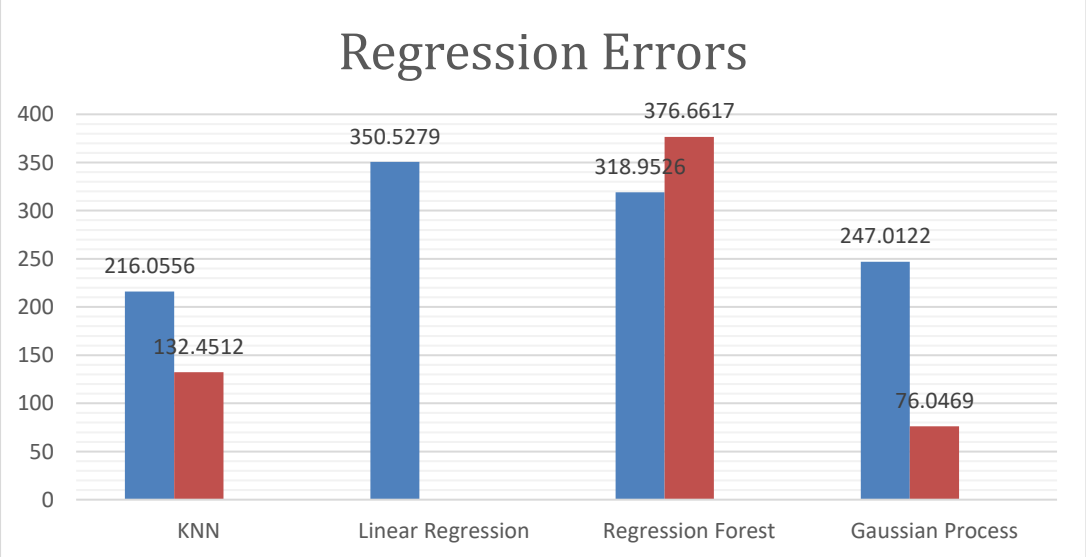


These results from the Gaussian process indicate the it is the best method for regression non-linear data akin to my toy function out of all the methods I have implemented.

4: Experiments and analysis

The Methodology I will be using on the SACROS data set will be the same as what I need with the toy problem, I will be normalising the data to have mean of 0 and variance of 1. I will then do same Hyperparameter optimization techniques for the 4 methods on the dataset. Due to the size of the data set given and the dimensions of the data, I will only do the hyperparameter optimization on a small subset of the training data to save the computation time required to run them.

Likewise, for testing the methods, I will use a small subset of the testing data, to get an idea of the accuracy of the methods on the SACROS data set. The for testing will be committed on the first 5000 elements of the testing set. The LSE results I got for the 4 methods whereas followed



Where the hyperparameters were set to the following respectfully

KNN: k=1, k=4
Regression Forest: (25 trees, 5 features, 25% data, max depth = 25)
(50 trees, 5 features, 15% data, max depth = 30)
Gaussian Process: SE Kernel with noise = 1, sigma = 26.42, l =1.35
RQ Kernel with noise = 1, alpha=2.245, sigma = 10, l = 3.57

These results show us the dramatic difference between the results of the regression methods. Such that the Gaussian Process with RQ kernel is miles better than linear regression. This is evident to indicate the data of the SACROS data set is very non-linear in nature. It also shows much better GP is at regression just by the fact that it is only trained on 10,000 data exemplars compared to the 26,000 exemplars the other methods are trained on.

I also experimented with the implementation of regularised linear regression which aims to make linear regression a better method for large dimensional dataset, but as In the SACROS the number of features per exemplar is much smaller than the number of training points used, the added benefit of doing regularised linear regression in so miniscule that it doesn't improve the results.

Although the GP is the best at estimating the values of torque in the data set, it isn't the most memory or computationally efficient as some of the other methods looked at. The GP training process can use quite a lot of memory usage as you need to store the N*N Covariance matrix, where N is the number of data points used to train the GP. This can get very large. Also for large number of N the computation of a prediction requires a O(N^3) operation when inverting the covariance matrix. Compared to the complexity and memory usage of the linear regression method which has time complexity of O(feature^2*N), then GP is a vastly infer way of regression.

The other methods also require quite a lot of training and predicting time. The Regression forest takes O(ntree * features * nlog(n)) time to build the forest, which for the size of training set is also infer to the time complexity for linear regression. The only other method that has a reasonable time complexity is K nearest neighbours with complexity of O(Nd) where d is the dimension of the training data.

Overall the performance of each method is dependent on the type of data you are using it on and the reason your doing regression. If you do want to find a quick linear function that approximates your data then linear regression is best, but if you want accurate results from a nonlinear data set your better of choosing another method, like nearest neighbour or Gaussian process, depending on the size of the data set and its dimensionality.