1 Introduction

The problem at hand is applying various regression methods to a given data set. This data set consists of 44,484 exemplars and 21 features, specifically the position, velocity and acceleration for 7 degrees of freedom of a robotic arm. The data set also consists of one target variable, which is the torque of one motor of the robotic arm. As mentioned previously, the task is to apply a range of regression methods to predict the target variable, and thus the torque of one motor. These regression methods are k nearest neighbour regression, linear regression, random forest regression, and Gaussian process regression.

2 Random forest

A random forest combines the two concepts of decision trees and ensemble learning. Ensemble learning is a method that combines multiple estimators. In the case of random forests it is a process known as bagging, which uses the same model and trains on randomised data from the original data set, so each is different.

An important concept for ensemble learning is diversity, the key idea is that the individual estimators should make different mistakes. The average over all mistakes will then tend towards no mistakes. The important insight here is that the errors are not correlated with each other. Therefore, it is often advisable to increase the diversity of estimators at the expense of their performance.

The name random forest comes from the fact that the model trains many decision trees and merges their predictions. This results in a more accurate prediction, compared to relying on single trees. Although each tree may have high variance with respect to a specific sample of the training data, the entire forest will have lower variance, without the risk of increasing the bias.

2.1 Bagging

Having many estimators and merging their predictions is desirable for a random forest, one solution to this would be more data. However, this is not a feasible requirement, therefore a process of bootstrapping is used. This consists of 'faking' more data. Effectively random samples (with replacement) are taken from the existing training set. The size of the sample taken is usually the size of the whole data set. An accuracy estimate is then calculated on the new data set.

The term bagging is short for Bootstrap Aggregating, this refers to the fact that the results of the estimators are aggregated. The general algorithm would be to choose an ensemble size, which will be the number of bootstrap samples taken. An estimator is then calculated on each of the samples, and the estimators are combined.

This was the original idea of the random forest. An extension of this concept which is now the standard procedure is to also apply bootstrapping to the features of the data set. This is known as the random subspace method. Where each decision tree node is split with respect to a subset of the available features.

Thus, the complete algorithm for a random forest is to select the ensemble size, which is the number of decision trees to train. Bootstrap that number of samples from the original data set, and then train a decision tree on each sample, using the random subspace method to split on features. The results of all the trees are then combined. In terms of regression this is the mean / median of all the estimates, the mean is usually more useful.

A good default for the random subspace method is to select a subset which has the size of the square root of the number of features. Further hyperparameter optimisation can then be done to improve the performance of the random forest. For further optimisation it should be noted that more trees are always better for the accuracy of the model, therefore the amount of trees should be chosen only so that it is large and computationally feasible. Moreover, deeper trees generally improve the accuracy of the model, with the condition that the amount of trees must increase too. In terms of bias and variance, the main concept of random forests is that many decision trees are combined, which are high variance and low bias. This then generates a random forest which is low bias and low variance. This helps avoid over fitting or under fitting, as each tree sees different data, and each tree has low bias.

2.2 Advantages and disadvantages

Random forests are useful in many different scenarios. They work both on classification and regression data and require no feature scaling or normalisation, as well as being able to handle binary, categorical, and numerical features. The bias also remains the same as that of a single decision tree. However, the variance decreases and thus we decrease the chances of over fitting.

Another advantage of random forests is that they perform feature selection, and thus provide information about feature importance. Furthermore, random forests are relatively efficient, even with very high dimensional data, this is due to the fact that we are often only considering a subset of the number of features of the original data set. Random forests are also 'embarrassingly parallel' as the trees are independent, which can further decreases the computation complexity of them.

Random forests do have drawbacks, for example they do not train well on small data sets, this is due to the fact that they fail to recognise patterns, unlike Gaussian process regression or linear regression. It is true that feature importance can be understood, however, there is no specific interpretable relationship between independent variables and the target variable, and as a result random forests are very hard to interpret. Moreover, they can have a large space complexity and be slow to evaluate, even in parallel.

3 Gaussian process

A Gaussian process is non-parametric much like k nearest neighbours in this sense. This can be understood when compared to linear regression. Instead of attempting to find the correct parameters for a linear regression $y = wx$. We instead try to find the correct distribution over functions of the data. Thus, a Gaussian process is a collection of random variables, where any finite subset of those random variables have a joint Gaussian distribution.

A Gaussian process is defined by a mean function, and a covariance function. The covariance function speaks of the covariance between pairs of random variables. The mean function $m(x)$ and the covariance function $k(x, x')$ are defined as:

$$\begin{aligned} m(x) &= E(f(x)) \\ k(x, x') &= E(f(x) - m(x))(f(x') - m(x')) \end{aligned}$$

For any set $X = x_1, \dots, x_N, f_X = f(x_1), \dots, f(x_N)$ is a Gaussian random vector characterized by the mean vector μ_x and the covariance matrix, known as the kernel, K_x.

$$\begin{aligned} \mu_x &= [m(x_1), \dots, m(x_N)] \\ K_x : [K_x]_{i,j} &= k(x_i, x_j) \end{aligned}$$

Many different kernels can be used and they all have different hyperparameters and properties. One of the most common kernels is the squared exponential covariance:

$$\exp\left(-\frac{1}{2\ell^2}(x_i - x_j)^2\right)$$

with the hyperparameter ℓ of the length scale. It should be noted that the one condition of the kernel is that it must be positive-definite.

We often assume a zero mean Gaussian process prior on the function f. Such that for any set $X = \{x_1, ..., x_N\}$, $f_X|X$ is a Gaussian random vector:

$$p(f_X|X) = \mathcal{N}(0, K_x)$$

Independent and identically distributed noise is also place on y:

$$p(y|X, f_X) = \mathcal{N}(f_X, \sigma^2 I)$$

Due to these results it is possible to obtain the joint distribution of y and $f(x')$:

$$\begin{pmatrix} y \\ f(x') \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} (K_X + \sigma^2 I) & k \\ k^T & k(x', x') \end{pmatrix}\right)$$

with $k = [k(x', x_1), ..., k(x', x_N)]^T$

Using the Gaussian conditional formula the posterior is obtained:

$$p(y'|x', y, X) = \mathcal{N}\left(k^T(K_x + \sigma^2 I)^{-1} y, k(x', x) - k^T(K_X + \sigma^2 I)^{-1} k\right)$$

To then get the predicted values of the data, it is usually enough to take the mean from the posterior.

3.1 Calculations

Performing these calculations in the manner as described would be time consuming and not feasible with large data sets. Therefore with python and numpy we can use Cholesky decomposition. This makes the calculation a lot faster and more numerically stable.

3.2 Advantages and disadvantages

A Gaussian process differs from the other regression methods discussed as it defines a predictive distribution. Therefore it is possible to understand uncertainty of the model. This is useful as it is possible to have confidence intervals for predictions. Another advantage is that there is the opportunity to add prior knowledge about the data in order to influence the predictive accuracy. Furthermore, Gaussian process regression can be very accurate with small data sets, with or without high dimensions. However, the biggest disadvantage of Gaussian process regression is the computational complexity, which scales cubically as the number of exemplars increase. Therefore working with large data sets is infeasible.

4 Validation on a toy problem

Validating on a toy problem that can be easily visualised helps confirm that the regression algorithms have been correctly implemented. The toy problem that acts as the underlying function of the data is a sine function $f(x) = sin(0.9 * x)$. It is important to note that there is no noise, and therefore the expected outcome is known. There will be one feature, and one target variable. The rationale for choosing this toy problem is that it gives a clear indication of whether the algorithms are working, and is not so trivially simple that a failed prediction could be mistaken for a successful one. Values are generated between -5 and 5 for the data of the feature variable, and the given function is applied to obtain the values of the target variable. A plot of the whole data set can be seen in Figure 4.1.

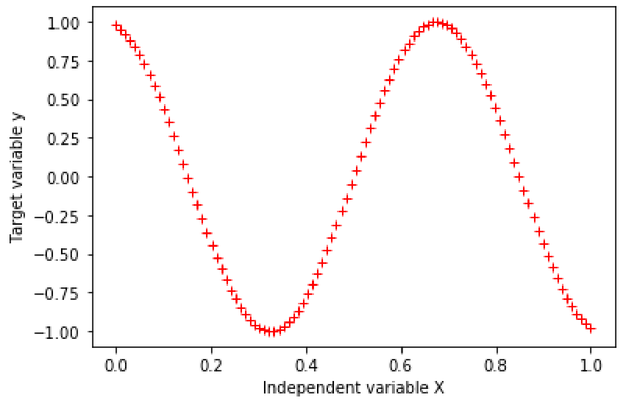

Figure 4.1: The whole data set plotted.

4.1 K nearest neighbours regression

To see if the implementation of the K nearest neighbours works as intended, the predictions for the target variable that the model makes can be plotted, and then compared with the actual target variable values.

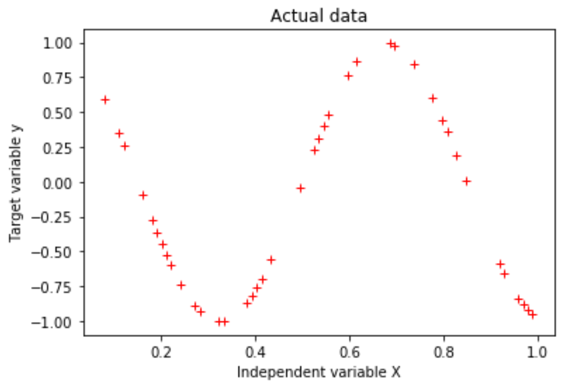

Figure 4.2: The actual data of KNN

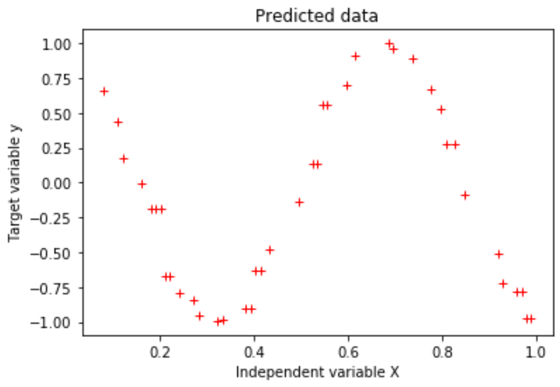

Figure 4.3: The predicted data of KNN

When comparing Figure 4.2 and 4.3 it is clear that the model is correctly predicting the target variable data. The model uses k-fold cross validation to discover the best k value, and once it has been found the target variables of the unseen test set are predicted, to obtain the result in Figure 4.3.

Furthermore, we can plot the line of best fit for the actual values of the target variable against the predicted values, which is shown in Figure 4.4.

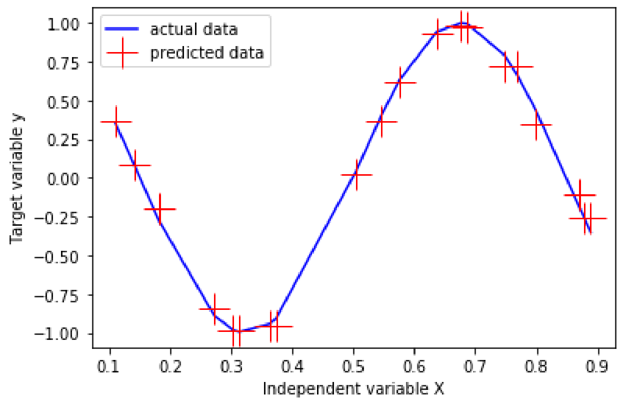

Figure 4.4: Comparison between actual and predicted data for KNN

4.2 Linear regression

A Linear regression model on a sine function such as the one in the toy problem should not perform significantly better than the mean. The reason for this is that the sine wave is non-linear and thus can not be fit to a linear equation. This is shown in Figure 4.5 and Figure 4.6, where the mean has been used to predict, and is compared with the predicted data from the linear regression model.

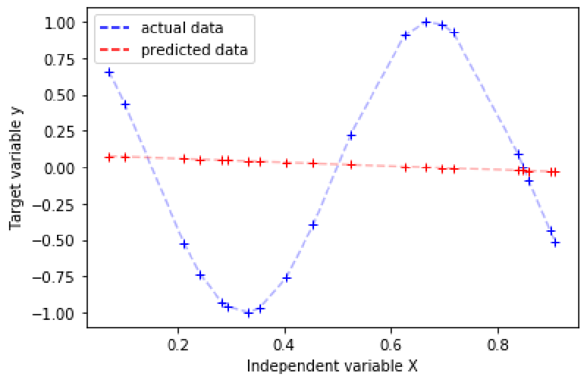

Figure 4.5: The actual against the predicted data for LR

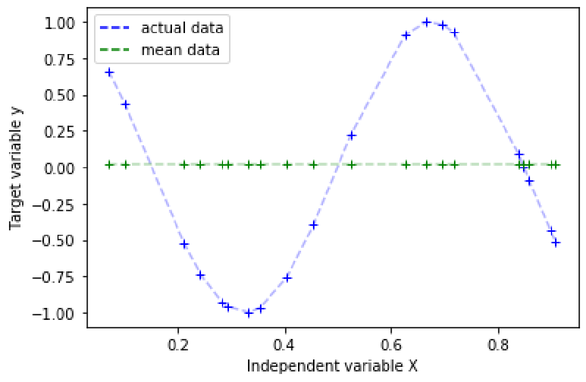

Figure 4.6: Using the mean of y_train as a model for LR

If instead a linear model was used, such as $f(x) = 2x + 4$, we can see that the prediction is much more accurate in Figure 4.7. Therefore the implementation of Linear regression on the toy problem is working as expected.

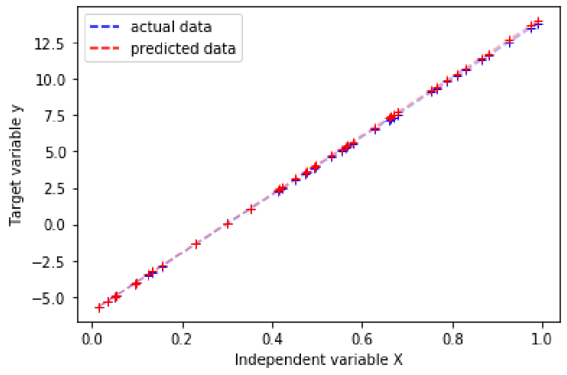

Figure 4.7: The actual against the predicted data for a linear function for LR

4.3 Random forest regression

With the random forest Figure 4.8 shows that the predictions are accurate, however not completely accurate. This is due to the low amount of data I initially gave the model to train on. This makes sense since random forests struggle to see patterns on smaller data sets. If the training data size is increased, we see that the prediction is near perfect in Figure 4.9.

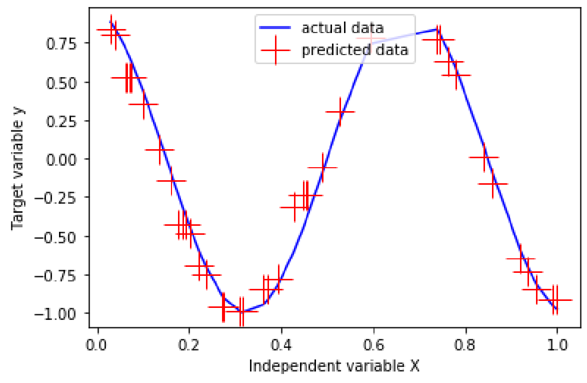

Figure 4.8: The actual against predicted data for small training set

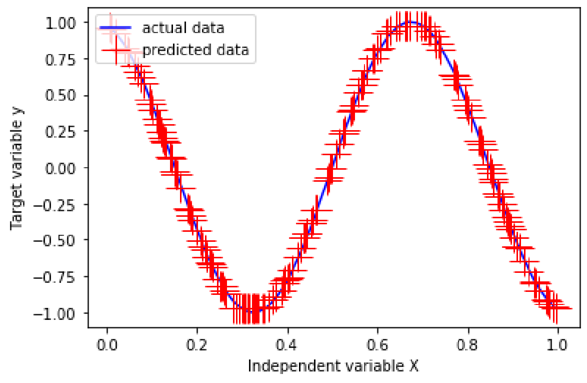

Figure 4.9: The actual against predicted data for larger training set

4.4 Gaussian process regression

To implement a Gaussian process on the toy problem, the prior is first defined over functions. Samples can then be taken from that prior. So using the squared exponential kernel, samples can be drawn to obtain Figure 4.10. The samples are plotted with the assumed mean of zero.

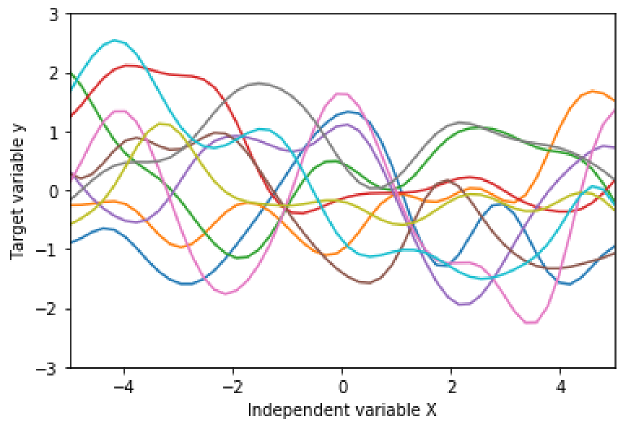

Figure 4.10: Samples from the Gaussian process prior.

The mean and covariance of the posterior distribution are then calculated and applied to the training data, and samples can be drawn from the posterior to obtain Figure 4.11. If the calculated mean is then taken as the prediction value for the values of the target variable of the test set, it's found that it is extremely accurate, as seen in Figure 4.12. As a result it is obvious that the Gaussian process is working as intended.

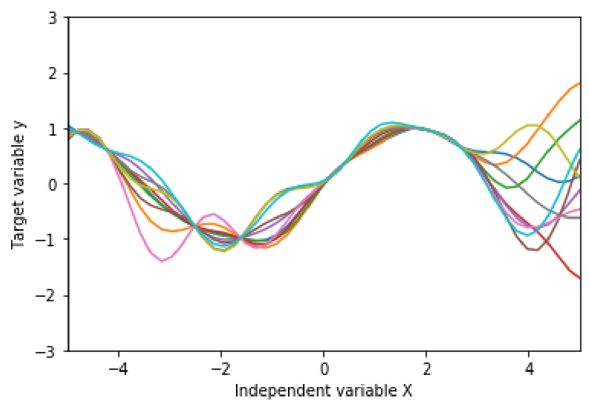

Figure 4.11: Samples from the Gaussian process posterior

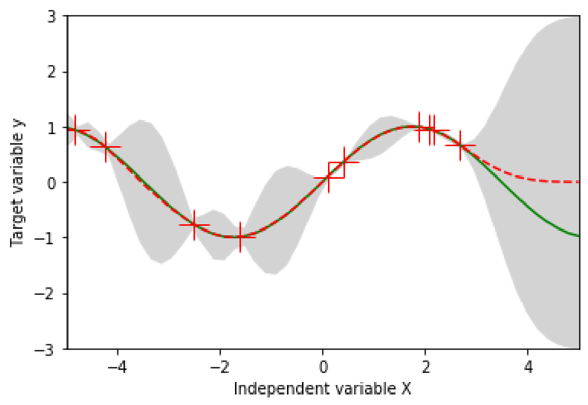

Figure 4.12: The actual data against predicted data for Gaussian process regression, including 95% confidence intervals

5 Experiments and Analysis

Training, verification, and testing sets have been used for some regression methods, while k-fold cross validation was used for others. This is so that data is trained with the training set, and the hyperparameters are tuned either with cross validation or a verification set. Finally the completely unseen testing set is used to measure the performance of the model.

5.1 Hyperparameter selection

5.1.1 K nearest neighbours regression

K nearest neighbours with regression works by selecting the k nearest neighbours of a test data exemplar and making the prediction for that exemplar be the average of the k nearest neighbours. There are different strategies that can be used, for example different meanings of 'closest'. In my implementation the distance matrix is calculated using Euclidean distance, otherwise known as the L2 norm. A method to calculate this in a more efficient way, rather than the naive method of looping through both the training set and testing set is to factorise the calculation to:

```
np.einsum('ij, ij -> i', X_test, X_test)[:, None])
    + np.einsum('ij, ij -> i', X_train, X_train)
    − 2 ∗ X_test.dot(X_train.T)
```

The hyperparameter for the k nearest neighbours algorithm is the amount of k neighbours to consider. The k value that results in the highest accuracy for the model is desired. Therefore the root mean squared error (RMSE) is iteratively calculated for the different k values. The best k has the lowest RMSE, and this is the k value that is then used for predicting the target variable values of the test set. The results can be seen in Figure 5.1. The optimal value is the 'elbow curve', which is when $k = 3$.

I also implemented k-fold cross validation with the algorithm to improve the accuracy of the model. The results of this can be seen in Figure 6.1 in the appendix.

5.1.2 Linear regression

I implemented gradient descent for linear regression, this is an optimisation algorithm that is used to find the parameters of the function that has the highest accuracy in terms of minimising a loss function. The parameters are iteratively redefined to get the 'optimal parameters'. Therefore the hyperparameters are the learning rate and the number of iterations to perform. The learning rate is effectively the step size taken

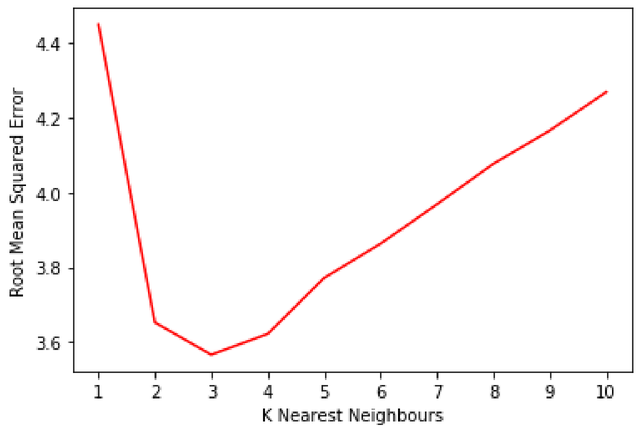

Figure 5.1: The K nearest neighbours against the RMSE

each iteration.

The algorithm has been implemented so that it measures the error of the prediction at each iteration, this ensures the error is always decreasing. If the error does increase, the step size will be reduced. This dynamic implementation means there is less need to decide on a fixed learning rate, as it will decrease if necessary. Various initial learning rates were tested from 0.001 to 0.3, but they all resulted in convergence.

The other hyperparameter is the number of iterations, in my implementation I have also introduced a convergence check, and this greatly reduces the computational complexity. As can be seen in Figure 5.2, both the methods converge to the same RMSE. Furthermore, without the convergence check the model takes significantly longer to train.

	RMSE	Time (seconds)	Max iterations
With convergence check	5.5688283358992825	4.05	100,000
Without convergence check	5.5688283359009025	278.49	100,000

Figure 5.2: Comparison between checking for convergence and not, for linear regression.

5.1.3 Random forest regression

The main hyperparameter to consider for random forest regression is the number of features that will be split at each node of a decision tree. The size of the sample when bootstrapping, and the maximum depth of a tree can also be hyperparameters.

The amount of decision trees does not technically need tuning as a hyperparameter, this is because a random forest is ‘embarrassingly parallel’, in the sense that each tree in a random forest is identically distributed. As a result the model’s loss decreases as the amount of trees increases. Therefore the only tuning that is needed is setting the amount of trees to a large and computationally feasible number.

However the model does benefit from tuning the amount of features to split on each node. The standard value is the square root of the number of features. I performed a grid search to find a suitable number of features, as shown in Figure 5.3 and Figure 5.4.

Feature importance plays a role with random forests, if there are uninformative features then the influence of the informative features can be diluted. Furthermore if there a large number of features then the computational complexity becomes much greater. As shown in Figure 5.3 the optimal number of features was

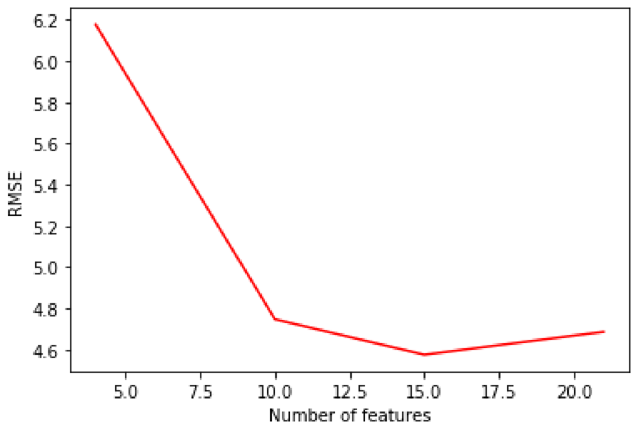

Figure 5.3: The number of features against the RMSE

Number of features	Root Mean Squared Error (RMSE)	Time taken (seconds)
12	4.74232	275
13	4.74598	309
14	4.72458	361
15	4.70033	402
16	4.71671	524
17	4.69802	448

Figure 5.4: Grid search of the number of features for the random forest.

found to be 15. However the marginal increase in accuracy causes a large increase in time taken, as shown in Figure 5.4. Allowing all the features decreased the accuracy of the random forest.

5.1.4 Gaussian process regression

The issue with a Gaussian process on this data set is that the computational complexity scales as $O(N^3)$, with N exemplars. Steps were taken, such as using the Cholesky decomposition to reduce computational complexity and improve the numerical stability. However, even so it was not possible to use the whole data set in one calculation due to the large matrix calculations needed. Therefore I used a form of bagging, where a Gaussian process is applied to smaller random samples of subsets of the original data. The results are then aggregated.

Length scale ℓ	RMSE
0.9	3.11557
5	2.51732
10	3.02673
15	3.36034
100	4.06417

Length scale ℓ	RMSE
2	2.41754
3	2.42259
4	2.52884
5	2.62153
6	2.71800

Figure 5.5: The length scale in relation to RMSE

The hyperparameters of a Gaussian process are dependent on the kernel being used. My implementation used the squared exponential. The length scale is a hyperparameter which determines the width of the kernel and how smooth the function is in the model. The best method to find the optimal length scale is to maximise the marginal likelihood, my implementation uses grid search to compute the length scale as shown in Figure 5.5. Once the optimal length scale been found, bagging is used with the unseen test set.

5.2 Comparing results

	RMSE	MAE
K nearest neighbours regression	3.65968	2.24573
Linear regression	5.53024	3.93417
Random forest regression	4.70342	3.38533
Gaussian process regression	2.65199	1.85054

Figure 5.6: Comparison of RMSE of regression methods.

Clearly the most accurate regression algorithm on the data set was Gaussian Process regression. There was difficulty with applying the regression method to the whole data set as it has $O(n^2)$ storage complexity and $O(n^3)$ time complexity. However using a bagging method resulted in the model being able to predict the unseen test data very accurately.

The second most effective regression algorithm was the k nearest neighbours. Interestingly this algorithm has similarities with Gaussian process regression, in the aspect that they are both interested in 'nearness', and that nearby data is important, and distant data is not. A potential reason for the poorer performance is that often the nearest neighbours algorithm has to deal with the curse of dimensionality with high dimensions, as the data can become sparse, tending towards the worst case where the average is as distant from a point as a neighbour.

Unlike Gaussian process and k nearest neighbour regression, linear regression is parametric. It assumes that the data is linearly distributed. This assumption means that it can not perform optimally on data with a different distribution, which seems to be the case with the given data set. However, it should be noted that linear regression had the best time complexity, and would therefore be useful if there was a constraint regarding this. While random forest regression had an extremely slow time complexity, in correlation to the number of trees, number of features selected, and maximum depth. It did however perform better than linear regression.

When comparing the results, Root Mean Error Squared (RMSE) and the Mean Absolute Error (MAE) are used. RMSE gives more weight to data that is further away, as a consequence of squaring, while MAE gives equal weight. With both these loss function all the regression method performances are ranked in the same order, supporting the analysis.

6 Appendix

K nearest neighbours	Cross validation RMSE	Average RMSE
k=1	4.722079056181955	4.7901117776748965
	4.837369245034528	
	4.7409461198649145	
	4.779716566070401	
	4.8704479012226845	
k=2	4.022495425214058	4.10535666865724
	4.061243991005536	
	4.009595390821807	
	4.2441461851786855	
	4.189302351066115	
k=3	4.00962524347925	4.0729340707583965
	4.0144499740943695	
	3.9792704207425875	
	4.18804346439506	
	4.173281251080716	
k=4	4.076102810659009	4.139714597980969
	4.1009888691620295	
	4.07668664709789	
	4.219359486136465	
	4.225435176849451	
k=5	4.21567805104086	4.226472859951608
	4.1680198329154035	
	4.140472234027918	
	4.298588565504249	
	4.30960561626961	
k=6	4.312105055347549	4.338110606807941
	4.2770869933810545	
	4.239722061760048	
	4.424083009761295	
	4.437555913789763	
k=7	4.393514214019737	4.451641622278561
	4.401193939871056	
	4.376386496645402	
	4.517166527739492	
	4.569946933117119	
k=8	4.500818743461401	4.560673335847057
	4.551126403440807	
	4.49843934887534	
	4.58846850217481	
	4.664513681282929	
k=9	4.610824017968063	4.686394590046139
	4.67214023331174	
	4.660680405757446	
	4.686036978713148	
	4.802291314480295	
k=10	4.71821350290231	4.80888581165281
	4.793982031052234	
	4.791541971014728	
	4.797451517687116	
	4.9432400356076664	

Figure 6.1: Cross validation RMSE results for k nearest neighbours.