MACHINE LEARNING COURSEWORK REPORT

Regression Solutions based on 4 different approaches

University of Bath

MSc Digital Entertainment

CM50264 - Machine Learning

Word Count: 2964

1. Introduction

We were assigned the task to produce implementations of 4 regression methods from scratch, as well as provide critical analysis of each method, based on the SARCOS data set. The SARCOS data set provides a multivariate regression problem, where given multiple input features, specifically position, velocity and acceleration with 7 degrees of freedom (21 features in total), the goal is to predict the torque of a single motor in a robotic arm. It consists of huge number of exemplars, more precisely 44483 exemplars, and we were also given the freedom to decide where to split the data for training and testing our regressors. This problem is considered a regression problem since the target values are continuous and infinite, opposed to a classification problem where the target output set is finite. The 4 different regression approaches to be discussed below include:

- Nearest Neighbour,
- Linear Regression,
- Regression Forest,
- And Gaussian Process.

The following sections will provide insights to how these methods were implemented, the hyperparameter selection strategies and various experimental results, which will be used to compare when and why should each algorithm be used. Please be aware that files for the implementations are provided separately and standalone for each algorithm except kNN and Linear Regression.

2. Algorithm Descriptions

With 4 different mechanisms to predict the torque of the motor by the robotic arm, given a new input sample, there are various parameters to be compared as well as complexity for each of the algorithms. The following subsections will briefly explain how Regression Forest and Gaussian Process algorithms are derived and what separates them from each other.

2.1 Regression Forests

Regression forests consist of several decision trees, providing an average of the predictions outputted. The process of building a single tree will be described. Firstly, we decide on the splitting criterion to find the split that will return the least Sum of Squared Error (SSE). This is implemented as a function that when building the tree would be called and eventually split the training set to produce 2 further nodes, a binary split. To stop the tree from growing indefinitely, we use a stopping condition. The stopping condition consists of checks on the whether the max depth allowed is reached, whether all the values reaching the node are the same or if the size of the subset at that node is 5% of the overall size. Then a prediction is made by traversing each test sample down the tree until it reaches a node. Until now, an explanation on how to construct a regression tree is given, but a tree does not produce a forest. In order to speed up the process of training and simultaneously predicting a target value given a sample, various smaller trees can be created each training on a random subset of the data and on a random subset of the features. A bundle of techniques is used to yield these subsets and each has its own preferences. To randomly select a subset of our exemplars, an ensemble method is required, and the method that is commonly used when considering random forests is bootstrap aggregation, also known as bagging, (Brownlee, 2018). Bagging randomly selects a subset of the data set, with replacement, meaning some exemplars may appear twice or even more in the subset, allowing us to be less concerned on whether a tree is overfitting our model. Based on this, we can also forget about pruning a tree, since an average of the predictions generated by each tree will conclude on a final prediction.

As for the features, they are randomly chosen based on a ratio. The features chosen are the features the new tree is to utilise and split upon, thus making the process of building a single tree faster. The random features are chosen without replacement. This is due to the fact that we already get only a subset of the features, having twice or even three times the same feature to split on will only make our tree inaccurate. With all the pre-processing mentioned, the following algorithm is used to produce a random forest:

RandomForest (train data, test data, n_trees)

1. Split the train data to train input, X, and train output, Y, (target values), and the test data to test input, X_t, and test output, Y_t (actual values)
2. Assign a bag ratio, e.g. 0.1, the subset size will be 1/10 of the overall data set size.
3. Assign a feature ratio, e.g. 1/3, the features used find splits will be 1/3 of the overall dimensions.
4. For i in range 1 to n_trees do:
    a. Produce a random subset with replacement of X, using bagging based on the bag ratio
    b. Produce a random subset without replacement of the features based on the feature ratio.
    c. Build a tree using the new random subset of data and features.
    d. Derive a prediction vector by recursively traversing down the tree for each sample in X_t and sum it over all prediction vectors derived thus far.
5. Divide the sum of all prediction vectorss by the number n_trees to find the mean of predictions
6. Output prediction vector

2.2 Gaussian Process (GP) for regression

Gaussian Process regression is a probabilistic approach where the final output is given as a predictive distribution for the dependent target variables. In fact, it is equivalent to Bayesian linear regression when combined with a nonlinear feature map, meaning multi-dimensional feature space,(Bailey, 2018).

The process of obtaining the desired predictive distribution, is by kernelizing Bayesian non-linear regression where the kernel is the covariance function of a GP. GP is a non-parametric approach as it utilises the consistency of observed data, considering the prior of the distribution to finally derive the posterior. The training and testing stages cannot be distinguished properly, except if one considers the process of building the kernel matrix, also known as Gram matrix, as the training stage, which can also be pre-calculated to avoid unnecessary addition to time complexity. Following are the operations steps that need to be done to derive a prediction for a test sample; it will be broken down into 4 steps:

Step 1: Creating a kernel function (Squared exponential)

The kernel function is a function that takes 2 parameters **a** and **b**, which correspond to samples of the data, i.e. rows of input data. This function will return a single value for each pair of parameters:

$$k(\boldsymbol{a}, \boldsymbol{b}) = \exp(-\frac{||\boldsymbol{a}-\boldsymbol{b}||^2}{\sigma_k^2}) \tag{5}$$

Where **a** and **b** are sample inputs and σ_k^2 is a variance consider as a hyperparameter. The kernel function calculates the exponent where the distance of the two samples is squared over the variance. This is called a squared exponential kernel function.

Step 2: Constructing a kernel vector

This vector is dependent on the input sample for testing, **x'**. It is a vector that contains the output of the kernel function (5) with inputs **x'** as **a** and **X** = $[\boldsymbol{x_1}, \dots, \boldsymbol{x_N}]$ as **b**, as shown in (6):

$$\boldsymbol{k} = [k(\boldsymbol{x'}, \boldsymbol{x_1}), k(\boldsymbol{x'}, \boldsymbol{x_2}), \dots, k(\boldsymbol{x'}, \boldsymbol{x_N})]^T \tag{6}$$

Where N is the number of training samples, rows, of our training data set.

Step 3: Producing the Covariance matrix

Each value in this matrix is also an output from the kernel function (5). Eventually, this covariance matrix will be an NXN matrix, with N being the number of training samples. This is also main reason for the GP's complexity being high. Eventually this matrix will contain row vectors produce by the method described in step 2, equation (6), where the corresponding **x'** will be every other training row. Therefore:

$$[K]_{i\,j} = k(\, x_i, x_j) \tag{7}$$

Step 4: Deriving the predictions

For the final step, what is left to do is derive a prediction for the target values of our test data using the previous steps. Given that each new sample to be predicted is different, the GP will produce a unique normal distribution, with different mean and variance, based on it. The following distribution is drawn:

$$p(y' |\, \boldsymbol{x'}, \boldsymbol{y}, \boldsymbol{X}) = N(\, \boldsymbol{k^T}\, (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}, k(\boldsymbol{x'}, \boldsymbol{x'}) - \, \boldsymbol{k^T}\, (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}\,) \tag{8}$$

Where y' is the predicted target value, **x'** is the input test sample, **y** is the training actual output for all training samples (vector). This is done for each new sample. But when it comes to predicting target variables, we only need the mean of the distribution: $\boldsymbol{k^T}\, (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}$**.** This Gaussian distribution mean is enough to provide a valid prediction and contains the second hyperparameter of GPs, the variance, σ^2. This value is a scalar multiplying the identity matrix, eventually producing a noise to the distribution, and it is very critical to the final result. Hyperparameter selection will be reviewed in the next sections.

2.3 Regression Forests vs. Gaussian Process: When and why are they most optimal?

Every machine learning algorithm has a unique time complexity and performs best on a certain type of data set. In the case of Regression Forest (RF) and Gaussian Process (GP), there are various reasons for choosing the best one, and they are weighted by the regression problem at hand. RF are most efficient when the data set is of great size. As mentioned above, in the case of RFs, multiple regression trees are built on randomly selected subsets of the data, and features, thus allowing for parallelisation with no chance of overfitting our regressor to the training set. Moreover, the time complexity comes out to be O(mn log n), where m is the number of the features used per tree building and n is the number of samples in the bagged training set, if the Random Forest is run in parallel, and O(tmn log n), where t is the number of trees, if the algorithm is not parallelised. GPs on the other hand, is most efficient when the number of features, n, is greater than the number of training exemplars, N, denoted as n>N. This is logical, as the covariance matrix, K, is created by performing NxN operations. In the case of the SARCOS regression problem, this is not the case. Since we have 21 features and over 44000 exemplars, the optimal training set would have to contain less than 21 exemplars for the GP to be both fast and efficient in predicting targets values. The overall time complexity for GP regression comes out to be O(N^3), a time-consuming procedure. In conclusion, regression forests are most optimal when the data set is of great size, while Gaussian process regression is a best fit when the dimensionality of the problem at hand is overpowering the size of the data set.

3.Toy Problem Formulation

A part of our assignment, we were given the additional task of running our implementations on smaller sized problem, as opposed to the SARCOS data set, a toy problem. This is to be done for validation purposes and to visualise the performance of our regression algorithms. The sole purpose of the toy problem is to try and break what is hypothesised to be unbroken. Therefore, a simple and humanly understandable toy problem was created, where X = 1, 2, 3, …, 100 and Y = X^2, making this a 1D problem with 100 exemplars, out of which 60% were used for training and 40% for testing. Moreover, the data set was shuffled before splitting it to avoid bias from continuation of values and it was done individually for every algorithm and therefore the following graphs are not based on the same shuffle.

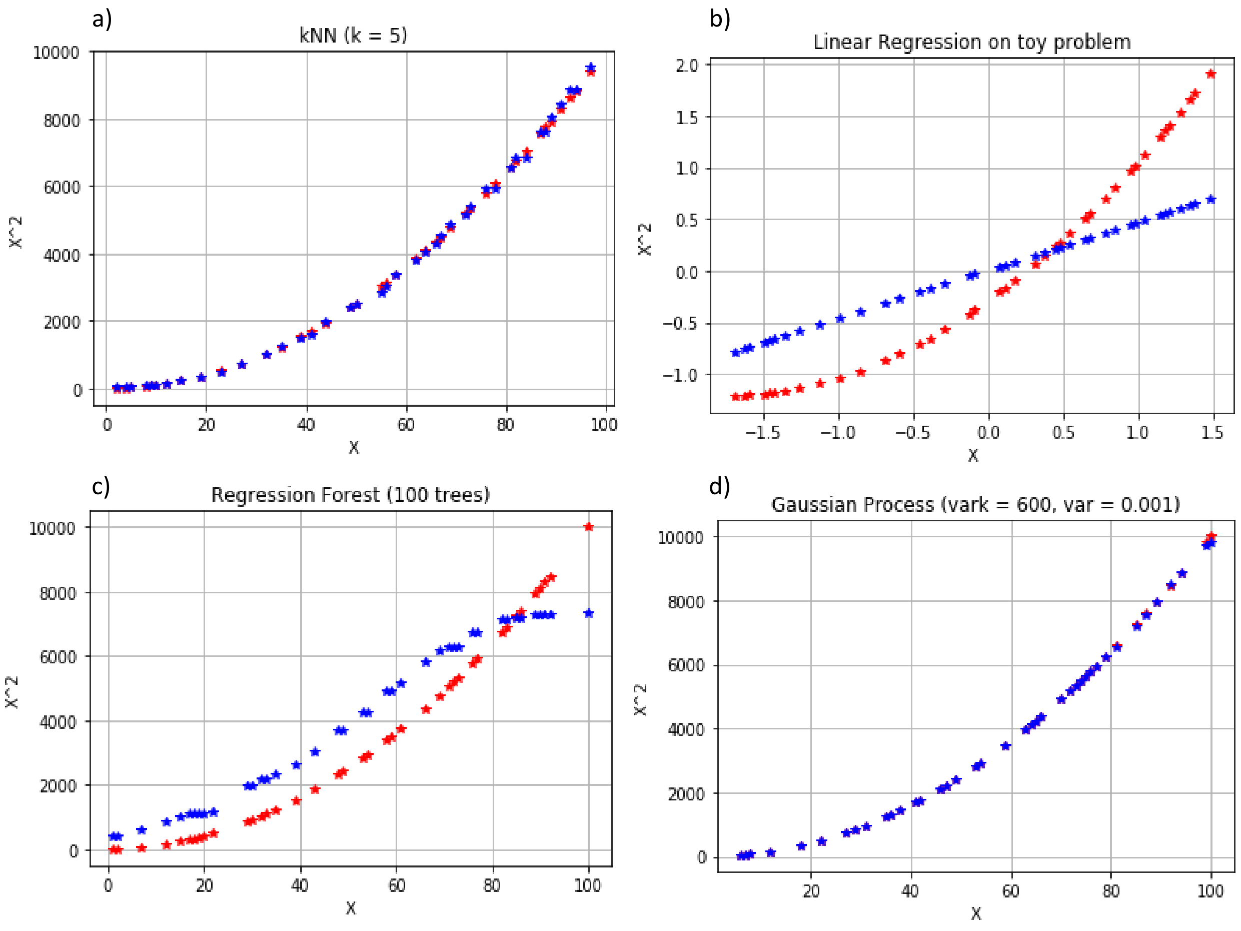


Figure 1: The performance of each of the algorithms implemented on the toy problem using the hyperparameters shown above the graphs. Since the sole purpose of running the algoritms on the toy problem was to validate whether the algorithms performed as expected, hyperparameter tuning was neglected and further studied on the actual SARCOS data set supplied.

The actual outputs for the test data are shown with red, and the predicted values are shown with blue asterisks. The hyperparameters used are noted in the titles of each plot. As you can see, the results are as expected. All the algorithms performed based on their efficiencies for the given data set. For example, regression forest was not as efficient as the kNN since it performs better on a larger data set.

4.Experiments and Analysis

Optimisations are crucial when producing a machine learning classifier or regressor, and they mostly depend on hyperparameter selection, based on the problem at hand, to make any approach optimal and efficient. Additionally, each approach will be put to the test and compared with the opposition algorithms, based on accuracy of predictions and computational complexity.

4.1 Hyperparameter selection strategy

Hyperparameters are values set before the training procedure and are kept constant throughout, but greatly affect accuracy of predictions. Since there is no fixed value for these hyperparameters prior to testing accuracy, and since their value could range from 0 to infinity, a strategy must be applied to avoid unnecessary exhaustive approaches. The first step of deciding hyperparameter values, used during evaluation, was through trial and error of sparsely chosen values. This aided with understanding the span in which these values produced the smallest error in the predictions. Once an interval was decided, then an exhaustive approach was performed with values within that interval. This strategy was followed for hyperparameter selection for each of the algorithms implemented. In order to validate whether a hyperparameter was chosen optimally, an error metric was introduced, which was the same for all algorithms tuned and tested to avoid bias in the constructions of results, the Mean Squared Error (MSE). To provide a valid MSE for the regression forest, each hyperparameter trial was performed three times and an average of their MSEs was calculated, due to the randomness of bagging and feature selection. Additionally, as advised, the Gaussian Process for regression was limited to 10000 exemplars for the training stage. For this reason, all algorithms were tuned and tested using a train data set consisting of 10000 exemplars and a test data set consisting of 5000 exemplars. This would provide an accuracy and run time given the same problem for all algorithms. More details can be seen in Table 1.

Algorithms	Hyperparameters to be tuned	Range of hyperparameter values to be tested
k-NearestNeighboors	k = the number of neighbours accounted for prediction	From k = 1 to k = 21
Linear Regression	a = learning rate num_iter = number of iterations to be performed	From a = 0.1 to a = 0.00001 with each step divided by 10 Kept constant at a large value (80000) since convergence checking was used

Regression Forest	br = Bagging ratio
	
	fr = Feature ratio
	
	max_depth = the max depth of each tree
	
	num_trees = number of trees the random forest will consist of
		*Kept constant at 1/10 of the overall data set with replacement
		
		*Kept constant at 1/3 of the overall features without replacement
		
		*Set to infinity as explained previously
		
		From num_trees = 5 to num_trees = 30

Gaussian Process	Var_k = the variance used to compute the covariance matrix, σ_k^2 Var = the variance used to produce the noise matrix, σ^2	From var_k = 300 to var_k = 600 From var = 10 to var = 0.0001 with each step divided by 10

Table 1: Hyperparameter pre-processing information

** Other than the fact that keeping these hyperparameters constant would reduce the amount of hyperparameters to be tuned, and in turn reduce the time taken for tuning, it was discovered that this is a common practise by many machine learners, (Brownlee, 2018).*

Below are graphs and MSE tables produced during hyperparameter tuning:

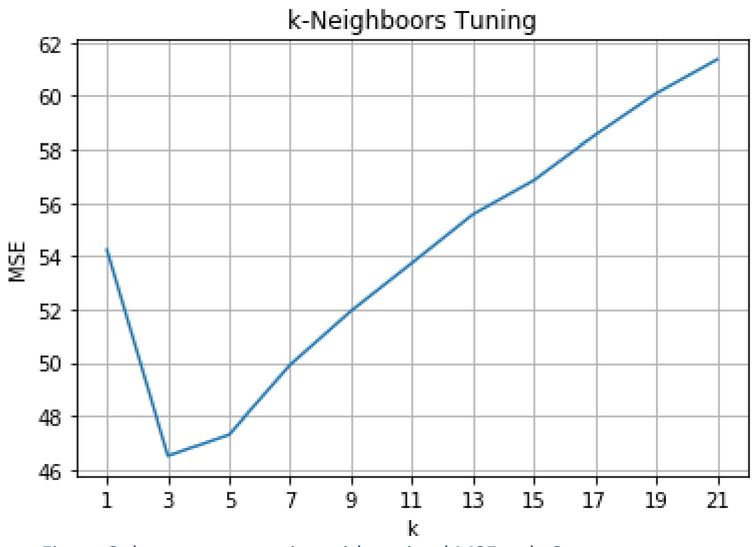
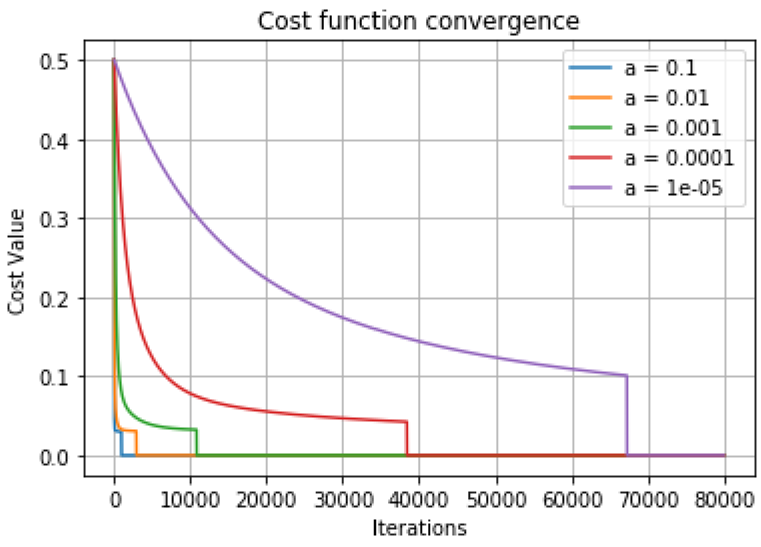


Figure 2: k parameter tuning with optimal MSE at k=3



α	MSE
0.1	0.295
0.01	0.284
0.001	0.262
0.0001	0.236
0.00001	0.262

Figure 3: Cost function convergence during gradient descent for Linear Regression, based on learning rate (a). The MSEs given the learning rates are shown on the table on the right with the best highlighted green.

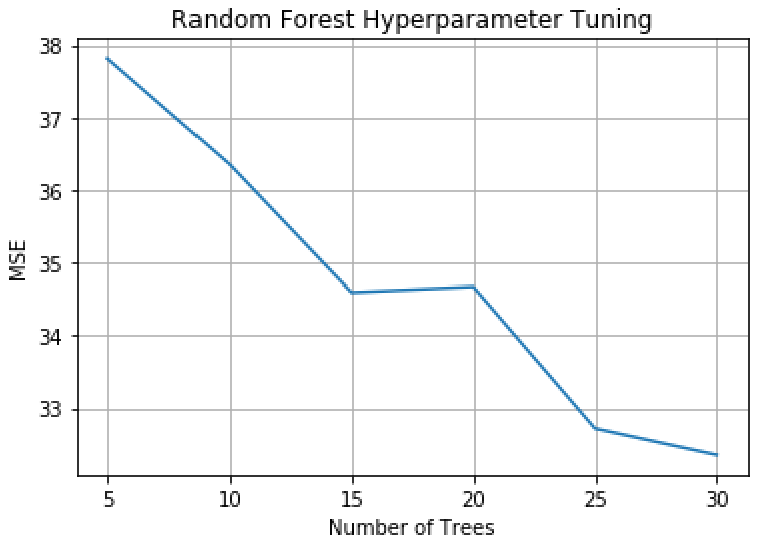


Figure 4: Number of trees hyperparameter tuning for Regression Forest. The MSE keeps decreasing but because of the large dataset could not continue any further.

MSE		σ^2					
		10	1	0.1	0.01	0.001	0.0001
σ^2_k	300	55.37	29.44	14.09	11.21	14.18	23.88
	310	55.59	29.56	14.06	11.13	14.11	23.63
	320	55.82	29.67	14.03	11.05	14.05	23.38
	330	56.04	29.79	14.01	10.97	13.98	23.13
	340	56.27	29.91	13.99	10.9	13.91	22.9
	350	56.49	30.03	13.97	10.83	13.85	22.69
	360	56.72	30.15	13.96	10.77	13.78	22.48
	370	56.94	30.27	13.95	10.71	13.7	22.29
	380	57.17	30.39	13.94	10.66	13.62	22.11
	390	57.39	30.51	13.94	10.61	13.54	21.94
	400	57.62	30.63	13.94	10.57	13.45	21.79
	410	57.84	30.75	13.94	10.52	13.36	21.64
	420	58.06	30.87	13.94	10.49	13.26	21.51
	430	58.29	30.99	13.94	10.45	13.17	21.38
	440	58.51	31.11	13.95	10.42	13.07	21.26
	450	58.73	31.23	13.95	10.4	12.97	21.15
	460	58.95	31.35	13.96	10.37	12.87	21.03
	470	59.18	31.47	13.97	10.35	12.77	20.92
	480	59.4	31.59	13.98	10.33	12.67	20.81
	490	59.62	31.71	13.99	10.31	12.58	20.7
	500	59.84	31.83	14	10.3	12.49	20.58
	510	60.05	31.95	14.02	10.28	12.4	20.47
	520	60.27	32.07	14.03	10.27	12.32	20.35
	530	60.49	32.19	14.05	10.26	12.24	20.23
	540	60.7	32.31	14.06	10.25	12.16	20.1
	550	60.92	32.43	14.08	10.25	12.09	19.97
	560	61.14	32.55	14.1	10.24	12.03	19.84
	570	61.35	32.67	14.11	10.24	11.96	19.71
	580	61.56	32.8	14.13	10.23	11.9	19.57
	590	61.78	32.92	14.15	10.23	11.85	19.44
	600	61.99	33.04	14.17	10.23	11.8	19.3

Table 2: The MSEs produced during hyperparameter tuning of Gaussian Process for regression. On the vertical axis are the values for var_k in equation (5) and on the horizontal are the values of var in equation (8). Optimal value is highlited green.

The hyperparameters producing the least MSE value during hyperparameter tuning for each algorithm can be seen in Table 3.

Algorithms	Hyperparameters to be tuned	Range of hyperparameter values to be tested
k-NearestNeighboors	k = the number of neighbours accounted for prediction	From k = 1 to k = 21
Linear Regression	a = learning rate	From a = 0.1 to a = 0.00001 with each step divided by 10
	num_iter = number of iterations to be performed	Kept constant at a large value (80000) since convergence checking was used
Regression Forest	br = Bagging ratio	*Kept constant at 1/10 of the overall data set with replacement
	fr = Feature ratio	*Kept constant at 1/3 of the overall features without replacement
	max_depth = the max depth of each tree	*Set to infinity as explained previously
	num_trees = number of trees the random forest will consist of	From num_trees = 5 to num_trees = 30
Gaussian Process	Var_k = the variance used to compute the covariance matrix, σ_k^2	From var_k = 300 to var_k = 600
	Var = the variance used to produce the noise matrix, σ^2	From var = 10 to var = 0.0001 with each step divided by 10

Table 3: Optimal hyperparameter values and their corresponding MSEs and run times

4.2 Performance Comparison

Tables 1 and 2 clearly show both accuracy and computational complexity comparison between the 4 algorithms. In terms of accuracy, the best approach came out to be Gaussian Process for regression with a very low MSE of 10.23 taking approximately 1425 seconds to run regardless of the hyperparameters used. When it comes to computational complexity, Linear Regression is the optimal learning algorithm taking only about 50 seconds to run, given that it converges, and with a good MSE of 23.63. Nonetheless, the Regression Forest approach could be optimised by making tree building parallel, which would significantly reduce run time. With this in mind, taking the run time and dividing it by the number of trees, from the information above, we can estimate, depending on the cores available, the average run time for Regression Forest. As for kNN, it produced the least accuracy and ranked last when comparing the run time, due to time complexity of both calculating the distance between all training samples with each of the test samples as well as the sorting procedure. Given the above results, on the given dataset, there is a trade-off between accuracy and complexity that one must decide when choosing the most efficient algorithm. For such a large data set, simple Linear Regression is thought to be optimal, given the significantly small run time and the fairly good MSE it provides.

5. Conclusion

Many more algorithms exist for regression problems, allowing for very good predictions of test samples, other than the algorithms mentioned and examined throughout this report. Nonetheless, the notion of machine learning suggests that optimisations never end; an algorithm can always be further improved. Something not utilised in the implementations above, is cross-validation. With cross-validation one can decrease the chance of overfitting on the training data set and produce more valid results, by further splitting the data set to smaller subsets, similar to how random forest and bagging work. Using the information from the experimental results, one is able to analyse the performance of each of the algorithms and decide on the best approach based on the problem at hand.

6. References

Alexopoulos, E. (2018). Introduction to Multivariate Regression Analysis. [online] Hippokratia. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3049417/.

Bailey, K. (2018). *Gaussian Processes for Dummies* ·. [online] Katbailey.github.io. Available at: http://katbailey.github.io/post/gaussian-processes-for-dummies/.

Brownlee, J. (2018). Bagging and Random Forest Ensemble Algorithms for Machine Learning - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/.

Resources from unit CM50264 webpage were mostly used.