Electric Motor Temperature

Mohit Jaju, Prachit Puranik, Adam Gard, Benjamin Yahle, Gongshun Wang

Table of Contents

EXECUTIVE SUMMARY

The permanent-magnet synchronous machine (PMSM) is an intricate sensor utilized in motion control applications and monitored via infrared sensor guns. The Paderborn University in Germany has been studying temperature regulation of these sensors in an effort to reduce sensor malfunctions. The dataset was acquired from Kaggle.com with the previously expressed goal of temperature prediction derived from eleven PMSM variables. Both machine learning and deep learning techniques were applied to this problem-set. The machine learning techniques included linear regression and random forest regression.

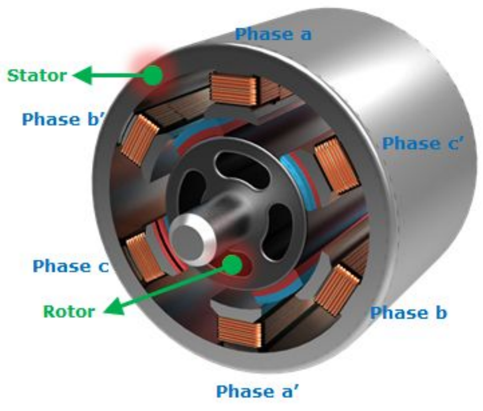
The linear regression returned a mean square error (MSE) of 0.1971 and adjusted R^2 (adj R^2) of 0.8000 on the test set, both results provided a promising baseline for future models. Random forest regression provided the best results of all the models with a MSE of 0.0002 and adj R^2 of 0.9998 with a runtime of less than 20 minutes, making it the team's proposed candidate to monitor the temperature of the PMSMs. Support vector regression machine learning was determined to be the least valuable model due to it's computing time requirements.

The long short-term memory deep learning models had differing epoch sizes (*epoch* = 5, 2, 5) and batch sizes (*batch_size* = 1024, 2000, 512). These LSTM models were never able to realize a MSE less than 0.0370 or accuracy greater than 0.9600 while also requiring high computing power for long periods of time, with the quickest epoch taking 1.78 hours to complete. Of the models tested, the random forest machine learning model provided the best results in the second fastest amount of time. This accuracy and speed makes it the most practical for field applications and should be deeply considered as the right choice for temperature determination of the PMSM sensors.

INTRODUCTION AND BACKGROUND

Introduction

The permanent-magnet synchronous machine (PMSM) drive is one of best choices for a full range of motion control applications. In order to monitor the temperature of these sensors while in operation, the existing method uses infrared sensor guns. These sensors are expensive and require extensive manual labor. Functional safety is an important concern that we want to address by training a model that will estimate the temperature of the rotor. Presently, overheating leads to increased power consumption, part malfunctions and inventory loss. The challenging points occur from the torque variations as well as external physical factors that inhibit maximum output while increasing motor wear. In an effort to realize potential peak efficiency and reduce the possibility of overheating, machine learning and deep learning models were developed and incorporated. we are going to use deep learning: LSTM Model, and Machine learning: support vector model, random forest regressor, linear regression.



The permanent-magnet **synchronous machine** (PMSM) drive is one of best choices for a full range of **motion control applications**. For example, the PMSM is widely used in **robotics**, machine tools, actuators, and it is being considered in high-power applications such as industrial drives and vehicular propulsion.

Background

Permanent-magnet synchronous machine (PMSM), A permanent magnet synchronous machine is a generator where the excitation field is provided by a permanent magnet instead of a coil. The term synchronous refers here to the fact that the rotor and magnetic field rotate with the same speed, because the magnetic field is generated through a shaft mounted permanent magnet mechanism and current is induced into the stationary armature.Long short-term memory (LSTM) is an artificial RNN architecture used in the field of deep learning. The main purpose is to eliminate the traditional method of recording temperature with infrared guns.

DATA AND METHODOLOGY

Data Description

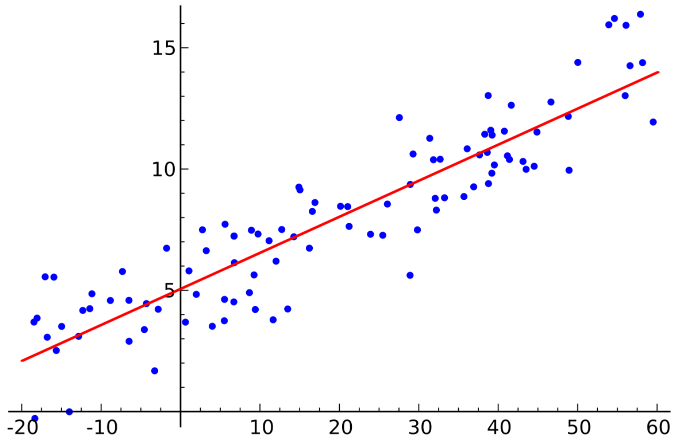
The data is a 140 hours recording from a permanent magnet synchronous motor (PMSM).
The dataset utilized in this project is composed of data collected from a permanent magnet synchronous motor (PMSM) that was operated on a test bench by researchers at Paderborn University in Germany. It is a large comma separated value (csv) file containing measurement sessions with multiple metrics per session. The measurement sessions (rows) represent a snapshot of sensor data at a specific time step. Each step/row is represented by a unique profile ID. There are 13 different metrics captured including…

Ambient temperature, Coolant temperature, Voltage D-Component, Voltage Q-Component, Motor Speed, Torque, Current D-Component, Current Q-Component, Permanent Magnet Surface Temperature, Stator Yoke temperature, Stator Tooth temperature, Stator Winding temperature, and Profile ID.

Methodology

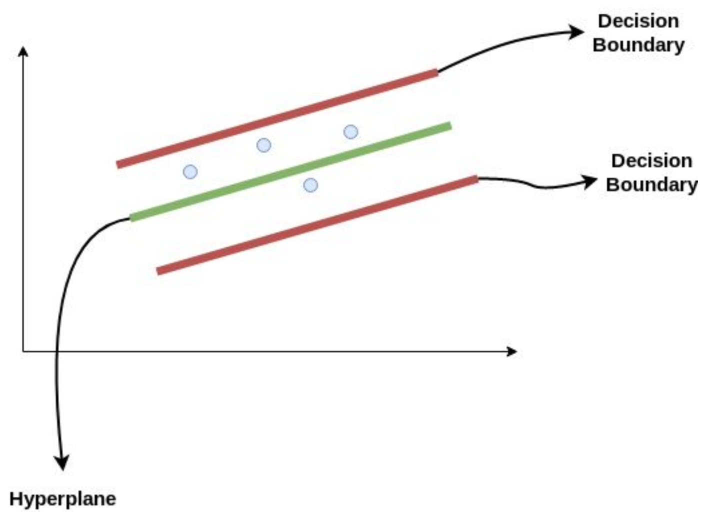
Pre-Processing-
These models were developed from the baseline dataset provided by the Paderborn University in Germany. Data transformation was conducted in such a way that the same temporary data could be used for both machine learning and deep learning models. Initially, the data was segregated by its *profile_id* so that it could be further separated into feature sets which resulted in a numpy array (*seg_data*). Next, the data was cleaned so that *temp* could be turned into a relational value with 11 dimensions and 60 vectorized values. Both the *train_x* and *train_y* were transformed in the same manner, resulting in a dimensional shape of (994951, 60, 11) and (994951,) respectively. Next, the *train_x* array was randomly distributed and the resulting values were used to set the new value order within the arrays. The data was then split into a 70/20/10 (training/validation/test) ratio for both *x* and *y*. Finally, the dimensionality of *x* was reduced across the training/validation/test from three dimensions to two in preparation for the machine learning and deep learning models.

Machine Learning Models
Multiple machine learning models were used to determine their viability in assisting with temperature monitoring and detection. As mentioned earlier, the three models used were Linear Regression, Support Vector Regression (SVR) and Random Forest Regression (RF). Each model had unique specific tuning parameters but were all tested against the *x/y_train/test/val_ml* datasets.



Linear Regression (Exhibit 1)
We did not expect great results from this model as it only finds the best line fit for the data and fits best only on linear data.

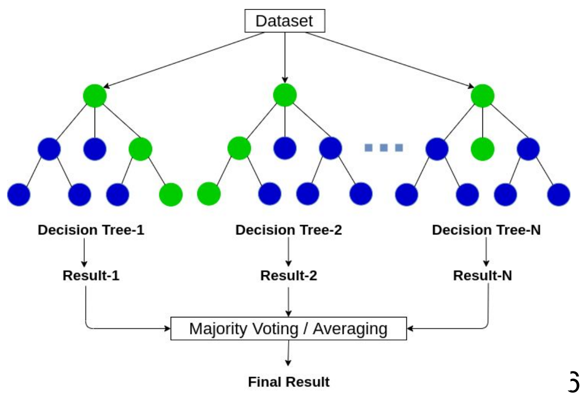
This is the most basic of all the models we used. However, this cannot be implemented on an industrial level as it can be inaccurate enough to be hazardous for safely levels



Support Vector Regression (SVR) (Exhibit 2)

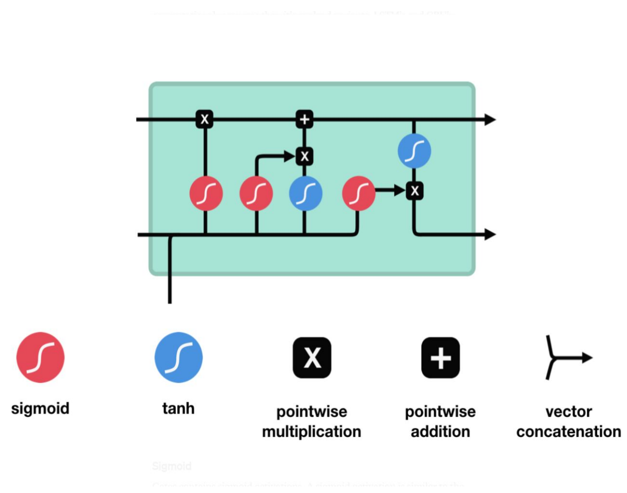
SVR considers the most fitting line in the data plot that has the least error from the data points. It is moderately complex and gives an average result in contrast with the other models run in this study.
The SVR also takes the longest processing time out of all the models and thus is not the best one for this type of data.

Random Forest Regression (RF) (Exhibit 3)
One of the most complex models in Machine Learning is the random Forest model and it gave us the best results out of all the algorithms we tried on this model. We achieved a testing r-square value of 0.994

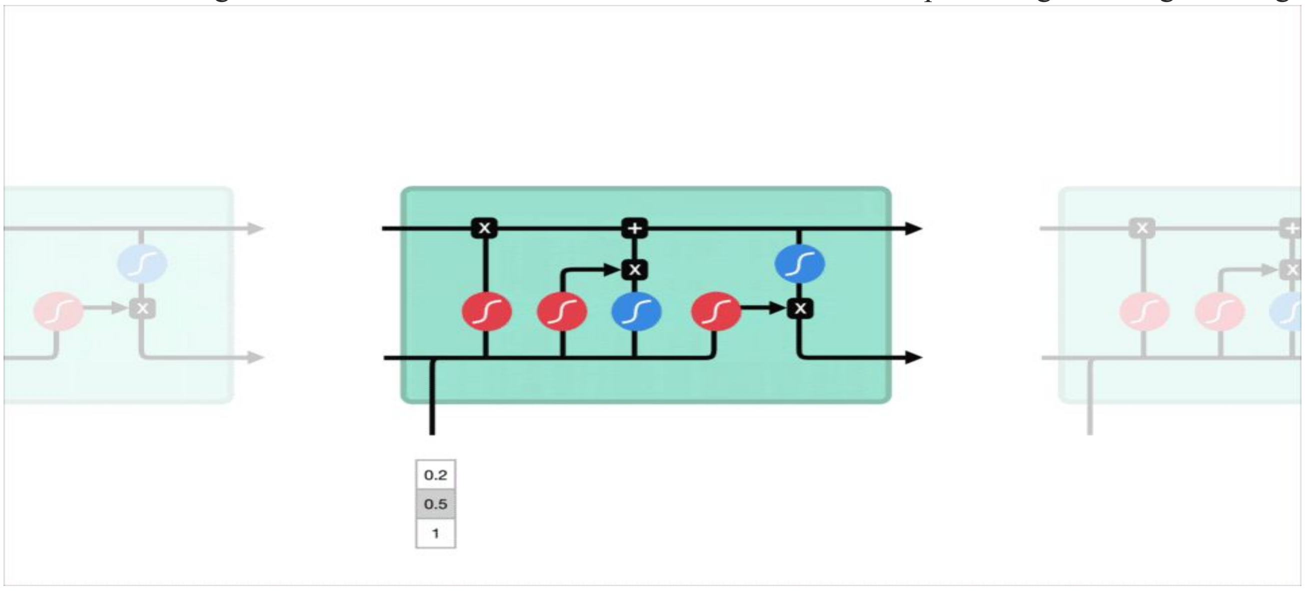


Deep Learning Models : Long Short-Term Memory Networks for Temperature Prediction

[Source]



The core concepts of LSTM models are the cell states, and it's various gates. The cell state acts as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.



All deep learning models performed were LSTM. The LSTM method is a robust recurrent neural network program that utilizes memory embedded decision nodes to assist in the training process. Model optimization was conducted via the adaptive movement estimation (Adam) algorithm to manage the "exponential moving average of the gradient and the square of the gradient" (*Adam*), the learning rate is set at *learning_rate = 0.0005*. The models criterions are set to determine the mean squared error loss (*MSELoss*) between the input *x* and target value *y*. The network itself utilizes 3 hidden layers with 512 hidden nodes each directed from an input size of 11 values (*nlayers* = 3, *hidden_size* = 512, *in_size* = 11). Within the LSTM model, the hidden layers become vectorized, resulting in half the hidden nodes in *t+1* (*hidden_size/2* = 256). Each model had unique specific tuning parameters but were all tested against the *x/y_train/test/val_ml* datasets.These parameters are held constant throughout all iterations of the LSTM model.

Base Parameters: *nlayers* = 3, *hidden_size* = 512, *hidden_size/2* = 256, *in_size* = 11, *criterion* = MSELoss, *optimization* = Adam.

LSTM Model 1:

The initial model utilizes large batch sizes of 1024 (*batch_size = 1024*), which is approximately $1.02\text{x}10^{-3}$% of the training dataset's size, resulting in 2,721 batch cycles within an epoch. Training and validation batch sizes were both set at 128 (*training batch size* = 128, *validation batch size* = 128). The large batch sizes couldn't be accommodated by Google Colab's capabilities and therefore we used an external GPU of 24 gb, but took nearly 8647s (Exhibit 3) to run. Initially the model was scheduled to run for 10 epochs (*epoch = 10*) this gave us the best results in the LSTM models that we ran. The train loss is as low as 0.0229 while the r2 score was as good as 0.976 - 97.6% Ultimately, the model performed better with the test dataset than it did with either the validation or training datasets, realizing a test loss of 0.037 and test adj R^2 score of 0.960 (Exhibit 4).

<u>Model Parameters</u>: *batch_size* = 1024, *tr_batch_size* = 128, *val_batch_size* = 128, *epochs* = 5

LSTM Model 2: (Change desc)

The initial model utilizes comparatively large batch sizes of 2000 (*batch_size = 2000*), which is approximately $2.87\text{x}10^{-3}$% of the training dataset's size, resulting in 349 batch cycles within an epoch. Training and validation batch sizes were both set at 1000 (*tr_batch_size* = 1000, *val_batch_size* = 1000). This change occurred due to the previous issues with the run-time issues experienced with *Model 1*. The batch sizes were determined by the max available GPU usage provided by Google's Colab to a single user. While the batch sizes were nearly ten times larger than *Model 1*, it took 6412s (Exhibit 5) to run the first epoch, resulting in a time savings of nearly 37 minutes. Only two epochs were scheduled to run with this model (*epochs* = 2) and both were completed. This model provides improved values over *Model 1* in both validation loss (0.0814) and validation adj R^2 score (0.9167) while also finishing the epoch faster (Exhibit 5). The second epoch also finished at nearly the same speed (6413s) while reducing validation loss by approximately .0084 (0.0730) and improving the validation adj R^2 score by .0087 (0.9253) (Exhibit 5). Ultimately, the model performed better with the test dataset than it did with either the validation or training datasets, realizing a test loss of 0.0726 and test adj R^2 score of 0.9258 (Exhibit 5).

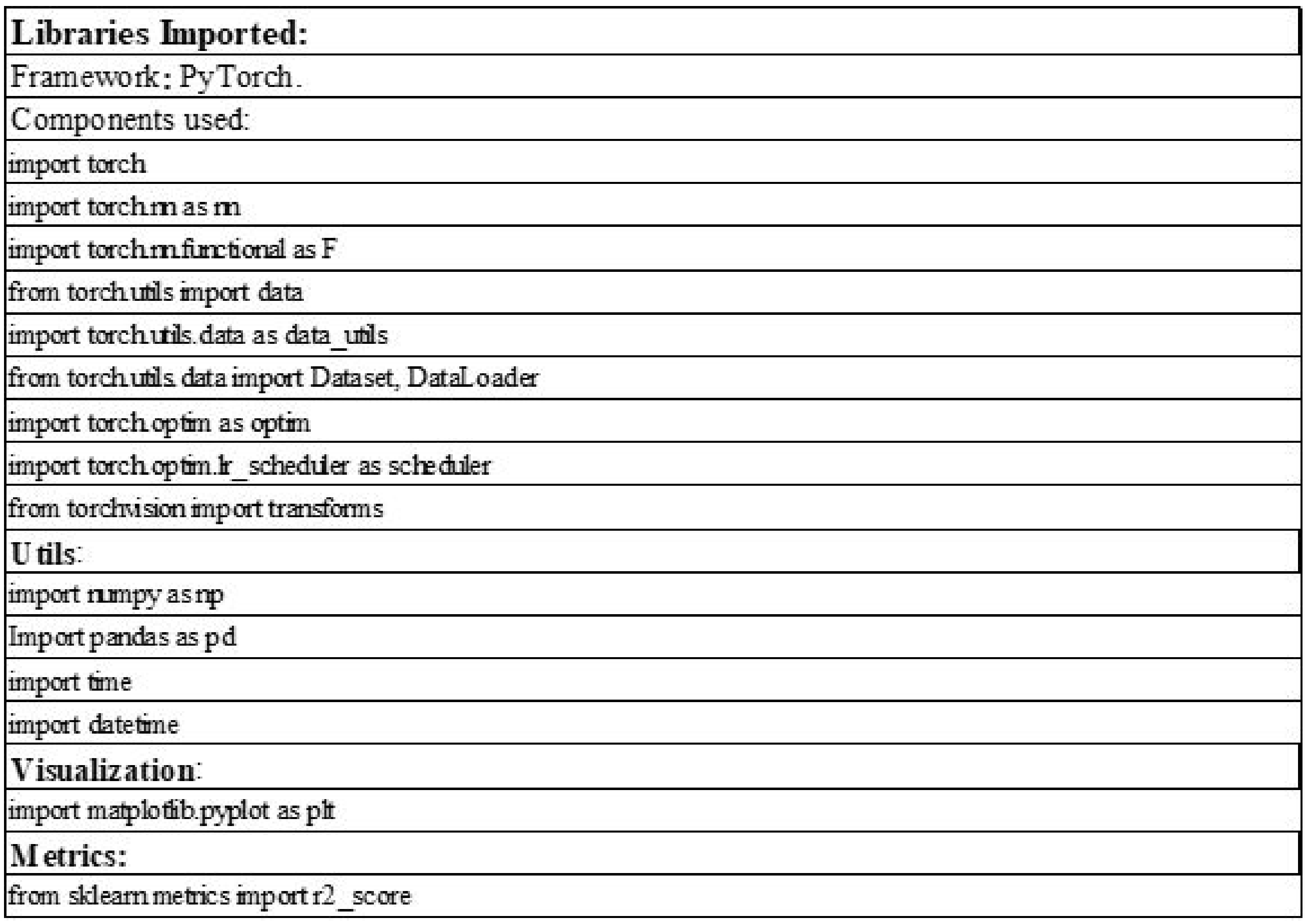
<u>Model Parameters</u>: *batch_size* = 2000, *tr_batch_size* = 1000, *val_batch_size* = 1000, *epochs* = 2

LSTM Model 3:

This was the initial model with a batch size of 512 (*batch_size = 512*), which is approximately $5.12\text{x}10^{-4}$% of the training dataset's size.
Training and validation batch sizes were both set at 128(*tr_batch_size* = 128, *val_batch_size* = 128). While the batch sizes were nearly ten times larger than *Model 1*, it took 6412s (Exhibit 4) to run the first epoch, resulting in a time savings of nearly 37 minutes. Only 5 epochs were scheduled to run with this model (*epochs* = 5) and all were completed. This model provides improved values over *Model 1* in both validation loss (0.038) and validation adj R^2 score (0.0.9607) while also finishing the epoch faster (Exhibit 6). Ultimately, the model performed better with the test dataset than it did with either the validation or training datasets, realizing a test loss of 0.037 and test adj R^2 score of 0.959 (Exhibit 7).

<u>Model Parameters</u>: *batch_size* = 512, *tr_batch_size* = 128, *val_batch_size* = 128, *epochs* = 5

LSTM Neural Network Architectures and dependencies :



Libraries Imported:
Framework: PyTorch.
Components used:
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils import data
import torch.utils.data as data_utils
from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
import torch.optim.lr_scheduler as scheduler
from torchvision import transforms
Utils:
import numpy as np
Import pandas as pd
import time
import datetime
Visualization:
import matplotlib.pyplot as plt
Metrics:
from sklearn.metrics import r2_score

RESULTS

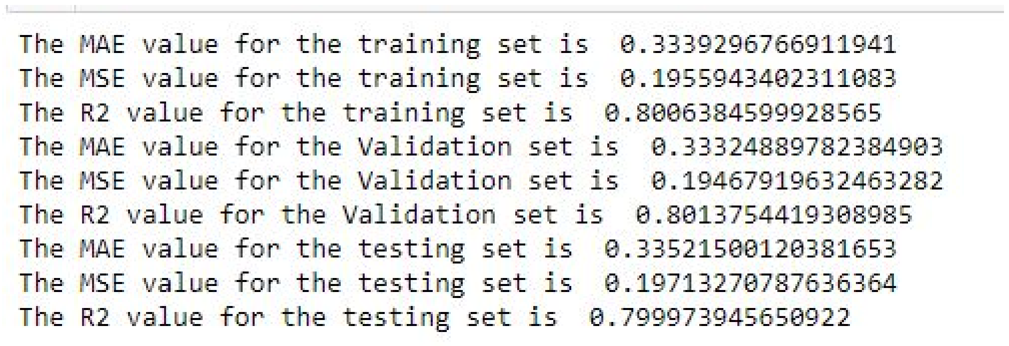
We were able to achieve similar results as that of the pre-trained models. Our model used lesser inference time due to lesser parameters used. We fine tuned our parameters such as kernel size, stride, padding, number of neurons in each layer using hit and trial method to achieve high accuracy.

Model Name	Training Accuracy	Validation Accuracy	Testing Accuracy
Linear regression	0.800	0.801	0.799
SVR	0.877	0.861	0.860
Random Forest	0.999	0.998	0.998
LSTM Model 1	0.960	0.960	0.960
LSTM Model 3	0.952	0.967	0.959

Part 1: Predefined Model

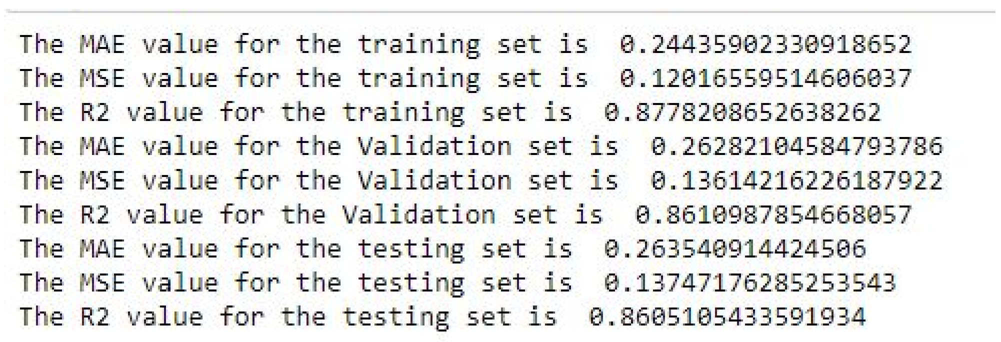
Exhibits :

Exhibit 1 : Linear regression Stats



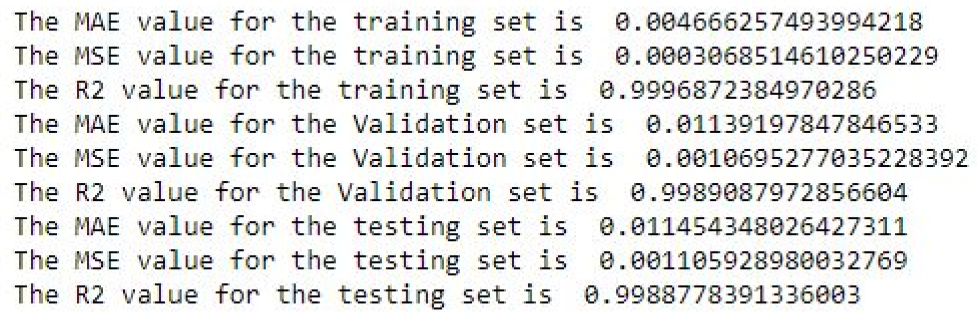
```
The MAE value for the training set is  0.3339296766911941
The MSE value for the training set is  0.1955943402311083
The R2 value for the training set is  0.8006384599928565
The MAE value for the Validation set is  0.33324889782384903
The MSE value for the Validation set is  0.19467919632463282
The R2 value for the Validation set is  0.8013754419308985
The MAE value for the testing set is  0.33521500120381653
The MSE value for the testing set is  0.19713270787636364
The R2 value for the testing set is  0.799973945650922
```

Exhibit 2: SVR Stats



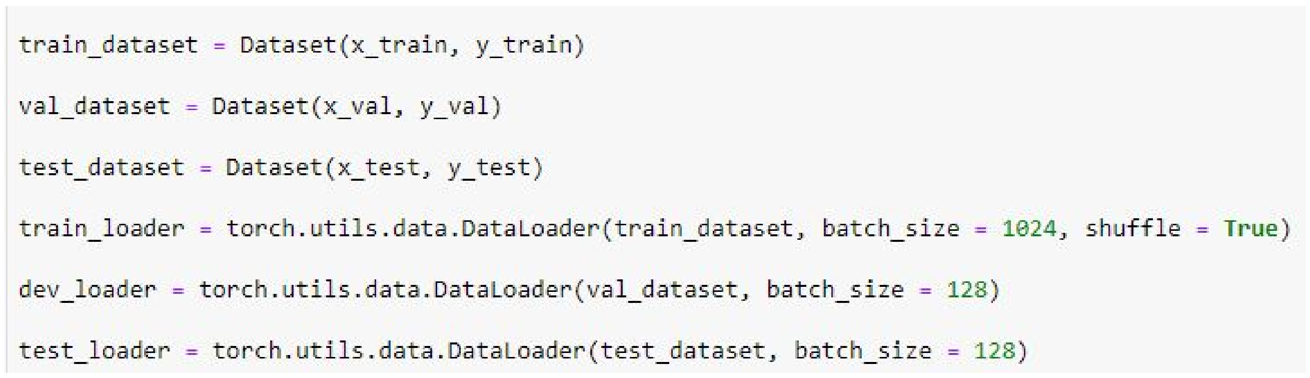
```
The MAE value for the training set is  0.24435902330918652
The MSE value for the training set is  0.12016559514606037
The R2 value for the training set is  0.8778208652638262
The MAE value for the Validation set is  0.26282104584793786
The MSE value for the Validation set is  0.13614216226187922
The R2 value for the Validation set is  0.8610987854668057
The MAE value for the testing set is  0.263540914424506
The MSE value for the testing set is  0.13747176285253543
The R2 value for the testing set is  0.8605105433591934
```

Exhibit 3: Random Forest Regressor Stats



```
The MAE value for the training set is  0.004666257493994218
The MSE value for the training set is  0.0003068514610250229
The R2 value for the training set is  0.9996872384970286
The MAE value for the Validation set is  0.01139197847846533
The MSE value for the Validation set is  0.0010695277035228392
The R2 value for the Validation set is  0.9989087972856604
The MAE value for the testing set is  0.011454348026427311
The MSE value for the testing set is  0.001105928980032769
The R2 value for the testing set is  0.9988778391336003
```

Exhibit 4: LSTM Model 1

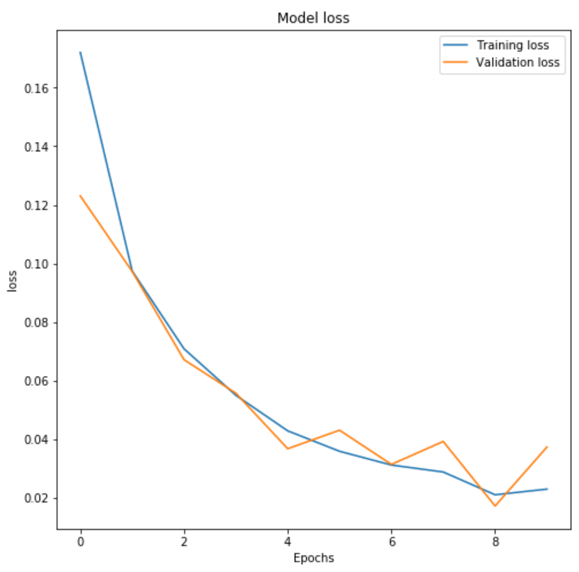


```
train_dataset = Dataset(x_train, y_train)

val_dataset = Dataset(x_val, y_val)

test_dataset = Dataset(x_test, y_test)

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size = 1024, shuffle = True)

dev_loader = torch.utils.data.DataLoader(val_dataset, batch_size = 128)

test_loader = torch.utils.data.DataLoader(test_dataset, batch_size = 128)
```

Model Performance (LSTM Model 1 with Batch Size 1024)

Accuracy and Loss Metrics:

Epoch Number	Training Loss	Training Score (r-square in %)	Validation Loss	Validation Score (r-square in %)	Test Loss	Testing Accuracy
0	0.171	82.49	0.123	87.16	0.037	96.06%
1	0.097	90.06	0.0972	89.85		
2	0.0708	92.77	0.0671	92.98		
3	0.0550	94.38	0.0557	94.17		
4	0.0428	95.62	0.0367	96.15		
5	0.0359	96.33	0.0430	95.51		
6	0.0312	96.81	0.0314	96.71		
7	0.0288	97.05	0.0392	95.90		
8	0.0210	97.85	0.0172	98.20		
9	0.0229	97.65	0.0373	96.10		



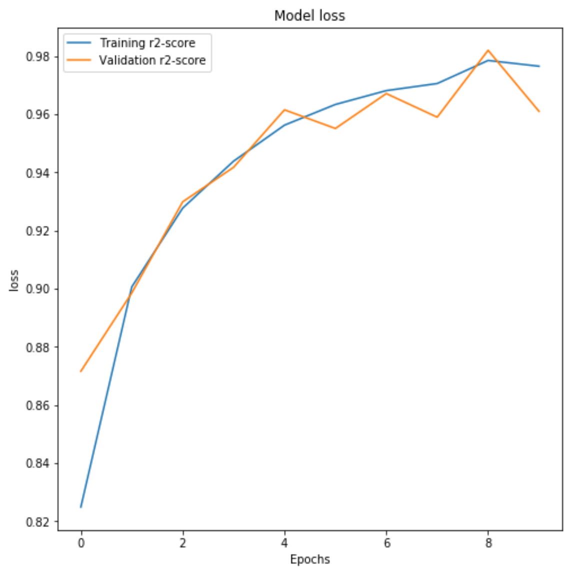
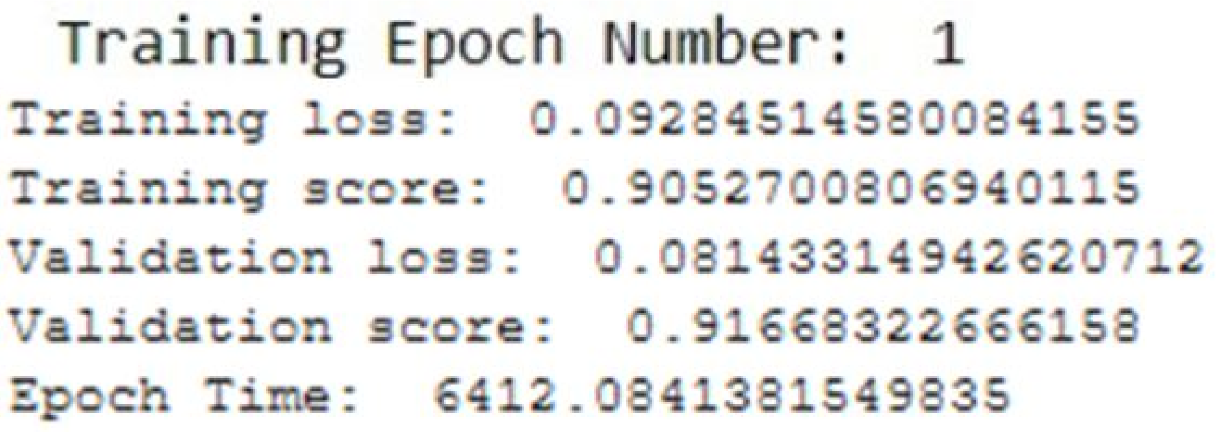
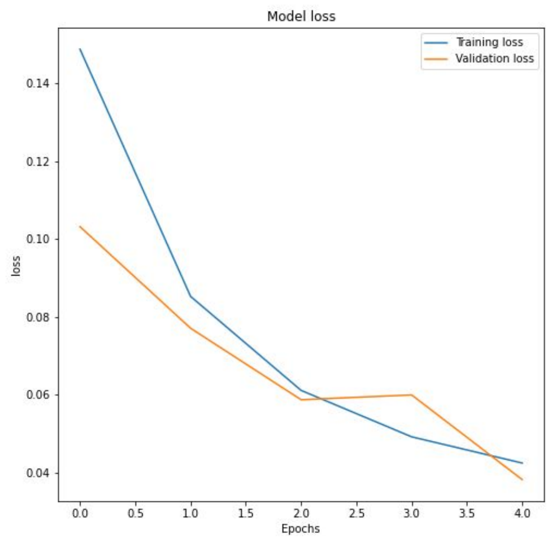


Exhibit 5:LSTM Model 2



```
  Training Epoch Number:  1
Training loss:  0.09284514580084155
Training score:  0.9052700806940115
Validation loss:  0.08143314942620712
Validation score:  0.9166832266158
Epoch Time:  6412.0841381549835
```

Exhibit 6 : LSTM Model 3



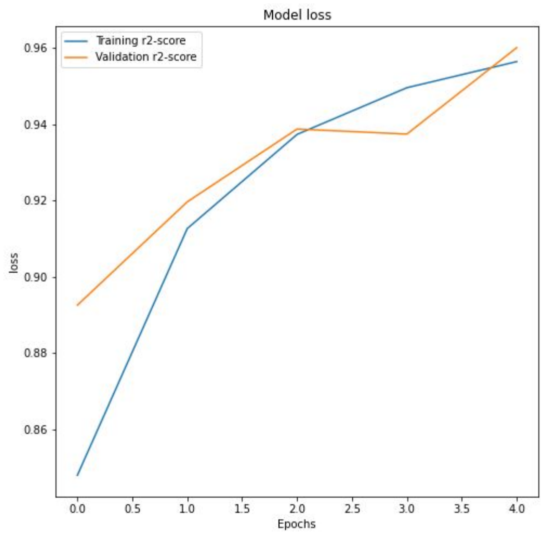
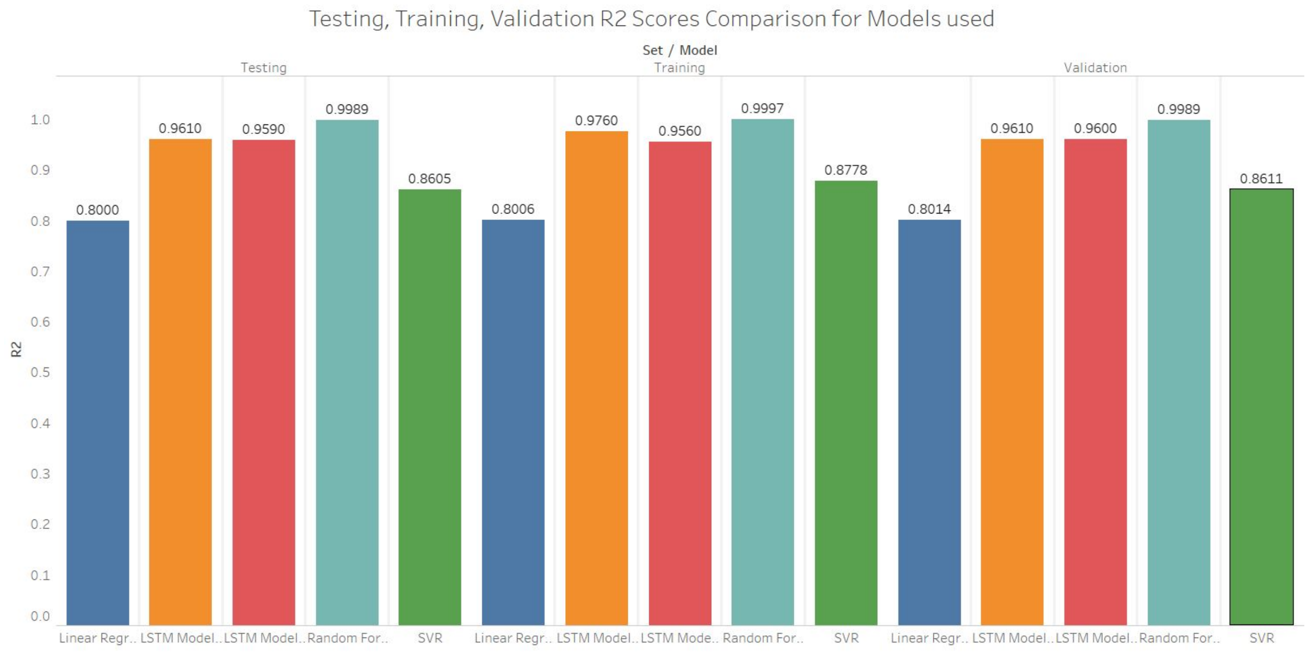
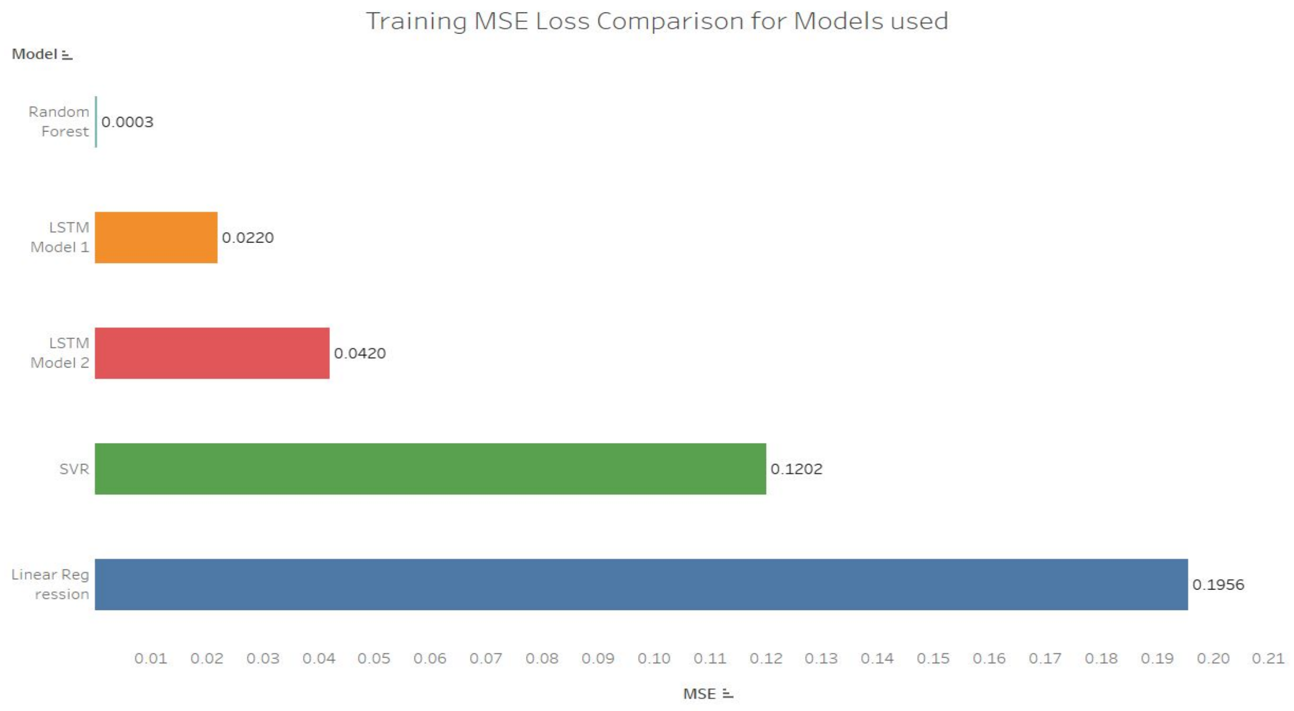
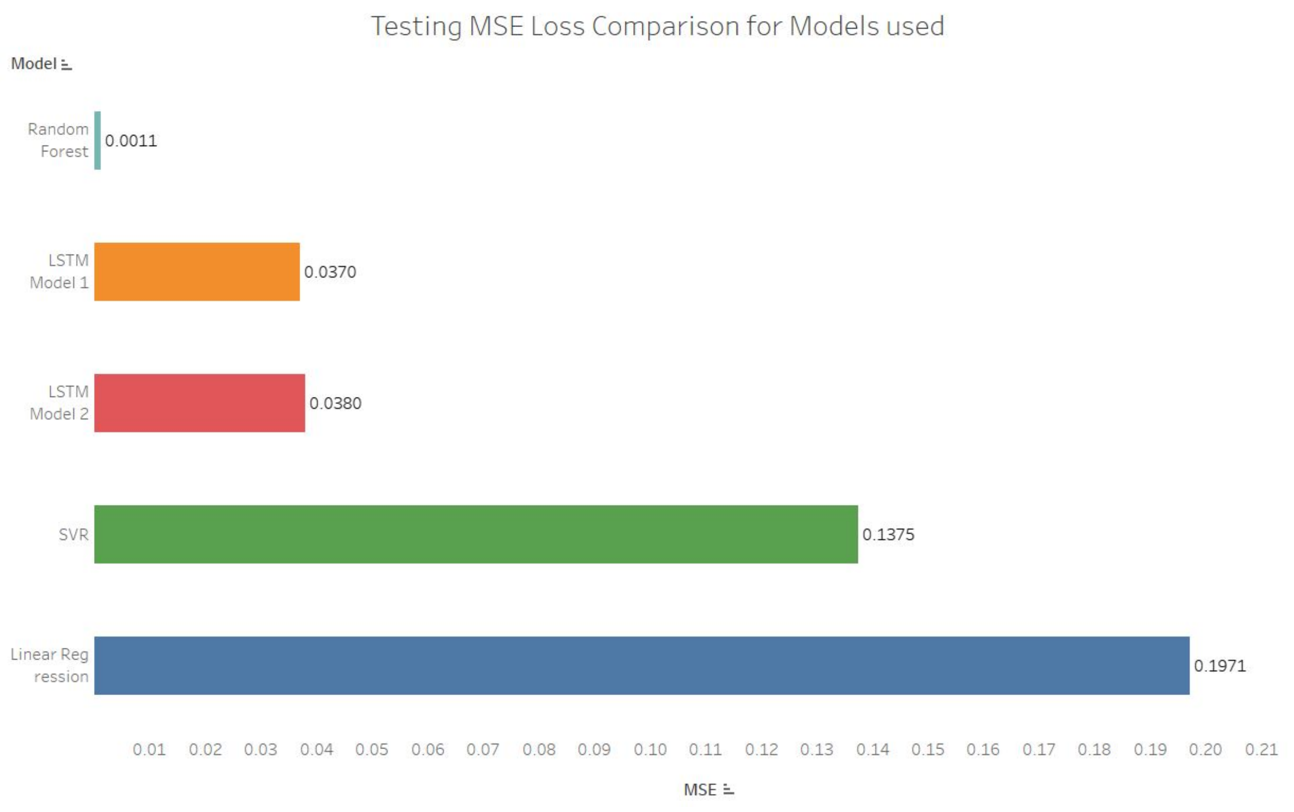
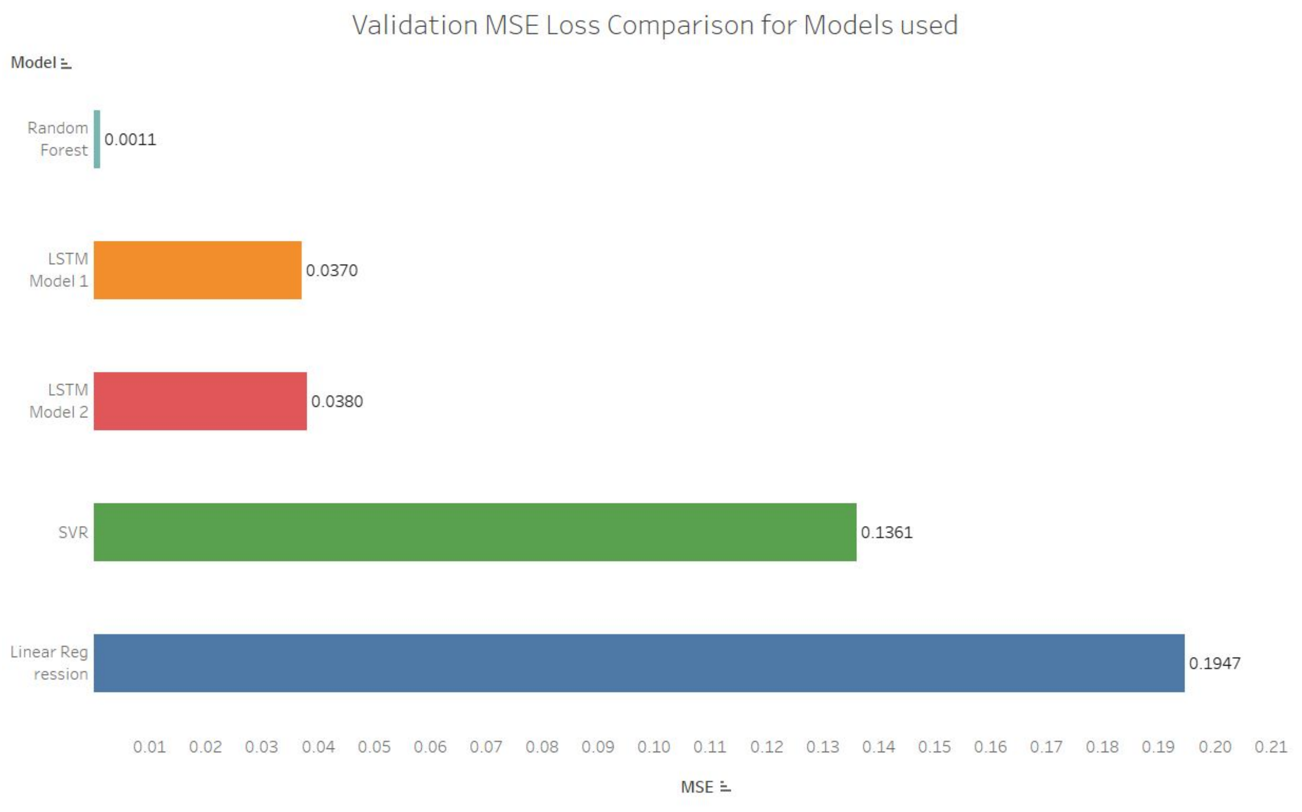


Exhibit 7 : Results , findings and Comparison

The Training, validation and testing R- square values for all the models used are listed in the graph below and are put side by side for comparison.






Testing MSE Loss Comparison for Models used
Model
Random Forest
LSTM Model 1
LSTM Model 2
SVR
Linear Regression
0.0011
0.0370
0.0380
0.1375
0.1971
0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21
MSE

Validation MSE Loss Comparison for Models used
Model
Random Forest
LSTM Model 1
LSTM Model 2
SVR
Linear Regression
0.0011
0.0370
0.0380
0.1361
0.1947
0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14 0.15 0.16 0.17 0.18 0.19 0.20 0.21
MSE

KEY FINDINGS

Machine Learning
While the linear regression model has a comparatively low score adj R^2 when put against more robust models such as Random Forest and LSTM, its value still remains for broad business decisions. When it comes to temperature determination, the model is the least accurate and should not be used. While it could be used for the temperature determination, the Random Forest and LSTM models were more accurate. Which brings us to the Random Forest model, its relatively quick run time (approximately 18 minutes) makes it the second fastest, and its adj R^2 score of 0.9998 makes it extremely accurate. This model has massive potential for projecting the potential failure of a PMSM.

Deep Learning
The deep learning models used to determine the failure rate of the PMSM all returned relatively high adj R^2 scores with none below 0.9006 after their first epoch. Also, the models were consistently found to be slightly better at prediction from the test dataset than the validation and training datasets. Since there is only a slight skew towards the test set we can recognize this fact as a reality of the bias-variance trade-off. Unfortunately, all of the LSTM models took extremely long periods of time to run, with the shortest single epoch taking 1.78 hours (1hr:48min). That being said, these models are the most robust and if given time they will eventually outperform the Random Forest model. As for the PMSM, these models would be useful in determining temperature based failures.

RECOMMENDATIONS AND CONCLUSION

The general value of machine learning is comparatively invaluable when recognizing the relatively low implementation costs. Linear regression is consistently used as a baseline process for present-day business decisions. While it does have some value in temperature determination for the PMSMs, the score of over 80% accuracy is not enough anymore, especially when safety is a concern. The support vector machine was determined to be the least viable of the models due to its long run-time. All of the deep learning LSTM models returned encouraging results, realizing over 90% accuracy after their first epoch and consistent improvement afterwards. The value in these models is that they are able to consistently improve without additional data, but they improve quicker with enormous amounts of data. The PMSMs are able to provide enormous amounts of data, but the time requirements increase in parallel with the data. Currently, the quickest epoch was just slightly under 1.78 hours, this would present a major problem in a manufacturing context. Finally, we have the Random Forest model, it provided over 99.98% accuracy and took under 20 minutes to run on Google's Colab. This model has the agility to determine temperature inconsistencies quickly, easily and accurately, making it the optimal candidate for implementation. (Exhibit 7)

The LSTM model has the highest level of complexity and with increasing batch size, the model gives better output. After epoch 9, the model curve (r-2 score) goes down all of a sudden and similarly in the loss it goes up. This is because the curve has not yet reached the flattening point and with more epochs there is more scope for improvement in the results.

We could run only ten epochs with a batch size of as much as 512 to 2000 for the training set and a maximum number of 10 epochs due to computational restrictions of our local machines as well as Colab.

Surprisingly, Random forest gave exceptionally good results, and the losses went as low as 0.002. Below are the testing losses for all the models run and the random forest model surpasses all. The LSTM model would reach a similar point with a large number of epochs.

CHALLENGES FACED

1) We used Google Collab to run our various image classification models. Initially, we had a training set of approximately a million rows. Due to the shortage of RAM on the Google Colab GPU, we were unable to run our models for the large data set. As a result, we used a 24 gb GPU attached externally to run the same.

2) We had to also do a lot of data processing in order to run the machine learning models and deep learning models. Both of them needed varied formats in their input and so we created a data processing code separately for theML models and combined the data processing code in the model itself for the deep learning model.

3) Initially, our model had poorly tuned parameters. The hyper parameters could be learnt with back propagation but we did use the Scheduler to decide how the model would perform on a range of learning rates.

REFERENCES

[1] https://functionbay.com/documentation/onlinehelp/default.htm#!Documents/pmsmpermanentmagnetsynchronousmachine.htm

[2] https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[3]https://www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/