An Empirical Comparison of Supervised Machine Learning Algorithms

Msc. Data Science - 2020
University of Bath

I. Introduction

This report reviews four supervised machine learning algorithms:

1) Linear Regression (LR)
2) kNearest Neighbor (k NN)
3) Random Forest Regression (RFR)
4) Gaussian Process Regression (GPR)

All of these will be implemented using regression predictive models resulting in a Mean Squared Error which is used as benchmark comparison for the performance of each model.

The four algorithms have been fit to the Sarcos data set which is an inverse dynamics problem for a seven degrees-of-freedom SARCOS anthropomorphic robot arm [1]. This data set is a well-defined functional mapping of continuous inputs $\mathbf{x} \in \mathbb{R}^n$ to outputs $\mathbf{x} \in \mathbb{R}^n$ of the same kind therefore we can view it as a regression problem; where its task is to to learn the mapping describing the relationship from input to target, using sample data.

In the following sections we will review GPR and RFR. GPR is known to achieve a high performance with minimal hyperparemeter tuning but simultaneously suffers from higher computational complexity and expense. RFR on the other hand has been observed to scale into very high-dimensional domains, however requires skillful and iterative tuning of the hyperparameters during the forest building and learning process in order to achieve competitive performance [2].

II. Methods

A. Gaussian Process Regression

A Gaussian Process Regression is a probability distribution over several possible functions. Since GPR processes let us describe probability distributions over functions we can use Bayes rule (given that it is is non-parametric) to update our distribution of functions by observing training data. This means we are able to shape our prior belief via the choice of a kernel and that the updated posterior distribution will contain both the prior distribution and the dataset, both of which can be assumed to be Gaussian.

The GPR assumes the following distribution for both the train and test data samples:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\mu}_* \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^{\mathrm{T}} & \mathbf{K}_{**} \end{bmatrix} \right) \tag{1}$$

Where $\mathbf{K}$ is the training data and $\mathbf{K}_*$ is the testing data set and $\mathbf{f}$ (plus noise) has : $//www.overleaf.com/project/5e0a1ec46dd7730001e0046c$ This assumption implies that $\mathbf{f}$ which equates to the $\mathbf{y}$ values have a μ = 0. Therefore LR is also implemented in the GPR model so that a line of best fit can be found between the $\mathbf{y}$ values that fall above and below the μ = 0 line.

Although GPR is known to suffer from high computational complexity and expense there are still many advantages to it.

There are three important advantages to using GPR; The first advantage is that it directly captures the models' uncertainty, meaning that the margin of error is transparent throughout the model predictions which is something none of the other algorithms are capable of displaying.

Another advantage is that GPR utilizes a Bayesian approach as opposed to Maximum Likelihood approach. This means that it is possible to inject prior knowledge on the parameters. Interval priors can be used to restrict one parameter within some given bounds [5] - meaning the parameters can only be positive, and are useful in avoiding outlier estimates and thus importing parameter estimation. Therefore the differentials of the three hyperparameters are included in this GPR as priors. The three hyperparameters are σ^2, the noise term within the cost function and tied to the kernel, s which is the signal variance of the kernel and l which is the length-scale of the kernel (defines how close two data points have to be in order to influence each other).

A final advantage is that the kernel, otherwise known as a covariance function, can be optimized exactly given the values of their hyperparameters; this often allows a precise trade-off between fitting and smoothing the data. The kernel identifies the statistical relationship between two data points within the input data - this is how the GPR identifies a change at $\mathbf{x}$ correlated with a change in the GPR at $\mathbf{x}'$. Generally, kernels incorporate a Euclidean distance between two data points, meaning they capture the linear transformation between them [3]. The Radial Basis Kernel was chosen here. This kernel consists of two hyperparameters; s and l .

$$k_{ij} = s \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{l^2}\right) \tag{2}$$

Given the assumption seen in Eq. (1), any finite set of points will have a joint multivariate Gaussian distribution with a mean function of:

$$\boldsymbol{\mu} = \mathbf{K}_{\mathbf{X}_*\mathbf{X}}(\mathbf{K}_{\mathbf{XX}} + \sigma^2\mathbf{I})^{-1}\mathbf{f} \tag{3}$$

and the covariance function of:

$$\sigma^2 = \mathbf{K}_{\mathbf{X}_*\mathbf{X}_*} - \mathbf{K}_{*\mathbf{x}}(\mathbf{K}_{\mathbf{XX}} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{\mathbf{X}_*} \tag{4}$$

With this probabilistic distribution defined, the cost function of the training data is given by:

$$l = -\frac{n}{2}\log 2\pi + \frac{1}{2}\log|\mathbf{K}| + \frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} \tag{5}$$

Where $\mathbf{K}$ is the radial basis kernel seen in (2), plus noise ($\mathbf{K}$ + $\sigma^2\mathbf{I}$). This means that the cost function now consists of three hyperparameters; σ^2, s and l. We define the hyperparameters to be the log of the variables in the equation since they must be positive scale-parameters.

The kernel is integrated into the cost function (5) and then differentiated with respect to the three hyperparameters:

$$\nabla_\sigma^2 \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = -\tfrac{1}{2}\mathrm{Tr}\big[\mathbf{K}^{-1}\sigma^2\big] - \mathrm{Tr}\big[\mathbf{y}^\mathrm{T}\mathbf{K}^{-1}\sigma^2\mathbf{K}^{-1}\mathbf{y}\big] \tag{6}$$

$$\nabla_s \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = -\tfrac{1}{2}\mathrm{Tr}\big[\mathbf{K}^{-1}\mathbf{K_k}\big] - \mathrm{Tr}\big[\mathbf{y}^\mathrm{T}\mathbf{K}^{-1}\mathbf{K_k}\mathbf{K}^{-1}\mathbf{y}\big] \tag{7}$$

$$\nabla_l \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = -\tfrac{1}{2}\mathrm{Tr}\big[\mathbf{K}^{-1}\mathbf{K_D}\big] - \mathrm{Tr}\big[\mathbf{y}^\mathrm{T}\mathbf{K}^{-1}\mathbf{K_D}\mathbf{K}^{-1}\mathbf{y}\big] \tag{8}$$

Where $\mathbf{K}_D$ is the Euclidean distance formula defined within the kernel function in equation 2 multiplied by the kernel itself. And $\mathbf{K_k}$ is the kernel without the noise.

The hyperparameter priors are then deducted from each of their respective gradients in order to make the computations truly Bayesian.

$$\nabla_s logp(s) = log(s) \tag{9}$$

$$\nabla_l logp(l) = log(l) \tag{10}$$

$$\nabla_\sigma^2 logp(\sigma^2) = log(\sigma^2) \tag{11}$$

The resulting gradient formulas can now be implemented and utilised to calculate the posterior mean of the GPR.

$$\nabla_s \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = \nabla_\theta logp(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) + \nabla_s logp(s) \tag{12}$$

$$\nabla_l \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = \nabla_\theta logp(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) + \nabla_l logp(l) \tag{13}$$

$$\nabla_\sigma^2 \log p(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) = \nabla_\theta logp(\theta,\sigma^2,|\mathbf{f},\, \mathbf{X},\mathbf{y}) + \nabla_\sigma^2 logp(\sigma^2) \tag{14}$$

Gradient descents are a valuable optimization method in machine learning as they aim to minimize some function by iteratively moving in the direction of steepest descent. Each of these gradient formulas update the hyperparameters of the algorithm throughout the training process.

In order to compute the parameters of the GPR predictive distribution and the partial derivative of the kernel function it was necessary to invert the matrix which can come at the cost of $\mathbf{O}(\mathbf{n^3})$. Given that we are dealing with a fairly large data set the Cholesky decomposition was used to invert the covariance matrix as it is much more computationally efficient.

Cholesky decomposition also produces the determinant of the matrix which is required to evaluate the maximum aposteriori likelihood. The remaining computations involved in evaluating the likelihood and its partial derivatives (vector by matrix multiplication and evaluation of the trace of a product) are of order $\mathbf{O}(\mathbf{n^2})$. Thus, the main computational obstacle is computation of equations 12, 13 and 14.

When fitting the Sarcos data to the Gaussian Process algorithm the hyperparameters were initialized to random values and then used in an iterative loop for the three gradient until they converged. [3]

B. *Random Forest Regression*

RFR is an ensemble learning method that uses a bagging technique (random sampling with replacement) as the forest building progresses. This technique is often implemented in order to reduce variance as it forces each decision tree to run independently and in parallel to each other, and then aggregates the output mean predictions at the final node, without preference to any of the models.

There are three advantages of the RFR, the first is that the feature splits that can occur at each node are limited to a fraction of the total sample size. This safeguards against the reliance of any individual feature too heavily; therefore the ensemble model utilizes all potentially predictive features equally. The second is that random sampling of training data points are taken when building each of the trees and are replaced each time; this is also known as bootstrapping. This means that each tree can be trained on a different set of sample training data and even though the individual trees may have a high variance the overall forest will have a decreased variance, but not at the cost of any increasing bias therefore improving the stability and accuracy of the prediction. This applies to the final advantage which is that RFR generates an internal unbiased estimate of the generalization error as the forest building progresses. (5). Both the first and second advantages mentioned here are considered hyperparameters that can be optimized during the forest building process.

As mentioned above, an RFR model is formed by initiating several decision trees; these train by recursively splitting the data into two halves and are then compared based on how well they split the dataset.They are initiated on a random vector Θ such that the tree predictor $h(\mathbf{x}, \Theta_k)$, where $k = 1, ..., K$ and $\mathbf{x}$ represents the observed input, takes on numerical values as opposed to class labels [4].

Random data samples are then taken from the training data resulting in multiple feature bootstrapping; therefore a subset of the p feature dimensions occurs at each split. When $\mathbf{K}$ separate bootstrapped samples of the training set are created, with separate model estimators $\hat{h}^K(x)$, then averaging these leads to a low-variance estimator model, $\hat{h}_{avg}$:

$$\hat{h}_{avg}(x) = \frac{1}{K}\sum_{k=1}^{K} h(\mathbf{x}, \Theta_k) \tag{15}$$

The output values are numerical and we assume that the training set is independently drawn from the distribution of the random vector $Y, \mathbf{X}$ [4]. Having calculated the above estimator equation (15) an average is taken over the k-tree resulting output values $h(\mathbf{x}, \Theta_k)$ and is used to predict the Mean Squared Error of the Random Forest.

Although RFR models do have several advantages they also suffer from of overfitting with noisy regression tasks, and can some times be biased in favor of varying levels or attributes of trees, if not controlled appropriately.

III. Validation on Toy Problem

A. Overview

The Sarcos data set provided is 21 dimensions making it difficult to visualize or run smaller training sets on it; therefore two toy problems were created in order to gain more intuition each models' learning behavior. A linear and non-linear toy problem were created that were used across all four machine learning algorithms and then used to calculate the respective Mean Squared Error. Noise was added in order to improve the MSE and the structure of the data mapping of the toy problem. Therefore the toy problems have been analysed both with and without noise.

$$MSE = \sqrt{\frac{\hat{y} - y}{n}} \tag{16}$$

B. Linear Toy Problem

The linear toy problem had the following equation:

$$f(x) = mx + c$$

Where $m \in \mathbb{R}^2$, $\mathbf{x} \in \mathbb{R}$ is sampled from $\mathcal{N}(0, 1)$, and c is a constant set to 10.

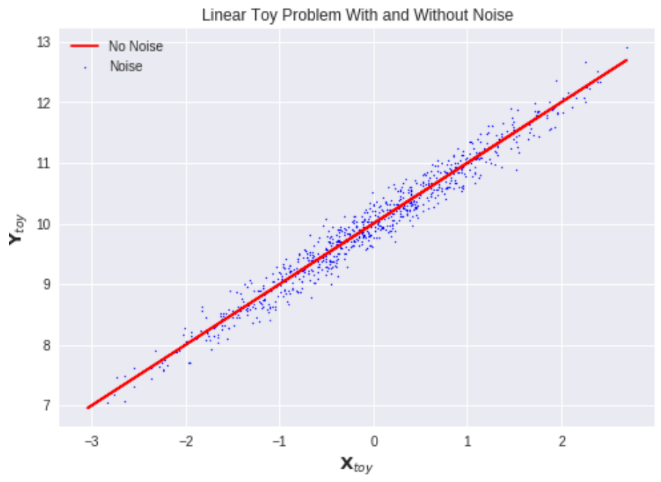

Fig. 1. Plot of linear toy problem both with noise (red line) and without noise (blue scatter).

C. Non Linear Toy Problem

A sine function was incorporated into the original linear toy problem so as to make it non-linear.

$$f(x) = sin(mx) + c$$

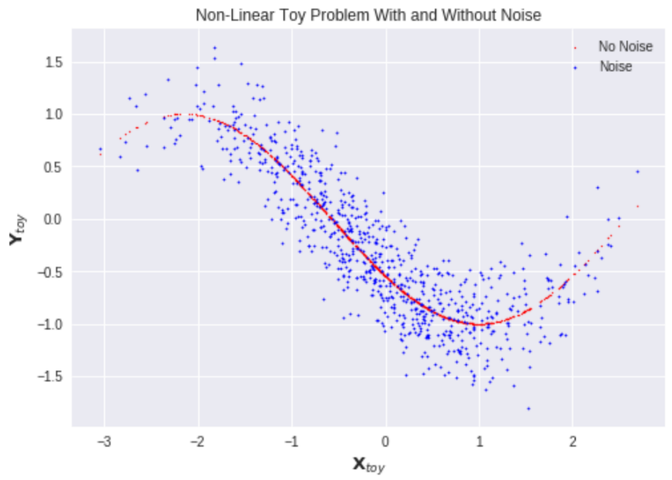

Fig. 2. Plot of non linear toy problem both with noise (red line) and without noise (blue scatter).

D. Toy Problem Predictions

1) Linear Regression: The Linear Regression algorithm aims to find the line that best fits the observed data points available so as to predict the output values for any unobserved data points. The MSE function calculates the distance between the predicted $\hat{y}$ value and the actual y value present in the dataset. Having said this we anticipate the mean function of the LR algorithm to fit directly over the linear toy problem's ground truth as they are both linear functions, but we expect very evident differences when fitting on the non-linear toy problem. Both can be seen in Fig 3.

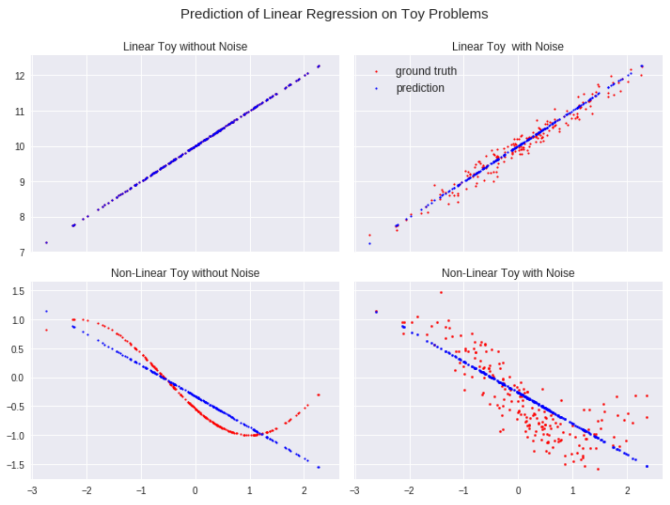

Fig. 3. Prediction of Linear Regression Algorithm on the two toy problems, both with and without noise the LR model fits directly onto the linear toy problem but fails in trying to find the line of best fit with the non-linear toy problem.

This result implies that although the LR algorithm will perform fairly quickly on the Sarcos data we should anticipate a high MSE.

2) k Nearest Neighbors: Given that kNN is a type of instance-based learning, where the function is only approximated locally and all computation is deferred until classification, we anticipate that it will not fit directly onto the linear toy problem. This is because kNN does not directly learn any functional form of the description of the data, but instead the prediction returns an average of the samples that are closest to the testing sample. It does not have any ”global” knowledge of the function, but can only give you the ”local” examples. This is why kNNs are generally defined as local function approximators.

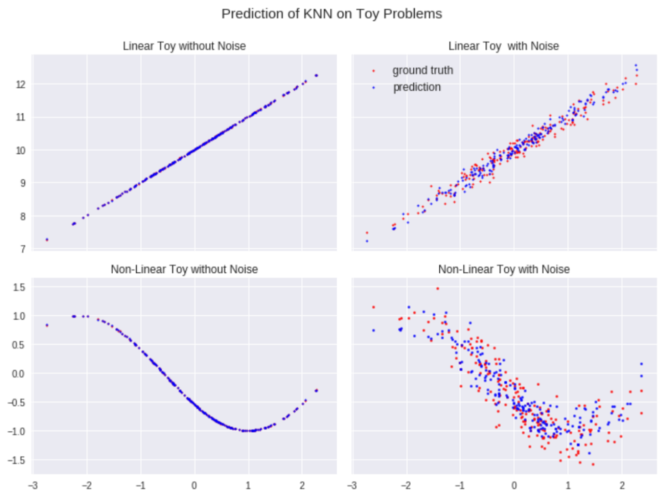

Fig. 4. Prediction of KNN Algorithm on the two toy problems, both with and without noise.

Although the kNN seems to have fit well to the linear toy problem (with noise), upon closer investigation it is observed that the kNN displays the behavior discussed above where it identifies training data points but when moving in between the points it is taking an average of the two points.

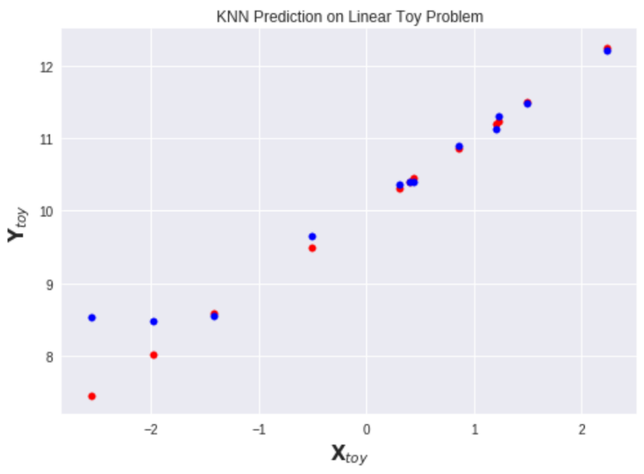

Fig. 5. Prediction of KNN Algorithm on the linear toy problem, without noise that displays the average sample distance between the data points.

That said, kNN performs relatively well on the non-linear toy problem, with noise so we can assume that it will result in a low MSE for the Sarcos data.

3) Random Forest: By definition, RFR is built on the training of many decision trees, trees that have an excessive max depth have a tendency to pick up very irregular patterns such as over-fitting their training data. Consequentially this can result in a low bias but a very high variance. This can be seen when plotting the prediction of the Decision Tree with the RFR and the toy problem's Ground Truth.

Therefore we can predict that RFR will perform well on the Sarcos dataset when including many decision trees and a large max depth.

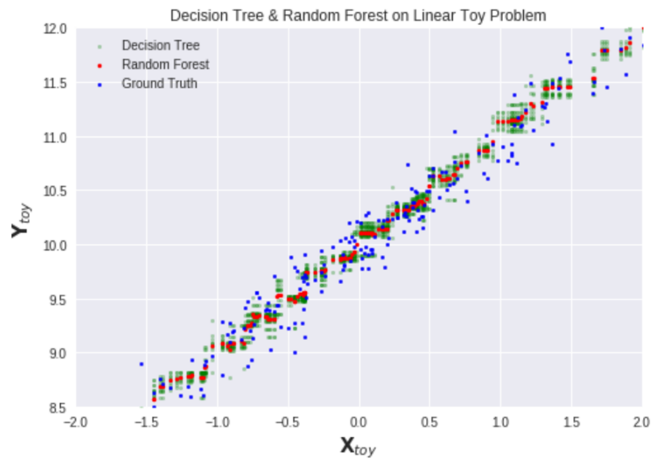

Fig. 6. Prediction of 10 Decision Trees (with a max depth of 5), the Random Forest and the Ground Truth of the Linear toy problem, with noise. It becomes evident that the more trees add the more accurate the random forest model become, resulting in a lower MSE

4) Gaussian Process: As discussed in the Methods section, GPR has a distribution over several functions with a continuous domain which simultaneously assigning probability to each of these functions. This said we can assume that the Gaussian Process will be the best predictor of the toy problem.

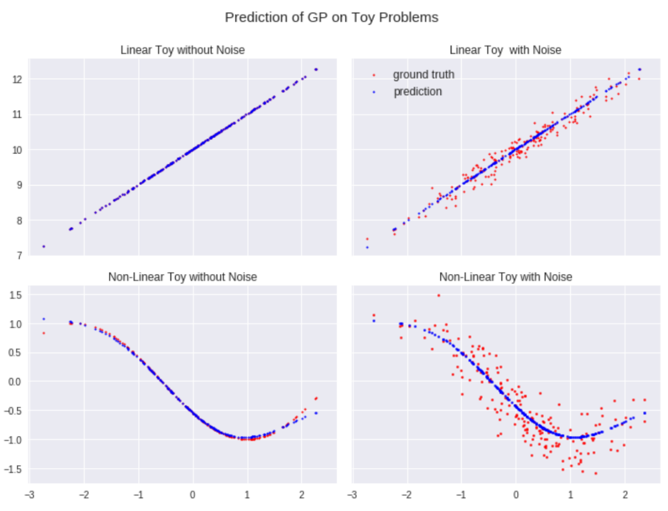

Fig. 7. Prediction the Gaussian Process on each of the toy problems, both with and without noise.

Here we can see how in comparison to the other three machine learning algorithms, the GPR has the best MSE prediction on the toy problem; ergo we can extrapolate that GPR will most likely have one of the best MSE results on the full Sarcos data.

E. Results

Having plotted and validated that each of the four algorithms were working correctly on both toy problems the MSE functions were then run and the resulting values were recorded below.

Linear Toy Problem:

Algorithm	No Noise	Noise
LR	0.153	0.507
RFR	1.614	1.903
*k*NN	0.060	0.350
GPR	8.441e-31	0.227

Table1: Results of each of the algorithms on the linear toy problem (rounded to 3 decimal places) with two columns; one is the resulting MSE with the function having no noise and the other has noise of scale 0.5. As anticipated Gaussian process performed the best on the linear toy problem both with and without noise. LR performed the worst alongside RFR which indicated that the hyperparameters still required some more optimizing.

Non-Linear Toy Problem:

Algorithm	No Noise	Noise
LR	0.569	0.7237
RFR	0.444	0.627
*k*NN	0.035	0.324
GPR	0.158	0.380

Table 2: Results of each of the algorithms on the non-linear toy problem (rounded to 3 decimal places) with two columns; one is the resulting MSE with the function having no noise and the other has noise of scale 0.5. *k*NN has performed the best on both versions of the non-linear toy problem.

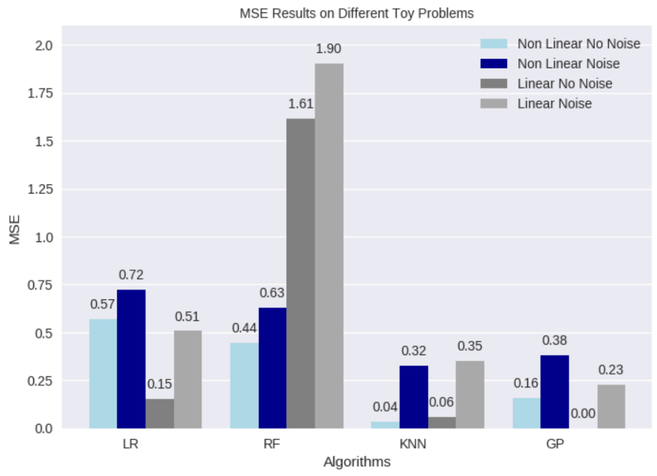

Fig. 8. MSE Results on each of the toy problems, both with and without noise added. RFR seems to have performed the worst on all the toy problems where GPR and *k*NN have performed well on on both. The RFR model still required hyperparameter tuning prior to running the full Sarcos data set on it therefore we can anticipate the MSE will drop and it will eventually perform better than the other models.

IV. Experiments and Analysis

The following section shall review the implementation, hyperparameter tuning, and predictive results of the four algorithms in order of ascending computational complexity:

1) LR
2) kNN
3) RFR
4) GPR

Prior to implementing the algorithms the x and y values were both scaled:

$$z = \frac{x - y}{\sigma} \tag{17}$$

This normalization of the data was performed in order to scale any irregular or irrelevant data within Sarcos.

A. Linear Regression

Although the LR model is computationally inexpensive and performs well on linear problems it does not perform as well on non-linear problems. The LR formula implemented is non-Bayesian and takes the weighted data point values into consideration while trying to find the line of best fit, which is eventually determined in terms of the lowest MSE:

$$\hat{\boldsymbol{w}} = (\boldsymbol{X^T X})^{-1} \boldsymbol{X^T y} \tag{18}$$

Given that the Sarcos data set is a multidimensional space the LR algorithm performs quickly but results with a high MSE of $\mathbf{30.484}$ when run on the full dataset (33363 training data, 11121 testing data). As this model is non-Bayesian there are no hyperparameters to optimize and therefore the MSE is considered to be the final accuracy point prediction for LR.

B. K Nearest Neighbor

k Nearest Neighbors takes a given data point (x) and searches for the k closest point to x in the sample training data. If the k chosen to implement is too small the model may be sensitive to outlier data points. Conversely, if the k is too large, the model may include too many data points and result in an inaccurate MSE. Therefore the *k*NN implemented here was weighted via the distance between each each k point. This means that the k nearest data points are given individual weights so that the closer the k points are too each other the move heavily they are weighted in the 'voting' process, and those farther away have less weight.

In order to calculate the MSE the distance between the given data point (x) and their k closest neighbor is found using a distance function. The most common distance function is the Minkowski distance which is a generalization of the Euclidean and Manhattan distance:

$$(\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}} \tag{19}$$

The p value is the dimension of attributes, meaning it can be optimized as a hyperparameter within the *k*NN model and can be performed in $\mathbf{O}(p)$ operations. Additionally the value of k nearest neighbors can be optimized to calculate the MSE.

It quickly became evident that *k*NN is extremely memory and computationally intensive, and does not perform well on high-dimensional data as it stores each sample training data observation that is *k* distance nearest to the *x* value and stores. These are then used to predict new observations and are also stored.Therefore the **scipy.spatial.cKDTree** was implemented in place of the distance and indexing calculation; this provided a much more rapid method of looking up the nearest neighbors of any point.

A grid search was used to identify the optimal p and k values that resulted in the most accurate MSE prediction.

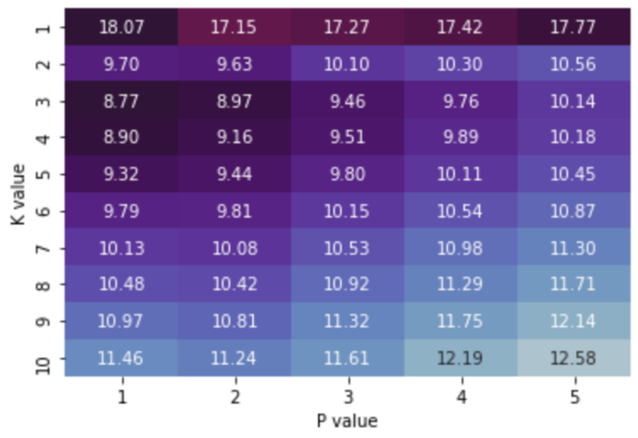

Fig. 9. Grid Search of optimal p and k values for the KNN MSE. 8.77 was found to be the lowest calculated MSE with a p of 3 and k of 1 when run on the full Sarcos dataset (33363 train, 11121 test) This p of 1 directly corresponds to the Manhattan distance.

C. Random Forest

As discussed in the methods section, there are several hyperparameters that can be optimized when building a Random forest; the n number of trees used, *max depth* and *feature split*. Therefore the hyperparameters were tuned via trial and error and the following results were found:

max depth	N	F Split	MSE
10	100	0.8	27.188
15	100	0.8	22.050
20	100	0.8	16.446
20	120	0.8	15.958
20	120	1.2	14.586
25	120	1.2	**14.087**

Table 3: Resuling MSEs of Random Forest when run on different hyperparameters. As the max depth, number of trees and feature split were optimized the MSE dropped as the prediction accuracy increased - given the behavior of Random Forests this was to be expected.

A validation curve or grid search could be performed on the RF model that could predict more optimal hyperparemeters.

D. Gaussian Process

The GPRs hyperparameters are confined to its kernel and are therefore optimized during the training process. This means that other parameters in the model were manually tuned so as to minimize the MSE result.

Epoch	alpha	momentum	MSE
50	5e-7	0.8	**6.906**
25	5e-7	0.8	7.411
50	1e-5	0.6	10.747
50	1e-5	0.8	10.188
50	1e-6	0.6	13.094
50	1e-6	0.8	12.678

Table 4: In order to predict an MSE with the calculated gradients (discussed in Methods), steps - otherwise defined as *alpha* - are taken in a positive direction whilst also maintaining momentum as they continue to move forward. This function was iterated over many epochs until the the model produced a resulting MSE that was considered competitive.

MSE	l	s	σ^2
6.906	2.69	1.18	0.72
7.411	2.54	1.20	1.11
10.747	3.31	1.12	0.36
10.188	3.20	1.31	0.21
13.094	2.51	1.19	1.34
12.678	2.61	1.17	1.13

Table 5: MSE results with their corresponding hyperparameters that were optimized internally as the GPR trained.

In addition to evaluating how the algorithm tuned the three hyperparameters the results were recorded both with and without the prior gradients and it was observed that although GPR is Bayesian, the high dimensionality of the Sarcos data set drowned out the priors; meaning they had little affect on the resulting MSEs.

E. Model Comparison

Given the complexity and size of the Sarcos data set not all of the algorithms were able to efficiently run and produce competitive MSEs on the entire Sarcos, therefore specific train and test data splits were chosen in order to maintain consistency and run a benchmark test of each algorithm.

Testing Size	LR	KNN	RF	GPR
5,000	31.661	14.966	14.087	**8.756**
24,484	30.572	14.282	16.446	**7.411**

Table 6: MSE results when changing size of testing data points from 5,000 to 24,484 while maintaining 20,000 train data points.

As anticipated GPR performed the best on both 5,000 and 24,484 testing data points.

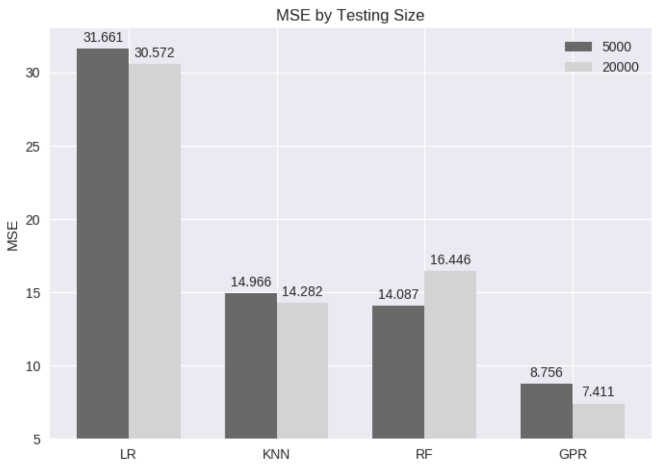

Fig. 10. MSE Values for each of the algorithms on 20,000 training data points and altering testing data points (5,000 and 24,484- the reaminder of Sarcos dataset). LR had the worst performance with a difference of more than twice the size of KNN and RFR, and almost four times the size of GPR.

A different set of results were seen when optimizing the algorithms and running them on the maximum amount of Sarcos data they could handle and efficiently run on.

Algorithm	MSE
Linear Regression Regression	30.484
k Nearest Neighbors	8.77
Random Forest Regression	14.087
Gaussian Process Regression	**6.906**

Table 7: Final MSE results of the four different algorithms. Gaussian Process has scored the lowest MSE meaning that it had the best fit and prediction on the Sarcos dataset. *k* Nearest Neighbors was the second best as a grid search was applied to find the best hyperparameters. Random forest did not perform as well as the hyperparameters were optimized via trial and error as

opposed to using a grid search, and Linear regression performed the worst which was to be expected as it did not perform as well as the others on the Toy Problem.

V. CONCLUSIONS

Supervised learning algorithms are used in a wide variety of domains and different performance metrics are accepted for each of those domains. Having used the MSE metric here we have identified that for non-linear supervised learning methods Gaussian Process has the highest performance. Unfortunately it is computationally expensive and complex but when tuned appropriately can return competitive results. *k*NN provided the next best MSE while also being one of the simpler implementations, but is considered a 'lazy' learner so not the best algorithm to use with complex data sets as it does not accurately capture the ground truth. RFR performed relatively well on the Sarcos dataset but a grid search or validation curve would have resulted in a higher MSE. Finally the LR algorithm scored well for its minimal complexity but is not as robust as the other models.was a good benchmark for the other. To conclude the Gaussian Process has proven to be the most maleable and proficient code on larger datasets and returns a prediction closest to the ground truth.

REFERENCES

[1] URL : http://gaussianprocess.org/gpml/data/

[2] Duy Nguyen-Tuong, Matthias Seeger Jan Peters (2009) Model Learning with Local Gaussian Process Regression, Advanced Robotics, 23:15, 2015-2034, DOI:10.1163/016918609X12529286896877

[3] Williams, C. and RAsmussen, C. 1996). Gaussian Process for Regression. [online] Mlg.eng.cam.ac.uk. Available at: http://mlg.eng.cam.ac.uk/pub/pdf/WilRas96.pdf [Accessed 10 Jan. 2020].

[4] Breiman, L. Machine Learning (2001) 45: 5. https://doi.org/10.1023/A:1010933404324

[5] Cousineau, D. and Hlie, S. (2013). Improving maximum likelihood estimation using prior probabilities: A tutorial on maximum a posteriori estimation and an examination of the weibull distribution. Tutorials in Quantitative Methods for Psychology, 9(2), pp.6171.

LaTeX word count: 3293