Temperature Prediction of Permanent Magnet Synchronous Motors using *k*-Nearest Neighbours

Nicholas Imperius
Software Engineering
Lakehead University
Thunder Bay, Canada
npimperi@lakeheadu.ca

Jimmy Tsang
Software Engineering
Lakehead University
Thunder Bay, Canada
jtsang@lakeheadu.ca

***Abstract*—With the rising demand of Permanent Magnet Synchronous Motors (PMSM), there is an absence of a good method to predict the temperature of components within these motors, which can result in severe health and safety risks. In this paper, we set out to determine a reliable and sustainable solution to predict and monitor the temperature of components within PMSMs through the use of various machine learning models. Different machine learning models will be used to demonstrate predictive abilities on the data set.**

Keywords—Permanent Magnet Synchronous Motor, Thermal Management, Machine Learning, Deep Learning, Temperature Estimation, Regression, Neural Network.

I. Introduction

Due to demand of automotive motors, requirements such as maintaining high power and torque whilst being efficient makes the permanent magnet synchronous motor (PMSM) the preferred selection by many. Ideally, you want the best estimates in terms of calculating the temperatures in the motor. A more accurate estimate allows automakers to build motors more efficiently and gain access to more performance. Torque measurements are crucial in determining peak performance. It has been found that temperature-sensitive components are prone to failure under high thermal stress and must be monitored [1]. There are many methods that have been put forward to assist in measuring the temperature of a PMSM, but it was found that a lot of experimental preparation was needed, in turn, raising the costs and complexity [2]. To combat this, in this paper, we experiment with multiple machine learning regression algorithms in order to develop a model that can precisely predict the temperatures of four parts: the permanent magnet, stator tooth, stator winding, and stator yoke. Machine learning is a methodology of data analysis that is used to predict certain attributes when given a dataset, finding patterns in these data to make generalizations and predictions about said data. If a machine learning model is able to properly predict overheating in a PMSM system, then manufacturers will be able to maximize their performance and develop control strategies to increase the capability of the motor while reducing overheating. The regression models we will be looking at are: linear regression, random forest regression, *k*-nearest neighbour, and artificial neural networks. A scheme of the various steps followed to develop our prediction model of the PMSM temperatures is shown in Fig. 1. We first take in our raw data and apply preprocessing techniques to it. After the preprocessing and curation steps have completed, we now build the model and train it, during this process we are optimizing the parameters of the model. Finally, we test the model on our test sets to validate the performance.

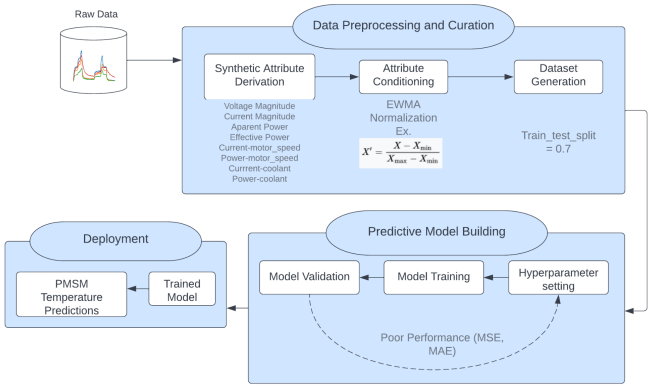

Fig. 1. Scheme of the project from input of raw data to testing our final model.

II. Literature Review

Different machine learning algorithms have been deployed to attempt to predict the temperatures of the motor components. Neural networks have been studied before, specifically, Recurrent Neural Networks and Artificial Neural Networks. Simpler machine learning methods, such as Linear Regression, have also been used to predict the temperature components and have achieved good results with proper preprocessing [3]. Another regression technique that we used was *k*-nearest neighbours (*k*-NN). In this technique, training observations are stored and new observations are estimated by taking the mean of the stored points that are nearest to the new

observations [4]. Decision trees, specifically Random Forests (RF), have also been used to predict temperature behaviours. Unlike other machine learning methods, decision trees can tend to greatly overfit the data if it is just a single tree; therefore, using multiple trees that are fit to random subsets of data reduces the risk of overfitting [3]. RFs are a collection of single trees that are compared against each other and are chosen. Neural networks are powerful machine learning methods. A multilayer perceptron regressor is a model that consists of an input layer, a hidden layer and an output layer [5]. The simplest model, linear regression is a supervised learning algorithm that fits a regression line to the data [6]. Linear regression models are the easiest to understand and can be widely used in various tasks which is why it is one of the more commonly used models in regression analysis.

III. Method

To gain a better understanding on how to make more educated estimations, we must look at the variables in our system and how they will be measured. In addition, the size of the data needs to be considered too.

A. Variables and Attributes

There are sixteen total input variables with eight being values from the data set and the remaining eight are derived values. In addition, there are four target variables that we are looking to predict at the same time with our models based on the sixteen inputs we have. The four target variables are the three stator temperatures and the permanent magnet surface temperature. In Table I. you will find a detailed list of every input and target variable, along with the type of parameter they are.

TABLE I. Variable Types and Attributes

Variable	Definition	Parameter Type
Ambient Temperature	The ambient temperature of the test environment.	Input
Coolant Temperature	The temperature of the coolant.	Input
Direct-Axis Current	The current of the d-component.	Input
Direct-Axis Voltage	The voltage of the d-component.	Input
Motor Speed	The speed of the motor.	Input
Permanent Magnet Surface Temperature	The temperature of the permanent magnet.	Target
Stator Tooth Temperature	The temperature of the stator tooth.	Target
Stator Winding Temperature	The temperature of the stator winding.	Target
Stator Yoke Temperature	The temperature of the stator yoke.	Target
Torque	The torque of the motor.	Input
Quadrature-Axis Current	The current of the q-component.	Input
Quadrature-Axis Voltage	The voltage of the q-component.	Input
Voltage Magnitude	The voltage magnitude computed using: $U = \sqrt{u_d^2 + u_q^2}$	Derived
Current Magnitude	The current magnitude computed using: $I = \sqrt{i_d^2 + i_q^2}$	Derived
Apparent Power	The electric power computed using: $S = U \times I$	Derived
Effective Power	The effective electric power generated by the motor computed using: $P = u_d \times i_d + u_q \times i_q$	Derived
Current - motor_speed	Computed using: $IMM = I \times \omega$	Derived
Power - motor_speed	Computed using: $SMM = S \times \omega$	Derived
Current - coolant	Computed using: $IMC = I \times coolant$	Derived
Power - coolant	Computed using: $SMC = S \times coolant$	Derived

B. Units

In this report, the types of units and what they measure being used include the following:

- Voltage: Volts (V)
- Temperature: Celsius (°C)
- Motor Speed: RPM
- Frequency: Hertz

C. Sample Size Plots

In the provided dataset, there are 69 profiles of entries with varying sizes. The range of entries per profile are between 2176, from Profile 47, and 43,971, from Profile 20. On average, a profile contains approximately 19,000 entries. There are a total of 1,330,815 recorded entries and the total sample size is determined to be 185 hours of recordings. To determine the frequency of the recordings, we must perform some calculations, the total amount of time in seconds is calculated in (1).

$$185\ Hours \times \frac{3600\ Seconds}{1\ Hour} = 666,000\ Seconds \quad (1)$$

We can compute our frequency, F, by dividing the number of recordings by the total time in (2) and achieve a frequency of 2 Hz which is equivalent to each sample lasting for 0.5 seconds (3).

$$F = \frac{1{,}330{,}815\ Recordings}{666{,}000\ Seconds} \quad (2)$$

$$F \approx 2\ Hz\ and\ P \approx 0.5\ Recordings/Second \quad (3)$$

Since we know that each session takes 0.5 seconds to complete, we can convert each profile's total session length

from seconds to hours to be easier to understand. Fig. 2 shows that there are some outlier profiles. For example, profile ID 46 and 47 contain approximately 20 minutes of recordings each, which is very miniscule compared to some of the other entry profiles and can be considered outliers and may not provide the most beneficial data.

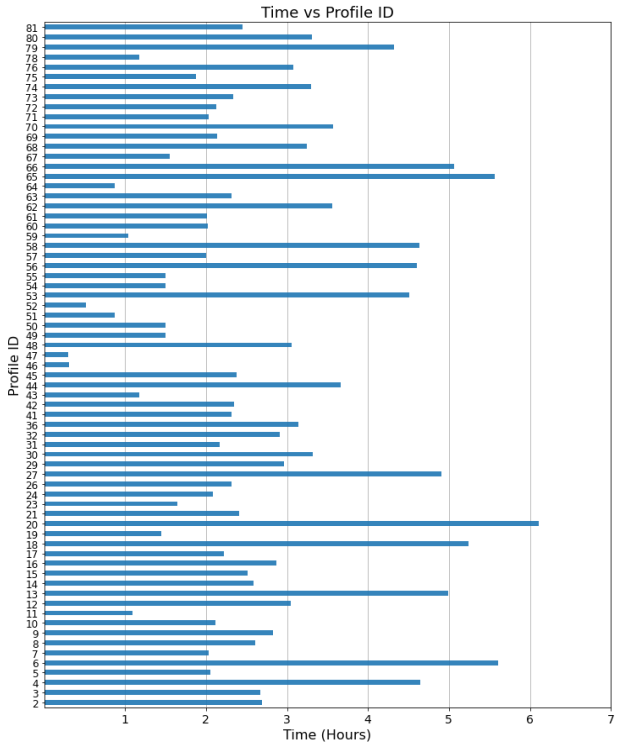

Fig. 2. The amount of time, represented in hours, that each.

D. Correlation

Within the dataset, there was definite visible correlation between some attributes. While graphing all the attributes against each other, we were able to visually determine what attributes had a correlation with one another, and which attributes had no correlation whatsoever. As shown in Fig. 3, there are positive correlations between some attributes.

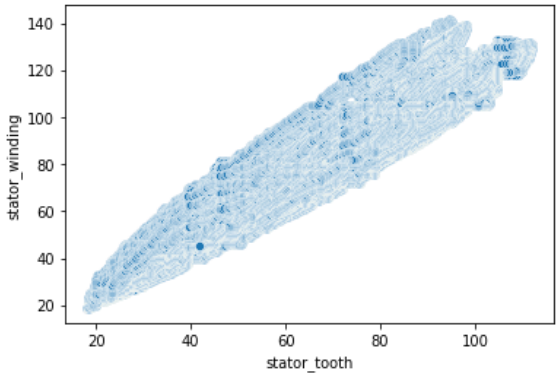

Fig. 3. Stator Winding vs Stator Tooth graph shows positive correlation.

However, there were some instances where negative correlation was observed, as seen in Fig. 4.

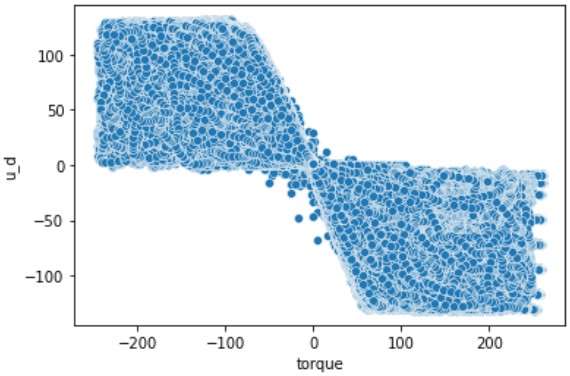

Fig. 4. u_d vs torque graph shows negative correlation.

Fig. 5, there is a correlation heatmap that shows the correlation between all attributes in the data set. In the heatmap, blue values represent high positive correlation whereas red values represent high negative correlation.

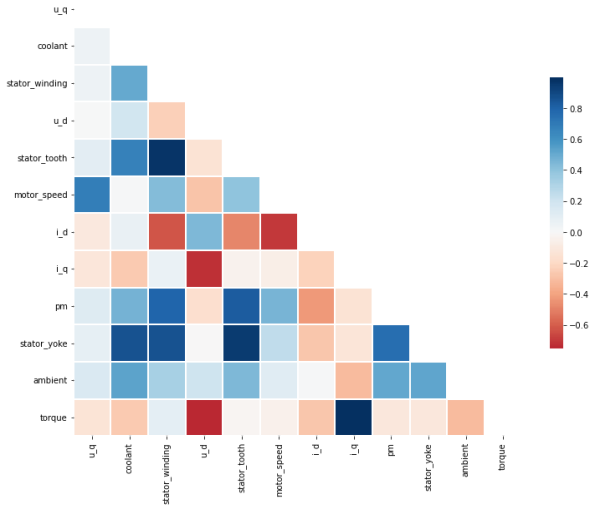

Fig. 5. A heatmap showing the correlation between all attributes. A red colour represents a strong negative correlation while a blue colour represents a strong positive correlation.

Fig. 6 and Fig. 7 contain two different profiles that show the distribution of their parameters over each measurement.

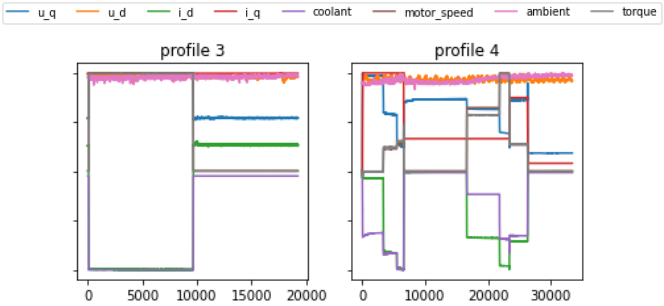

Fig. 6. Distribution of input variables over time in profile 3 and 4.

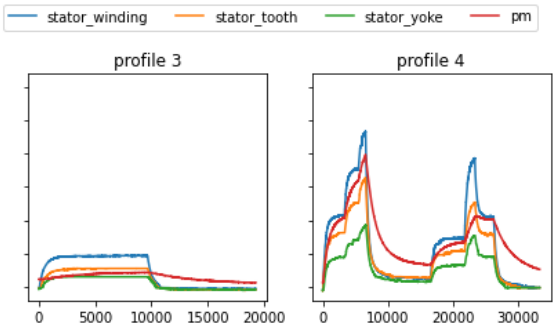

Fig. 7. Distribution of target variable temperatures over time in profile 3 and 4.

Fig. 8 shows the frequency distribution, or histogram, of all the variables that were used in the data set. This figure only shows 3 of the variables from the plot. We can see that some of the data is multimodal with a large range of possible values and other variables that consist of a small range of values with larger frequencies which indicates a larger range of values.

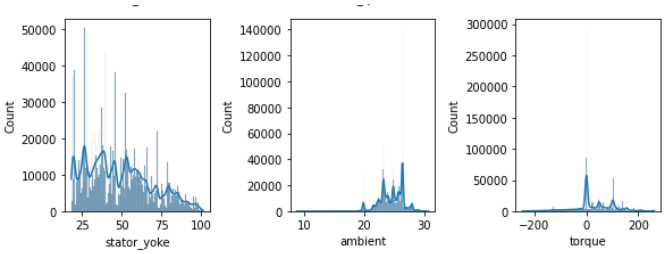

Fig. 8. Histogram of stator_yoke, ambient and torque showing the frequency of each value.

E. Feature Engineering

From our original input variables, we can derive more synthetic attributes to train the model on. We can determine whether using the synthetic input variables would allow for a more accurate model. Fig. 9 contains the distribution of the input parameters when we have a combination of both the original and synthetic input variables.

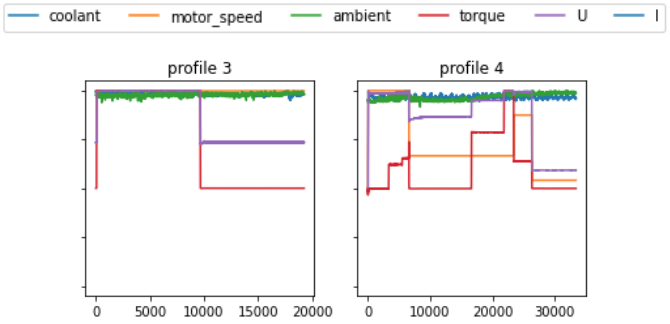

Fig. 9. Distribution of input variables, coolant, motor_speed, ambient, torque with derived attributes, U and I, over time in profile 3 and 4.

F. Training and Testing

In our project, we have decided to randomly separate our data into a 7:3 ratio to train and test our model. This ratio provides us with the most accurate and valid results [8]. The process of randomly separating the data will be automatically conducted by the function *train_test_split()*. As there are 1,330,815 entries before preprocessing, which totals to 185 hours of recordings, we will allocate 129.5 hours at random to the training set, while the remaining 55.5 hours will be allocated to the testing set. This means that for the datasets that do contain the outliers, whether they have been adjusted or not, approximately 933,000 entries will be used to train the model, with the remaining approximate 400,000 entries being used for testing.

G. Data Preprocessing

For data preprocessing, we split the dataset into 2 preprocessing methods, one where we performed synthetic attribute derivation, and one where we did not. Synthetic attribute derivation involves deriving extra values to be implemented in the dataset; therefore, we created another dataset that contained only the synthetic attributes. We wanted to compare the results from testing based on the recorded inputs vs synthetic attributes derived from the input variables. The values that were synthesized are derived in Table I. We will compare the results from using both types of preprocessing methods and then determine whether the models performed better with synthetic attribute derivation or without it.

To continue preprocessing the dataset, we needed to verify that there were no missing values in the dataset, as well as remove any outliers that occur. Removing outliers allows for a more accurate estimation of the temperatures. Visually, the outliers are shown in Fig. 10.

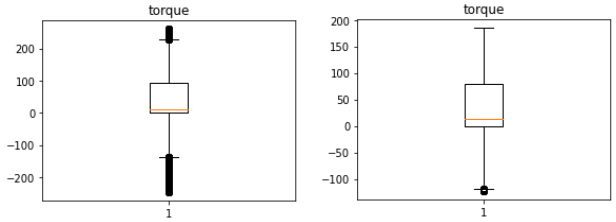

Fig. 10. Box plot of the Torque variable before and after outlier removal.

The dots located outside of the whiskers are considered outliers. For one of our preprocessing methods, we replaced the outlier values with the features median value. The median was chosen to replace the outlier because it impacts the overall data the least when the outlier is removed [9]. After removing the outliers, we found that more outliers were created at the same time, this is because of masking. When the outliers were removed, it changed the interquartile range, which then could cause more outliers to appear. Although there are some outliers, the number compared to before has drastically decreased and we left those outliers there since they were not in the original set of outliers that needed to be removed.

Additionally, we also wanted to observe how the model performed if we kept the outliers in the dataset, as well as if we completely removed the row with the outliers; therefore,

we created separate datasets for each of these three cases. This brought our preprocessing methods to a total of six. Furthermore, we also wanted to see how attribute conditioning would affect the performance of the models, so we applied an exponentially weighted moving average (EWMA) to it. EWMA is used to reduce the noise in data by weighing the number of observations and using their average as you move further along. This now brings the total final number of preprocessing combinations to twelve.

The next step in preprocessing data is to normalize it. Data normalization involves normalizing all of the data points to values we specify, in this case, we normalized all the data to values between 0 and 1. Data normalization is done in order to change all the different attributes to the same scale to make it easier for the program to find trends and generalize data. In the dataset, each attribute has a different scale, with some varying over large amounts of data; therefore, by normalizing all of the data, we shrink the scale of all attributes to [0,1].

H. Model Selection

To implement our models, we used scikit-learn, a free online library that contains all of the algorithms needed for regression, classification, or other types of machine learning algorithms. To model our dataset, we chose the following implementations: linear regression, random forest regressor, *k*-nearest neighbour, and artificial neural network. Each model was trained with all three datasets with and without the EWMA applied to the data, as we wanted to see the effects of keeping outliers or removing outliers would have on the accuracy of the models. As such, each model was ran twelve times, with a different combination of preprocessing methods each time, and compared the results.

Linear regression is a regression algorithm that assumes there is a linear relationship between the input attributes and the output attributes. In (4), a multi class linear regression equation is described where y_i is the dependent variable, x_i is the independent variable, β_0 is the y-intercept, β_p are the coefficients for the slope for each independent variable, and ϵ is the residual value.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_p x_{ip} + \ \ldots \ + \beta_p x_{ip} + \epsilon \qquad (4)$$

By using multi class linear regression, we are able to map the relationship between many dependent variables and many different input variables. In our dataset, we have multiple input variables, as discussed earlier; therefore, multi class linear regression is required compared to single variable linear regression. This method was chosen because we have a good understanding of how it works.

The next method we implemented was a random forest regressor. This is a method that involves combining multiple decision trees in order to perform regression on the dataset. This method was chosen because we have an understanding of decision trees already, but with the forest, it allows for a more accurate prediction of the output variables. Random forest regressors involve running a series of decision trees in parallel and outputting the mean prediction of each individual tree. In a decision tree regressor, to measure the quality of a split, a measure known as mean squared error, mean absolute error, or poisson is used. Mean squared error is the measure of error squared, whereas mean absolute error is the absolute value of all the errors. Poisson is a measure that uses Poisson deviance to find the goodness of fit for a split, as seen in (5), where D is the measure of Poisson deviance, y_i is the observed values, and $\hat{U}_i$ is the mean of *i*.

$$D = 2 \sum_{i=1}^{n} [y_i \, log(\frac{y_i}{\widehat{u}_i}) - (y_i - \widehat{u}_i) \qquad (5)$$

In Fig. 11, a visual representation of a random forest is shown. In the figure, we can see that when given an input, there is *n* number of trees created, which can be specified by the user. The grey dots are the path that is followed in the tree, with the bottomost leaf being the prediction of a specific tree. As seen below, the prediction of the random forest regressor is the average of all the predictions of the individual trees. There are no interactions between the trees within the random forest. We can aggregate the prediction of each tree into a random forest that combines the results.

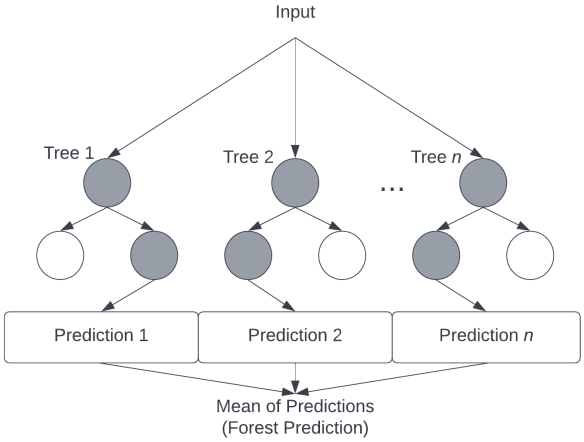

Fig. 11. A figure illustrating the setup process of a Random Forest Regressor.

After deciding on these two methods, we wanted to implement some models that we have not done before, which is why we implemented *k*-nearest neighbour and an artificial neural network (Multilayer Perceptron Regressor) after extensive research, which is discussed in our literature review. *k*-nearest neighbour is an algorithm that assumes that similar values are close in proximity, thus, it predicts values based on how close they are to each other [10]. The algorithm will compute the distances from the input to the closest neighbours. To compute the distance from the input to their neighbour, we can use either the Euclidean, Manhattan or Minkowski methods shown in (6), (7), and (8), respectively. In these equations, *d* represents the distance, and *i* and *j* are two

points, where (x_{in}, x_{jn}) are the coordinates of the points. For (8), the h value represents the order of the norm, usually set equal to 1 or 2 which equates to the Manhattan distance and the Euclidean distance, respectively.

$$d(i,j) = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \ \ldots} \quad (6)$$

$$d(i,j) = \left|x_{i1} - x_{j1}\right| + \left|x_{i2} - x_{j2}\right| + \ \ldots \quad (7)$$

$$d(i,j) = \sqrt[h]{\left(x_{i1} - x_{j1}\right)^h + \left(x_{i2} - x_{j2}\right)^h + \ \ldots} \quad (8)$$

In addition, the algorithm can be designed to limit the search distance based on a parameter in the initialization function of the model. This method was chosen because there was a copious amount of supporting research that used *k*-nearest neighbours as a regression model for this type of problem.

Lastly, we used an artificial neural network, built with the multi-layer perceptron library that scikit implements, as our last machine learning algorithm. Artificial neural networks are a type of machine learning algorithm that mimics how neurons in a human brain send signals to each other [11]. In neural networks, each neuron is considered a node, and each node has an activation function that determines whether the neuron is activated or not. This method was chosen because neural networks are becoming increasingly popular due to the predictive abilities they provide, and we believe that it would be one of the better machine learning regression models for this type of problem. For the artificial neural network, two things are needed, the maximum number of iterations and an activation function, which determines whether a neuron should be fired or not, is required. For the development of our model, we used the *tanh* activation function, represented in (9), which takes in an input value and outputs a value from [-1, 1]. In (9), z represents the net input value, which would be the summation of the weight of a neuron multiplied by the input of that neuron.

$$tanh = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (9)$$

In Fig. 12, a visual representation of a MLP Regressor, is shown. The blue nodes represent the input neurons, which is equal to the number of input variables being used. The green nodes represent the neurons in the hidden layer, in which

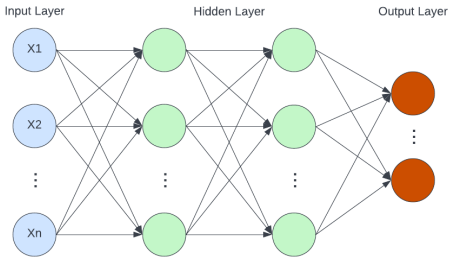

Fig. 12. A figure illustrating an example of an artificial neural network.

activation functions are applied to determine whether a neuron should fire or not. Lastly, the red neurons represent the output layer, in which the final result is sent to.

I. Performance Metrics

To measure the performance of our models, we looked at 3 specific scores: mean squared error, mean absolute error, and R^2 score. Mean squared error (MSE), one of our performance metrics, is the measure of the average difference between the actual and estimated values squared computed using (10). In (10) and (11), y_i represents the actual value, $\bar{y}_i$ is the predicted value and n is the number of samples.

$$MSE(y, \bar{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \bar{y}_i)^2 \quad (10)$$

Mean absolute error (MAE) is the average measure of absolute error between the actual and estimated values computed using (11). Ideally, the MSE and MAE should be as close to 0 as possible to indicate the most accurate prediction.

$$MAE(y, \bar{y}) = \frac{1}{n}\sum_{i=0}^{n-1}\left|y_i - \bar{y}_i\right| \quad (11)$$

Lastly, R^2 score is the measure of goodness of fit. It is an indication of how well the regression formula fits the data points and is computed using (12) where $\hat{y}_i$ is the prediction and $\bar{y}$ is the mean value.

$$R^2(y, \bar{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (12)$$

IV. Experimental Results & Analysis

A. Model Performance

When creating our models, there were specific values set for the parameters of each of the models. From these, we would determine the best performing model. We would then apply hyperparameter tuning in order to optimize performance of our chosen model. When a parameter is not specified upon creation of the model, the values are set to the default state as outlined in the sci-kit learn documentation.

For linear regression, all values were set to their default values, as specified in the sci-kit learn documentation.

For our random forest regressor, all values were also set to the default values outlined in sci-kit learn aside from *n_estimators*, which is a count of how many trees are in the forest. The default value for the number of trees in the forest is 100, but we specified it as 10 due to the sheer size of the data set and the time constraints we faced. It is important to note that the default criterion, which is the method used to determine the quality of a split, is defaulted to using mean

squared error, as it is the fastest method. Additionally, the trees in the forest defaultly expand nodes until all leaves are pure.

Progressing further, for our *k*-nearest neighbour, all values were set to their default counterparts, aside from *n_neighbors,* and *metric*. The *n_neigbors* specifies the number of neighbours that any given number searches for, while *metric* is the methodology used to calculate the distance between the input and the neighbours, as detailed in (6), (7), and (8). For our model, we used the minkowski method from (8), as that is found to be one of the most popular methods to measure distances [12]. Our *p* parameter was set to 2 so that we would use the euclidean distance in the model.

Lastly, for our artificial neural network (MLP Regressor), the parameters that we specified were *max_iteration* and *activation*. *max_iteration* is the maximum number of iterations; however, the solver can also iterate until the convergence point is reached if it is before the maximum number of iterations. The maximum number of iterations we specified is 250, compared to the default 200, as we noticed the model did not consume a lot of time. *activation* is the activation function used for the hidden layer. We selected to run our model using the *tanh*, as stated above in (9).

When determining the best model, we compared each model against each other by observing their mean squared error and mean absolute error. Lower MSE and MAE values indicate stronger performing models, as those are measures of error. Between the three preprocessing methods involving how outliers are dealt with as well as using these datasets with only the original data, synthetic data, and EWMA tuned data, it was found that *k*-nearest neighbour had the lowest MSE and MAE, as seen in Table II. While comparing the previous results using synthetic data, it was found that *k*-nearest neighbour with the original real data still performed the best. In Fig. 13, there are four plots that show the model's predicted temperature against the original temperature from the datasets. The density on the y-axis is a normalization of the temperature values to demonstrate the accuracy of the model.

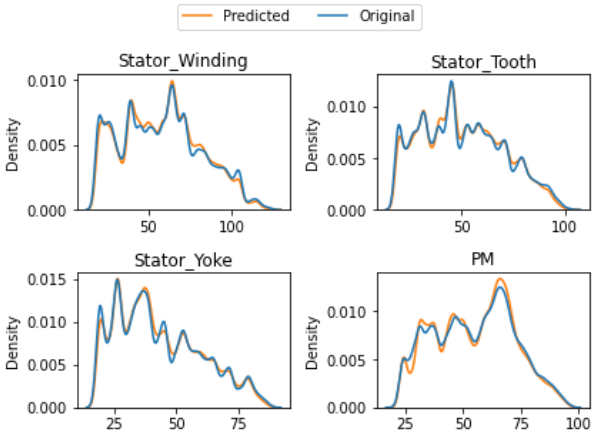

Fig. 13. *k*-NN with outliers removed tested on original input variables, shows the differences in the predicted vs original temperatures.

We found that when comparing the three preprocessing methods, replacing outliers with the median would sometimes cause the model to run more poorly. This could be because we are just replacing a single entry from the row instead of the whole row itself, skewing the information in that row.

From the previous preprocessing methods, we also applied EWMA to both the real and synthetic data, we chose to create a moving average over a span of four samples which would help smooth the data slightly since as the span increases, the data deviates further away from the true value which could lead to incorrect predictions as shown in Fig. 14.

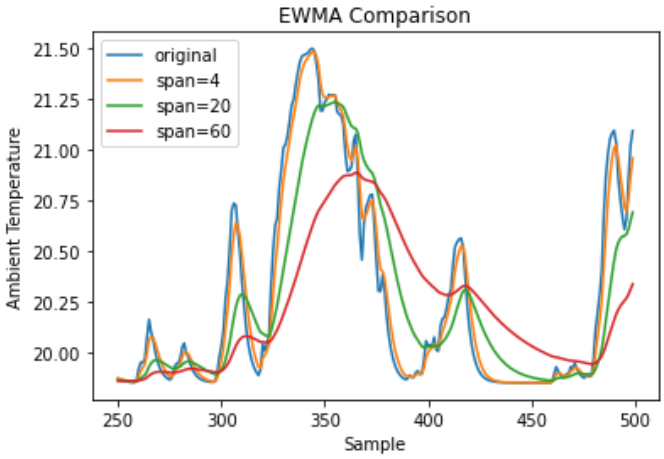

Fig. 14. Comparing different span values on a subset of samples from the ambient temperature feature.

It was found that with EWMA applied, *k*-nearest neighbour with only the original input variables was the model that performed the best. Since EWMA smoothes data, making it less volatile, it was highly probable that applying EWMA would make the MSE and MAE lower compared to not including the EWMA. In Fig. 15, the results of the predictions are compared to the original values for the 4 output attributes.

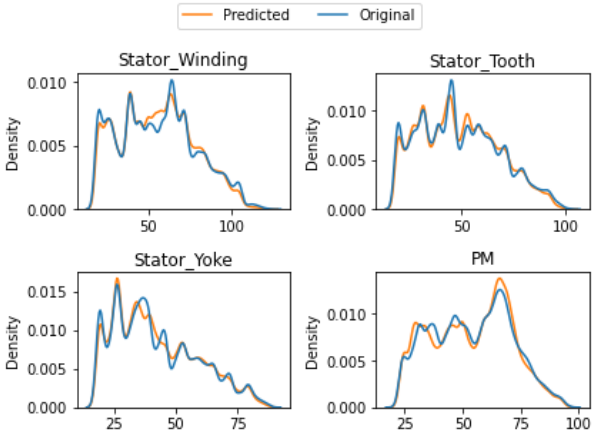

Fig. 15. *k*-NN with only the real data, outliers removed and EWMA smoothed data, shows the differences in the predicted vs original temperatures.

TABLE II. RESULTS ORIGINAL TESTING PHASE

Experiment 1 - Results From Original Data												
Model	**With Original Data**						**With EWMA Tuned Data**					
	With Real Data			**With Synthetic Data**			**With Real Data**			**With Synthetic Data**		
	MSE	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**
Linear Regression	148.880	9.25864	0.71833	270.554	13.0907	0.47553	148.229	9.26023	0.71894	270.235	13.0831	0.47609
Random Forest Regressor	183.134	9.13145	0.67278	88.1670	5.88342	0.83377	173.493	8.69455	0.68555	84.5486	5.88387	0.84341
k-Nearest Neighbour	163.668	7.34729	0.71222	50.3565	4.28971	0.90479	154.596	7.32402	0.72425	47.5857	4.07468	0.90978
Artificial Neural Network	64.5924	5.40484	0.88005	97.3392	6.71712	0.81244	83.7126	6.41025	0.84148	97.8378	6.77530	0.81120
Experiment 2 - Results From Replacing Outliers With Median Value												
	With Original Data						**With EWMA Tuned Data**					
	With Real Data			**With Synthetic Data**			**With Real Data**			**With Synthetic Data**		
	MSE	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**
Linear Regression	235.283	11.5253	0.51393	316.316	14.4684	0.31446	234.914	11.5283	0.51448	315.319	14.4378	0.31642
Random Forest Regressor	179.678	8.13842	0.63940	126.449	6.00545	0.73383	193.206	8.80564	0.60115	129.285	7.10755	0.74002
k-Nearest Neighbour	96.6547	4.54337	0.81164	48.8321	3.80289	0.89688	109.905	4.99977	0.78489	48.6792	3.77731	0.89696
Artificial Neural Network	113.532	7.06537	0.76315	110.943	7.27380	0.76235	153.503	8.26637	0.67841	111.918	7.31452	0.75949
Experiment 3 - Results From Removing Rows That Contained Outliers												
	With Original Data						**With EWMA Tuned Data**					
	With Real Data			**With Synthetic Data**			**With Real Data**			**With Synthetic Data**		
	MSE	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**	**MSE**	**MAE**	**R^2**
Linear Regression	102.304	7.24373	0.72325	202.689	11.1346	0.44035	101.758	7.26827	0.72470	201.979	11.1199	0.44227
Random Forest Regressor	49.8870	4.14274	0.86968	92.2215	6.51211	0.75659	47.0780	3.91182	0.87579	93.3755	6.37281	0.75110
k-Nearest Neighbour	40.4600	3.44815	0.89494	65.4049	4.82447	0.82871	**27.9247**	**2.75045**	**0.92759**	48.6773	3.87998	0.87322
Artificial Neural Network	48.1575	4.35696	0.87348	81.0112	5.97116	0.78108	46.6857	4.30836	0.87771	79.2604	5.85875	0.78578

Therefore, we conclude that the best model is *k*-nearest neighbour. As such, we will apply hyperparameter tuning to that model to maximize the performance and further decrease the MSE and MAE.

In Table II, the results from all of our preprocessing models, linear regression, random forest regressor, *k*-nearest neighbour, and artificial neural network are shown. Vertically, experiment 1 has the results from when outlier data is left in the dataset, untouched. Experiment 2 is when the outlier values were removed and replaced with the median value for that column. Lastly, experiment 3 represents the results of the models when the rows containing outliers are removed in full. Horizontally, we have four separate methods, original data with both real and synthetic data introduced, and then attribute conditioned data with both real and synthetic data. The results displayed include the MSE, MAE, and R^2 score for each model with each method. The best method, which we determined was the *k*-nearest neighbour method with EWMA tuned real data, is highlighted in grey.

B. Model Finalization

As we have selected our best performing model from the sanity tests, we will now perform hyperparameter tuning as a form of model regularization. Hyperparameter tuning is the act

of selecting the most optimal parameters for a given model. For a *k*-nearest neighbour model, we decided that we must determine the optimal number of neighbours, the optimal power parameter, and optimal metric. Additionally, we wanted to determine if using a combination of both synthetic and real data would allow for more accurate predictions in our model as well as improve the testing and training time required. The synthetic data we chose to use in our mixed dataset included *U*, the v*oltage magnitude*, and *I*, the *current magnitude*, as discussed in Table 1. *U* and *I* replaced the *i_d*, *i_q*, *u_d*, and *u_q* that were initially in the input set of variables. These derived features were combined with the other original *coolant*, *motor_speed*, *ambient*, and *torque* variables to reproduce a dataset with decreased number of inputs that improves speed up as well as performance.

Since our dataset was quite large, we first performed an analysis of the MSE, MAE, and R^2 on a baseline *k*-NN model to determine what number of neighbours performed the best in Fig. 16.

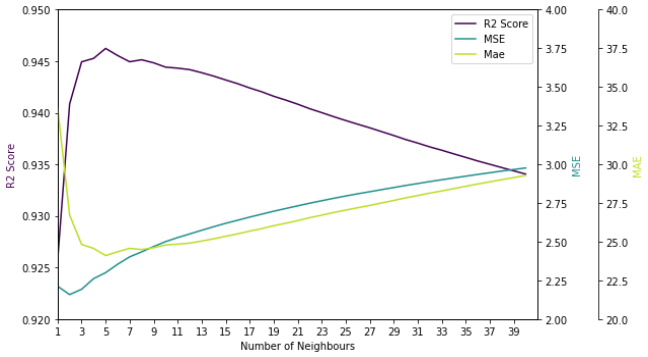

Fig. 16. Mapping the performance of a *k*-NN model with the *n_neighbors* ranging from 1 neighbour to 40 neighbors.

The reason for this was to improve the time it took to perform the cross validation search. Our parameters we were testing on the *k*-NN model to gauge performance were *n_neighbors*, *p*, and *metric*. For *n_neighbors*, since we performed that first step of mapping out the first 40 neighbour's performance, we can choose a small subset of values based on the graph in Fig. 16. For the cross validation steps, we compared 2 folds, model fits, of each combination against each other in an exhaustive search function provided by the scikit *GridSearchCV* function. The *GridSearchCV* will compute a fit and store a score of each parameter combination and upon completion, we would be able to see the parameters that will provide the best performance of our model. The cross-validated grid search was performed on the following parameters:

- *n_neighbors* : [1, 2, 3, 4, 5, 6]
- *p* : [1, 2]
- *metric* : [euclidean, minkowski, manhattan]

This gives a total 36 combinations and when using a *cv* value of 2, we get a total number of fits equal to 72. If we did not perform that preliminary step of finding the optimal number of neighbours, we would have had to run 480 fits which would take a very long time to compute. From performing hyperparameter tuning, it was found that the optimal *n_neighbor* was 2, the optimal *p* was 1, and the optimal *metric* was 'minkowski'. When training and testing our model with the new, optimized parameters, we trained the model with two different sets, one with the mixed data but no EWMA applied, and then one with the same data but applying EWMA with a span of 5. The reason behind a span value of 5 is because as the span of the EWMA function increases, the data becomes farther away from the true structure of itself as shown in Fig. 14 earlier. We decided upon a span of 5 since the model performed better than a span of 4 from the initial baseline tests and the integrity of the data remained very similar to before. The model provided the following results in Table III.

TABLE III. RESULTS OF HYPERPARAMETER TUNED MODEL

	With no EWMA		With EWMA	
Model	**MSE**	**MAE**	**MSE**	**MAE**
k-Nearest Neighbour	26.71741	2.15793	**13.92034**	**1.29990**

Fig. 17 shows the predicted values vs the original values of the data from running our model on the dataset with the EWMA applied to it.

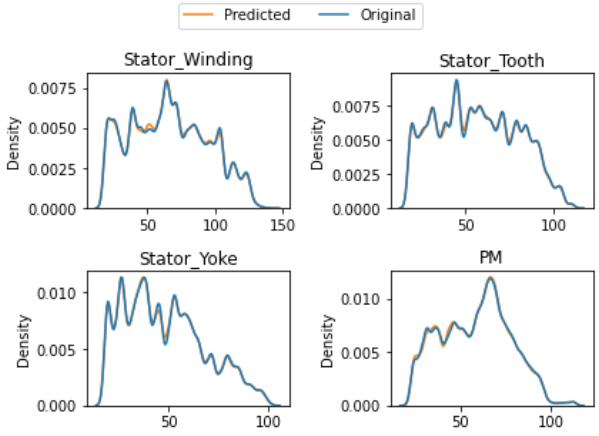

Fig. 17. *k*-NN model that has been hyperparameter tuned and trained on data that has had EWMA applied to it.

V. DISCUSSION & CONCLUSION

It can be understood that when provided a substantially large dataset, a well developed model can be deployed that can assist engineers in predictive analysis of the temperature variables inside of permanent magnet synchronous motors. In our sanity testing, the *k*-NN model trained using real data with removing outliers and applying EWMA was the most accurate model when training all four models with the same parameters and datasets. As such, we performed hyperparameter tuning to

find the most optimal set of parameters for the model, and developed an improved version of the original model. Applying EWMA functions allowed for smoothing to help create better predictive models. Data preprocessing methods such as synthetic attribute derivation, attribute conditioning, and data normalization allow for a much greater accuracy in the training of the models.

In Table IV, the mean average performance of three of our models: *k*-NN, linear regression, and random forest regression, when tested on profile id no. 65 and 72 are displayed. Table IV shows that linear regression performs better on these profiles despite the *k*-NN model being our best performing model in our sanity testing. In our sanity testing, we did attempt to predict all four temperature values at the same time unlike here where we were predicting each individual value.

TABLE IV. PROFILE ID 65 AND 72 TESTING RESULTS

	Mean Average Performance (MAp) - Profile 65		
Model	**MSE**	**MAE**	**R^2**
k-Nearest Neighbour	39.11932	3.96	0.8507
Linear Regression	9.62675	1.74425	0.96125
Random Forest	30.91025	2.851	0.87
	Mean Average Performance (MAp) - Profile 72		
Model	**MSE**	**MAE**	**R^2**
k-Nearest Neighbour	64.7772	4.967	0.38375
Linear Regression	18.2075	2.593	0.8155
Random Forest	37.2495	3.20375	0.61675

VI. FUTURE WORK

Due to the limitations associated with the built in scikit library functions, we were not able to properly implement time series analysis. Specifically, splitting the dataset into sequences and samples that would allow the model to better predict the temperatures since it could use the previous samples as reference too. We did attempt to make this work with the KNearestRegressor but since it could not handle a 3-D array, flattening it defeated the purpose and took extremely long to run.

Therefore, for future work, we would have implemented a model not using a built in library like scikit but instead using tensorFlow that allows for us to apply proper regularization and time series sequencing techniques. After developing the models and preprocessing methods, the largest limitation would be computational power for large datasets as well as more complex models such as recurrent neural networks which we were unable to implement.

VII. GOOGLE COLAB LINKS

Exploratory Data Analysis:
https://colab.research.google.com/drive/1_zQx1B8aJT2g_-v1NT_THB-6aWZKKMV3?usp=sharing

Data Curation and Conditioning:
https://colab.research.google.com/drive/1D43J626PSbIxEU-uqH-K_Vmy0ztse79o?usp=sharing

Model Finalization:
https://colab.research.google.com/drive/1Nic4qoW9HnDAW1_oQsh7UcUj1Cz2grVp?usp=sharing

Testing PID 65 and PID 72:
https://colab.research.google.com/drive/1W1zMj07HE8oH2T_OSKwZithGknA8X3jr?usp=sharing

VIII. ACKNOWLEDGMENTS

We greatly appreciate the contributions of Kirgsn (Paderborn University) for providing us with the *Electric Motor Temperature* dataset and additional supporting documentation. In addition, we also appreciate Sci-kit learn for providing thorough documentation on the implementation of their regression models.

IX. REFERENCES

[1] Kirchgassner, Wilhelm, et al. “Deep Residual Convolutional and Recurrent Neural Networks for Temperature Estimation in Permanent Magnet Synchronous Motors.” 2019 IEEE International Electric Machines & Drives Conference (IEMDC), 2019, https://doi.org/10.1109/iemdc.2019.8785109.

[2] Guo, Hai, et al. “Predicting Temperature of Permanent Magnet Synchronous Motor Based on Deep Neural Network.” Energies, vol. 13, no. 18, 2020, p. 4782., https://doi.org/10.3390/en13184782.

[3] Kirchgassner, Wilhelm, et al. “Data-Driven Permanent Magnet Temperature Estimation in Synchronous Motors with Supervised Machine Learning: A Benchmark.” *IEEE Transactions on Energy Conversion*, vol. 36, no. 3, 2021, pp. 2059–2067., https://doi.org/10.1109/tec.2021.3052546.

[4] Le, Ran, and Kaijun He. “[PDF] Motor Temperature Prediction with K-Nearest Neighbors and Convolutional Neural Network: Semantic Scholar.” [PDF] Motor Temperature Prediction with K-Nearest Neighbors and Convolutional Neural Network | Semantic Scholar, 1 Jan. 1970, https://www.semanticscholar.org/paper/Motor-Temperature-Prediction-with-K-Nearest-and-Le-He/cbf8e666a666c95b76318b4b1e9376c4f439064b.

[5] T. Yidirim and H. K. Cigizoglu, "Comparison of generalized regression neural network and MLP performances on hydrologic data forecasting," Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02., 2002, pp. 2488-2491 vol.5, doi: 10.1109/ICONIP.2002.1201942.

[6] B. Sravani and M. M. Bala, "Prediction of Student Performance Using Linear Regression," 2020 International Conference for Emerging Technology (INCET), 2020, pp. 1-5, doi: 10.1109/INCET49848.2020.9154067.

[7] Kirgsn. “Electric Motor Temperature.” *Kaggle*, 26 Apr. 2021, https://www.kaggle.com/wkirgsn/electric-motor-temperature.

[8] Gholamy, Afshin; Kreinovich, Vladik; and Kosheleva, Olga, "Why 70/30 or 80/20 Relation Between Training and Testing Sets: A

Pedagogical Explanation" (2018). Departmental Technical Reports (CS). 1209.

[9] R. K. Pearson, “Outliers in process modeling and identification,” *IEEE Xplore*, 7-Aug-2002. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/974338?casa_token=lL9X0EkLe8cAAAAA%3AXC_5nb5l9jxWQOS2NbEm5_sM9YZ31zkeZk2TwPfGQarK2DSQnyjer-hvQzZejAekXcKBmDqTcQ. [Accessed: 08-Mar-2022].

[10] Y. Song, J. Liang, J. Lu, and X. Zhao, “An efficient instance selection algorithm for K nearest neighbor regression,” *ScienceDirect*, 13-Apr-2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217306884?casa_token=KsTaEogaF_sAAAAA%3A0d7BGye0eOgCTGWsw8yVmDwt1HYMTrSuk557mniGgbzK1A71zb8wIsZykMgQiGxH0ZTGVkO_ow. [Accessed: 08-Mar-2022].

[11] Y.-chen Wu and J.-wen Feng, “Development and application of Artificial Neural Network - Wireless Personal Communications,” *SpringerLink*, 30-Dec-2017. [Online]. Available: https://link.springer.com/article/10.1007/s11277-017-5224-x. [Accessed: 08-Mar-2022].

[12] M. Mailagaha Kumbure and P. Luukka, “A generalized fuzzy k-nearest neighbor regression model based on Minkowski distance - granular computing,” *SpringerLink*, 25-Sep-2021. [Online]. Available: https://link.springer.com/article/10.1007/s41066-021-00288-w#citeas. [Accessed: 20-Mar-2022].