Problem Statement

"HW2Dataset.csv" file contains 8900 data points of 15 prioritized potential inputs (i.e., these regressors are expected to generally show decreasing levels of importance in predicting the output) (array shape 8900x15) that reflect salient measures of material structure and their corresponding FE (finite element)-predicted elastic stiffness values (serves as output) (array shape 8900x1). Your goal in this homework is to build a number of different models that can predict the stiffness values for new inputs. You are not required to use all 15 regressors in your models. In building your models, please employ suitable train-test splits and/or cross-validations to demonstrate your model performance. Specifically, please demonstrate the following strategies:

1) Use simple linear regression methods with or without regularization. Explain the rationale behind your choices.

2) Use the Gaussian Process regression method. Explain your choice of the kernel and comment on the methods used for the optimization of the hyper-parameters in the kernels. Hint: GPR models might face convergence issues when dealing with large datasets and high computational time for training. You can start with an initial number of the most important input regressors (5 PCs) based on your observations from Q1 and add additional regressors for better models if necessary. You can choose a random set for training datapoints consisting of 20-50% of the entire dataset. Depending on the choice of your kernel the computational time would increase with the number of input points and input regressors as well as the number of optimisation restarts.

3) Use Neural Networks. Discuss the choices made in your network architecture (number of layers, activation functions, etc.)

Discuss the relative pros and cons of all the models you have built for this homework. Which ML method performed the best? Why?

Solution Before any models were created, the data was plotted and analyzed in an attempt to observe relationships amongst the data. Additionally, prior to using the data as given, the data was shuffled to ensure that the order the data was provided in was not a factor in the training of the model.

First, 2D plots were created for each critical point plotted versus the output value. These plots can be seen in Figure 1. Through visual inspection of the plots, it appears that regressors 1 and 3 show some trend in the output value. There is somewhat of an inverse relationship between PC1 and the stiffness value (as PC1 increases, stiffness decreases), and there somewhat of a direct relationship between PC3 and the stiffness value (as PC3 decreases, stiffness decreases). All other plots don't appear to show any convincing trends, and appear to be somewhat random.

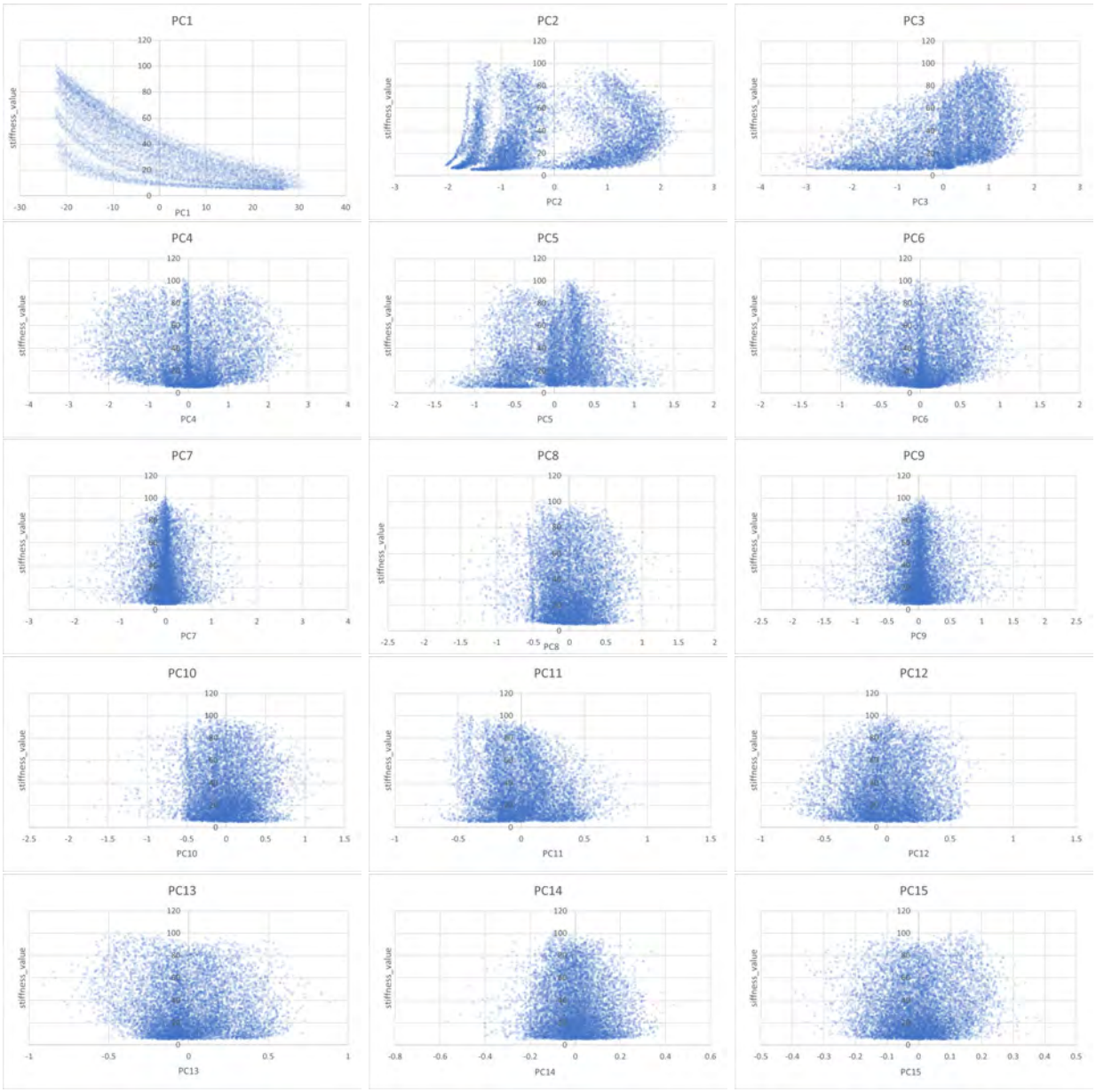


Figure 1: All 15 regressors plotted against the output, "stiffness_value".

To further explore the data, a pairwise heatmap depicting the absolute value of the standard correlation coefficient between each input and the stiffness value output was created and is shown in Figure 2. It shows that the first 5 regressors, PC1 - PC5, show the strongest correlation with the output, followed by the last 5 regressors, PC11-PC15, showing a weaker, yet still somewhat significant relationship with the output. The middle 5 regressors, PC6 - PC10 show the weakest correlation with the output. PC1 shows the strongest correlation with the stiffness value, followed by PC3, then PC11. Additionally, in Figure 2, another heatmap ordering the regressors from most to least correlated is shown.

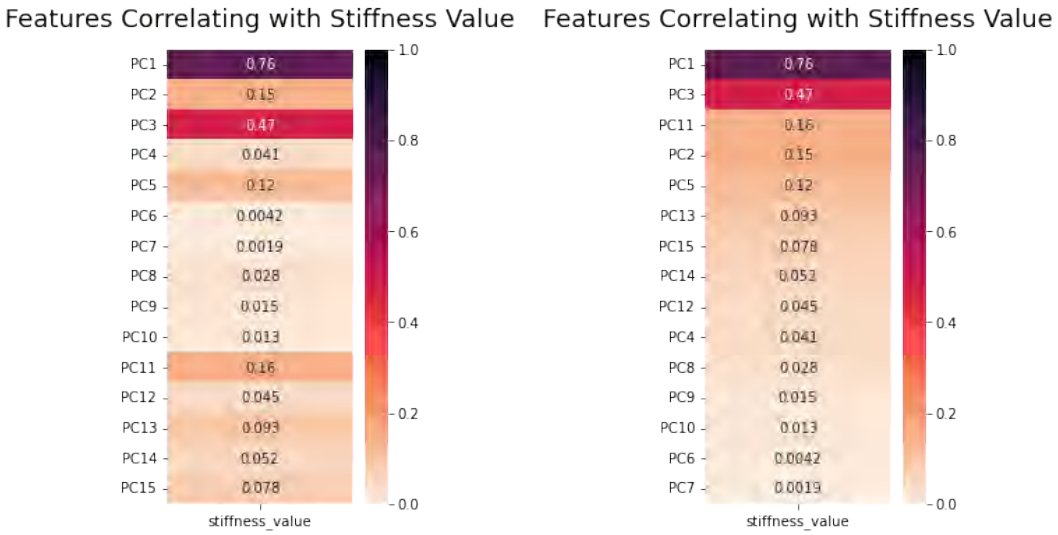


Figure 2: Unordered and ordered feature correlation heatmap with stiffness_value .

The importance of these regressors will be verified using Lasso regularization in part 1 of the problem statement. This information serves to show which regressors may be most important to determining the output stiffness, and may serve to create simpler models with fewer inputs that can still be relatively accurate.

Of the 8900 data points, different splits of testing and training data were used for each part of this problem statement. For the linear regression models, the test set was chosen to be 30% of the provided data points - the rest of the data was used for training the models.

For the Gaussian Process Regularization Models tested, different sizes of test/train splits were tested to see what split would provide the best results for the model. Additionally, different numbers of regressors were tested in the given order of priority to see if the model could be simpler, yet produce good results. Three training sets were used for the GPR models - One set containing regressors PC1 - PC5, one set containing regressors PC1 - PC10, and the final set containing all the regressors. The results of this study are shown in the exploration of part 2 of this problem statement.

Finally, for the Artificial Neural Network models tested, a training/test split of 80% to 20% was used.

SKLearn models were used for Linear regression and GPR, and TensorFlow was used to create the ANN's tested in this solution.

Note: All errors referred to in this report are root mean square errors. If error is mentioned, it refers to RMSE unless otherwise specified (MAE is used for machine learning models).

Another very important note to make is that every model, except for all GPR models, have been put through a 5 - fold cross validation to obtain a cross validation error for performance metrics. Only the final GPR model had a 5-fold cross validation run on it to avoid extremely long run times and potential crashes.

Problem 1

Use simple linear regression methods with or without regularization. Explain the rationale behind your choices.

Solution

Four models were considered for this problem : Ordinary Least Squares Regression, Ridge Regularization, Lasso Regularization, and Linear Regression with Polynomial Features.

OLS was chosen as a starting baseline. It appears that not all regressors are significant in determining the output, therefore both Ridge and Lasso regularization models were used in an attempt to create simpler, sparser models. Additionally, regularization was used since a large amount of the data seemed to show minimal correlation with the stiffness values. Regularization was used as a method to weed out these unimportant variables in an effort to potentially create good models with fewer regressors, yielding simpler models and shorter training time. Finally, a linear regression model using polynomial features, taking advantage of SKLearn's pipeline class, was created to test for more complex relationships between the data. To start, polynomial features of degree 3 were used for preliminary testing.

Ridge and Lasso regularization models must be optimized with respect to λ. In order to do this, SKLearn has a built-in class that can optimize λ and perform a k-fold cross validation on the data. These classes are called RidgeCV and LassoCV. They were chosen over the basic version of Lasso and Ridge models due to their ability to automatically find and optimize a λ value for the training set. One parameter that may potentially be optimized for these models is the number of folds to be used for cross validation. A sensitivity study was run to check for the optimal number of folds to use for these models. The training error, cross validation (CV) error, and test error were calculated for 5 different CV values ranging from 5 to 100 (values larger than 100 were not used since this is not a good split between testing and training data that would produce meaningful results - the test sets would be too small and the model is likely to be overfit). These results are shown in Figure 3. Since no significant deviation in error was detected through changing the number of folds for these models, a 5 folds were chosen for the models to be compared to the other linear regression models to be trained and tested. This value was chosen since it is a standard value used in data science, providing a good ratio of training to testing during cross validation.

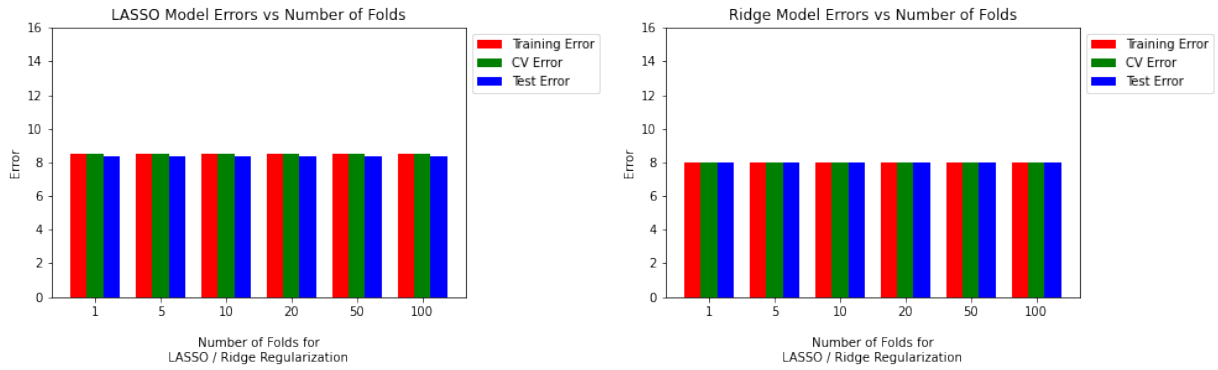


Figure 3: Bar Plot of LassoCV and RidgeCV models trained on a varying number of folds used in cross fold validation.

Next, in an attempt to validate the conclusions made when initially analyzing the data, Ridge, Lasso, and Linear Regression models were trained on all the regressors in order to evaluate the importance of each regressor. The absolute value of the weights from each model for each regressor are plotted in Figure 4. Lasso regularization is most likely to produce a sparse model, and it does. Based on the Lasso regularization model weights, it appears that the most important regressors are PC1 - PC5, as well as PC11 and PC13. Note that the Ridge and OLS models produce exactly the same model with same feature weights, and same performance metrics. This indicates that Ridge regularization does not make the feature space any more sparse than a regular OLS.

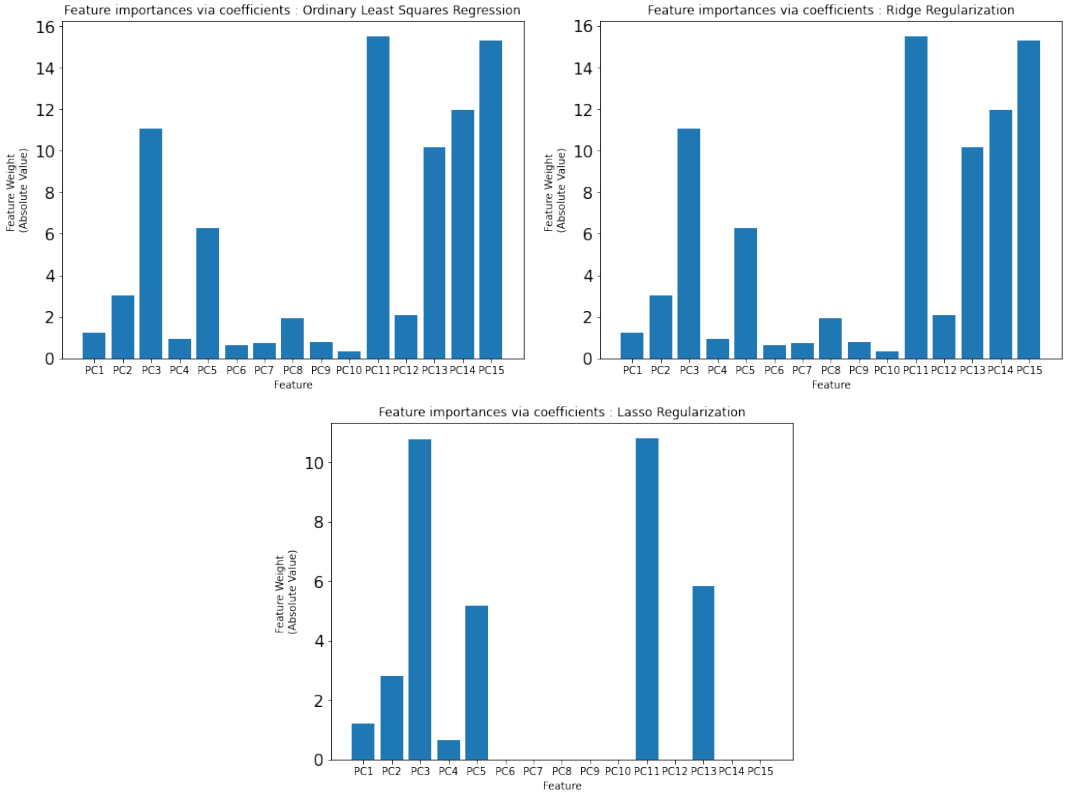


Figure 4: Feature importance bar charts displaying the absolute value of feature weights determined from different linear regression models.

Next, the models were trained 16 times each, training each model with an increasing amount of regressors ([PC1], [PC1, PC2], ... , [PC1, PC2, ... , PC15]), and a 16th set was created evaluating each model with just the (seemingly) most important regressors determined via Lasso Regularization (Pc1 - PC5, PC11, PC13). The training error, cross validation error, and test error were calculated for each model at varying numbers of regressors. Bar charts showing these errors as they evolve with the number of regressors the model is trained on is shown in Figure 5.

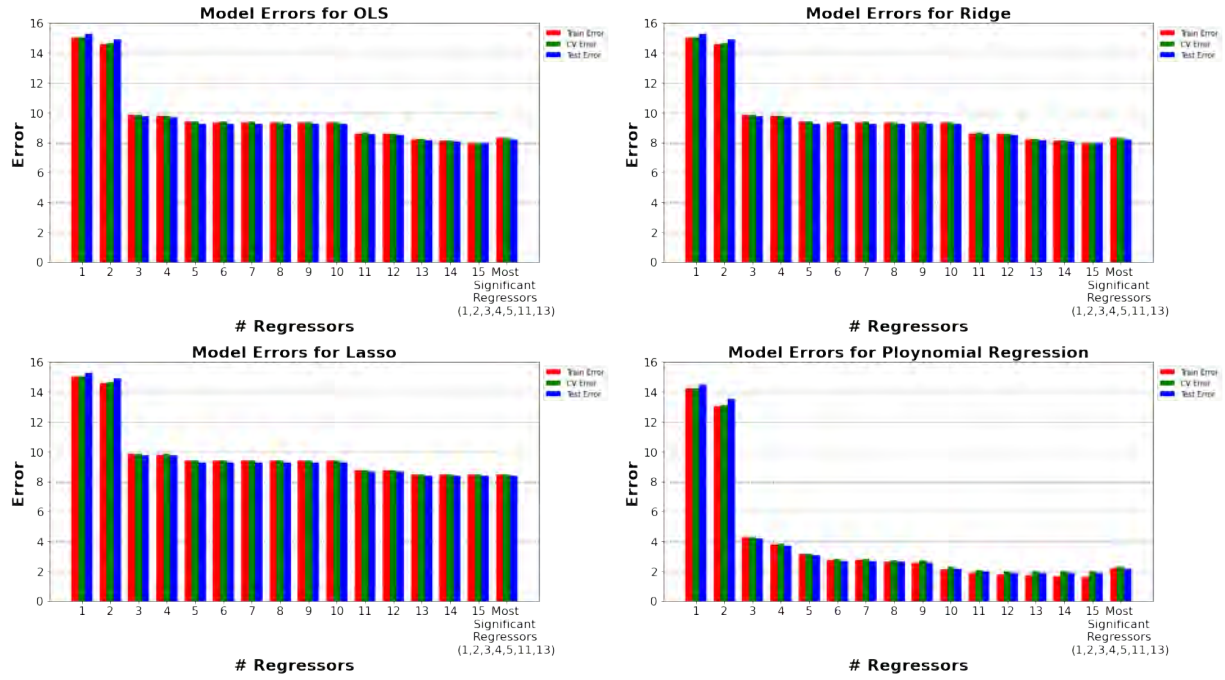


Figure 5: Bar Plot of OLS, Ridge, Lasso, and Polynomial Feature Linear REgression model training, cross validation, and testing errors versus the number of regressors the model was trained on.

All models show a trend of decreasing error with an increase in the number of regressors used to train each model. In an effort to reduce the number of models to compare, for each model, the optimum number of regressors was selected. This optimum was calculated by choosing number of regressors that resulted in best R^2 score for each model. For every model, the number of regressors producing the best R^2 score was all 15, except for Lasso, where the best number of regressors was 13.

To more directly compare these models, the best version of each model was compared directly using parity plots. An OLS model with 15 regressors, a 5 fold RidgeCV model with 15 regressors, a 5 fold LassoCV model with 13 regressors, and a Linear Regression model with polynomial features of degree 3 with 15 regressors were determined to be the best based on the methodology pursued and were trained on the training data accordingly. Parity plots for each model were made by plotting the acutal stiffness values on the x axis, and the predicted stiffness values on the y axis. The better the scatter follows the line $y = x$, in other words, the higher the R^2 score, and the lower the root mean square testing error, the better the model is. The parity plots from this study are shown in Figure 6. The performance metrics for each of these models are tabulated in Table 1.

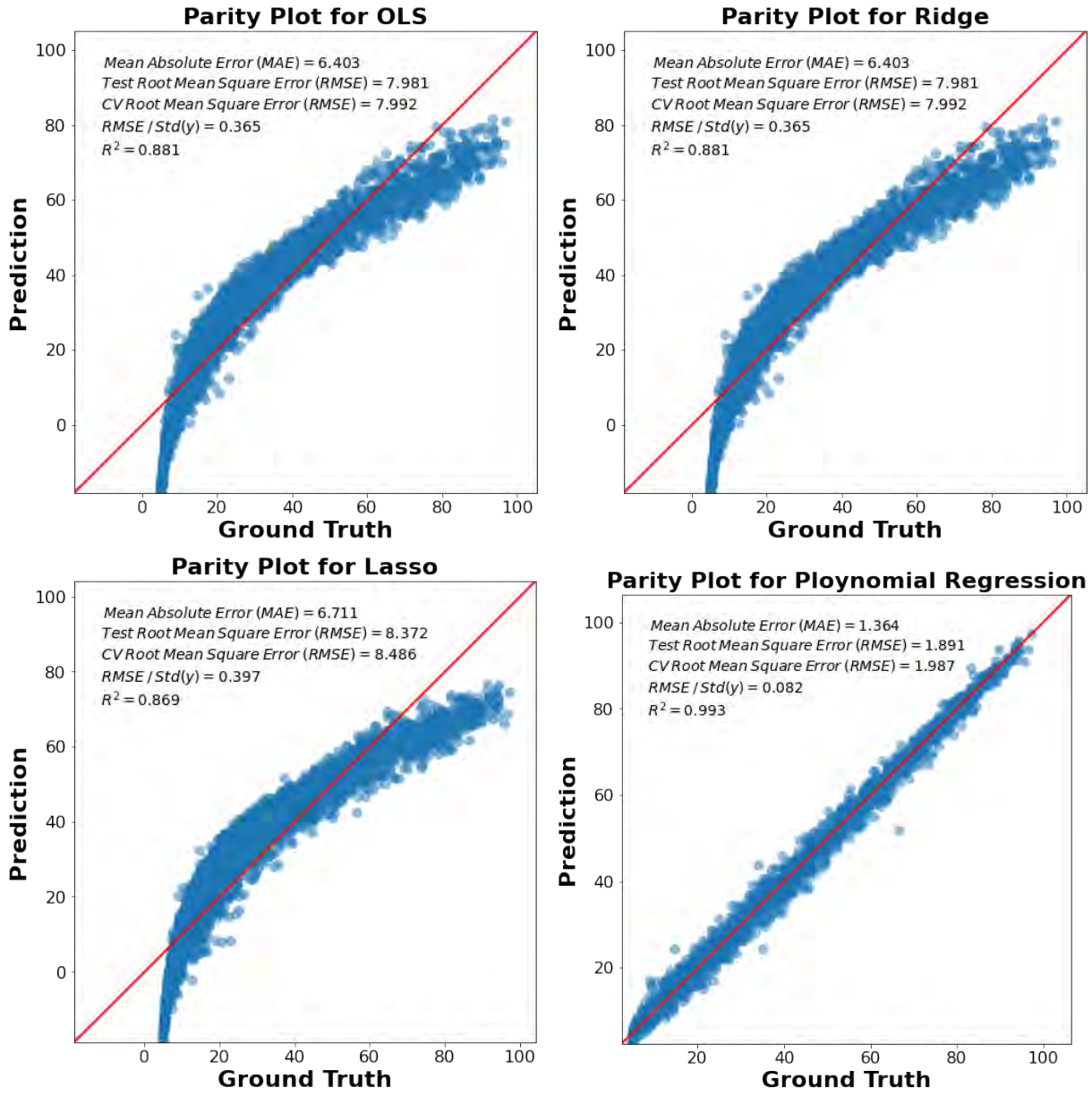


Figure 6: Parity Plots of OLS, Ridge regularization, Lasso regularization, and polynomial features linear regression models.

Table 1: Model Performance for OLS, Ridge regularization, Lasso regularization, and polynomial features linear regression models.

Linear Regression Model	R^2	Test RMSE	CV MAE	Test MAE	CV RMSE - Test RMSE
Ordinary Least Squares	0.881	7.981	7.992	6.403	0.011
Ridge Regularization	0.881	7.981	7.992	6.403	0.011
Lasso Regularization	0.869	8.372	8.486	6.711	0.114
Polynomial Features	0.993	1.891	1.987	1.364	0.096

Through a visual inspection, it can be seen that the purely linear models (OLS, Ridge, and Lasso) produce similarly shaped parity plots. Each purely linear model exhibits a shape similar to an n^{th} root curve. The linear regression using polynomial features of degree three shows a very strong linear relationship, along with the smallest test error, and is easily determined to be the best model tested.

Having chose linear regression with polynomial features as the best linear regression model to fit the data, one step more could be taken to optimize the model even more. The degree of the polynomial features were tested to see if a degree other than 3 (which was what was used up to this point) may do a better job in approximating the data.

Polynomial features of the training data of degrees 1-5 were created and linear regression models were trained on each of these feature sets. Note, a linear regression model on degree 1 polynomial features is simply a linear regression model, and is used as a baseline comparison to previous models for this study. The resulting parity plots from the predicted stiffness values of these models are shown in Figure 7. The performance metrics for each of these models are tabulated in Table 2.

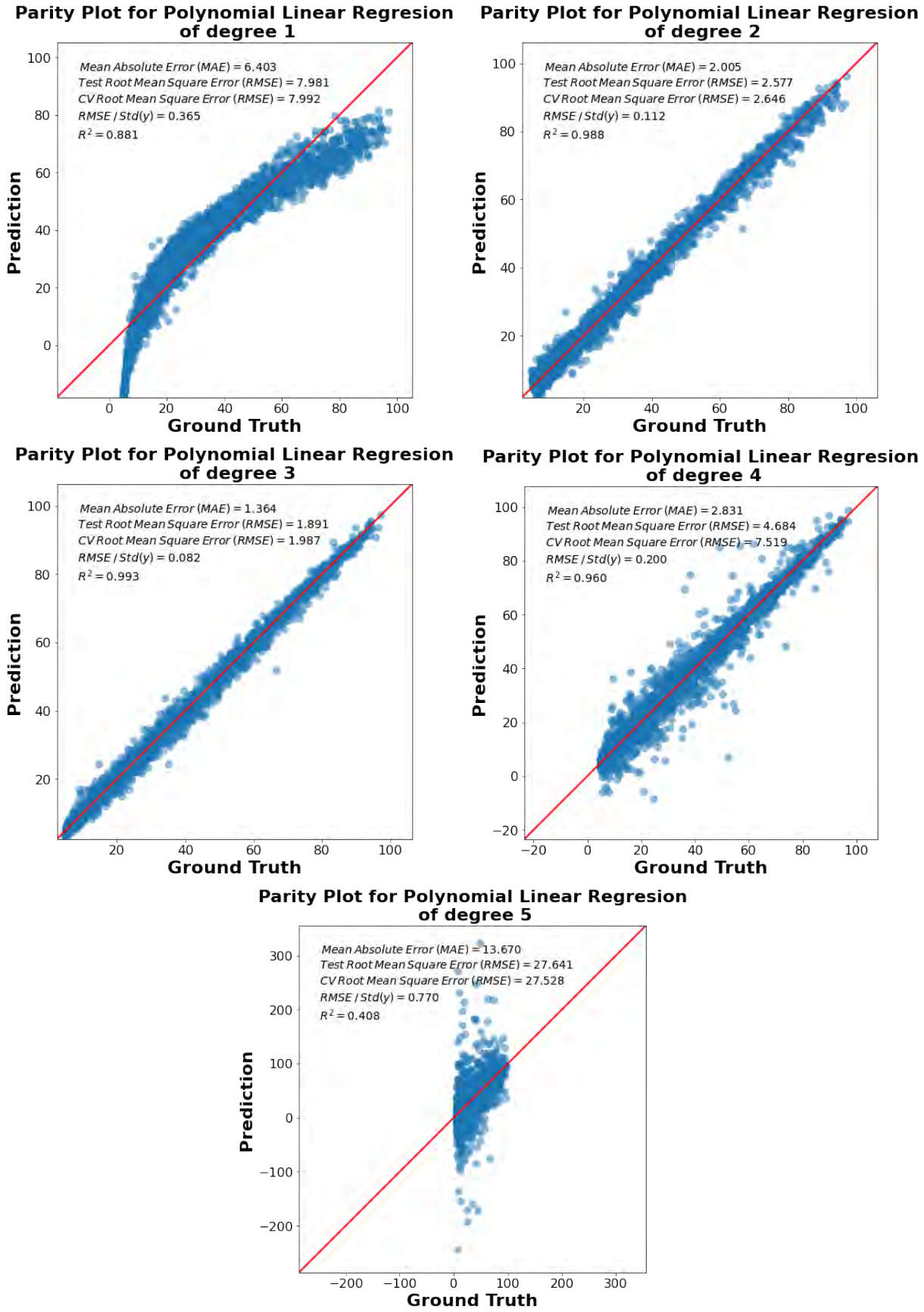


Figure 7: Parity Plots of linear regression models with polynomial features of varying degrees (Degree 1 through Degree 5).

Table 2: Model Performance for linear regression models with polynomial features of varying degrees

Degree	R^2	Test RMSE	CV MAE	Test MAE	CV RMSE - Test RMSE
1	0.881	7.981	7.992	6.403	0.011
2	0.988	2.577	2.646	2.005	0.069
3	0.993	1.891	1.987	1.364	0.096
4	0.960	4.684	7.519	2.831	2.835
5	0.408	27.641	27.528	13.670	-0.113

Degrees 1 and 2 show underfit models, whereas degrees 4 and 5 show increasingly overfit models. The optimum results were obtained for degree 3. Furthermore, the training time for the degree 3 model was very short, has the lowest error of all the models, and the highest R^2 score as well.

The best linear regression model tested in this study was a linear regression model with polynomial features of degree 3. This model will be compared to parts 2 and 3 of the problem statement. Note that many more models can be created: Linear regression models with Gaussian basis functions, Polynomial features implemented with regularization models, etc., however this 3^{rd} degree polynomial linear regression appears to give really great results and is satisfactory for this data set. There is considerable confidence that this model would provide good predictions for future data points. The parity plot for this model is shown in Figure 8. The model performance metrics are shown in Table 3.

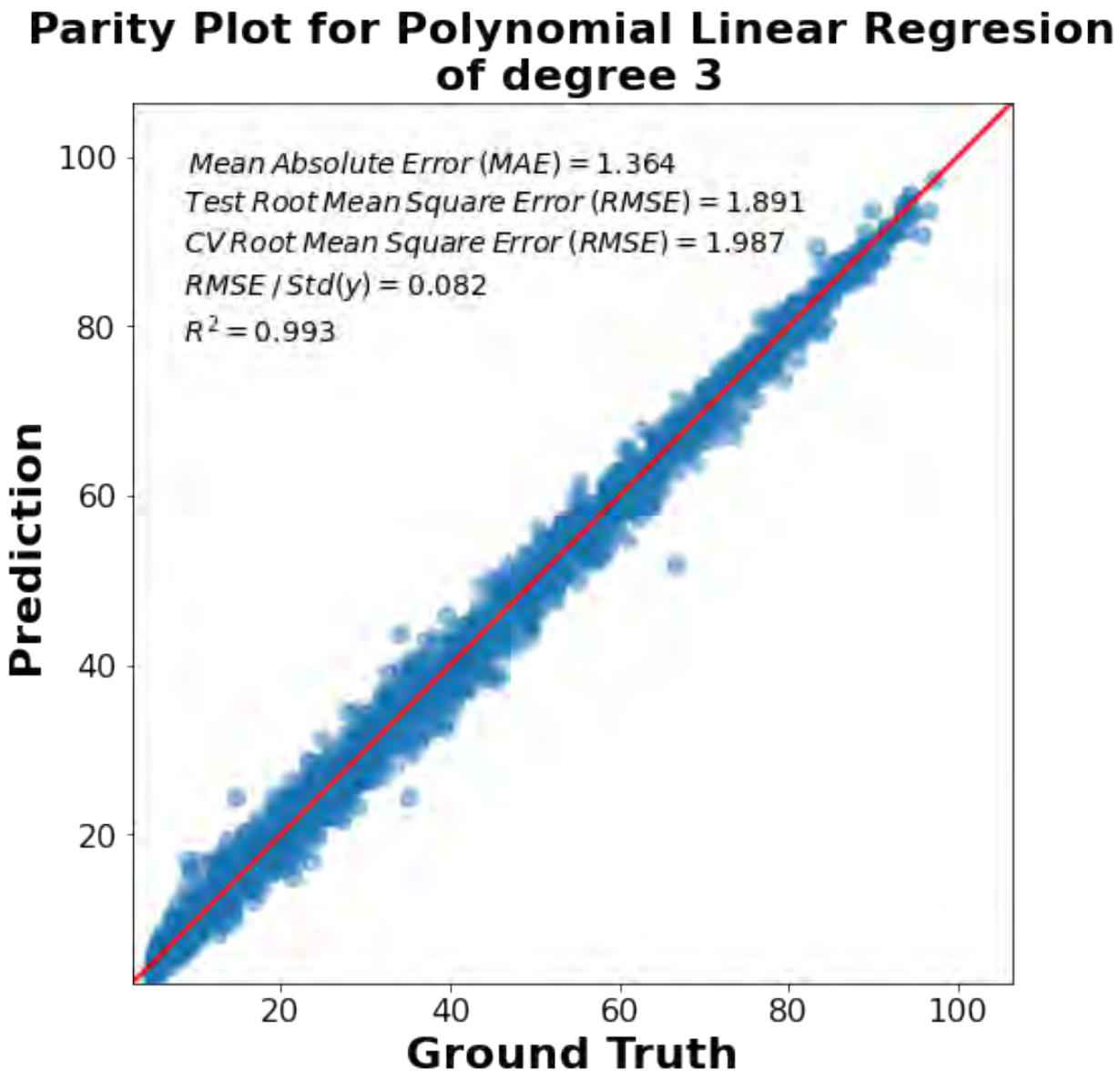


Figure 8: Parity Plot for the final linear regression model with polynomial features of degree 3.

Table 3: Model Performance for the final linear regression model with polynomial features of degree 3.

R^2	Test RMSE	CV MAE	Test MAE	CV RMSE - Test RMSE
0.993	1.891	1.987	1.364	0.096

Problem 2

Use the Gaussian Process regression method. Explain your choice of the kernel and comment on the methods used for the optimization of the hyper-parameters in the kernels. Hint: GPR models might face convergence issues when dealing with large datasets and high computational time for training. You can start with an initial number of the most important input regressors (5 PCs) based on your observations from Q1 and add additional regressors for better models if necessary. You can choose a random set for training datapoints consisting of 20-50% of the entire dataset. Depending on the choice of your kernel the computational time would increase with the number of input points and input regressors as well as the number of optimisation restarts.

Solution

In order to determine the best GPR model, different parameter sweeps and optimizations were performed. One group of parameters studied was the percentage of datapoints the GPR model is trained on. Different training / testing splits of 20%/80%, 30%,70%, 40%/60%, and 50%/50% were used for this study. The other group of parameters investigated were the number of parameters the models were trained on. Due to the large amount of time needed to train these GPR models, 3 training sets were used for this study : one set containing regressors PC1 - PC5, another set containing regressors PC1 - PC10, and the final set containing all the regressors.

The kernels used for training all of these models were the same during these parameter optimizations. An anisotropic RBF kernel (ARDSE) and White Noise kernel were used for these studies. As stated in SKLearn documentation on Gaussian Process Regression, the hyperparamters of the kernels are optimized during model fitting by maximizing the log-marginal-likelihood (LML). White Noise was used since much of the data seemed to be relatively noisy, and anisotropic RBF was used for its expected good performance. Although this isn't a perfect way to make conclusions about the best parameters to use, it is assumed that the change in training / test splits and number of regressors trained on are generalizeable to any GPR model with different kernels.

Finally, GPR models with differnt kernels were studied. Different combinations of Anisotropic RBF and White noise kernels were tested to see how the model would behave.

First, a GPR model using a 20%/80% training / testing split was trained on 3 different sets of regressors varying in number of regressors. The first model used the first 5 regressors, the second used the first 10, and the last model used all 15 regressors. The models were compared using parity plots. These plots are shown in Figure 9. The metrics of each model tested are shown in Table 4. The more regressors that were used, the better the model performed. Higher number of regressors to train resulted in lower training error, a higher R^2 score, and tighter spread on the predictions in terms of the average standard deviation of the predicted values. The error bars shown in the plots are 1 std deviation of error about the point. The model with 15 regressors shows the best performance.

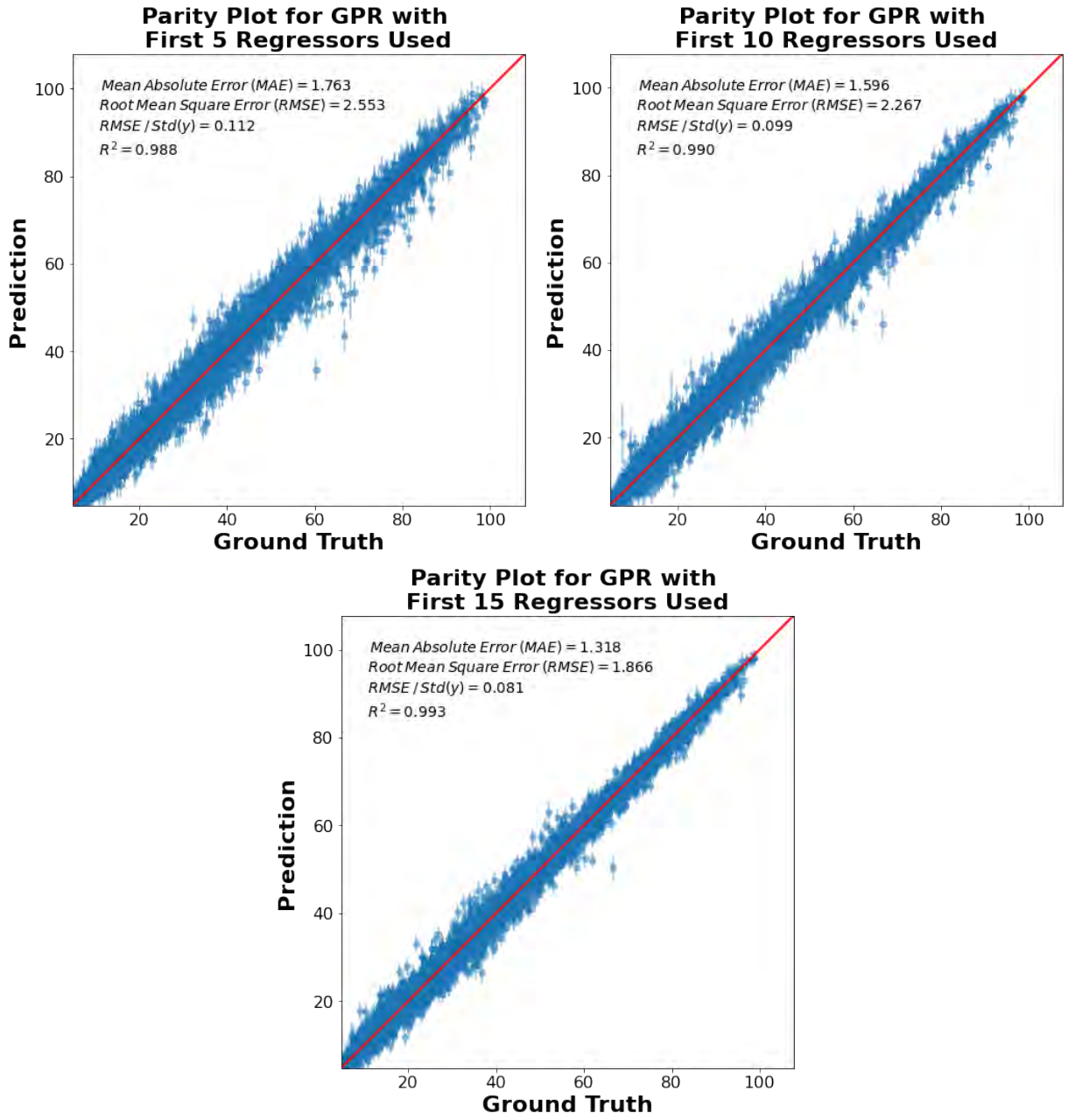
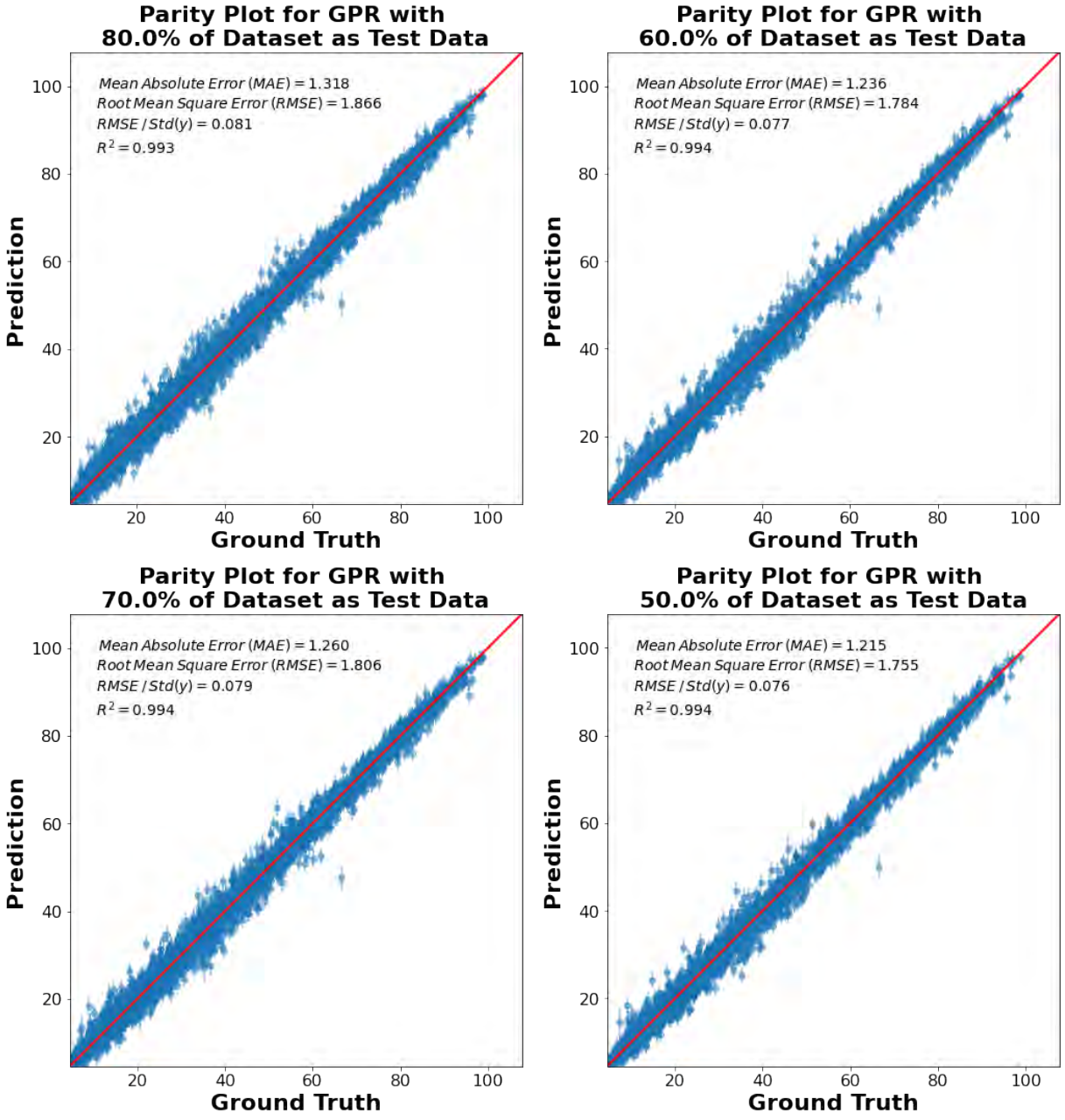


Figure 9: Parity Plots for GPR models with an anisotropic and White Noise kernel, trained on a 20%/80% training/testing data split, with varying number of regressors that the model was trained on.

Table 4: Model Performance for GPR models with an anisotropic and White Noise kernel, trained on a 20%/80% training/testing data split, with varying number of regressors that the model was trained on.

# of Regressors	R^2	Test RMSE	Test MAE
First 5	0.988	2.553	1.763
First 10	0.990	2.267	1.596
All 15	0.993	1.866	1.318

Next, GPR models with different training splits were used. Four models were created and trained on 20%, 30%, 40%, and 50% of the data. The parity plots obtained from this study are shown in Figure 10. The metrics of each model tested are shown in Table 5. Although GPR models trained with more data exhibited better R^2 values and lower errors, the performance gains were marginal, especially when considered with respect to the amount of time required to train the model with more data points. Training time increases very quickly with a higher number of training points. For this reason, GPR models trained on 20% of the data were chosen as the best model to build.




```
Kernels and their Parameters: RBF(length_scale=[26.4, 2.01, 2.41, 23.7, 1.51, 7.61, 33, 3.48, 9.76, 3.93, 1.85, 2.33, 4.39, 1.82, 3.43]) + WhiteKernel(noise_level=0.0048)
GPR_score =  0.9934204019682028
Kernels and their Parameters: RBF(length_scale=[26.5, 1.85, 2.49, 29.8, 1.39, 6.63, 30.3, 3.18, 21, 3.61, 1.87, 2.79, 4.3, 2.79, 3.41]) + WhiteKernel(noise_level=0.00509)
GPR_score =  0.9938504499934386
Kernels and their Parameters: RBF(length_scale=[25.9, 1.85, 2.26, 37.7, 1.33, 5.53, 20.3, 3.38, 12.8, 3.13, 1.92, 2.55, 3.79, 3.17, 2.54]) + WhiteKernel(noise_level=0.00494)
GPR_score =  0.9940398244557848
Kernels and their Parameters: RBF(length_scale=[25.6, 1.72, 2.43, 39.4, 1.29, 5.86, 30.5, 3.11, 14.3, 3.27, 1.96, 2.3, 3.68, 2.99, 2.65]) + WhiteKernel(noise_level=0.00501)
GPR_score =  0.9942564361300571
```

Figure 10: Parity Plots of GPR models with an anisotropic and White Noise kernel, trained on all 15 regressors, with varying sizes of testing / training data splits (training data split varies from 20% - 50% of the dataset). Also, model hyperparamter outputs are displayed below the parity plots.

Table 5: Model Performance for GPR models with an anisotropic and White Noise kernel, trained on all 15 regressors, with varying sizes of testing / training data splits (training data split varies from 20% - 50% of the dataset).

Testing Split	R^2	Test RMSE	Test MAE
80%	0.993	1.866	1.318
70%	0.994	1.806	1.236
60%	0.994	1.784	1.260
50%	0.994	1.755	1.215

At this point, the GPR model is being trained on 20% of the entire dataset, and all 15 regressors are being used as inputs. The optimizer being used is Up until this point, all models were trained on a kernel consisting of an anisotropic RBF and White Noise kernel. The results from this model seem to work very well, however different permutations of these two models as well as an isotropic RBF were tested. The kernel combinations tested were just an isotropic RBF, anisotropic RBF, isotropic RBF + White Noise kernel, anisotropic RBF + White Noise kernel, White Noise kernel only,and a combination of all 3 kernels. Figure 11 shows the parity plots along with error from the models with each kernel. The metrics of each model tested are shown in Table 6.

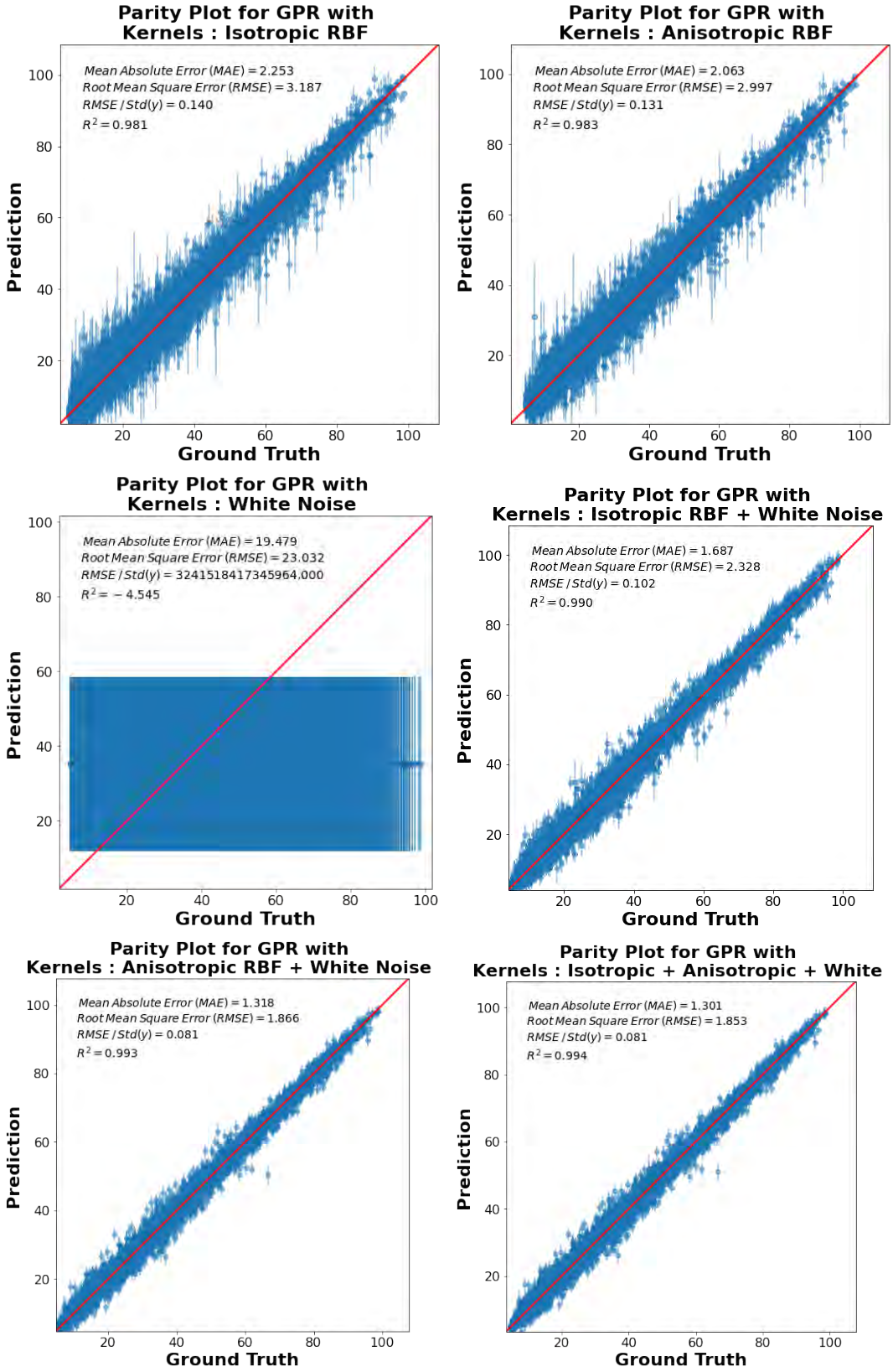


Figure 11: Parity Plots for GPR models trained on a 20%/80% training/testing data split, trained on all 15 regressors, varying the kernels used to train them.

Table 6: Model Performance for GPR models trained on a 20%/80% training/testing data split, trained on all 15 regressors, varying the kernels used to train them.

Kernel	R^2	Test RMSE	Test MAE
Isotropic RBF	0.981	3.187	2.253
Anisotropic RBF	0.983	2.997	2.063
White Noise	-4.545	23.032	19.479
Isotropic RBF + White Noise	0.990	2.328	1.687
Anisotropic RBF + White Noise	0.993	1.866	1.318
Isotropic RBF + Anisotropic RBF + White Noise	0.994	1.853	1.301

The best performing model used a kernel that utilized a combination of all 3 tested kernels. A close contender was the isotropic RBF + White Noise kernel model however. While slightly worse, the isotropic RBF + White Noise kernel model took much less time to train and prodcued comprable results. similarly, the anisotropic RBF + White Noise Kernel model performed nearly as well as the model using all three kernels. The White noise kernel produced extremely poor resutls, while the isotropic RBF kernel alone had good error and R^2 value compared to linear models, but had room for improvement.

To further refine this model, a series of N-restarts values were tested. N-restarts optimization allows for potentially better optimization of the kernel hyperparameters. With N-restarts, an intial guess is made for the hyperparameters, they are maximized using LML, and then a new initial guess is randomly sampled and the optimization is started again. This optimization method was run for N-restarts from 1-5, however the results were identical to not using N-restarts. Figure 12 shows the hyperparameter output from each kernel optimization during this study. There is no difference between any of these models, and therefore N-restarts optimization was abandoned as a method for optimizing the model.

```
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
Kernels and their Parameters: RBF(length_scale=20.2) + RBF(length_scale=[42.2, 2.08, 2.25, 14.8, 1.48, 6.8, 27.9, 2.82, 9.72, 3.14, 1.67, 2.27, 4.29, 1.73, 3.74]) + WhiteKernel(noise_level=0.00472)
```

Figure 12: GPR model parameters obtained from employing 1-5 N-restart optimization runs. There is no change in the model using the N-restarts optimizer.

The best model obtained from GPR was determined to be a model trained with 15 regressors, trained on 20% of the dataset, with an isotropic RBF kernel + anisotropic RBF kernel + White Noise kernel. Finally, The parity plot for this model with CV RMSE, Test RMSE, and R^2 value are shown in Figure 13. The metrics for this model are shown in Table 7. Cross validation error was determined and compared to test error to ensure no overfit. The Test RMSE was determined to be 1.853, whereas the cross validation RMSE was determined to be 1.936. Since there is no big difference between these two errors (they are relatively

close), it can be concluded that this model is not too much of an overfit.

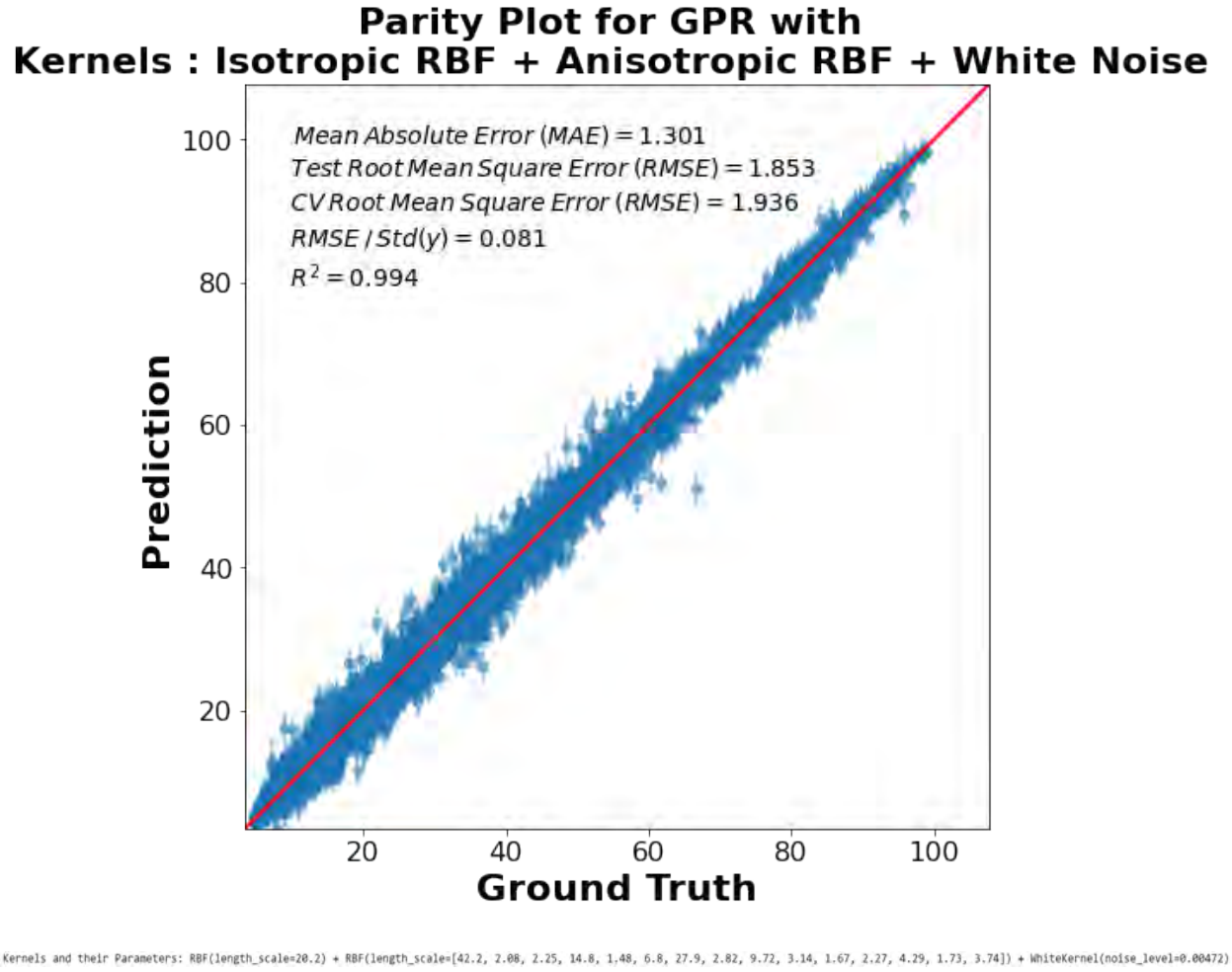


Figure 13: Parity Plot for the final GPR model with Isotropic RBF, Anisotropic RBF, and White Noise kernels, trained on 20% of dataset using all regressors.

Table 7: Model Performance for the final GPR model with Isotropic RBF, Anisotropic RBF, and White Noise kernels, trained on 20% of dataset using all regressors.

R^2	Test RMSE	CV RMSE	Test MAE	CV RMSE - Test RMSE
0.994	1.853	1.936	1.301	0.083

Problem 3

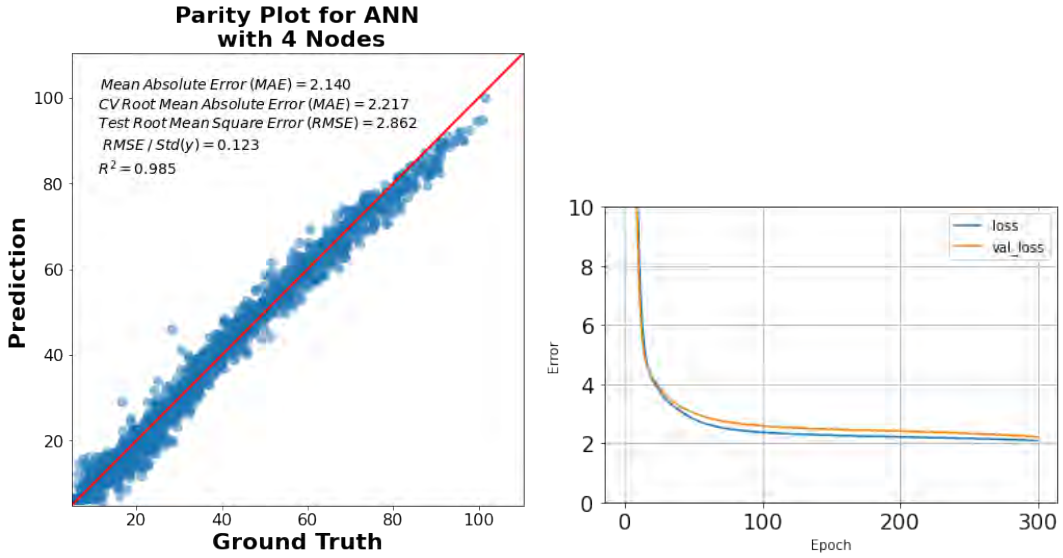
Use Neural Networks. Discuss the choices made in your network architecture (number of layers, activation functions, etc.)

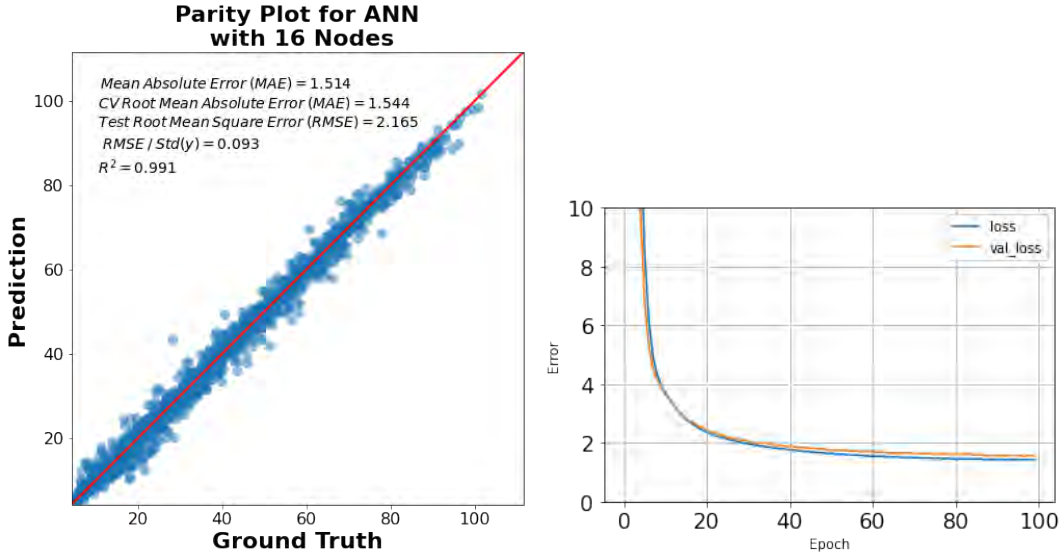
Solution

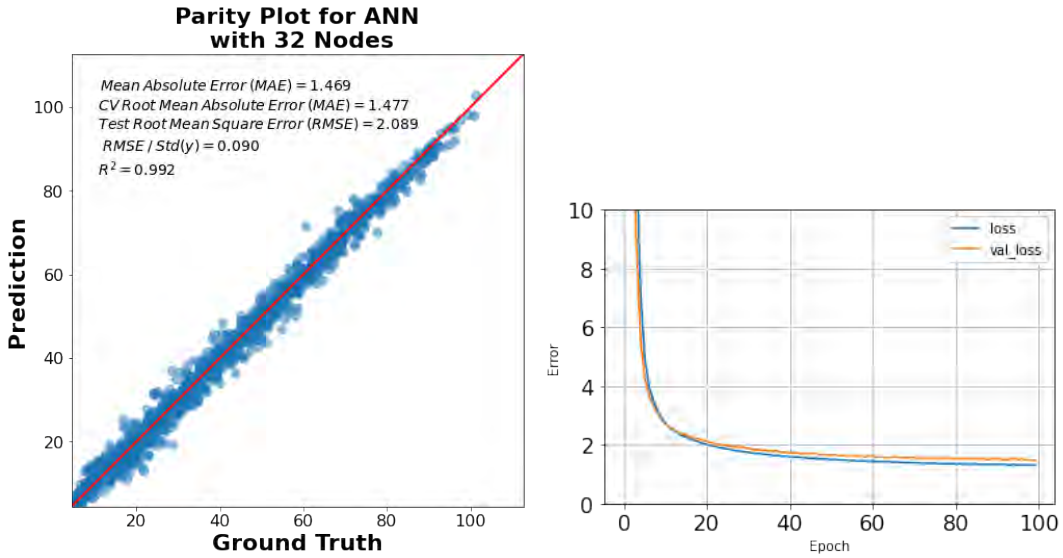
Different ANN models were tested. First, 1 hidden layer models were testing, varying the number of nodes in the hidden layer. Then, a second layer was introduced, and the effect on the model was determined. For these first two model tests, all activation functions used were ReLu since it is widely used as a default for its good results. Additionally, each model was trained for 100 epochs since this seemed to be a good point to stop training the model during preliminary testing. Any further training generally resulted in overfitting - a growing difference between validation loss and test loss would develop. The best and simplest of these models was chosen, and different activation functions were tested to see if the model could be improved further. Note that mean absolute error is used for training the ANN models. This was done to allow the neural networks trained to be more robust to noise in the training data.

Note that for all ANN models, all 15 regressors were used to train the model, and Mean Absolute Error was used to compare the test loss to validation loss. Additionally, since the loss did not settle for the 4 node model after 100 epochs, the 4 node model was trained for 300 epochs to get more stable loss.

The first models tested had 1 hidden layer. The number of nodes in this layer tested were 4, 16, 32, 64, and 128 nodes. The parity and loss plots for these models are shown in Figure 14. The metrics for each of these models are tabulated in Table 8. As the number of nodes in the hidden layer increases, so does the accuracy of the model, however it can be seen that the difference between the validation loss and testing loss at the end of 100 epochs seems to change depending on the number of nodes in the model. The model that seems to give the best balance between accuracy and minimizing overfit is the 128 node model, which presents the smallest difference between validation and test loss, as well as a very accurate model with an R^2 value of 0.993.


Parity Plot for ANN with 4 Nodes
Mean Absolute Error (MAE) = 2.140
CV Root Mean Absolute Error (MAE) = 2.217
Test Root Mean Square Error (RMSE) = 2.862
RMSE / Std(y) = 0.123
R^2 = 0.985
Prediction
Ground Truth
loss
val_loss
Error
Epoch


Parity Plot for ANN with 16 Nodes
Mean Absolute Error (MAE) = 1.514
CV Root Mean Absolute Error (MAE) = 1.544
Test Root Mean Square Error (RMSE) = 2.165
RMSE / Std(y) = 0.093
R^2 = 0.991
Prediction
Ground Truth
loss
val_loss
Error
Epoch


Parity Plot for ANN with 32 Nodes
Mean Absolute Error (MAE) = 1.469
CV Root Mean Absolute Error (MAE) = 1.477
Test Root Mean Square Error (RMSE) = 2.089
RMSE / Std(y) = 0.090
R^2 = 0.992
Prediction
Ground Truth
loss
val_loss
Error
Epoch

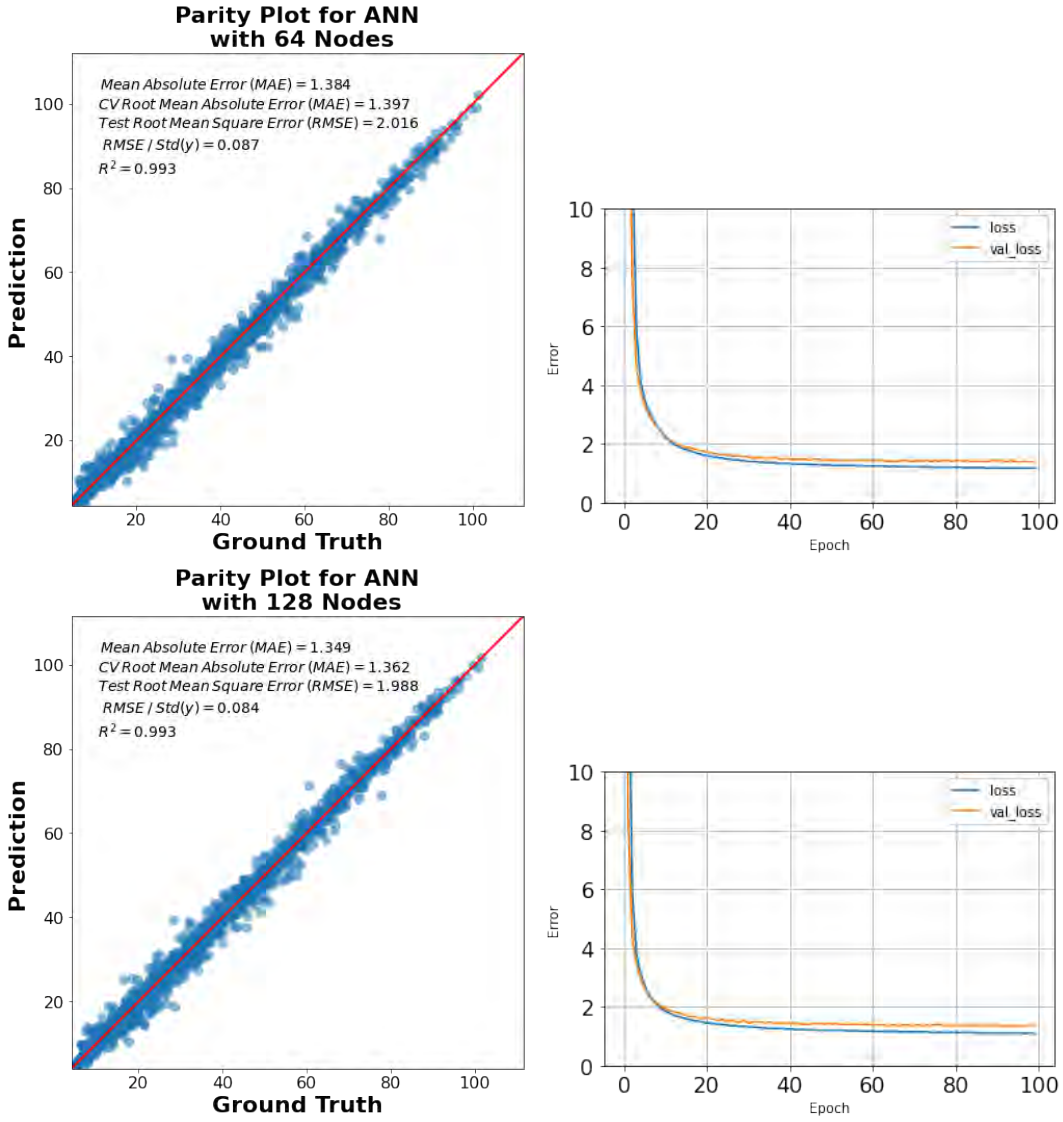


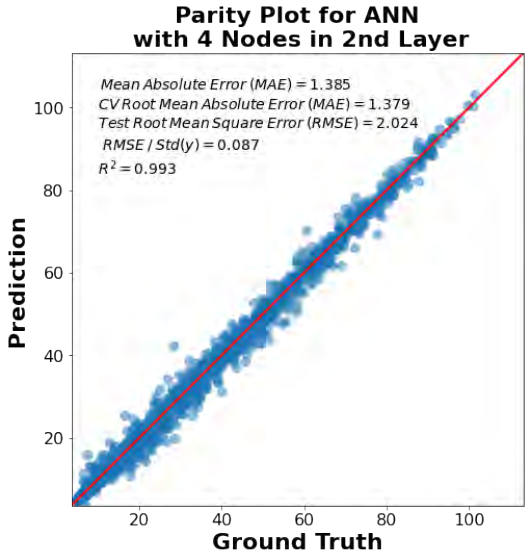
Figure 14: Parity Plots for 1 hidden layer ANN models trained on a 80%/20% training/testing data split, trained on all 15 regressors, varying the number of nodes in the first hidden layer.

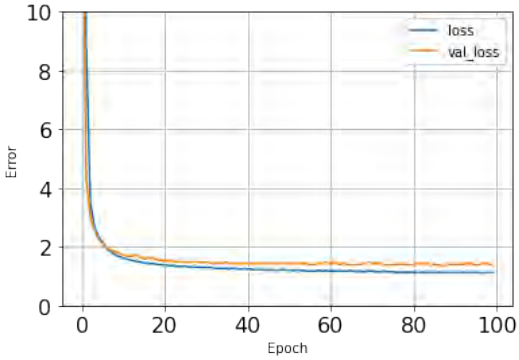
Table 8: Model Performance for 1 Hidden Layer ANN Varying Number of Nodes in Hidden Layer

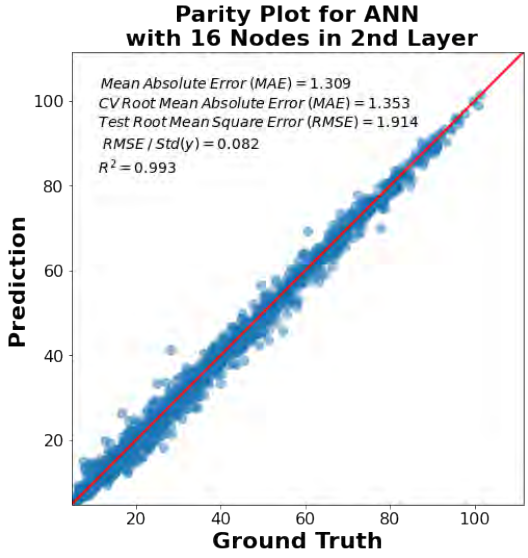
# Nodes	R^2	RMSE	Test MAE	CV MAE	CV MAE - Test MAE
4	0.985	2.862	2.140	2.217	0.077
16	0.991	2.165	1.514	1.544	0.030
32	0.992	2.089	1.469	1.477	0.008
64	0.993	2.016	1.384	1.397	0.013
128	0.993	1.988	1.349	1.362	0.013

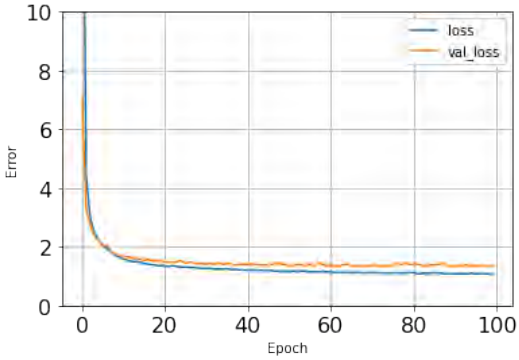
Next, a similar test was performed to see if adding another layer of nodes could improve this 64 layer model. The same procedure was carried out as the previous study, but with a second layer with varying number of nodes in second layer.

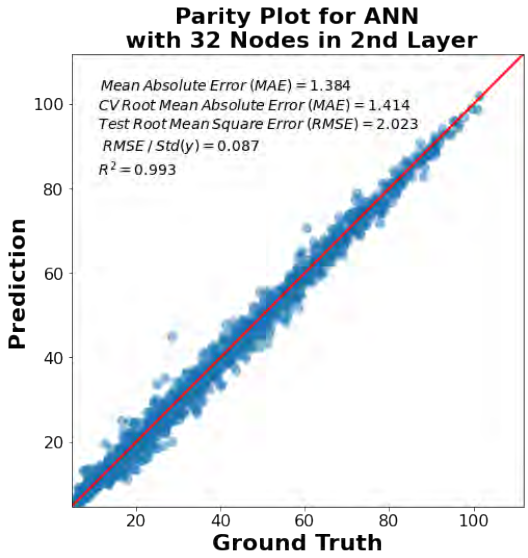
The parity and loss plots for varying the number of second layer nodes are shown in Figure 15. The metrics for each of these models are tabulated in Table 9. It can be seen that adding a second layer adds marginal improvement over only one layer in the model. The best model with 2 layers is the model with 16 nodes in the second layer. However, this model's performance is essentially the same as that of an ANN with 128 nodes in a single hidden layer, and the 2 layer model takes much longer to train. Therefore, the simpler model is chosen as the better model in this case. Further studies are performed on an ANN with 1 hidden layer with 128 nodes.

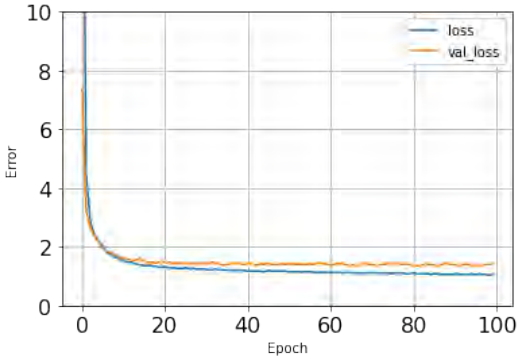

Parity Plot for ANN
with 4 Nodes in 2nd Layer
Mean Absolute Error (MAE) = 1.385
CV Root Mean Absolute Error (MAE) = 1.379
Test Root Mean Square Error (RMSE) = 2.024
RMSE / Std(y) = 0.087
R^2 = 0.993
Prediction
Ground Truth


loss
val_loss
Error
Epoch


Parity Plot for ANN
with 16 Nodes in 2nd Layer
Mean Absolute Error (MAE) = 1.309
CV Root Mean Absolute Error (MAE) = 1.353
Test Root Mean Square Error (RMSE) = 1.914
RMSE / Std(y) = 0.082
R^2 = 0.993
Prediction
Ground Truth


loss
val_loss
Error
Epoch


Parity Plot for ANN
with 32 Nodes in 2nd Layer
Mean Absolute Error (MAE) = 1.384
CV Root Mean Absolute Error (MAE) = 1.414
Test Root Mean Square Error (RMSE) = 2.023
RMSE / Std(y) = 0.087
R^2 = 0.993
Prediction
Ground Truth


loss
val_loss
Error
Epoch

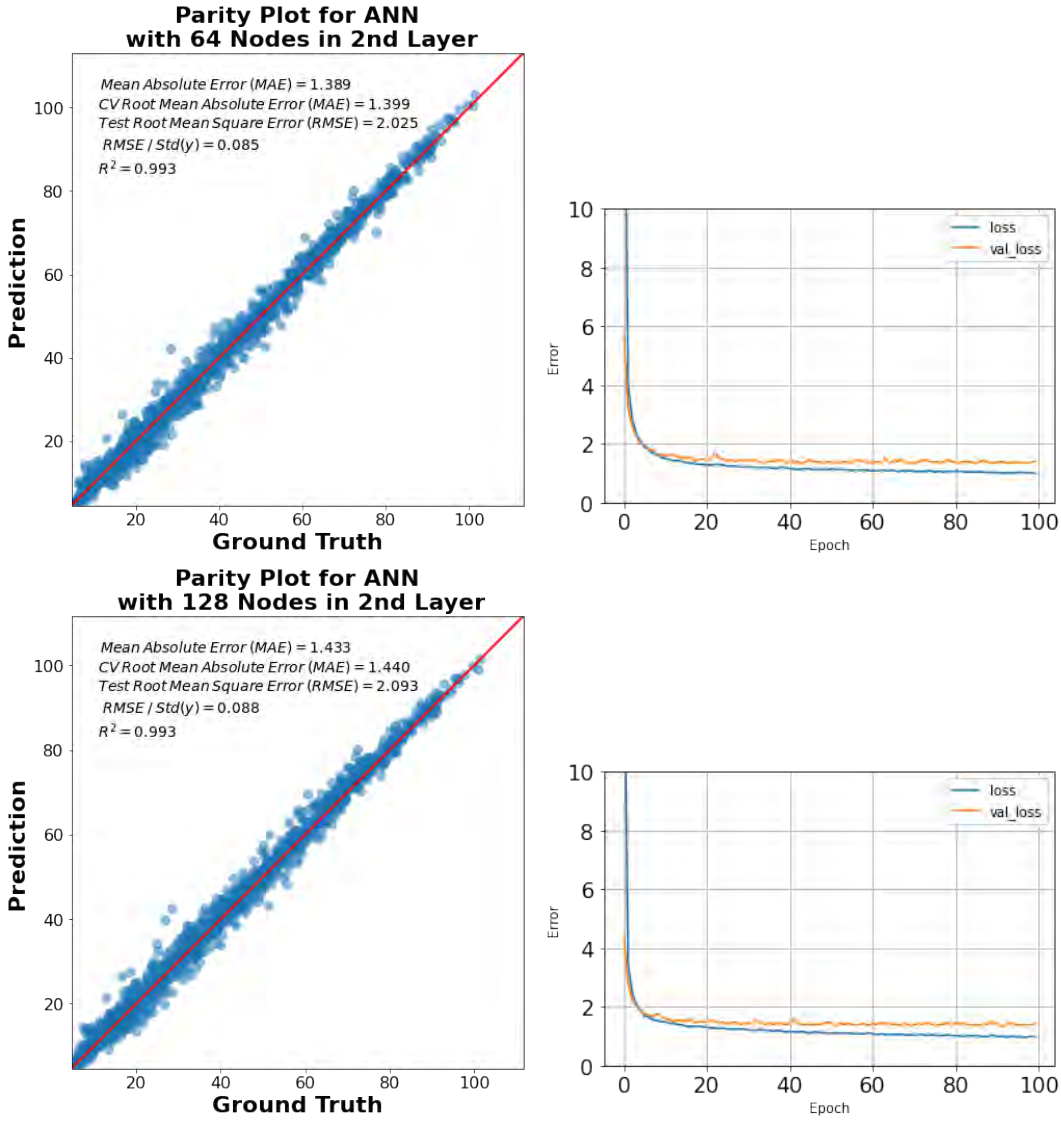
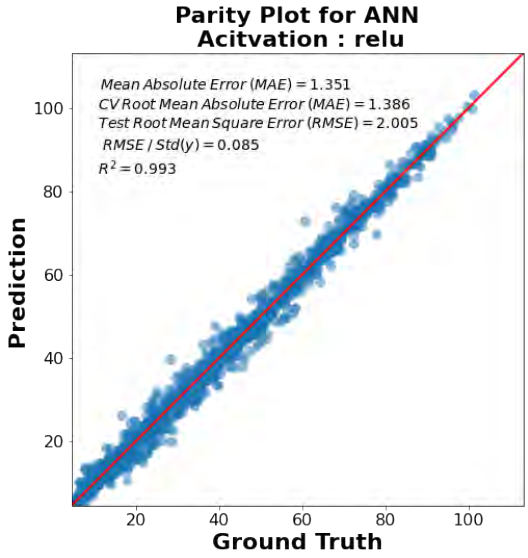


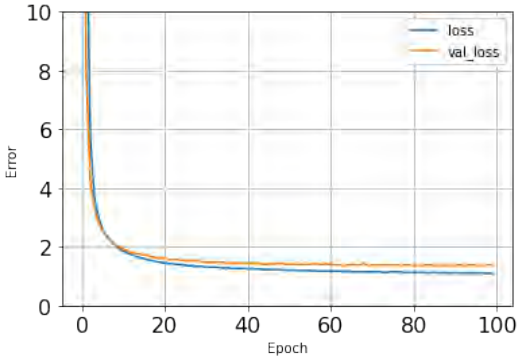
Figure 15: Parity Plots for 2 hidden layer ANN models trained on a 80%/20% training/testing data split, trained on all 15 regressors, varying the number of nodes in the second hidden layer. There are 128 nodes in the first hidden layer.

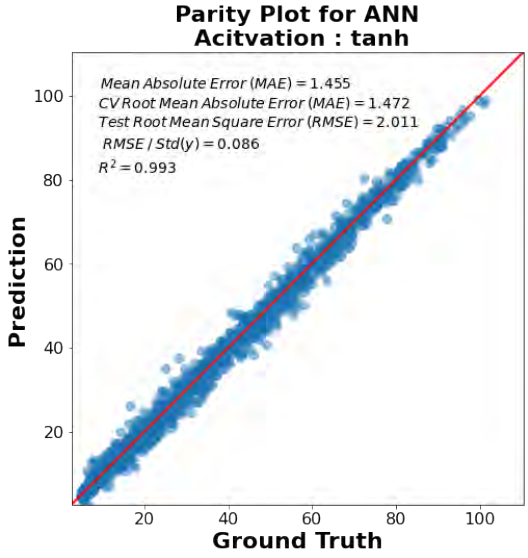
Table 9: Model Performance for 2 Hidden Layer ANN Varying Number of Nodes in the 2^{nd} Hidden Layer

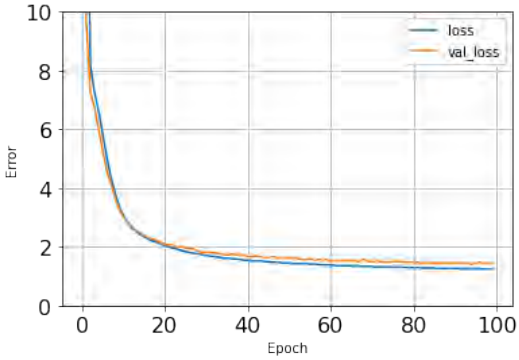
# Nodes in Layer 2	R^2	RMSE	Test MAE	CV MAE	CV MAE - Test MAE
4	0.993	2.024	1.385	1.379	-0.006
16	0.993	1.914	1.309	1.353	0.044
32	0.993	2.023	1.384	1.414	0.030
64	0.993	2.025	1.389	1.399	0.010
128	0.993	2.093	1.433	1.440	0.007

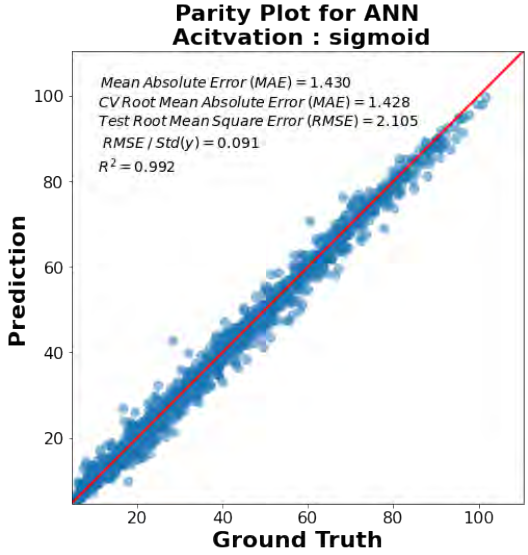
Finally, different activation functions were compared to the performance of ReLu. Sigmoid, Softmax, Softmax, and *tanh* activation functions were tested and compared. The parity plots and corresponding information from this study is shown in Figure 16. The metrics for each of these models are tabulated in Table 10. It can be seen that the best performance is obtained from the ReLu and *tanh* activation functions, which have the lowest errors and insignificant difference between validation and test error. This makes sense since ReLu is widely regarded as a good performing activation function. The ReLu activation function produces great performance. The final model chosen is the model with a ReLu activation function, since the RMSE and MAE errors are lowest across the board, with a low indication of overfitting via the difference between validation error and test error.

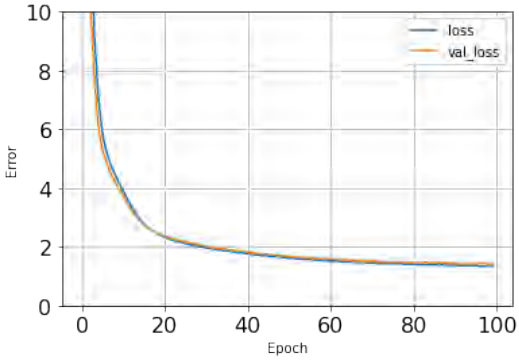

Parity Plot for ANN
Acitvation : relu
Mean Absolute Error (MAE) = 1.351
CV Root Mean Absolute Error (MAE) = 1.386
Test Root Mean Square Error (RMSE) = 2.005
RMSE / Std(y) = 0.085
R^2 = 0.993
Prediction
Ground Truth


loss
val_loss
Error
Epoch


Parity Plot for ANN
Acitvation : tanh
Mean Absolute Error (MAE) = 1.455
CV Root Mean Absolute Error (MAE) = 1.472
Test Root Mean Square Error (RMSE) = 2.011
RMSE / Std(y) = 0.086
R^2 = 0.993
Prediction
Ground Truth


loss
val_loss
Error
Epoch


Parity Plot for ANN
Acitvation : sigmoid
Mean Absolute Error (MAE) = 1.430
CV Root Mean Absolute Error (MAE) = 1.428
Test Root Mean Square Error (RMSE) = 2.105
RMSE / Std(y) = 0.091
R^2 = 0.992
Prediction
Ground Truth


loss
val_loss
Error
Epoch

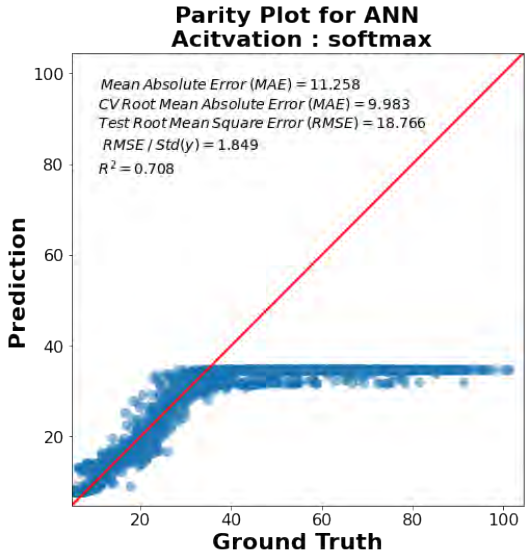


Figure 16: Parity Plots for 1 hidden layer ANN models with 128 nodes, trained on a 80%/20% training/testing data split, trained on all 15 regressors, varying the activation function used in the model.

Table 10: Model Performance for 1 Hidden Layer ANN with 128 Nodes Varying the Activation Function Used

Activation Function	R^2	RMSE	Test MAE	CV MAE	CV MAE - Test MAE
ReLu	0.993	2.005	1.351	1.386	0.035
tanh	0.993	2.011	1.455	1.472	0.017
sigmoid	0.992	2.105	1.430	1.428	-0.002
softmax	0.708	18.766	11.258	9.983	-1.275

The final model chosen for the ANN is a 1 hidden layer network trained on all 15 regressors, using 80% of the data, and using ReLu as the activation function. The parity plot for this model is shown in Figure 17. The metrics for this model are tabulated in Table 11.

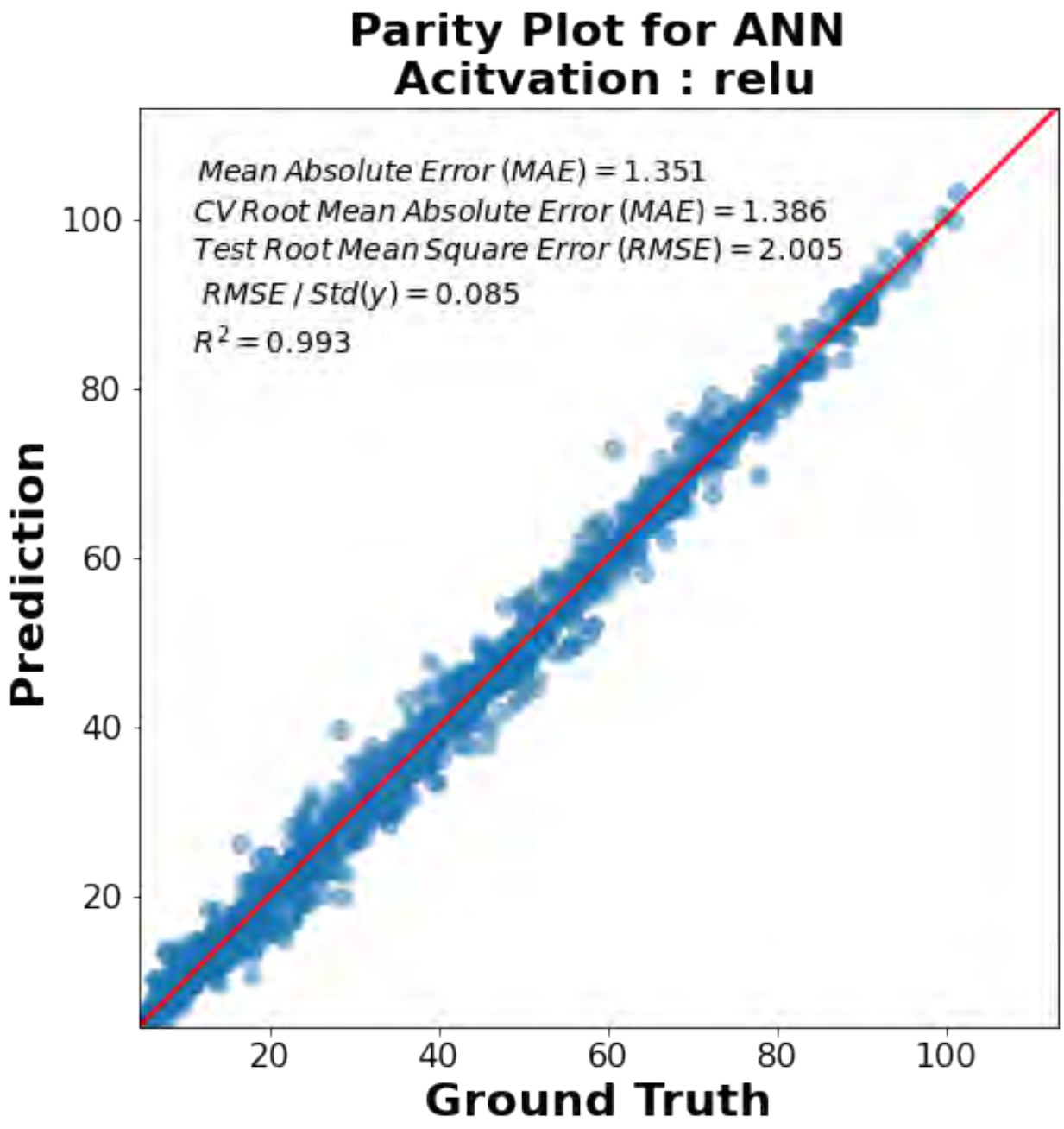


Figure 17: Parity Plot for the final ANN model with 128 nodes in 1 hidden layer trained on 20% of dataset using all regressors, using ReLu as the activation function.

Table 11: Model Performance for the final ANN model with 128 nodes in 1 hidden layer trained on 20% of dataset using all regressors, using ReLu as the activation function.

R^2	Test MAE	CV MAE	Test RMSE	CV MAE - Test MAE
0.993	1.351	1.386	2.005	0.035

Final Comparison

Discuss the relative pros and cons of all the models you have built for this homework. Which ML method performed the best? Why?

Solution

The final models produced from each model study are as follows:

1. Linear Regression: a Linear Regression model with polynomial features of degree 3, trained on 80% of the data, with all 15 regressors.
2. GPR: A GPR model trained on 20% of the data with all 15 regressors. The kernel used is a combination of an isotropic RBF kernel, an anisotropic RBF kernel, and a white noise kernel. `enumerate` environment.
3. ANN: A 1 hidden layer model trained on all 15 regressore with 80% of the data as training data, and ReLu activation function

The parity plots for these models are shown in Figure 18, and a table comparing metrics of these models is displayed in Table 12.

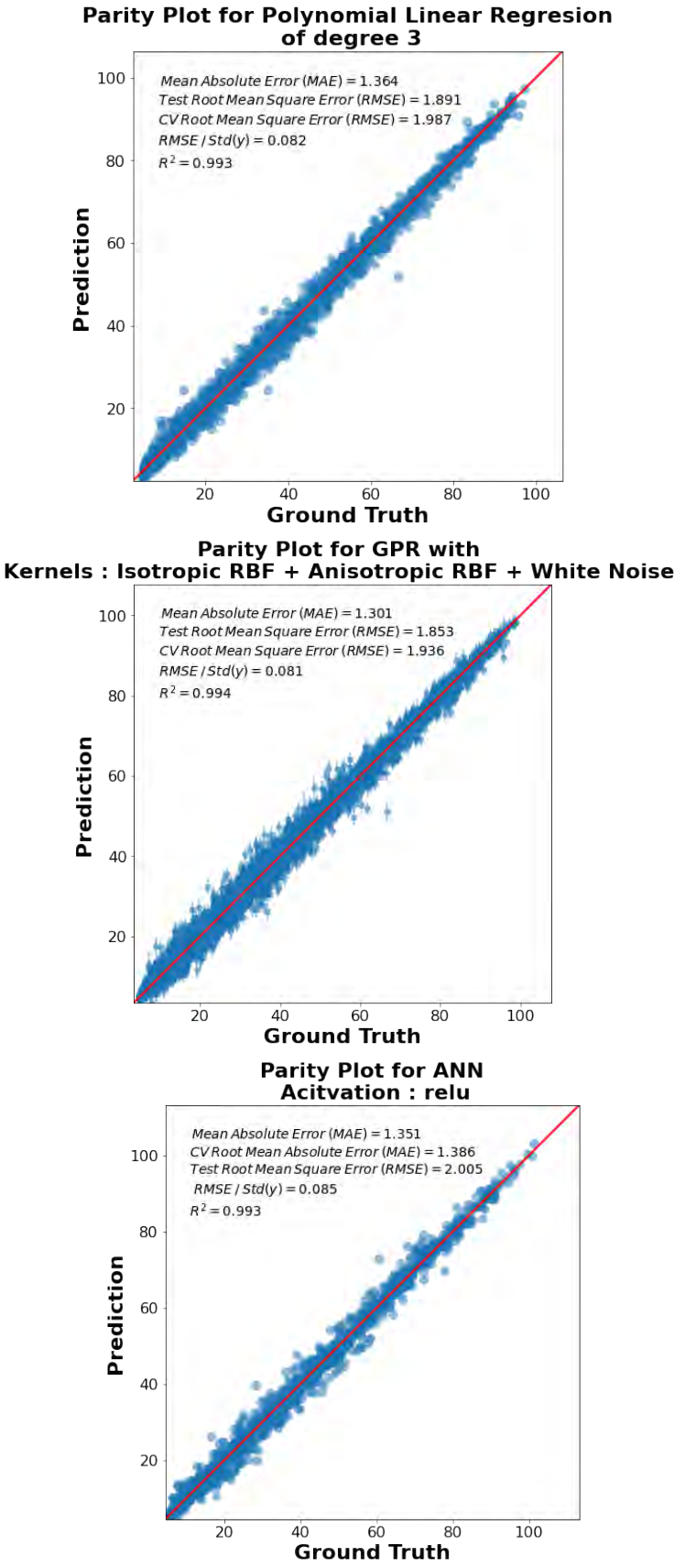


Figure 18: Parity Plots for the final machine learning models for Linear Regression, GPR, and ANN models.

Table 12: Model Performance for the final machine learning models for Linear Regression, GPR, and ANN models.

Model	R^2	Test RMSE	CV RMSE	Test MAE	CV MAE	CV Loss - Test Loss
Linear Regression	0.993	1.891	1.987	1.364	-	0.096
GPR	0.994	1.853	1.936	1.301	-	0.083
ANN	0.993	2.005	-	1.351	1.386	0.035

All 3 models have nearly the same accuracy, however each model takes increasing amounts of time to train. The linear regression model was very quick to train, whereas the GPR and ANN models took a much longer time to train, and are more complex than the linear regression model. In my opinion, the best model is the linear regression model. This model is the simplest of the 3, takes the least time to train, and produces an incredibly good fit to the data for its computational cost. However, by looking at the metrics, it is clear that the best performing model is the GPR model. It produces the lowest error and has the lowest indication of an overfit of the three machine learning models.

One downside of both linear regression and ANN models is that they require a lot of data to be trained on. GPR on the other hand is very insensitive to the amount of data its model is trained on. From extra testing done, it was seen that the amount of training data can be reduced to surprisingly low levels (5% of the entire data set), and the GPR model could still produce really great results. If data acquisition is difficult for the specific application that a machine learning model should be applied to, then GPR is the best choice. It can produce great predictions with only a small fraction of the data points required to train linear regression and ANN models.

Additionally, whereas the ANN model may not be the machine learning model of choice for this dataset, it may be more desirable for more complicated relationships between inputs and outputs that a linear regression of GPR model just would not be able to capture. This dataset exhibits a relatively simple (polynomial) relationship between its inputs and outputs. However, if there is nonlinearity in the data, or a classification must be obtained, then a linear regression or GPR model may not be good enough for that specific application. It is important to note that an ANN model for this data is somewhat of an overkill. No nonlinearities are present and combined with its extensive training time compared to linear regression, it makes ANN a bad choice for this data set. However more nonlinear relationships would require an ANN to capture the relationships that a linear regression, and even (likely) a GPR could not capture.

The best model in this case is the model produced with GPR, however I would be hesitant to use it on an actual project due to the amount of training time it demands.