A COMPARISON OF MACHINE LEARNING ALGORITHMS FOR THE PREDICTION OF THERMAL-EXPANSION OF SPHERICAL SHELLS

by

Kolby Bumgarner

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Mechanical Engineering

Charlotte

2022

Approved by:

_____
Dr. Harish Cherukuri

_____
Dr. Yawo Amengonu

_____
Dr. Russell Keanini

ABSTRACT

KOLBY BUMGARNER. A Comparison of Machine Learning Algorithms for The Prediction of Thermal-Expansion of Spherical Shells. (Under the direction of DR. HARISH CHERUKURI)

Machine Learning (ML) applications in material science is an active area of research due to its accurate predictive capabilities while also being generalizable. Experimentation where a physical model of the problem is constructed has been widely used in the past and is still used today but is very costly in time and capital. Simulation using virtual models can be much more efficient than experimental modeling but still takes time to construct and solve computationally. ML uses statistics and data science methods to predict a variety of different problem configurations quickly while remaining accurate. While becoming an increasingly popular method, different ML model types have their own advantages and disadvantages. In this work, ML models are trained to predict the thermal expansion of a hollow sphere so that these advantages and disadvantages may be studied. Six ML model types were explored by training on instances of data generated from Finite Element Analysis (FEA) data then compared using error metrics. Sample size analysis was completed to determine required amounts of input data to produce an accurate model for each ML type. A noise study was conducted to observe how each model type would react to varying levels of noise added to the data. Sensitivity analysis was carried out on the optimal model to test how each predictor variable affected the prediction value. Finally, the optimal model was tested against new data that had not been encountered before to simulate how a production ready model would work in the real world.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANN Artificial Neural Network

FEA Finite Element Analysis

GPR Gaussian Processes Regression

KNN K Nearest Neighbors

ML Machine Learning

MPR Multivariable Polynomial Regression

OLS Ordinary Least Squares

RMSE Root Mean Square Error

SVM Support Vector Machine

SVR Support Vector Regression

CHAPTER 1: INTRODUCTION

1.1 Problem Description

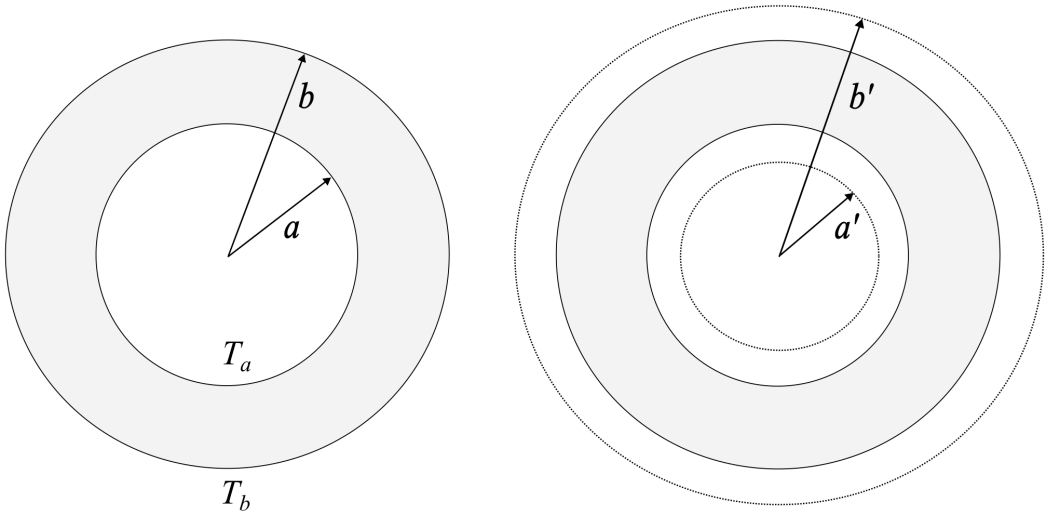


Figure 1.1: The hollow sphere before and after thermal expansion

In this study, the thermal expansion of a hollow sphere due to thermal gradient across the thickness is studied. The sphere is at an initially uniform temperature. At $t = 0$, the temperatures at inner and outer radii are suddenly changed resulting in a contraction or expansion of the sphere. The goal of this exploratory study is to use various learning algorithms to predict the change of thickness of the hollow sphere given input parameters that govern the deformation of the sphere. The predictive and target variables are shown in Fig.1.1 and listed in Table 1.1.

Table 1.1: Parameters to be used in prediction

Symbol	Parameter
ν	Poisson's Ratio
E	Young's Modulus (N/m^2)
α	Thermal Expansion Coefficient (C^{-1})
a	Inner Radius (m)
b	Outer Radius (m)
T_a	Inner Temperature (°C)
T_b	Outer Temperature (°C)
$(b' - a') - (b - a)$	Thickness Change (m)

1.2 Machine Learning

For this problem, machine learning (ML) will be used to attempt to find a predictive method for estimating the thickness change of the hollow sphere. Machine learning algorithms use data driven problem solving methods to create models that predict new information after training on previously known information. Instead of discovering an equation that explains exactly how a set of independent variables relates to a dependent variable, an ML model predicts dependent variables after the model is trained on a set of data with dependent and independent variables.

1.2.1 Machine Learning Example

As an example of machine learning and its application in a mechanical engineering problem, a simple beam deflection problem can be solved. A cantilever beam with a load applied at the unsupported end has a derivable equation which describes the deflection at the unsupported end which is outlined in Eqn.1.1 and viewed in Fig.1.2. F is the force in Newtons applied at the free-end of the beam, L is the total length of the beam in meters from the supported end to the unsupported end, E is Young's Modulus which encapsulates the beam's material properties, and I is the Second

Moment of Inertia which is a term that is derived from the beams cross-sectional geometry. Delta (δ) describes the deflection of the right end due to the force.

$$\delta = \frac{FL^3}{3EI} \tag{1.1}$$

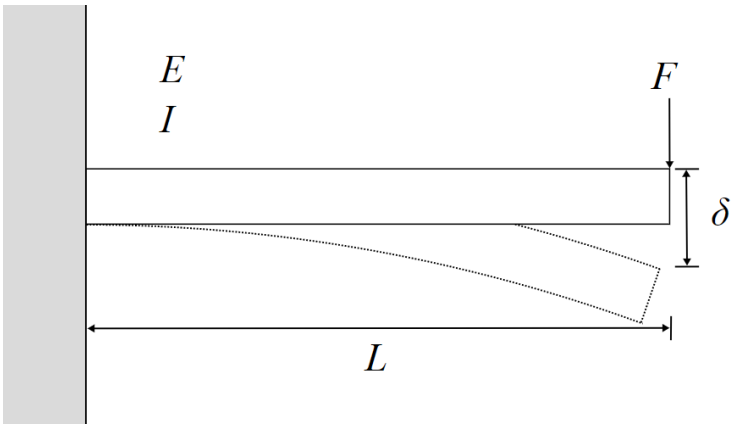


Figure 1.2: Deflection of cantilever beam with force applied unsupported end

Since there is an equation that fully describes this particular problem, a ML predictive model may be tested against the known truth of the equation as the example. For a mental visualization of the problem, the beam can be thought of as a diving board of varying length. If a person stands on the edge of a very short diving board, the deflection down where the person stands will be small. As the length of the diving board increases with the person at the tip of the board, the deflection down increases (in a cubic manner). Table 1.2 shows data generated by using Eqn.1.1 of said diving board to train a machine learning model. The person stays the same weight, thus applying the same force on the edge of the board at any length. The material of the board (aviation grade aluminum) stays the same so the Young's Modulus (E) is constant. The cross sectional area (0.5m wide, 0.005m thick) stays the same which results in the same second moment of inertia (I) for each length.

Table 1.2: Deflection data from a diving board of varying length

F	L	E	I	δ
100	0.25	340000000	0.0005208333	0.000002941176659
100	0.5	340000000	0.0005208333	0.000002941176659
100	0.75	340000000	0.0005208333	0.00002352941327
100	1	340000000	0.0005208333	0.00007941176979
100	1.25	340000000	0.0005208333	0.0001882353062
100	1.5	340000000	0.0005208333	0.0003676470824
100	1.75	340000000	0.0005208333	0.0006352941583
100	2	340000000	0.0005208333	0.001008823594
100	2.25	340000000	0.0005208333	0.001505882449
100	2.5	340000000	0.0005208333	0.002144117784
100	2.75	340000000	0.0005208333	0.002941176659
100	3	340000000	0.0005208333	0.003914706133
100	3.25	340000000	0.0005208333	0.005082353266
100	3.5	340000000	0.0005208333	0.006461765119
100	3.75	340000000	0.0005208333	0.008070588752
100	4	340000000	0.0005208333	0.009926471224
100	4.25	340000000	0.0005208333	0.01204705959
100	4.5	340000000	0.0005208333	0.01445000092
100	4.75	340000000	0.0005208333	0.01715294227
100	5	340000000	0.0005208333	0.0201735307

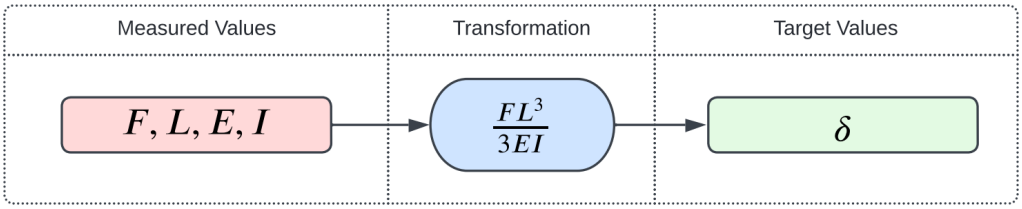


Figure 1.3: How an equation is used to calculate deflection

Figure 1.3 shows how the deflection would be calculated using a transformation of the measured values. If the deflection of the diving board was required, the values could be measured and entered into the equation which transform the values to predict how far down the diving board would bend if a person were to stand at the unsupported edge.

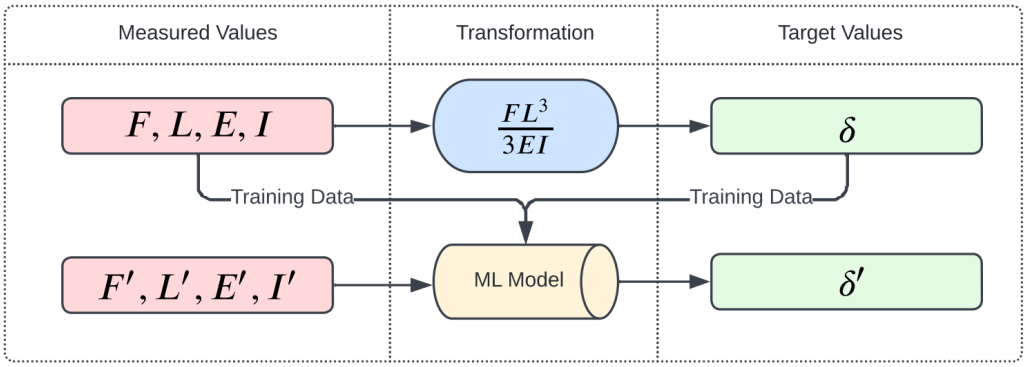


Figure 1.4: How a ML model is used to predict deflection

Figure 1.4 shows how a machine learning model (ML Model) is used in the diving board example specifically. The equation is used to find the deflection using measured values. All the values including the dependent variable (deflection) are used to train a ML model which can then predict new deflection values (δ') from previously unencountered input parameters (F', L', E', I').

After training a machine learning model, the accuracy of the model can be visually evaluated using graphs. Figure 1.5 shows the ML model predictions using the values that trained the model listed in Table 1.2 against the true values calculated using the equation.

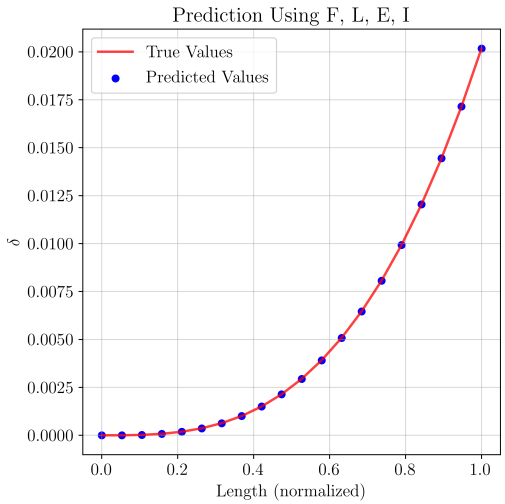


Figure 1.5: ML model predictions on original dataset (Table 1.2)

Figure 1.6 shows predictions of previously unencountered input values for L. Using the new input values, predictions still match closely to the true values that would be calculated with the beam equation.

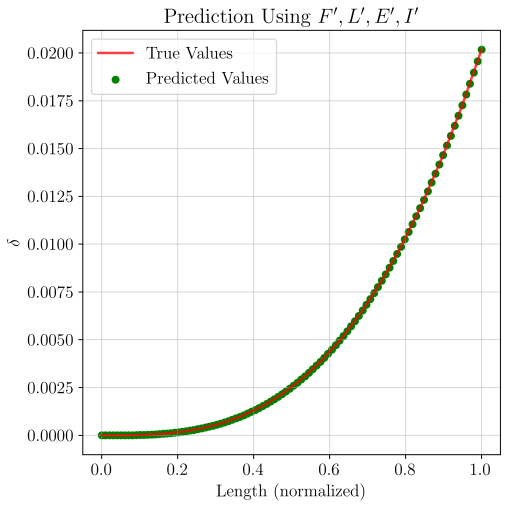


Figure 1.6: ML model predictions on new instances of data

A different graphing technique allows a more general visualization of how accurate a model is as seen in Fig.1.7. With the true value being the x component and y value being the predicted value, the closer a point is to the $x = y$ line, the more accurate the prediction is.

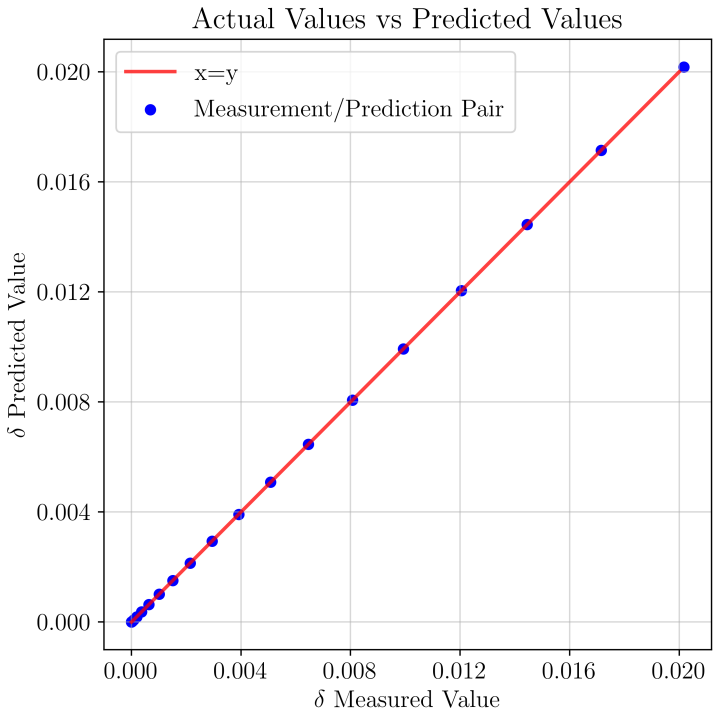


Figure 1.7: Cantilever beam example ML predictions vs actual values

With this simple example, it can be shown that ML models can be used to predict physical phenomena. Instead of needing an equation to produce the target variable to test, the target variable can be measured and fed into the machine learning model as visualized in Fig.1.8.

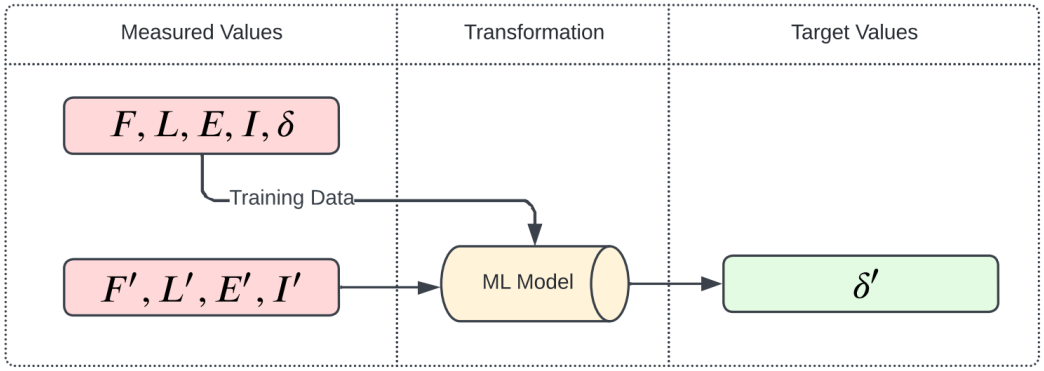


Figure 1.8: ML model training only using measurement data

When all the values are measured and the underlying relationships cannot be as closely tested against a known true value, many more questions arise. How do different ML models perform with the data? How can the models be evaluated mathematically? How many measurements would be needed to produce the most accurate ML model? If the measurements have noise (measurement error), how well will the models perform? Is there a particular input variable that strongly predicts the outcome?

CHAPTER 2: Literature Review

To get an idea of how researchers are approaching problems of similar scope, a literature review was performed to get an idea of what methods would be most useful in the hollow sphere case. Machine learning based prediction is a key focus of today's research of deformation and failure of materials. Research was found which was published within the last year related to machine learning and materials science. The papers reviewed all had similar focuses on ML models in material science but differing focuses in their hypothesis' and conclusions.

2.1 Datasets

The datasets used focused on measurements taken from experiments like different concrete mixtures using different ratios of eco-friendly aggregate [5] to varied loading on fatigue strength of steels [6]. Some of the papers did not mention dataset size but of the ones that did, sample sizes ranged from 125 instances of data to 437 instances. The majority of data used to train the ML models were measurement data but also Finite Element Analysis (FEA) simulation data was used in one of the experiments. Of these instances of data, there were only one target variable for each of the studies and independent predictor variables ranging from five to 25. Some data used raw measurement data like distance or mixture percentage but others used data like location parameter and size of deformity in additively manufactured parts which were measured using x-ray technology.

2.2 ML Models

There were a wide range of ML models used with some performing better than others. Ordinary Least Squares (OLS) linear regression was used with the highest

accuracy on the study of steel fatigue strengths [6]. Support Vector Regression (SVR) was the most accurate on a study of additively manufactured material fatigue life [7]. The eco-friendly concrete mixture study found that Multivariate Polynomial Regression was the most accurate in predicting fatigue limits [5]. A two-step loading fatigue life experiment found that Gaussian Processes Regression (GPR) was the most accurate in predicting the remaining serviceable lifetime of different material types [8]. Other model types used in the literature were Artificial Neural Networks (ANN), K Nearest Neighbors (KNN), and many more that did not perform as well as those already mentioned.

2.3 Error Types and Ranges

Many different error types were used across the literature but two were common amongst almost all of the papers. Coefficient of Determination (R^2) and Root Mean Square Error (RMSE) were the most commonly used error types as a base with more error types depending on the study. Values to be accepted as the most accurate in the literature varied from a R^2 of 0.81 to 0.99 meaning that accuracy metrics depend greatly on what the experiment is trying to achieve.

2.4 Additional Inquiries

After these general steps were completed, the literature changed direction into different focuses. Some of the papers focused on which ML model performed the best on noiseless data while others tested models with and without noisy data. The study on noisy data also tested measured data versus a dataset mixed with measurement data and simulation data. Sensitivity analysis was also conducted in one of the studies where the resulting predictions where two of the five predictive measurements accounted for a combined 80% of change in fatigue strength.

2.5 Conclusion

It is common for researchers to attempt to predict outcomes using ML models in materials science. The datasets used were mostly measured data which small or medium sized numbers of total variables. Usually, a few different model types are chosen and optimized for the data then compared against each other. Coefficient of Determination and RMSE are used the most often with other error types added in depending on what the researcher found relevant to measure the accuracy of each model. A few hundred instances of data which can make a surprisingly accurate model with R^2 values of 0.8 to 0.99. The data comes into question sometimes in the form of what predictive data should be used, can simulated data be used to accurately predict measurement data or a mixture of the two, how sensitive a model is to each predictor variable, and how robust a model is to noise.

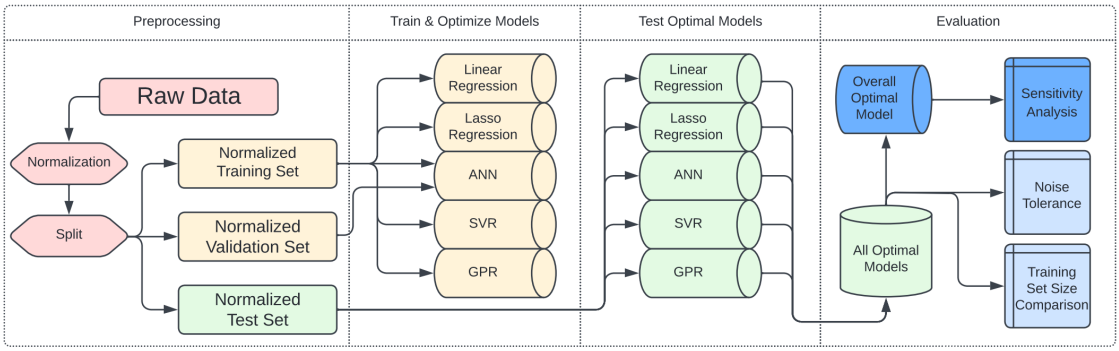


Figure 3.1: Experimental method flow chart

Using current literature on the topic of ML and materials science, an experimental method was constructed to find an optimal ML model for the hollow sphere expansion problem. The experimental method can be visualized in 3.1 starting with preprocessing.

3.1 Preprocessing

3.1.1 Software

In general, ML models are built to be used with computational software. Theoretically, the algorithms could be completed by hand but that isn't feasible when computers may be used. Table 3.1 shows a list of all software libraries used in building and optimizing the ML models in this thesis.

Table 3.1: Software Used

Software Name	Purpose
Numpy	Allows the use of array calculation in python
pandas [9]	DataFrame functionality
matplotlib	Math plotting
seaborn [10]	DataFrame plotting functionality
sklearn [11]	Machine learning functionality for all used models

Python was used exclusively for the exploratory data analysis, preproccessing, model generation, evaluation, and visualization of the data. Google Colab [12] has Python functionality built in with many libraries that are as easily accessible as including them with one line of code. All the libraries used were included in Google Colab. Pandas [9] is a library used in data analysis that gives the user access to DataFrames. A DataFrame is a type of labeled array where data can be stored under labeled columns. Pandas allows for many difficult functions to be applied to DataFrames automatically like histograms which visualize the data distribution with one command. Seaborn [10] was used to generate the heatmap of the total dataframe in the exploratory data analysis section. The library sklearn [11] was first used to split the training, validation, and testing data. Each machine learning scheme was also available using sklearn along with many statistical functions needed for error calculation.

Since ML is a data-focused approach to problem solving, data management is important. Preprocessing of data refers to any formatting, normalization, or randomization of data before it is used to train ML models.

Table 3.2: Data minimums and maximums

Parameter	Minimum Value	Maximum Value
Poisson's Ratio	0	0.49
Young's Modulus	7.00e10	3.99e11
Thermal Expansion Coefficient	2.00e-6	3.00e-5
Inner Radius	1.02e-2	1.19
Outer Radius	1.26e-1	2.37
Inner Temperature	2.50e1	1.99e2
Outer Temperature	3.86e1	3.98e2
Change in thickness	2.96e-5	9.79e-3

	PoissonsRatio	E	ThermalExpansion	InnerRadius	OuterRadius	InnerTemp	OuterTemp	ThicknessChange
0	0.05	274963176062	0.000019	0.349704	1.161172	60.850524	121.833290	0.001732
1	0.24	157115249336	0.000016	0.624374	0.883991	128.775652	263.986620	0.000859
2	0.07	201347914159	0.000018	0.503166	0.789903	112.375774	156.150195	0.000707
3	0.22	76687027854	0.000007	1.106494	1.271535	93.803190	287.237049	0.000221
4	0.20	345899561776	0.000029	0.440460	1.151209	163.173818	184.293644	0.003680
...	...	...	...	...	...	...	...	...
9995	0.24	88133103773	0.000021	0.211720	0.318512	26.975601	116.153576	0.000183
9996	0.13	159276320771	0.000005	1.181847	1.890586	193.093802	380.461531	0.001167
9997	0.32	230207049267	0.000009	0.958243	2.149861	172.554720	321.157120	0.002944
9998	0.10	87477571953	0.000028	1.044543	1.599917	166.190199	310.416325	0.004018
9999	0.07	104677609333	0.000027	1.025651	1.591533	62.517424	142.295664	0.001768

10000 rows × 8 columns

Figure 3.2: Raw data shown in a pandas Dataframe

The raw data was given in an excel file and can be seen in Fig.3.2 as a pandas DataFrame after it was imported using python. There are a total of 10,000 instances of data. An instance of data refers to one instance of the problem where all the variables have been measured which is listed as one full row of data in the DataFrame.

Before training any ML model or altering the raw data, it is important to first understand the data in general. The minimum and maximum values may be viewed in table 3.2. From the table, the value ranges are obviously very different with the Young's Modulus being on the order of 10^{10} while Change in Thickness is on the order of 10^{-4}. Since the ranges are so large, normalization will be important.

When exploring the data, the distribution of the data is also important for understanding how the ML models will behave. Figure 3.3 shows histograms of each variable including the target variable, ThicknessChange. A histogram looks at each variable's values and sorts them into bins to show how many instances of specific values there are. In this case, 100 bins were used meaning the total range of values from maximum to minimum for each variable was split into 100 bins and any value falling within a bin range would be added to the total. From the figure, there are uniform distributions for all of the variables except for OuterRadius, OuterTemp, and the target variable ThicknessChange. A uniform distribution means the probability of having any value in the range is the same for each value. The OuterRadius and OuterTemp seem to be closer to a Gaussian distribution which is more common in natural phenomenon. Finally, the target variable ThicknessChange, shows many more measurements on the lower end of the spectrum. This type of distribution signals that it may be difficult for the machine learning models to correctly predict higher dependent variable values due to a lack of training data in that range.

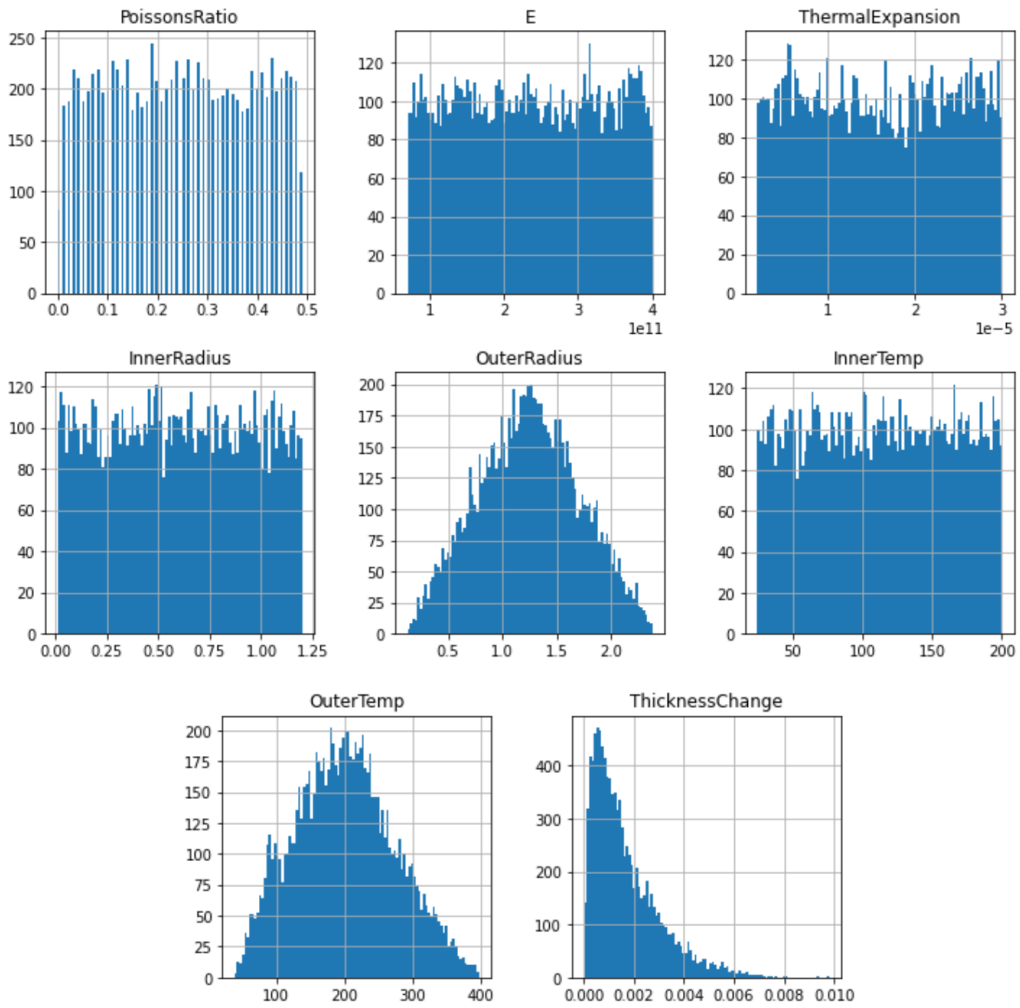


Figure 3.3: Variable distributions

A correlation between two variables is a calculation of how similar the values are to each other from instance to instance. If two variables are similar in each instance, the correlation will be high. If two variables are exactly the same in each instance, the correlation will be 1. Correlation of parameters are important in exploratory data analysis because some of the variables may be highly correlated leading to the possibility of omitting the need of predictive variables. Pandas uses the Pearson correlation calculation as a default whose equation is found in Eqn.3.1.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}} \tag{3.1}$$

Figure 3.4 shows a correlation heatmap where light blue denotes a high correlation between variables and dark blue for negative correlation. The highest correlations are between InnerRadius and OuterRadius, and InnerTemp and OuterTemp. The ThicknessChange is the most correlated with ThermalExpansion, InnerTemp, and OuterRadius.

Although there are some higher correlations, none of the variables have higher than a correlation of 0.7 meaning all the predictors should be kept to predict the target variable until otherwise indicated in the machine learning models.

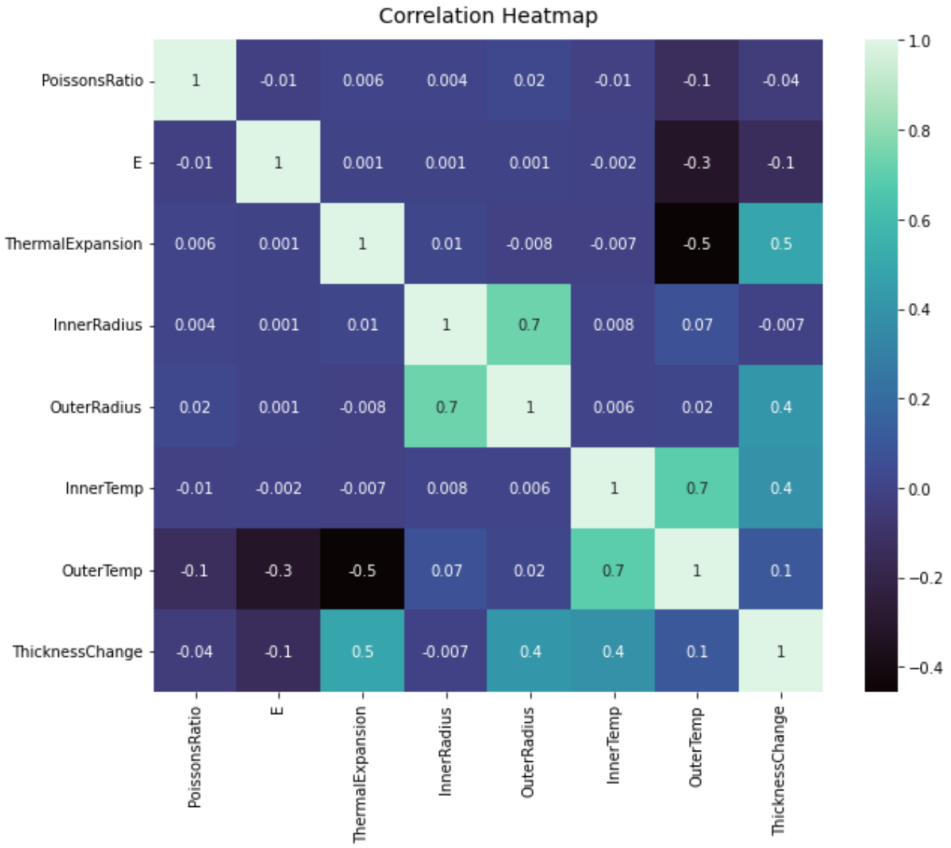


Figure 3.4: Correlation Heatmap

After exploring the data, it is ready to be preprocessed. The first step in preprocessing is normalization. Normalization refers to transforming the data to make it more digestible by the ML models. There are many different ways to normalize data but the most convenient for this application is the Min-Max normalization. Min-

Max normalization [13] takes a variable (one column of data) and scales it down between two values as calculated in 3.2. In the equation, A is the column of values to be normalized with C and D as the lower and upper bounds of the normalized data respectively. The values after normalization are shown in Fig.3.5. Importantly, the values of the target variable are not normalized since our goal is to not have to postprocess the target variable.

$$\hat{A}_i = \frac{A_i - min(A)}{max(A) - min(A)} * (D - C) + C \tag{3.2}$$

	PoissonsRatio	E	ThermalExpansion	InnerRadius	OuterRadius	InnerTemp	OuterTemp	ThicknessChange
0	0.102041	0.621084	0.613571	0.285377	0.459455	0.204800	0.231461	0.001732
1	0.489796	0.263931	0.486786	0.516268	0.336389	0.593030	0.626820	0.000859
2	0.142857	0.397984	0.554286	0.414379	0.294615	0.499295	0.326903	0.000707
3	0.448980	0.020184	0.168571	0.921546	0.508455	0.393143	0.691484	0.000221
4	0.408163	0.836065	0.958214	0.361667	0.455032	0.789634	0.405176	0.003680
...	...	...	...	...	...	...	...	...
9995	0.489796	0.054873	0.661786	0.169385	0.085323	0.011187	0.215664	0.000183
9996	0.265306	0.270481	0.115357	0.984888	0.783308	0.960643	0.950762	0.001167
9997	0.653061	0.485445	0.240357	0.796923	0.898423	0.843251	0.785823	0.002944
9998	0.204082	0.052886	0.929286	0.869469	0.654254	0.806874	0.755951	0.004018
9999	0.142857	0.105013	0.909643	0.853587	0.650531	0.214328	0.288371	0.001768

Figure 3.5: Normalized data with the target variable unchanged

Following normalization, the dataset is split into three subsets: training, validation, and testing sets. To accomplish this, sklearn has a function that splits input and target variable datasets into two subsets randomly which alleviates the need for instance randomization manually. First the dataset was split into two subsets: `test` and `train_val`. This was done because some of the ML model functions have internal validation splitting which makes pre-splitting unnecessary. After the first split, another split was applied to the `train_val` set to split into the training and validation sets. At the end of preprocessing, 80% of the data was used as a training set, 10% as a validation set, and 10% as a test set.

The training subset is used to train the model. A training subset is usually a larger portion of the total dataset so that maximum information may be gained from the dataset.

The testing subset is used specifically for error measurement after a model has been trained. Having a set of information prepared for testing means that data that should be correctly predicted by a well trained model. The error between a known target value and predicted target value using the test set is an important metric in evaluating the accuracy of a model in general.

The validation subset is used to test models for overfitting. Overfitting of a model is when the model is trained too much on a dataset making it less accurate on a dataset of values the model was not trained on. The validation subset is used to have a disconnected subset of valid data to check prediction accuracy during the training or optimization of a model.

3.2 Machine Learning Models

3.2.1 Linear Regression

Linear regression is one of the oldest used regression schemes and in some respects is the most simple which is why it was chosen as the first regression algorithm. The concept applies a linear combination of each input parameter multiplied by a coefficient as seen in Eqn.3.3. The coefficients can be found using a few different methods and there are schemes to reduce some coefficients to low values or zero depending on the minimization of the error.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...\beta_n x_n \tag{3.3}$$

3.2.1.1 Ordinary Least Squares (OLS)

Ordinary least squares, or OLS, is an optimization method that minimizes the OLS error when finding the Linear Regression coefficients. In a generalized multiple

input linear regression as seen in Eqn.3.4, the target variables (Y), coefficients (β), predictor variables (X) are all in matrix form. An error matrix (ϵ) is added to the calculation so that it may be minimized.

$$\boldsymbol{Y} = \boldsymbol{\beta X} + \boldsymbol{\epsilon} \tag{3.4}$$

To find the optimal values for the linear coefficient matrix, a minimization is performed on the square of the errors as seen in Eqn.3.2.1.1.

$$\boldsymbol{\epsilon}^2 = (\boldsymbol{Y} - \boldsymbol{\beta X})^2 \tag{3.5}$$

$$\boldsymbol{S}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{j=1}^{p} X_{ij}\beta_j \right|^2 = (\boldsymbol{Y} - \boldsymbol{\beta X})^2$$

3.2.1.2 Lasso

Least absolute shrinkage and selection operator, or Lasso, linear regression is a method that seeks to minimize the absolute sum of the coefficients. It accomplishes this by adding a new penalty term into the minimization target as seen in 3.6. The tuning parameter (λ) controls the strength of the penalty with a zero being the same as the OLS regression.

$$\boldsymbol{S}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left| y_i - \sum_{j=1}^{p} X_{ij}\beta_j \right|^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.6}$$

The major strength of the Lasso method is that it will remove predictive variables from the calculation by reducing the associated coefficient to zero if the variable does not contribute to the prediction. Looking at the coefficients after the lasso method can show if any of the variables are less important for prediction.

3.2.2 K Nearest Neighbors

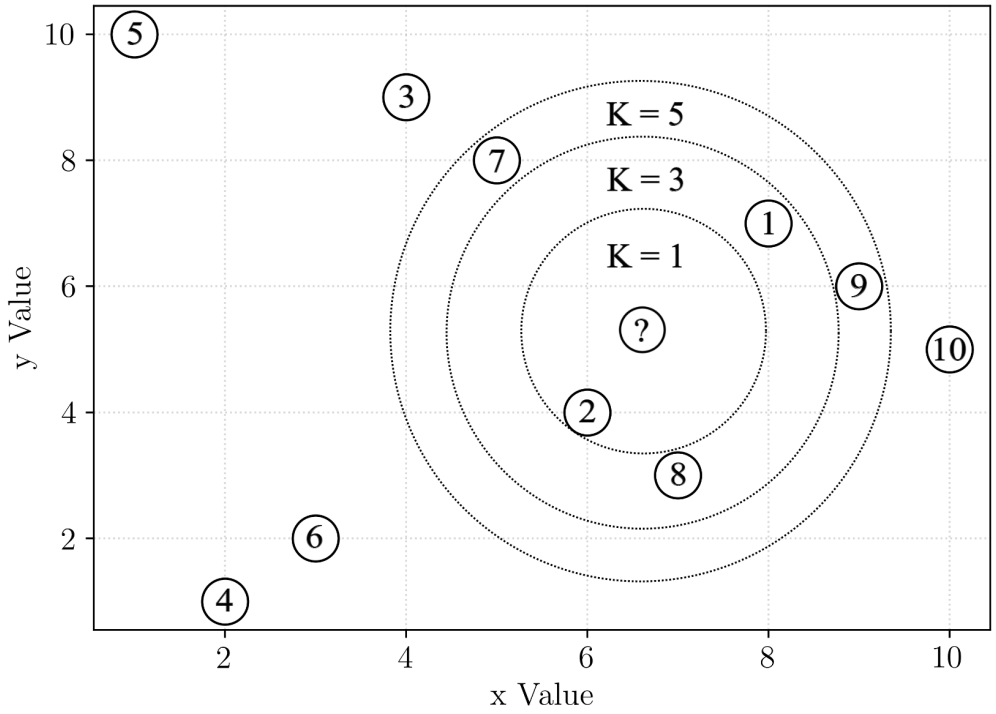


Figure 3.6: KNN visualization

Table 3.3: KNN visualization data

x	1	2	3	4	5	6	7	8	9	10	6.5
y	10	1	2	9	8	4	3	7	6	5	5.5
z	5	4	6	3	7	2	8	1	9	10	?

K Nearest neighbors (KNN) is a machine learning scheme that is simple in theory and can most easily be described in an example. Figure 3.6 shows an example figure of a two input, one target variable model. The x and y values of each instance are visualized by the location of a circle at their combined coordinates and the z value is inside each associated coordinate circle. To predict the value of the unknown z value, the nearest neighbors to the circle can be consulted.

If only one neighbor is consulted, the predicted value takes on the same value as the neighbor since there are no other points to average with.

$$? = 2 \tag{3.7}$$

If the three closest neighbors are consulted, the predicted value takes the form of the average of the three closest points.

$$? = \frac{2 + 8 + 1}{3} \tag{3.8}$$

When $K = 5$, five neighbors are consulted and the predicted value is the average of those values.

$$? = \frac{2 + 8 + 1 + 7 + 9}{5} \tag{3.9}$$

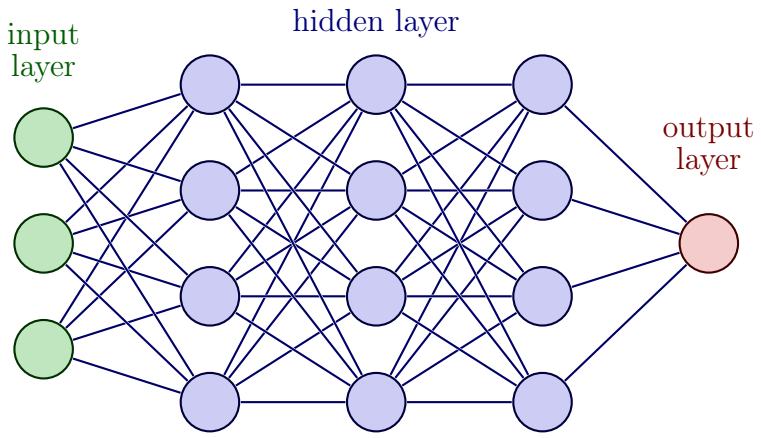
There are two main ways to optimize the KNN model which are distance measurements and the K-value. A distance measurement refers to the distance of training data to the testing prediction variables. As in the example, the Euclidean distance is calculated using Eqn. 3.10. Other types of distance can be used depending on the data type.

$$\sqrt{(x_1 - \hat{x}_1)^2 + (x_2 - \hat{x}_2)^2} \tag{3.10}$$

The K-value refers to the number of neighbors the algorithm will average. A K-value of one means that the one closest distance value will be made to be the prediction. For a K-value of two, the two closest distance data points are averaged and that average is used as the prediction. Each possible K-value can be tested up to the total value of instances of data in the dataset.

3.2.3 Artificial Neural Network

Figure 3.7: ANN Architecture



An artificial neural network or ANN can be visualized in Fig.3.7 and is built using a few fundamental parts. Each column of the ANN represents a "layer". A layer is a set of nodes (the colored circles) that are not interconnected but will be connected to the previous or next layer. In the figure, the left most layer is the input layer. The input layer takes altered or unaltered predictor data. The second layer represents a "hidden" layer. Each line denotes some calculation connecting nodes meaning in the figure's case that the layers are "fully connected" or each node in a layer is connected to each node of the previous layer. Eventually, after calculations are completed, the final layer calculates the predicted value at the output layer.

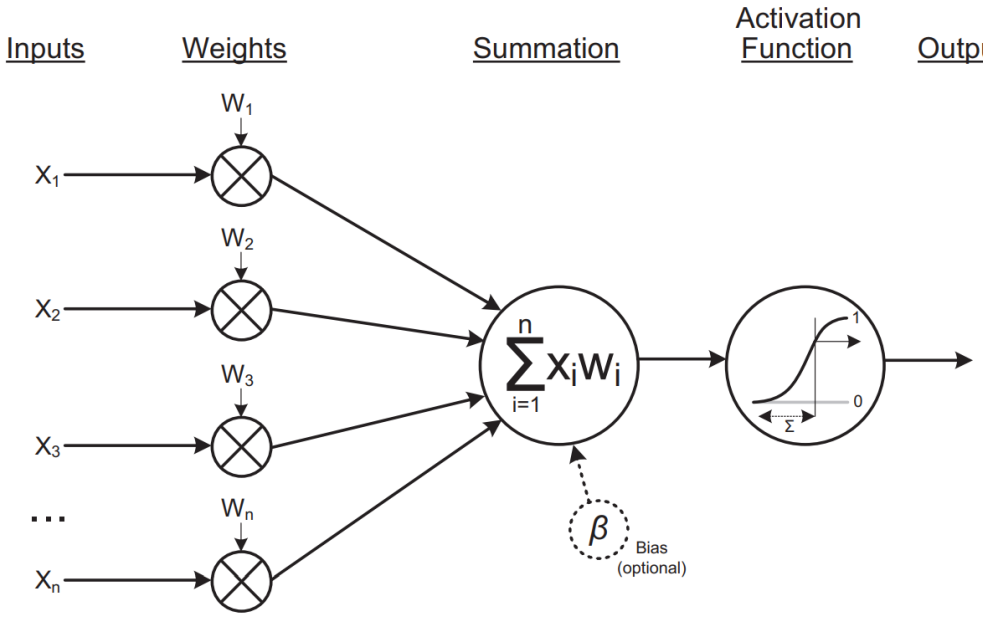


Figure 3.8: Node calculation [1]

Each node, or neuron, has an associated calculation as seen in Fig.3.8. The node takes in values from previous layer nodes, multiplies them by weights (coefficients), adds them together, then transforms them depending on an activation function. The activation function is a function that is applied after the summation and multiplication of each node. An activation function allows the ANN to be non-linear since each layer would essentially be a summation of linear regressions.

After the architecture has been decided and constructed, there needs to be a method of updating the weights associated with each node so that the model can be nudged into more accurate predictions. The method of going back to update weights is called back propagation which uses something called an optimizer.

An optimizer goes into effect after some data has been sent through the model and a few predictions have been calculated. These predictions are compared to the training data and errors are calculated and used to go back and update weights in the model.

Clearly there are many different options for optimizing the ANN model which

can lead to some unwanted outcomes. One of the unwanted outcomes that can be common for ANN models is overfitting. Overfitting is when a model is trained too much on it's training data. A model should be able to generalize, meaning it should accurately predict on unseen data but an overfitted model will be most accurate on data that it had been trained on. A simple way to measure overfitting is to provide a validation dataset. The validation dataset is different data than the training dataset so it can be used during the training process to test against. When the accuracy of the prediction on the validation dataset begins to decrease, the model becomes overtrained and should be stopped.

3.2.4 Support Vector Regression

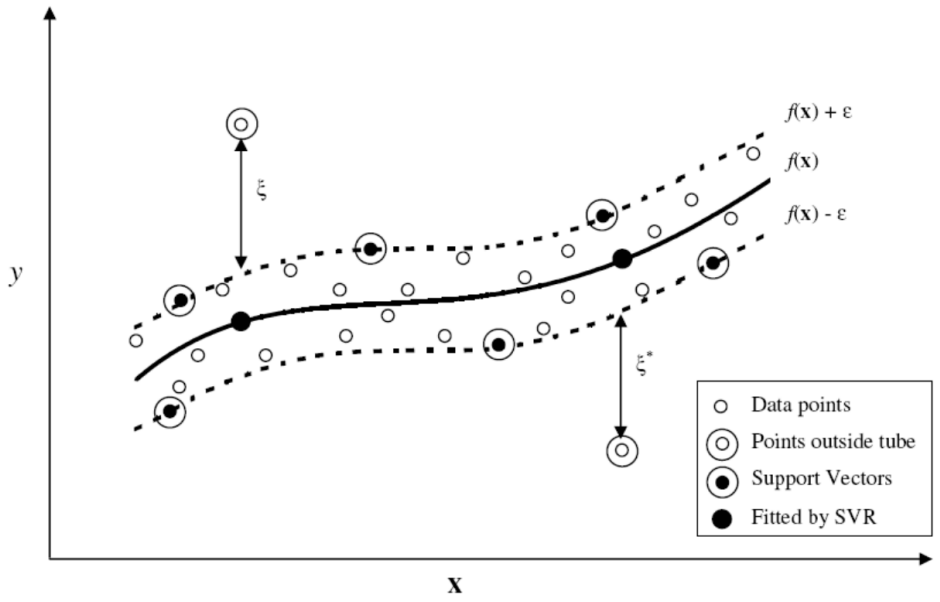


Figure 3.9: SVR visualization [2]

Support Vector Regression (SVR) is a ML model that builds a "tube" around training data in order to predict test data. Figure 3.9 shows a visualization of a one dimensional case along with some of the pertinent hyperparameters. SVR aims to optimize the function shown in Eqn.3.11 [14].

$$y = f(x) =< w, x > +b = \sum_{j=1}^{M} w_j x_j + b \tag{3.11}$$

SVR uses hyperparameters like ϵ which is half the width of the tube which the support vectors will construct and ξ which are slack parameters that can guard against outliers in the training data. Using these hyperparameters, an optimization process is performed on Eqn.3.12.

$$min||w||^2 + C\sum_{i=1}^{N}(\xi_n + \xi_n^*) \quad ||w||^2 = \langle w, w \rangle \tag{3.12}$$

$$s.t. \begin{cases} y_i - w^T\varphi(x) \leq \epsilon + \xi_i \\ -y_i - w^T\varphi(x) \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \leq 0 \\ i = 1, 2, ...N \end{cases}$$

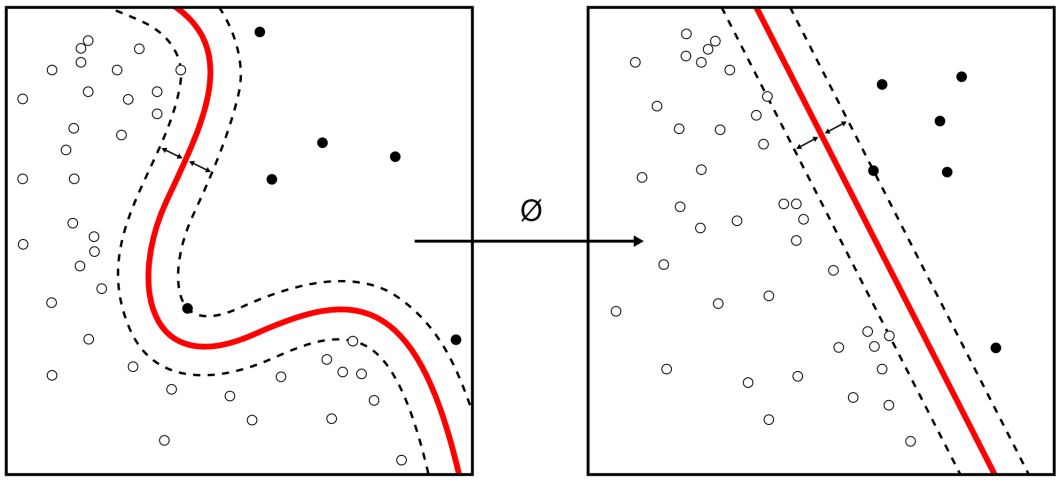


Figure 3.10: Kernel visualization [3]

The kernel of SVR is a function that maps the original space of the dataset into a new coordinate system that may be more easily fitted to a line or more simplified

boundary. This is a common practice in ML since data that can be more easily divided can have higher accuracy than working with the original data distribution.

3.2.5 Gaussian Processes Regression

A Gaussian Process is an expansion of the idea of a random variable with a probability that is normally distributed, or Gaussian distributed. A normal distribution is defined by it's Probability Density Function (PDF) in Eqn.3.13 which defines the probability of the random variable being a value given some mean (μ) and standard deviation (σ).

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)\right) \tag{3.13}$$

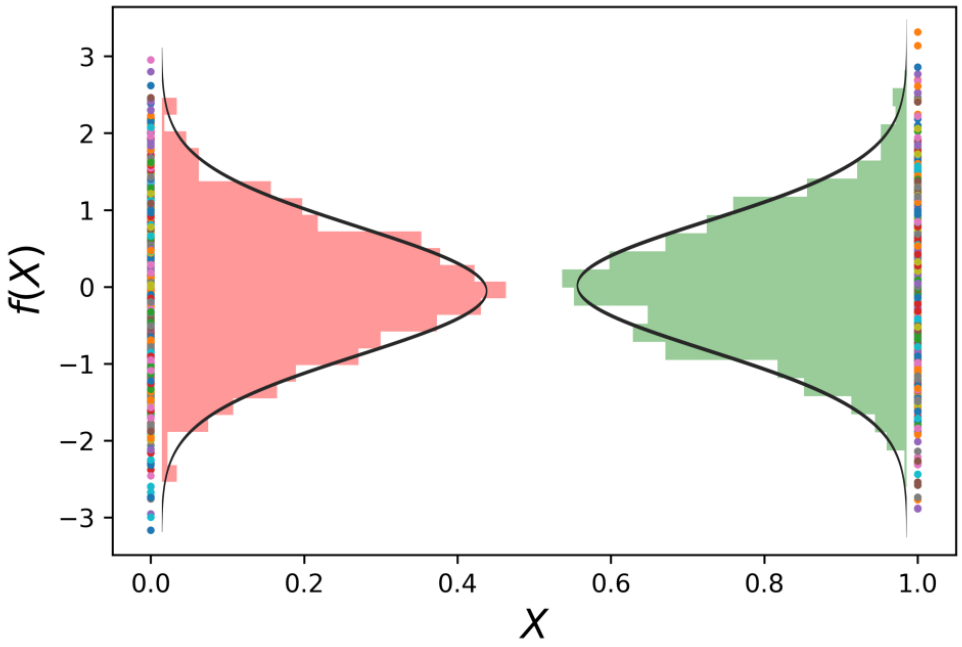
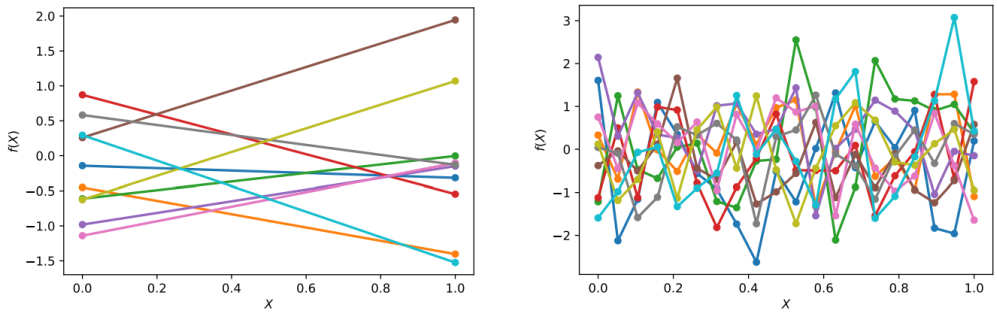


Figure 3.11: Two random variables sampled 1,000 times from the Gaussian distribution [4]

Figure 3.11 shows two independent normally distributed random variables with 1,000 samples from each with one random variable at $X = 0$ and one random variable at $X = 1$.



(a) Two Gaussian vectors **(b)** Twenty Gaussian vectors

Figure 3.12: Uncorrelated Gaussian distributed random variables connected [4]

Figure 3.12 shows how connecting these lines gives unsmooth connections because the random variables are independent of each other. The way to fix this is to define the random variable's correlation with each other.

To correlate the random variables together, kernel equations are used. In the most common case, the RBF kernel function (Eqn.3.14) is used to describe the correlation between all the random variables with each other. After the correlation matrix has been built, Fig.3.13 shows how the random variables are smoothed.

$$k(x_i, x_j) = cov(x_i, x_j) = exp\left(-\frac{(x_i - x_j)^2}{2}\right) \tag{3.14}$$

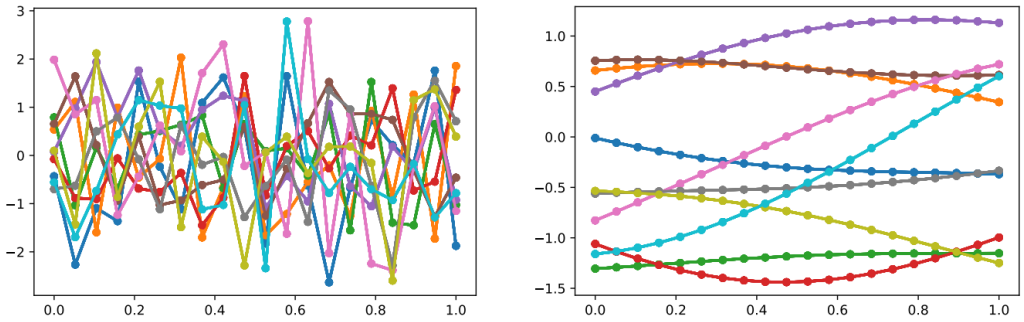


Figure 3.13: Correlated Gaussian distributed random variables [4]

To bring all this together, the training data is used as a known condition in the GPR model (Eqn.3.15) then the prediction data (Eqn.3.16) are also added to the

correlation matrix. The combined matrix can be seen in Eqn.3.17 where f_* denotes the predicted data [4].

$$\boldsymbol{K} = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{bmatrix} \tag{3.15}$$

$$\boldsymbol{K}^* = \begin{bmatrix} k(x_*, x_1) & k(x_*, x_2) & \dots & k(x_*, x_n) \end{bmatrix} \quad \boldsymbol{K}^{**} = k(x_*, x_*) \tag{3.16}$$

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f_*} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}^{*T} \\ \boldsymbol{K}^* & \boldsymbol{K}^{**} \end{bmatrix}\right) \tag{3.17}$$

3.3 Evaluation

To evaluate the models, error measurements will be used during the training and optimization phases to gauge which models will be the most advantageous to continue further with. The two most common error measurements used in literature were the Coefficient of Determination and the RMSE which are included in the scikit metrics library and shown in Eqns.3.19 and 3.18 [15].

$$R^2 = 1 - \frac{\sum (y_i - f_i)^2}{\sum (y_i - \bar{y})^2} \tag{3.18}$$

$$RMSE = \sqrt{\sum (y_i - f_i)^2} \tag{3.19}$$

The Coefficient of Determination is an error type which takes values from negative infinity to positive one. Any negative number is associated with no correlation at all between the predicted and true training or testing values. The closer the Coefficient of Determination approaches one, the more closely the model's predictions match

the correct target values meaning a R^2 of one would be a perfect match between the prediction and true values. The RMSE error type is a calculation that gives the summed squared errors across all tested instances. Since any error of this type is unwanted, the goal is to find a model with the lowest RMSE value.

After the selected models have been constructed, trained, and tested using the initial error metrics, further testing is needed to check for robustness to other common issues for ML models. One of the more important problems with ML models is the amount of data needed to train an accurate model. To test for this, the optimal model for each model type is to be trained using varying amounts of training data to see how the accuracy of the models are effected by training set sizes. Another common problem for ML models are how they perform when noise is introduced. Noise in the dataset refers to a value that may be measured incorrectly which are usually due to tolerance issues in the measurement device. To make the noisy data more realistic, each measurement can be viewed independently for how accurate those measurements would most likely be if they were made in the real world. When referring to the predictor variables, measurements like distance and temperature are fairly accurate with current measurement methods while Young's Modulus, Poisson's Ratio, and coefficient of Expansion would be more difficult to measure in each sphere. For this reason, only the measurements with a high likelihood of error will have noise added.

To simulate noise in the given FEA data which is noiseless, normally distributed noise can be added to the synthetic data to produce noisy data. The normal, or Gaussian distribution was covered earlier in Eqn.3.13. From the PDF, one can surmise that the distribution of the noise depends on the mean and standard deviation given in the PDF. The mean of the noise is set to zero so that the noise can either add or subtract from the data randomly. Finally the standard deviation may be varied to generate random variable variables useful test with. In the case of this thesis, percentages of noise would like to be compared. For example, if a noise level of 1%

was needed, what standard deviation value should the normally distributed noise be set to? To solve this, the expected value of the random variable can be used but the expectation of the normal distribution would be the mean or zero in this case. To combat this, the expected value of a normal distribution of only positive values was used called the half-normal distribution. The expected value of a half normal distribution is calculated by Eqn.3.20 which can be rearranged to solve for standard deviation as seen in Eqn.3.21.

$$E(X) = \sigma\sqrt{\frac{2}{\pi}} \tag{3.20}$$

$$\sigma = E(X)/\sqrt{\frac{2}{\pi}} \tag{3.21}$$

The expected value for the noise should be a percentage of the normalized data. Since the normalized data ranges from zero to one, a percentage of one makes the most sense meaning that 1% of 1 would be 0.01. For any percentage to be tested, it can be input into the equation to solve for an appropriate standard deviation to construct the noise.

Finally, a sensitivity analysis on the best model would be helpful in determining how important each predictor variable is to the output. Sensitivity analysis refers to how sensitive an output is to changes of an individual input variable or groups of input variables. To accomplish this analysis, a library called SALib [16], [17] was used. SALib creates a uniformly distributed input array of specified range which is then fed into the model in question instead of testing data. The model predictions are then taken and the variances are calculated following Eqn.3.22.

$$V_i = Var_{X_i}(E_{X\ _i}(Y|X_i)) \qquad S_i = \frac{V_i}{Var(Y)} \tag{3.22}$$

CHAPTER 4: Results

Each model was imported using the scikit library and optimized. The most accurate model of each type was then tested using the test set and graphed below.

4.1 Linear

4.1.1 OLS Model

The OLS model was created using the scaled training set and the results may be viewed in Fig.4.1. The R^2 of the model was 0.91 when fitted to the training set. When testing against the test set, the R^2 was 0.762 and the RMSE was 0.000628.

OLS Regression Results

Dep. Variable:	ThicknessChange	R-squared (uncentered):	0.910
Model:	OLS	Adj. R-squared (uncentered):	0.910
Method:	Least Squares	F-statistic:	1.153e+04
Date:	Wed, 20 Jul 2022	Prob (F-statistic):	0.00
Time:	16:20:37	Log-Likelihood:	47387.
No. Observations:	8000	AIC:	-9.476e+04
Df Residuals:	7993	BIC:	-9.471e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	-0.0006	2.35e-05	-26.164	0.000	-0.001	-0.001
x2	-0.0011	2.29e-05	-49.465	0.000	-0.001	-0.001
x3	0.0018	2.41e-05	75.442	0.000	0.002	0.002
x4	-0.0031	3.7e-05	-82.844	0.000	-0.003	-0.003
x5	0.0053	4.86e-05	109.237	0.000	0.005	0.005
x6	0.0020	3.93e-05	51.513	0.000	0.002	0.002
x7	-0.0009	4.88e-05	-19.306	0.000	-0.001	-0.001

Omnibus:	1472.032	Durbin-Watson:	1.984
Prob(Omnibus):	0.000	Jarque-Bera (JB):	4367.791
Skew:	0.960	Prob(JB):	0.00
Kurtosis:	6.069	Cond. No.	11.6

Figure 4.1: OLS Results from training on training set

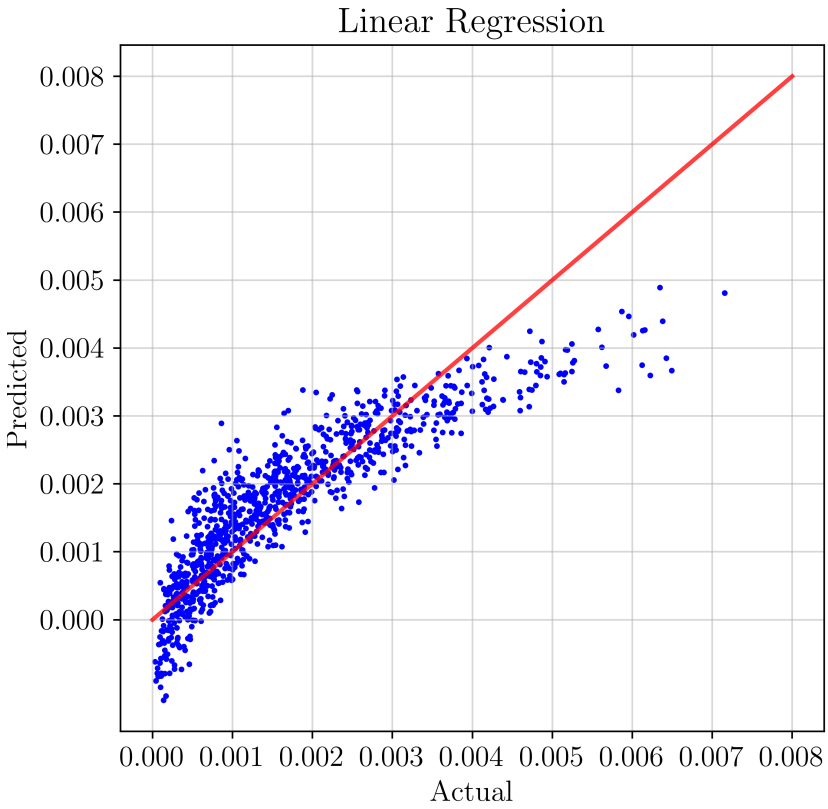


Figure 4.2: OLS predicted vs actual values

4.1.2 Lasso

When optimizing the Lasso regression, the only parameter to vary was the tuning parameter (λ). The optimal tuning parameter was found by sweeping through many different values ultimately resulting in a value of 0.000002. Figure 4.3 shows how the test set targets compared to the predictions of the Lasso model. The final Lasso model had a R^2 of 0.827 and RMSE of 0.000535 on the test set.

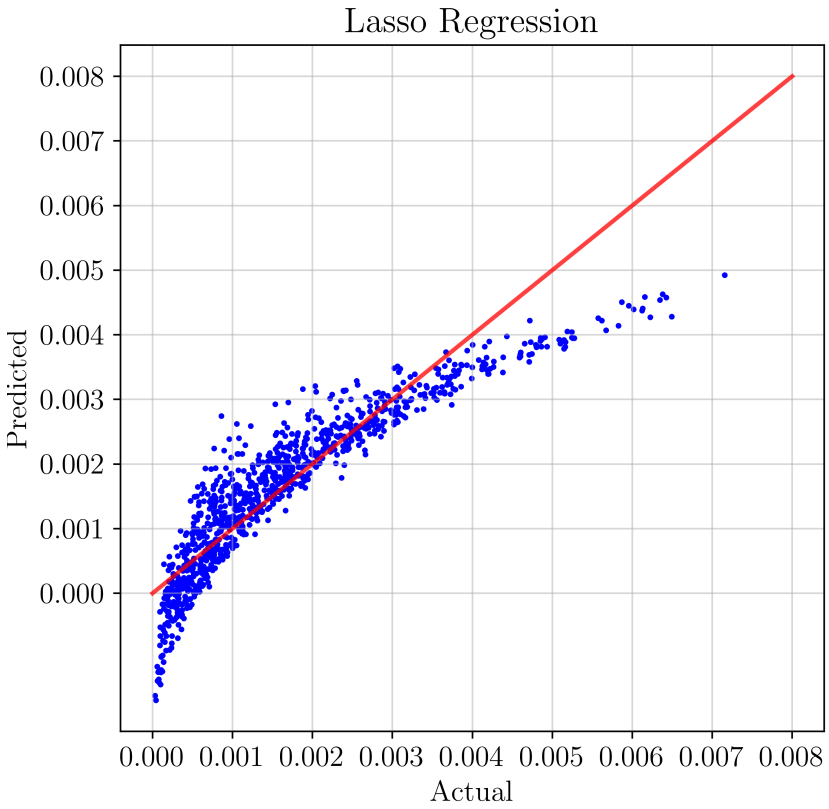


Figure 4.3: Lasso predicted vs actual values

4.2 KNN Model

For the KNN model, a loop was used to first compare the R^2 and RMSE values for all values of K so that the optimal value of K could be chosen for the most accurate model. Figure 4.4 and 4.5 shows the R^2 and RMSE values for each K value respectively when tested on the test set. The R^2 is highest and RMSE is lowest for K values from five to ten so those values were tested individually.

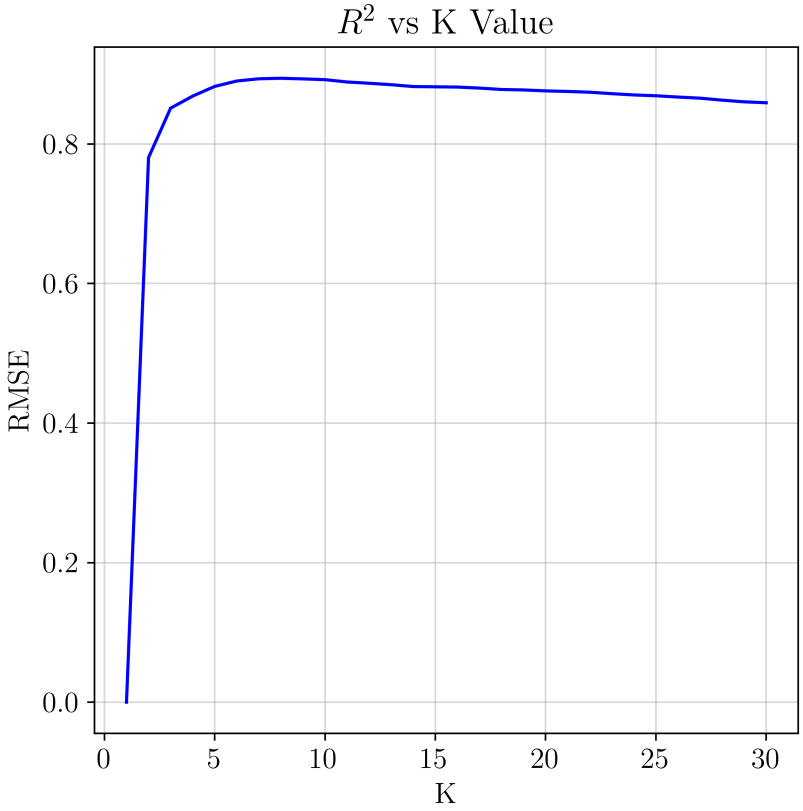


Figure 4.4: R^2 values for sweep of K values

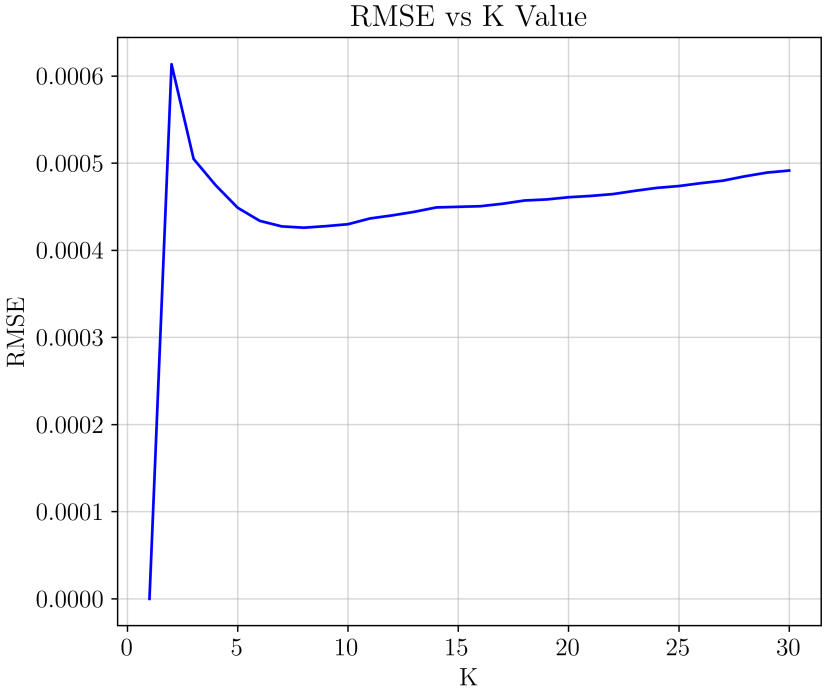


Figure 4.5: RMSE values for sweep of K values

Each K value between five and ten were individually checked for accuracy to find the optimal K value. The K value of 6 gave the highest R^2 value of 0.889 and RMSE of 0.000429. Figure 4.6 shows the graphs of predicted values vs actual values. This graph shows an improvement from the linear and lasso results by predicting all positive values and having less of a curve in the data. The higher the predicted value, the worse a prediction was due to having less training data in the higher range of target variables.

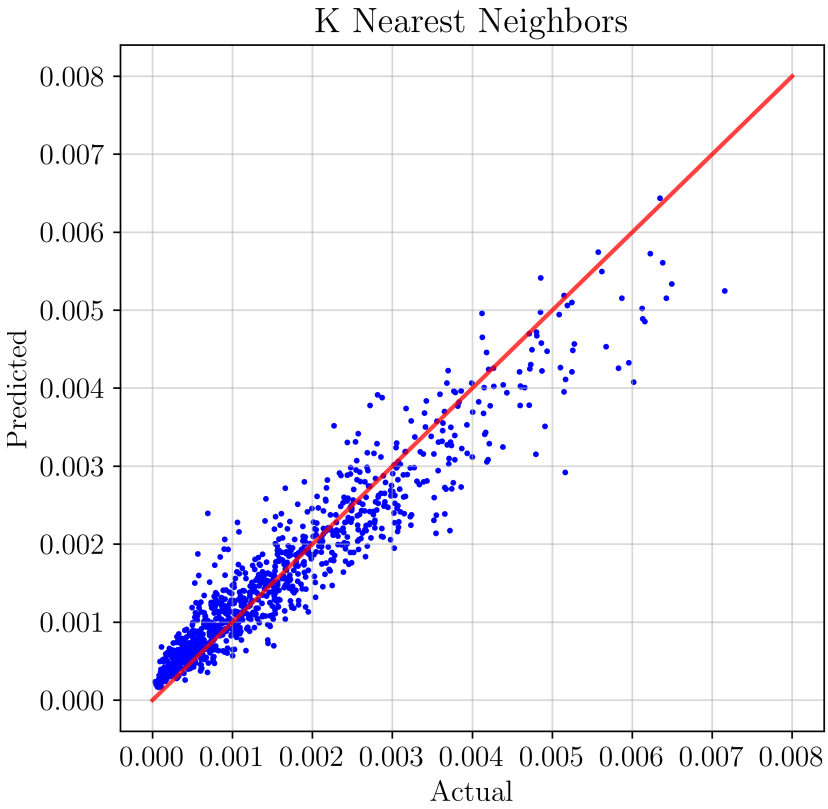


Figure 4.6: KNN predicted vs actual values

4.3 Artificial Neural Network

For the ANN, different architecture types were tested using the grid search method. First activation type and solver were fixed while different numbers of nodes and layers were tested. Four to five layers of 100 to 1,000 nodes each would produce models that gave a R^2 of around 0.94 consistently. Since accuracy was fairly high, different activation types and solvers were tested but relu and adam produced the most accurate models. Finally, more architectures were tested with higher node counts and different layers but the most accurate model had four fully connected layers of 3,000 nodes which produced an R^2 of 0.972 and a RMSE of 0.00022 on test data.

The ANN took the longest to train by far with the final model taking around forty

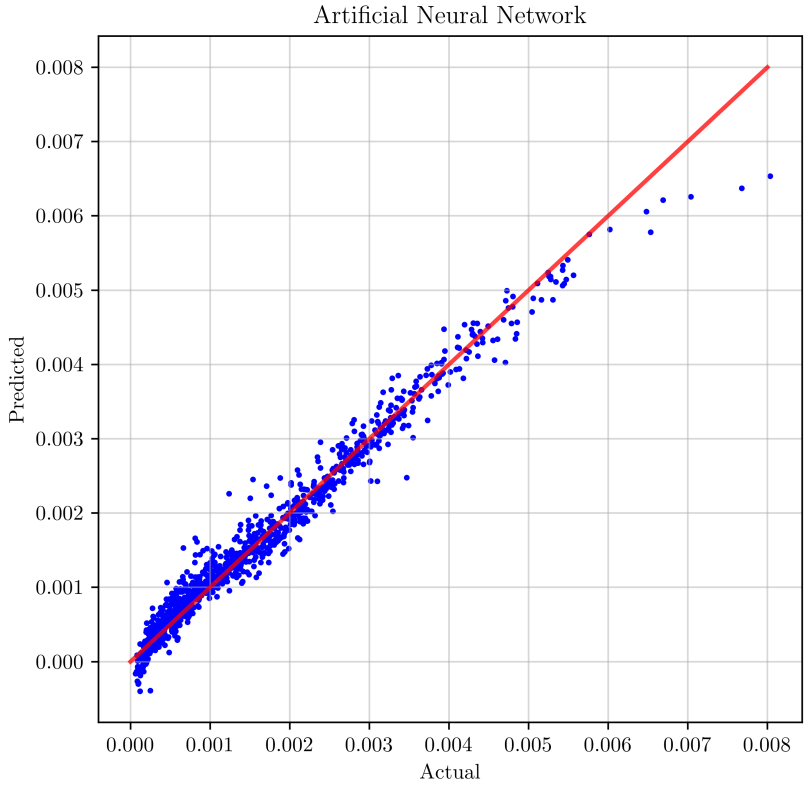


Figure 4.7: ANN predicted vs actual values

minutes to train. An early ending option was used where the model would continue to train until the error did not change for five iterations.

4.4 Support Vector Regression

Table 4.1: Parameters for SVR model

Parameter	Purpose
Kernel type	linear, polynomial, rbf, sigmoid
Degree	In the polynomial kernel, you can set the degree of the polynomial used.
Gamma	Kernel coefficient for rbf, poly and sigmoid
coef0	Independent term in kernel function. Only significant in poly and sigmoid.
C	Regularization parameter. Strength of the regularization is inversely proportional to C.
Epsilon	The amount of error from the support vector that is acceptable.

To optimize the SVR model, a grid search was conducted to find the optimal value of each SVR parameter listed in Table 4.1. The most important parameter was the kernel of which the polynomial type performed the best. Finding the best degree of the polynomial and the coefficient (coef0) finalized the optimization. The final model had the parameters listed in 4.2. An ϵ of 0.000001 meant that the tube boundaries were only 0.000002 wide making this a very accurate model. The final SVR model had an R^2 of 0.994 and RMSE of 0.000103 on the testing data. Figure 4.8 shows the predicted values of the test set with the actual values.

Table 4.2: SVR Final Parameters

C	coef0	degree	ϵ	gamma	kernel
1	0.9	8	0.000001	scale	poly

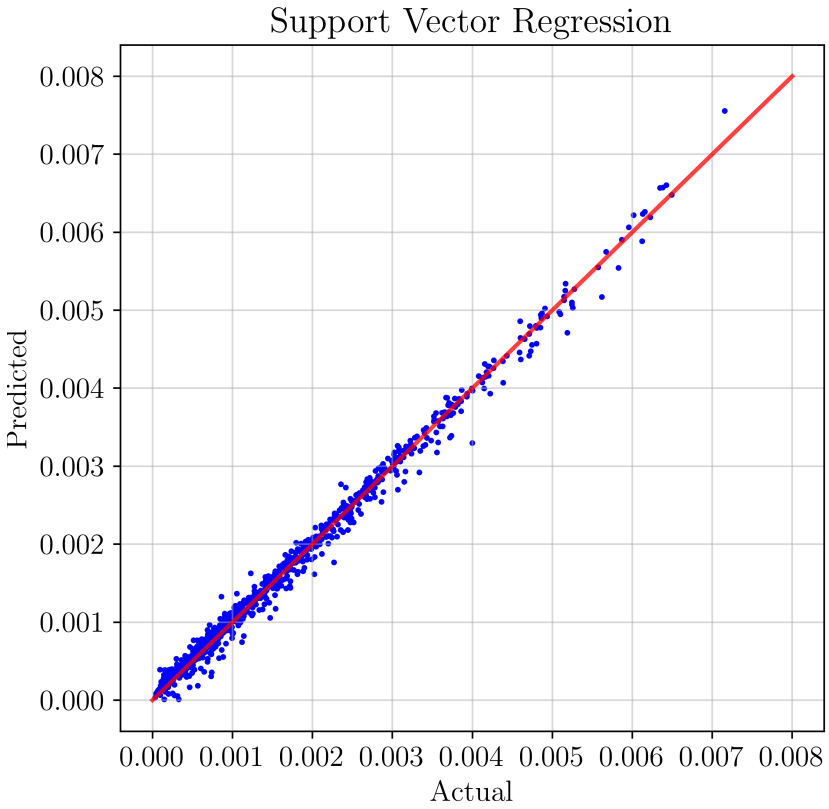


Figure 4.8: SVR predicted vs actual values

4.5 Gaussian Processes Regression

For the GPR model, no optimization or changing of parameters was needed. The first training of the GPR gave a R^2 of 0.99999 and a RMSE of 0.00000119.

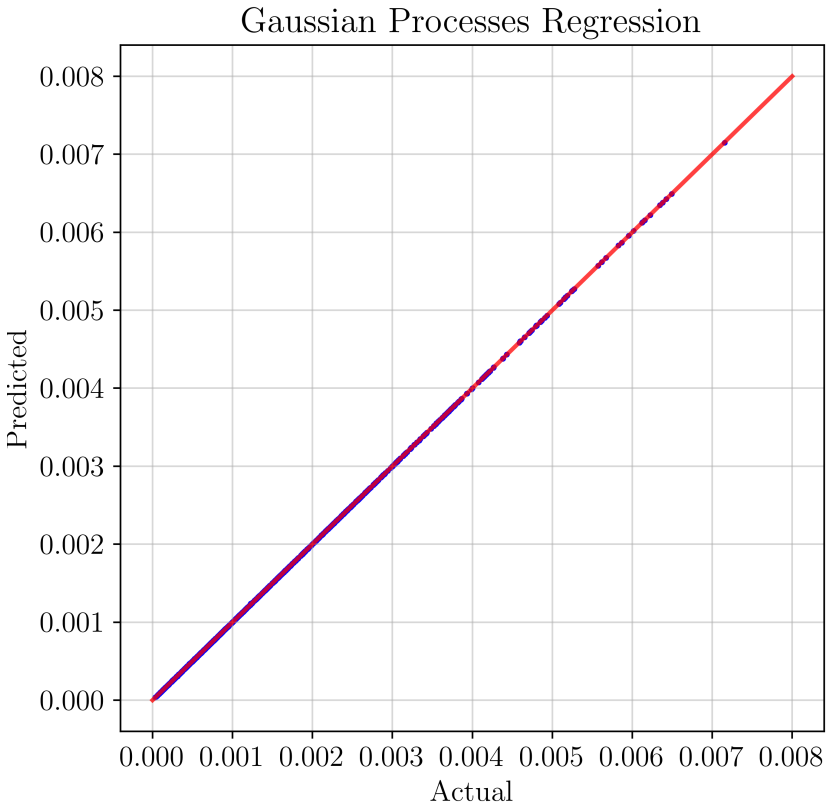


Figure 4.9: GPR predicted vs actual values

Clearly the SVR and GPR models performed the best when only looking at the error calculations but further testing was needed on all the models for training set sample size and noise tolerance.

4.6 Response to Different Training Sample Sizes

The amount of data needed is important to ML because not many sets of data may be available or measuring may be expensive and time consuming so the smallest amount of data possible to produce an accurate model is vital. To first check the full data range for each ML model, different sample sizes were selected where instances were randomly chosen from the full pool for 10,000 instances. Figures 4.10 and 4.11 show the errors at each sample size. Once the initial sample size accuracies were calculated, it was clear that not much was to be gained from viewing the range of

samples from 2000 to 9000 so a smaller set of sample sizes were tested. Figures 4.12 and 4.13 show the error calculations on smaller sample sizes.

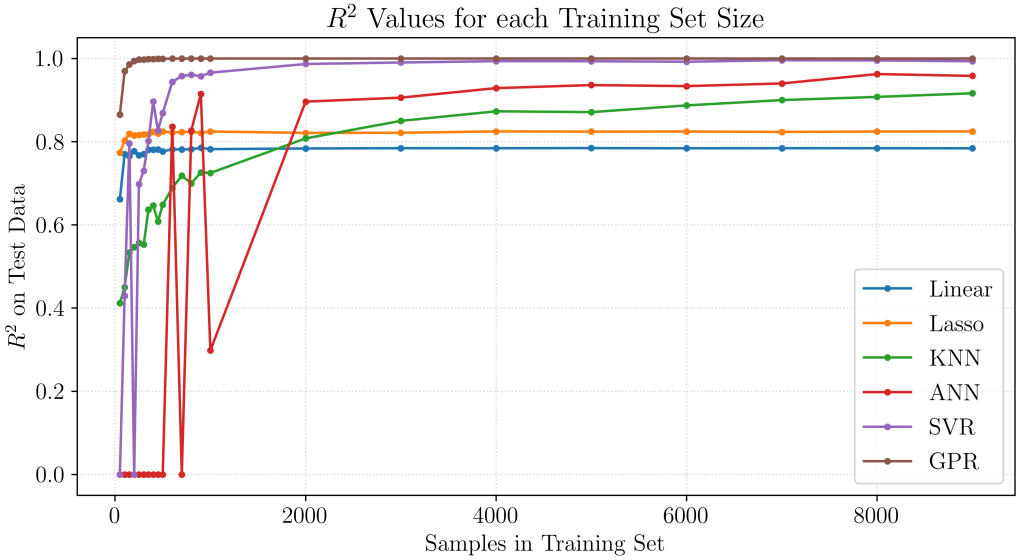


Figure 4.10: R^2 values for up to 9,000 training instances

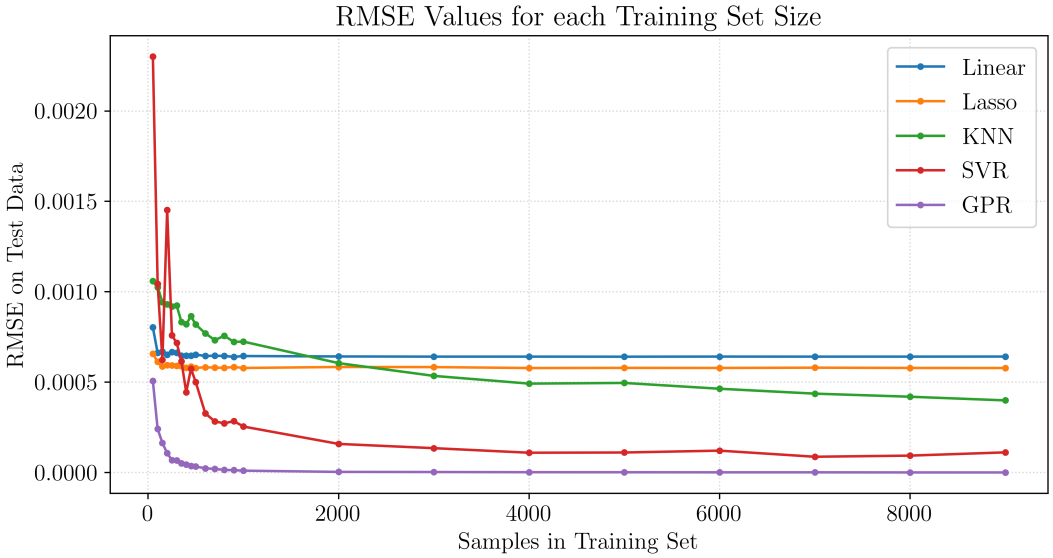


Figure 4.11: RMSE values for up to 9,000 training instances

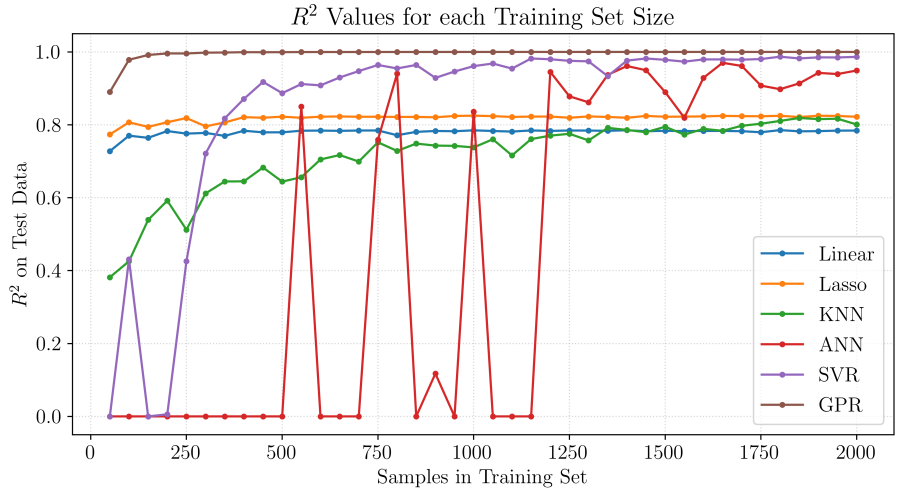


Figure 4.12: R^2 values for up to 2,000 training instances

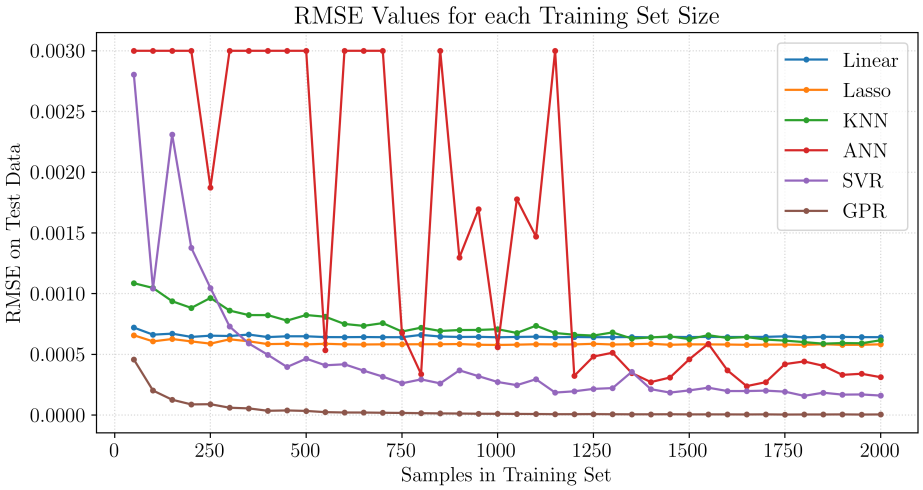


Figure 4.13: RMSE values for up to 2,000 training instances

Clearly, again the GPR model is the most accurate at any sample size with the SVR model performing worse than other model types until around 300 samples are used in the training set. The ANN may take multiple full attempts at making a model which makes the calculations erratic until around the 1200 sample size. The KNN model does best with more instances of data because it directly relates to the training data for averaging so even at the 9000 instance mark, the KNN accuracy is increasing.

4.7 Response to Noise

For the noise response test after conducting the sample size test, it was decided to use 3000 of the 10,000 available instances to train the models to save time while also being because the GPR and SVR model types were performing so much better than the other model types. The percentages of noise chosen to test against are listed below in Table

Table 4.3: Noise percentages and associated standard deviations

Percent Noise	Standard Deviation
0.01	0.000125
0.1	0.00125
1	0.0125
1.5	0.0188
2	0.0251
2.5	0.0313
3	0.0376
3.5	0.0439
4	0.0501
4.5	0.0564
5	0.0627
7.5	0.0940
10	0.125
12.5	0.157
15	0.188

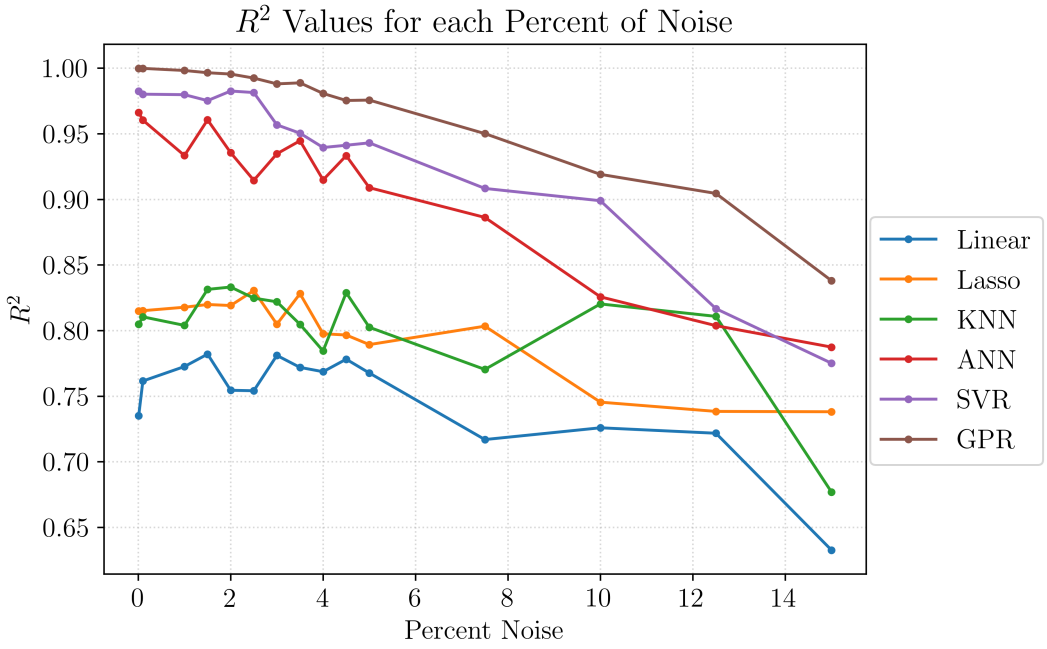


Figure 4.14: R^2 values for each percentage of noise

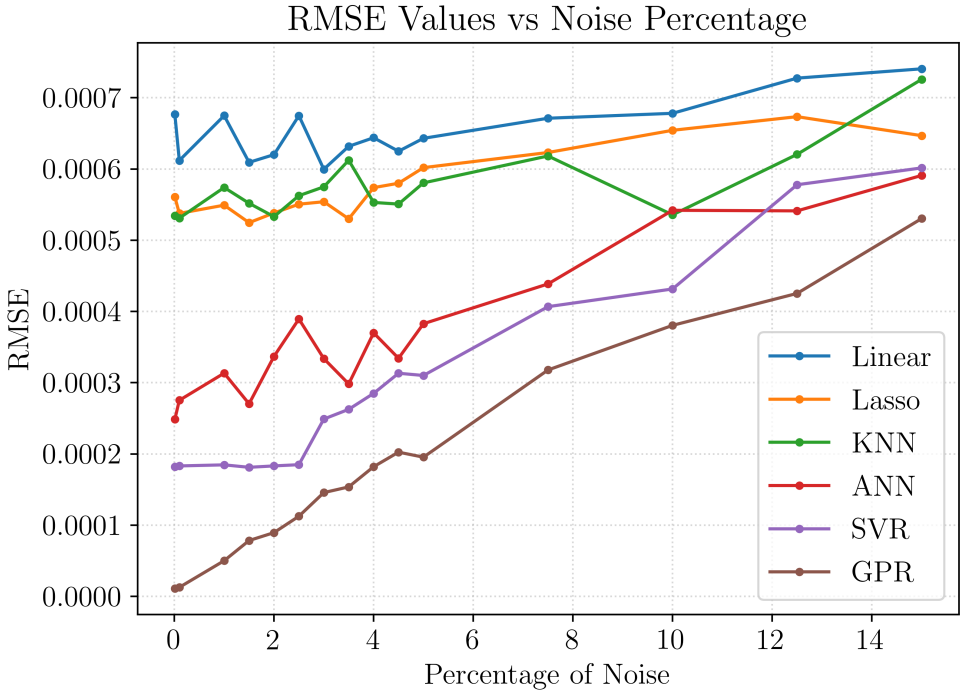


Figure 4.15: RMSE values for each percentage of noise

4.8 Sensitivity of GPR Model

Since the GPR model was the most accurate in every way, sensitivity analysis was performed on it. Figure 4.16 shows the first degree sensitivity values which relate the correlation between each predictor variable individually with the prediction.

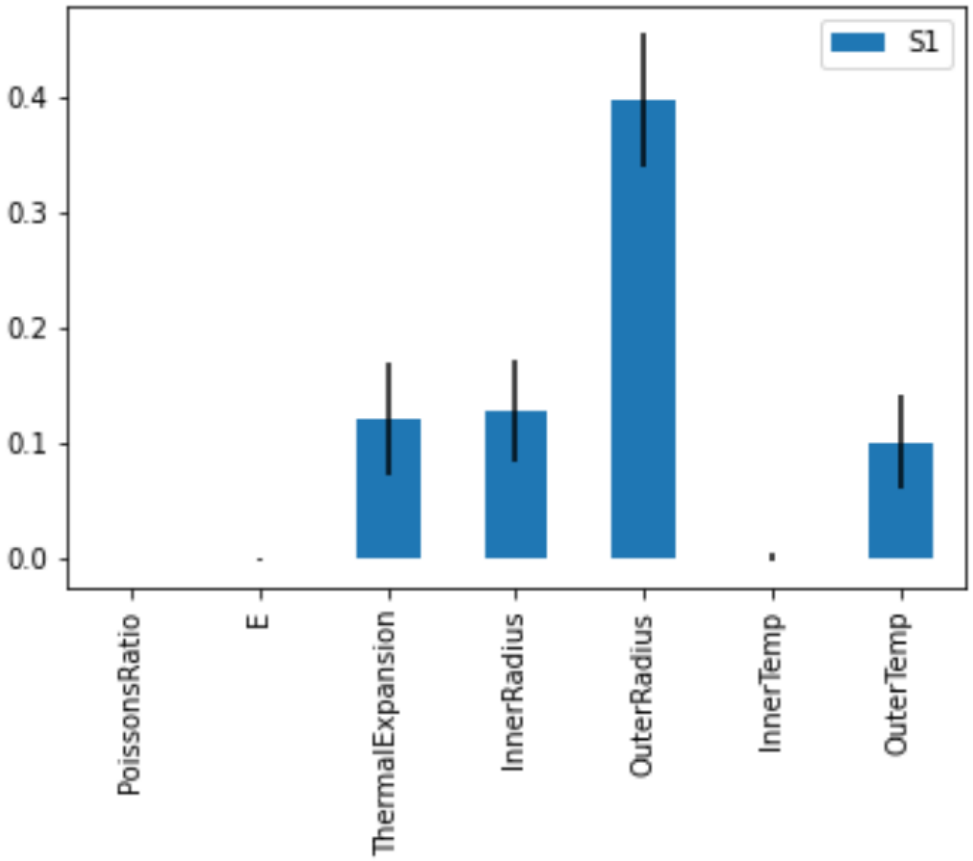


Figure 4.16: First degree sensitivity graph

4.9 Production Ready Model

A production ready model is a model that would be put in place in a production setting. This model would be what would let manufacturing personnel know if produced materials were within tolerance. For the production ready model, the most optimal model is trained from the original 10,000 instance dataset then tested on a new 500 instance dataset that has not been encountered before. Since the GPR model type was the most accurate in all settings, that type model was chosen for the

production ready model.

Since a new dataset is being introduced to the experimental method, data analysis should be performed on it. As before, a histogram of the new data is helpful in understanding how the model will perform when it encounters the new data. Figure 4.17 shows the histograms of both the original 10,000 instance dataset that would be used to train the production ready model along with the new 500 instance production data that the model would be tested on. When viewing the distributions of the datasets, there are some obvious issues the production dataset could encounter when being tested. The Young's Modulus values for the production data ranges lower and much higher than the training data meaning that the model may not interpret the values that do not overlap with the training data ranges.

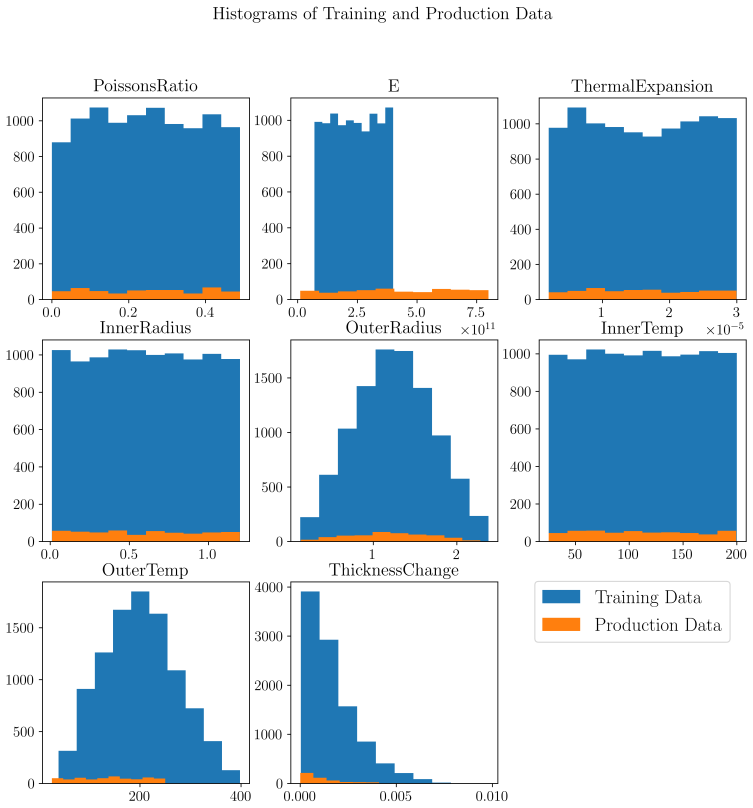


Figure 4.17: Histogram of training data and production data

When training and testing the production ready model without removing any instances of data, the performance of the GPR model was expectedly bad with a R^2 of 0.25 and RMSE of 0.00115. The predictions and true values are shown in Fig.4.18.

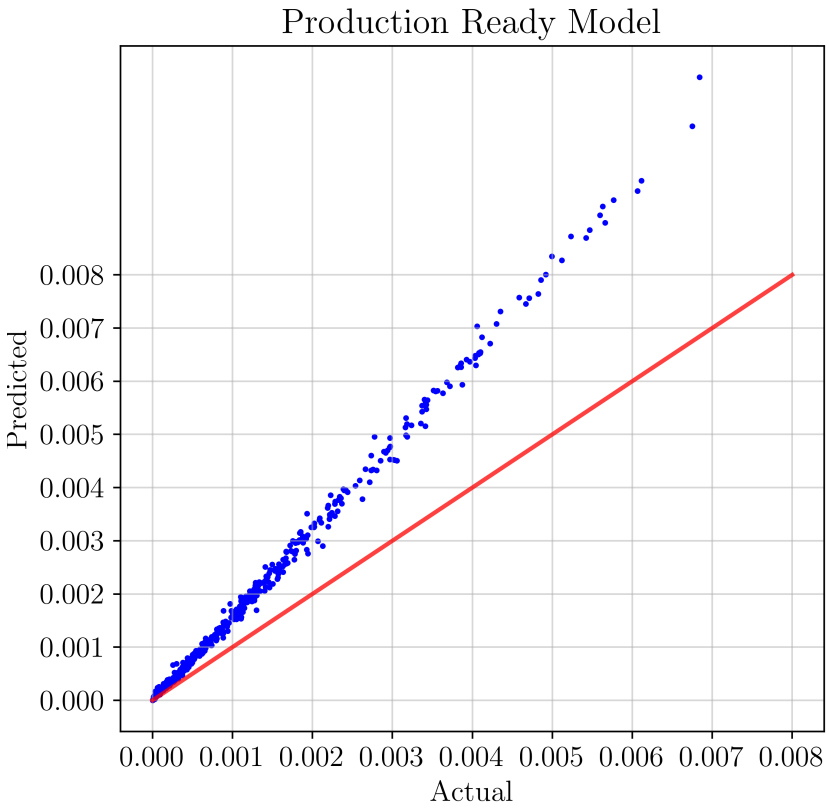


Figure 4.18: Predicted vs actual values of full production ready dataset

Since the GPR performed was very inaccurate, some extra work was needed. The obvious issue was different ranges of values between the two datasets so instances with values in ranges that did not match were pruned and a new model was trained and tested. The histogram of the two new pruned datasets may be viewed in Fig.4.19. Now that the ranges of the values matched more closely, the model performed much better with a R^2 value of 0.97 and RMSE of 0.000224. The prediction and actual values are graphed in Fig.4.20 which shows a much closer prediction set to the actual values.

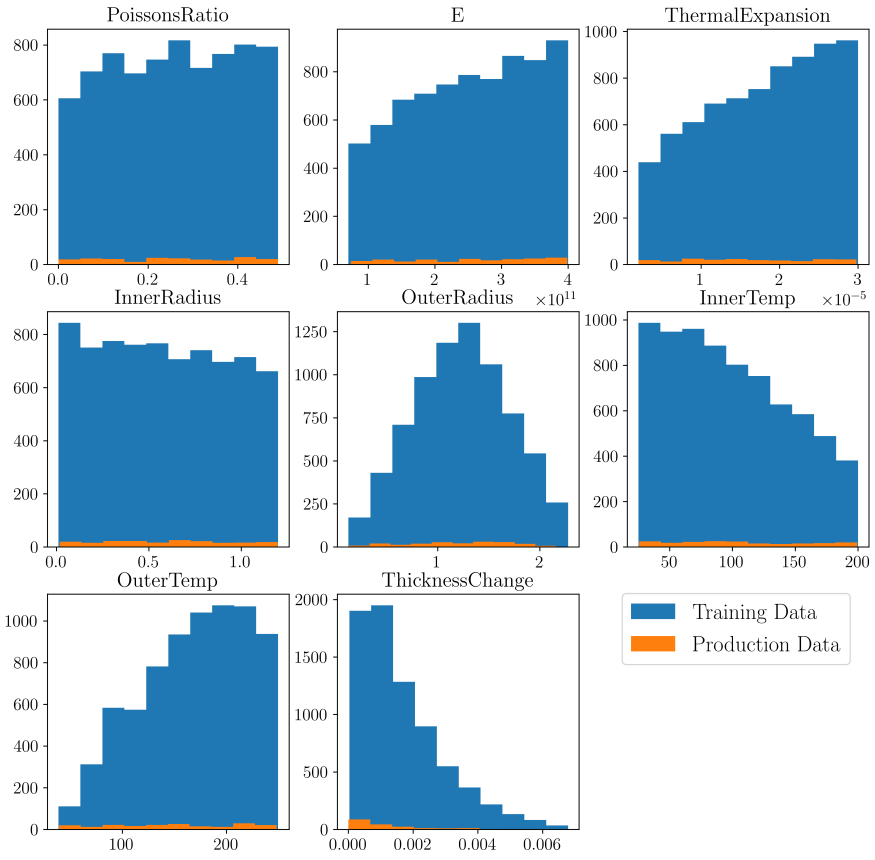


Figure 4.19: Histogram of training data and production data that overlap in value ranges (pruned dataset)

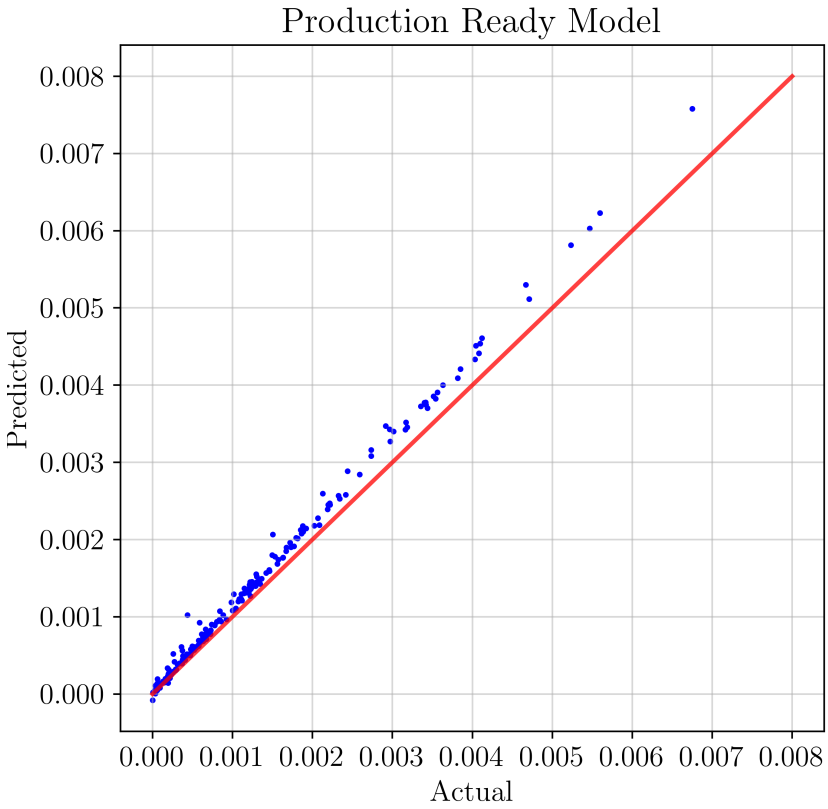


Figure 4.20: Predicted vs actual values with pruned dataset

No more optimization was completed on this production ready model because 300 instances of data of the 500 production dataset were already removed in the pruned model.

CHAPTER 5: Conclusions and Future Work

ML models can be a powerful tool in predicting information in materials science but caution must be taken before implementing any model into production. Many factors can effect the outcome of a model which come in a variety of forms. The dataset itself is the most important part since not having data for a specific range can have a catastrophic effect on the final model. Depending on the model used, a large amount of measured data may be required to sufficiently train and test a ML model. A work around for noise is also available but knowing where and how the noise is present can increase the accuracy of the ML model before training begins. There are many ML model types available with as many parameters to optimize in each. One could spend their whole lives attempting to optimize a ML model but there needs to be a trade-off with accuracy and time spent training and testing ML models that go into production. Having an understanding of the limitations of each ML model is a helpful tool in knowing when to stop optimization and possibly looking for different model types.

For this experiment, the GPR model was the clear winner at sample size response and noise tolerance but that could not alleviate the differences of data ranges in the production data. For future work, there are many avenues that could be followed. More simulation data could be produced for the ranges of data in the production dataset so that a more accurate model could be produced. Measuring and using experimental data for model training would give much more insight into how models would perform on real world data along with if the noise addition was accurate compared to the real world measurements which almost certainly would contain noise.

REFERENCES

[1] C. Patterson, *Managing a real-time massively-parallel neural architecture*. PhD thesis, 01 2012.

[2] S. Lahiri and K. Ghanta, "The support vector regression with the parameter tuning assisted by a differential evolution technique: Study of the critical velocity of a slurry flow in a pipeline," *Chemical Industry and Chemical Engineering Quarterly*, vol. 14, 07 2008.

[3] Alisneaky, "Kernel Machine." Accessed Sep. 6, 2022 [Online].

[4] J. Wang, "An intuitive tutorial to gaussian processes regression," 09 2020.

[5] H. Imran, N. M. Al-Abdaly, M. H. Shamsa, A. Shatnawi, M. Ibrahim, and K. A. Ostrowski, "Development of prediction model to predict the compressive strength of eco-friendly concrete using multivariate polynomial regression combined with stepwise method," *Materials*, vol. 15, p. 317, Jan 2022.

[6] A. Agrawal, P. Deshpande, A. Cecen, B. Gautham, A. Choudhary, and S. Kalidindi, "Exploration of data science techniques to predict fatigue strength of steel from composition and processing parameters," *Integrating Materials and Manufacturing Innovation*, vol. 3, 04 2014.

[7] H. Bao, S. Wu, Z. Wu, G. Kang, X. Peng, and P. J. Withers, "A machine-learning fatigue life prediction approach of additively manufactured metals," *Engineering Fracture Mechanics*, vol. 242, p. 107508, 2021.

[8] J. Gao, C. Wang, Z. Xu, J. Wang, S. Yan, and Z. Wang, "Gaussian process regression based remaining fatigue life prediction for metallic materials under two-step loading," *International Journal of Fatigue*, vol. 158, p. 106730, 2022.

[9] pandas, "pandas Homepage." Accessed Jul. 18, 2022 [Online].

[10] seaborn, "seaborn Homepage." Accessed Jul. 18, 2022 [Online].

[11] SKLearn, "SKlearn Homepage." Accessed Jul. 18, 2022 [Online].

[12] Google, "Google Colab Introduction Page." Accessed Aug. 24, 2022 [Online].

[13] S. G. PATRO and D.-K. K. Sahu, "Normalization: A preprocessing stage," *IARJSET*, 03 2015.

[14] M. Awad and R. Khanna, *Support Vector Regression*, pp. 67–80. Berkeley, CA: Apress, 2015.

[15] scikit learn, "3.3. Metrics and scoring: quantifying the quality of predictions." Accessed Jul. 18, 2022 [Online].

[16] T. Iwanaga, W. Usher, and J. Herman, "Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses," *Socio-Environmental Systems Modelling*, vol. 4, p. 18155, May 2022.

[17] J. Herman and W. Usher, "SALib: An open-source python library for sensitivity analysis," *The Journal of Open Source Software*, vol. 2, Jan 2017.