Regretion Forests & Gaussian Process

UNIVERSITY OF BATH

DEPARTMENT OF COMPUTER SCIENCE: CS50264
CANDIDATE NUMBER: 02151
08/01/2018

1 Introduction

Regression problems can be described as making predictions based on a known data set, D, of n data points,

$$D = (\boldsymbol{x_i}, y_i)|i = 1, ..., n, \boldsymbol{x_i} \in X, y_i \in \mathbb{R} \quad (1)$$

where the inputs are vector $\boldsymbol{x_i}$ and the targets are y_i. The new predictions, y_*, we wish to make are from new input vectors $\boldsymbol{x_*}$. These problems can be described by an underlying function. In many cases we can expect this function to be linear in nature and make assumptions about the input data.

The simplest approach is to use a straightforward method, such as least-squares, to fit a straight line to the data (linear regression). Alternatively, if the underlying function is suspected to be more complex, such as quadratic or non polynomial, other approaches can be used to model the function and predict target values. Regression problems can be even more complex when data points are noisy, as is common with most real world problems and measured data sets. Gaussian process (GP) regression allows a finer approach to the regression solution. Rather than claiming a single underlying function, GPs extend multivariate Gaussian distributions to infinite dimensions. Formally, any finite subset of the data range follows a multivariate Gaussian distribution [1].

In this report, four regression methods will be discussed, implemented, and critically analysed with a simple toy problem and the more complex SARCOS data set [2]. Approximately 8000 training and 900 test exemplars will be used, each with 21 features to predict the target value. The methods used will be as follows:

- Linear Regression
- K-Nearest Neighbours
- Regression Random Forest
- Gaussian Process Regression

The report will conclude with a comparison of the various methods effectiveness on different types of problems.

2 Methods

2.1 Linear Regression

Arguably one of the most understood algorithms in machine learning, linear regression assumes linear relationships between input variables and a single output. The most common method used is the ordinary least squares method, the gradient descent method is also common and is presented here.

With more than one input variable the values of the coefficients used in the regression process can be optimized by iteratively minimizing the error of the model with the training data. This process is known as gradient descent. Starting with estimated (or random) values for each coefficient, the sum of the squared errors is calculated for each pair of inputs and outputs. A hyperparameter of learning rate is used to control the iterative change of the coefficients and the process is repeated until a minimum error is found.

Using the ordinary least square method ro minimise the sum of squared residuals, it is possible to find a closed form solution for the coefficient vector, B, as follows.

$$B = (X^T X)^{-1} X^T y \quad (2)$$

where X^T indicates the transpose of the input data. B can then be used to predict y_* for new $\boldsymbol{x_*}$ as,

$$y_* = X_* B + B_0 \quad (3)$$

where B_0 indicates the error term and is returned as the first row in B in the implemented code. In large multivariate problems, its unlikely a closed form solution exists, however in the SARCOS data set a closed form solution can be found and it is this solution that will be implemented and used for comparison with the other methods.

2.2 K-Nearest Neighbors

Unlike the other methods presented in this report, the nearest neighbor algorithm does not learn, instead predictions are made using the entire dataset directly. Predictions are made for each input by searching through the whole training set for the most similar data points and providing a summarized output for the k nearest neighbors (where k is a hyperparameter).

For the implementation presented here the distance measure used if the Euclidean distance,

$$Dist(x_*, x_i) = \sqrt{\sum_{j=1}^{n}((x_{*j} = x_{ij})^2)} \tag{4}$$

where x_i is an existing point across all inputs j. Alternative measures of distance are used for various applications, for example the Hamming distance is often used for text classification.

The prediction for the target value, y_*, is then found by taking the weighted average of the target values, y_i. In the implementation in this report, the weighting is based on the distance from of the x_i values from the x_* values. The weighting could alternatively be calculated by $1/k$ where k is the neighbour position.

2.3 Regression Random Forest

First created by Tin Kam Ho [3], Leo Breiman extended the early algorithm into the modern random forest algorithm by combining bagging and random feature selection [4]. Random forests construct multiple decision trees from the training data and output based on a mean prediction of the individual trees.

The method implemented in this report functions by creating the each tree in the forest by first bagging (creating a random sample of the training set to fit a tree to) and then building a tree with the bagged sample. The tree has the following hyperparameters; maximum depth, minimum size, number of features to consider.

The choice of best split for each tree is found by choosing the split which minimises the residual sum of squares (RSS). This is a standard error measure used for linear regression and in the case of S splits is given as:

$$RSS = \sum_{s=1}^{S}\sum_{i}(y_i - y_{R_s})^2 \tag{5}$$

By summing across all partitions and observations, i, in each partition, the squared difference of the response, y_i, and the mean of the responses in the partition, y_{R_s}. The algorithm is known as a greedy algorithm as it carries out the evaluation for each iteration of the recursion rather than the full set and continuing to branch before making a final evaluation of the splits.

Predictions are then made by passing x_* through each tree in the forest and selecting the mean of all the trees outputs.

2.4 Gaussian Process Regression

Multivariate Gaussian distributions can define a distribution over a finite set of random variables. An extension of this is a Gaussian process (GP), which can define a distribution over an infinite set of random variables [1]. A GP is specified by its mean, μ, and covariance, k. Both μ and k are functions,

$$\begin{aligned} \mu &: X \to \mathbb{R} \\ k &: X \times X \to \mathbb{R} \end{aligned} \tag{6}$$

We assume $\mu(\boldsymbol{x}) = 0$ for all $\boldsymbol{x}$ as we can shift the dataset for any given mean. This means the covariance function completely defines the behaviour of the process.

The covariance function (often referred to as the kernel) is the modelling problem we are trying to solve. Choosing a suitable covariance function is analogous to choosing a suitable prior. One example of a commonly used kernel is the squared exponential kernel:

$$k(x, x^{'}) = \theta_0 exp\left(\frac{-(x - x^{'})^2}{2\theta_1^2}\right) \tag{7}$$

where k is the kernel function and $theta$ is a hyperparameter. The exponential allows the function to look smooth for similar neighbours. However, if x and $x^{'}$ are distant from each other $k(x, x^{'})$ approaches zero, meaning the two points cannot 'see' each other and thus do not influence each other. This means the process is entirely dependent on the euclidean distance between x and $x^{'}$.

A second kernel example is that of the linear kernel,

$$k(x, x^{'}) = x^T x^{'} \tag{8}$$

where x^T is the transpose of x. A third is the periodic kernel:

$$k(x, x^{'}) = exp\left(-\frac{2sin^s(\frac{x - x^{'}}{2})}{\theta^2}\right) \tag{9}$$

All three of these kernels will be implemented and the results compared to show the effect of choosing different kernels.

Once the covariance function has been found, the full Gaussian function of the data set can be found via single value decomposition. What remains is a conditional probability function of $p(y_*|\boldsymbol{y})$, given the training data observations, $\boldsymbol{y}$, how likely is a particular prediction for y_*. This probability follows a Gaussian distribution (fully derived in [1]), the result of this is the estimate for the y_* is the mean of the distribution.

$$\bar{y}_* = K_* K^{-1} \boldsymbol{y} \tag{10}$$

where K is the kernel function found from the training data, K_* is the matrix of the kernel functions

for x_* and all x_i. The uncertainty in this estimate is found from the variance which also introduces the need for the kernel function of x_* with all other x_*:

$$var(y_*) = K_{**} - K_* K^{-1} K_* \tag{11}$$

3 Validation

To test the algorithms behaviours, a simplified toy problem is used. In this report two problems will be used to assess the behaviour of the algorithms on different types of data set. Both toy problems are three dimensional data sets with two inputs and one output. The first is a plane described by

$$y = x_1 + x_2 \tag{12}$$

where there are 500 random x values between 5 and -5 for each input. Figure 1 shows the data set in full. The second toy data set is a superposition of two waves described by

$$y = (sin(x_1) + cos(x_2))/2 \tag{13}$$

again with 500 random x values between 5 and -5 for each input. Figure 2 shows the data set in full.

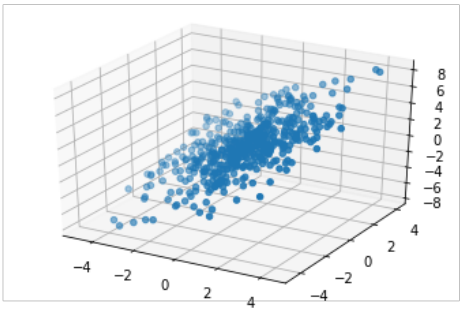


Figure 1: A graphical display of the data created for the plane toy problem described by equation 12.

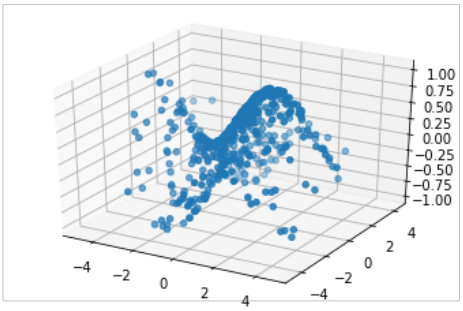


Figure 2: a graphical display of the data created for the wave toy problem described by equation 13.

These two problems were chosen due to their simple form as well as the reduced dimensionality. Two problems were chosen over one to test the algorithms behaviour on different types of data. The plane toy problem should be predicted well by algorithms that deal with linear relationships well, whereas the wave toy problem was chosen to test algorithms on a periodic, nonlinear function.

3.1 Validation Results

What follows is the performance of each of the algorithms implemented on the toy problems. Accuracy and will be assessed as well as any issues with the specific implementations.

3.1.1 Linear Regression

Predictably, the linear regression algorithm was significantly more accurate on the plane toy problem, with a root mean square error (RMSE) of 4.1×10^{-16}, as opposed to the wave toy problem, which had a RMSE of 0.45.

The difference in RMSE indicates that the algorithm is performing as intended. The linear model perfectly fits the plane data whereas the periodic wave data will have a linear regression model fit to a plane within the dataset leading to an overall error that is significantly higher.

3.1.2 K-Nearest Neighbors

For the toy problem validation a fixed value of the hyperparameter k is chosen with $k = 6$. Accuracy for both toy problems was much more similar than in the linear regression case with the plane problem producing a RMSE of 0.17 and the wave problem an RMSE of 0.09.

In the case of the nearest neighbor algorithm, similar accuracies can be expected due to the full training set being used and the algorithm choosing the most similar points. To validate the functionality of the algorithm here, it will be run again with a larger training set. As all training points are used in the prediction of target points, a larger training set should reduce the RMSE if behaving correctly. With a training set of 700 used the plane RMSE was found to be 0.12 and the wave RMSE 0.07, thus validating its functionality.

3.1.3 Regression Random Forest

For the validation of the random forest algorithm, fixed hyperparameters were used of; max depth of 5, minimum size of 1, sample size for each tree of 0.01 of the dataset, 1 feature considered at each split, 100 trees in the forest. This is iterated 3 times to get an average RMSE for the forest. The plane RMSE was found to be 2.2 and the wave RMSE

0.39. Random forests are significantly better at capturing non-linear features and so the better RMSE for the wave problem is expected. To confirm the random forest is functioning as expected, a larger sample size for each tree should reduce the RMSE for the two problems. With a sample size of 0.5 of the training data per tree the RMSE for the plane and wave problems becomes 1.2 and 0.34 respectively. The reduced RMSE with larger sample size is indicative of the behaviour of a random forest and thus validating the algorithms functionality.

3.1.4 Gaussian Process Regression

The Gaussian process is best viewed graphically. Figure 3 shows the output from the Gaussian process algorithm for the plane and wave problems.

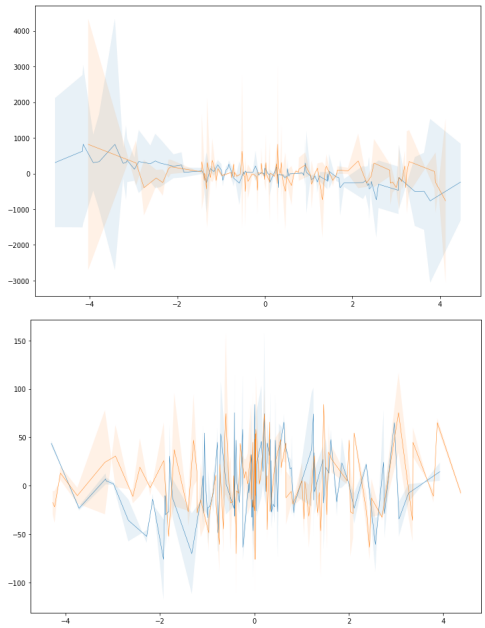


Figure 3: The output graph for a plane toy problem using the linear kernel (top) and the wave problem using the periodic kernel (bottom). The two predictions are indicated by the blue and orange lines with the shaded region indicating the error.

It is clear by looking at the output graphs that the Gaussian process algorithm is not functioning as intended, the predicted values for each input value are exceeding the range that the values exist in by a large amount. The toy data set target values do not exist beyond 1 and -1 and yet the predicted values in the plane problem with the linear kernel extended over 1000. It is unsure why this is the case as the output with the SARCOS dataset and exponential kernel appears much more successful. This will be discussed later in the results section, however, it can be concluded that there is significant issues in the implementation of the Gaussian process.

4 Results

The results of the algorithms with the SARCOS data set will be presented in this section for each of the algorithms. Both accuracy and speed will be discussed for various hyperparameters and data set sizes. Unless otherwise specified the training data set is the same random sample of 8007 exemplars and 890 test.

4.1 Linear Regression

The linear regression method, as implemented here, uses the closed form solution. As such is computationally very fast when calculated in a vectorised way. The resulting time taken for prediction is only 0.013 seconds and is by a large margin the fastest algorithm.

As discussed before the linear regression method is not as effective at considering non-linearities in datasets, however, the method achieved a respectable RMSE of 5.67. This suggests that the dataset has a significant linear relationship.

4.2 K-Nearest Neighbor

With this algorithm the speed is entirely dependent on the size of the data set and is independent of the choice of k. With this data set the time is uniformly 109 seconds which renders this method one of the slower methods.

Optimisation of this method relies on choosing the best value for k which varies based on the data set. An optimal value of 5 was found for this data set, this is shown in table 1.

k	RMSE
1	7.76
3	6.52
4	6.48
5	6.38
6	6.44
10	6.75

Table 1: Various k values showing an optimised value of $k = 5$ for this set of data.

4.3 Regression Random Forest

The speed of this algorithm is highly dependent on the hyperparameters chosen. The size of the sam-

ple to use for each tree causes the greatest change to the speed, this is seen in table 2 below.

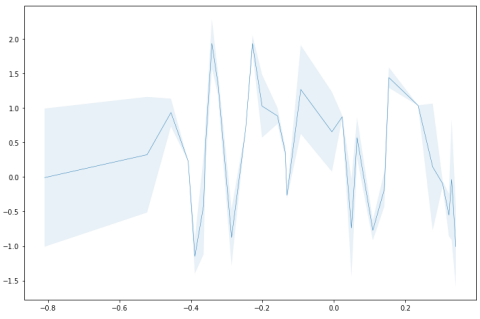
The number of features chosen was $f/3$ where f is the total number of features. The RMSE appears to not change much regardless of the hyperparameters chosen, as well as having a relatively high RMSE. Larger tree depth, smaller sample size and more trees do slightly reduce the RMSE but the reduction is minimal and the RMSE does not improve much beyond 20. This could be an error with the implementation of the algorithm, however, considering the correct functionality was seen with the toy problem it is more likely that the algorithm does not perform particularly well with the data set.

Depth	Sample	Trees	RMSE	Time (s)
5	0.01	10	21.9	1.5
5	0.01	100	20.8	15
10	0.01	100	20.3	18
10	0.005	100	19.9	5
10	0.005	200	19.7	10

Table 2: Various hyperparameter combinations showing the very minor changes in RMSE.

4.4 Gaussian Process

Using a small sample of the data set to train (1000 training points) and testing with 30 points generates a graph as seen in figure 4. The output is not interpolated between points but appears to generate a functional distribution. The line shows the predicted target value based on the input value in that feature, the shaded area shows the error in the prediction.



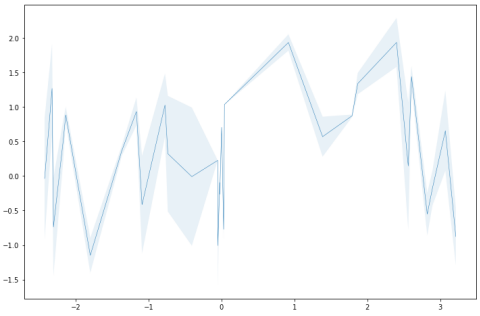


Figure 4: The GP outputs for features 1 and 10 using 100 training points and 30 test points. The squared exponential kernel is used with $\theta_0 = 1$ and $\theta_1 = 5$.

The other kernel functions all created outputs with exceptionally large errors and volatile predictions so the exponential kernel is used.

There still appears to be an issue with this implementation, as discussed in the validation section, where the predicted output values are sometimes very wrong. By taking the output values as the predictions a RMSE of 10.9 is found, no changes in parameters appear to change this value significantly with it ranging between 10 and 11. A periodic kernel leads to an RMSE of 65 and a linear kernel an RMSE of 2167. The time taken for this method appears to be directly dependent on the size of the training data set, the larger the set the longer the time taken to compute.

5 Conclusion

To summarise, the best predictions were formed with the linear regression method (RMSE = 5.67) with the k-nearest neighbors method similar in accuracy (RMSE = 6.38 with k = 5). The Gaussian process had a RMSE close to 10.5, however, it is unsure if the implementation functions correctly due to its failure on the toy problems in the validation section. The random forest performed worst, this may well be due to the data set being poor for predictions via random forest or may be that there is an error in the implementation in the higher dimensional problem of the SARCOS set compared to the toy problems which the forest dealt with effectively.

In reference to algorithm speed, the linear regression algorithm was the simplest and fastest to compute, followed by the forest and the gaussian process. The nearest neighbor algorithm was the slowest by a significant margin due to it referring to the whole training set for each test point prediction.

References

[1] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[2] The sarcos data. `http://gaussianprocess.org/gpml/data/`. Accessed: 05-01-2018.

[3] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, Aug 1995.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.