Predictive Maintenance Using LSTM and GRU on NASA Turbofan Engine Dataset Report

Bonvie Fosam; AndrewID: bfosam

Introduction

The objective of this project is to predict the number of remaining operational cycles before failure and to predict whether or not an engine would fail within a preset number of cycles. This was done using the CMAPSS Jet Engine Simulated Data in which the metrics (operational settings and sensors) of 100 jet engines were tracked over a various number of cycles until failure (or before failure for the test set). The setting and sensor measurements were used as model features, and the output was a binary classification predicting engine failure.

Approach

In addressing this problem set, I began by ensuring that I had all packages required for this project installed. These packages included tensorflow, scikit-learn, and torchinfo. From there, I imported the libraries that would be required for the implementation of my code. The data was loaded from NASA's Open Data Portal. Specifically, the FD001 training, test, and real remaining useful life data was used. The training, test, and real output data were stored respectively in data frames prepared to be manipulated. In processing the data, I first reset the column names of the training and test data frames to be more interpretable. Next, I assessed the data by column and dropped columns with low variance across the engines and cycles because leaving them would increases the complexity of the model without contributing much learning value. Then, I derived the remaining useful life at each cycle for the engines in the training data by extracting the maximum cycle for each engine, subtracting the current cycle from the maximum cycle (named 'RUL'), then merging these differences to the training data set. Using the calculated RUL, I used a binary label named 'label1' to identify whether the specific engine was going to fail within a preset quantity of cycles. Lastly, I normalized all the data excluding the engine ID, the cycle, RUL, and the binary labels. I took a very similar approach in processing the test data, but instead of deriving the RUL from within, I used the real RUL data and merged it with the test data's maximum cycles. Binary labels were also created for the test data.

Once the training data and test data were manipulated this way, it could be reshaped to be more compatible with the LSTM and GRU modeling architectures. This shape was (# samples, time steps, #features). The input data includes the data excluding the dropped columns and the labels. The output data (values to be predicted) are the labels themselves. Both the input and output data were reshaped. Both LSTM and GRU models were built with hyperparameters specified. Both models were evaluated using accuracy as the primary metric. A confusion matrix, and various graphs were also created to aid in drawing conclusions and recommendations.

Methodology:

The specific features I identified as having too low variance were the operational setting 3, sensor measurements 1, 5, 6, 10, 16, 18, and 19. The observations for these features were identical in each sample. It was appropriate to drop these columns because the features would not contribute anything to both models' learning and would make the running the models more computationally expensive. In the end, there were 18 features included in the modeling process.

I used three functions to help reshape the features and the target label. First, generate_sequences() takes a data frame, a preset sequence length, and a list of features and generates a NumPy array of sequences. The function, generate_sequences_for_all_ids(), executes generate_sequences() for each engine ID. This sequence array is used as feature inputs for the model and takes the shape (15731, 50, 18). A similar methodology was used in the third function, generate_labels(), which takes a data frame, a preset sequence length, and list of features and generates a NumPy array of labels representing the target label for the model. The resulting shape is (15731,).

The LSTM model has two layers. I experimented with a model with one layer to be conservative with the complexity, however, I realized that there is a moderate amount of data and ended up with better results when using a two-layer LSTM model. This was suitable in generating a well-trained model. I mitigated my concerns of overfitting my model by adding dropout layers and early stopping. I also used a sigmoid activation function. This was most appropriate because the output is a binary classification. I used 40 epochs to ensure the model had enough time to be trained and the early stopping condition ended the training after 15 epochs. After running the model, I plotted the model accuracy and loss for the training and validation data, calculated precision and recall, and plotted a confusion matrix.

My GRU model was set up similarly to my LSTM model. I used the same rationale and used two layers. The model stopped training after 18 epochs due to the early stopping condition. I plotted the model accuracy and loss for the training and validation data, calculated precision and recall, and plotted a confusion matrix. I was able to take these outputs and access, compare, and contrast the performance of these models, contributing to my overall conclusions and recommendations.

LSTM Model Results

The accuracy score of this model is 98%, the precision is 97%, and the recall is 77%. These metrics decline when accessing a subsection of the data.

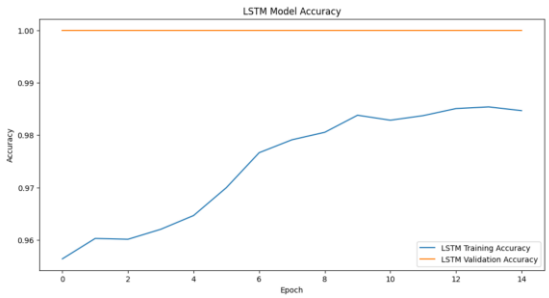

Figure 1: LSTM Model Accuracy

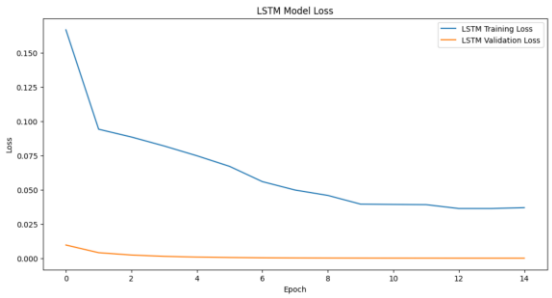

Figure 2: LSTM Model Loss

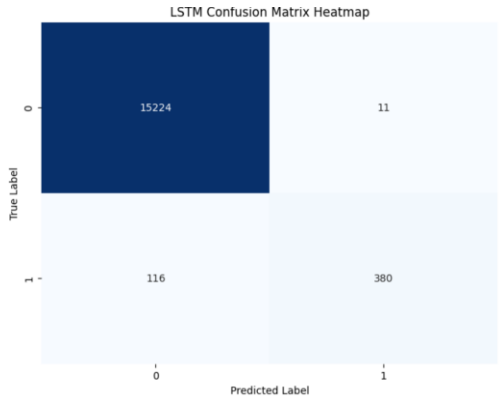

Figure 3: LSTM Confusion Matrix

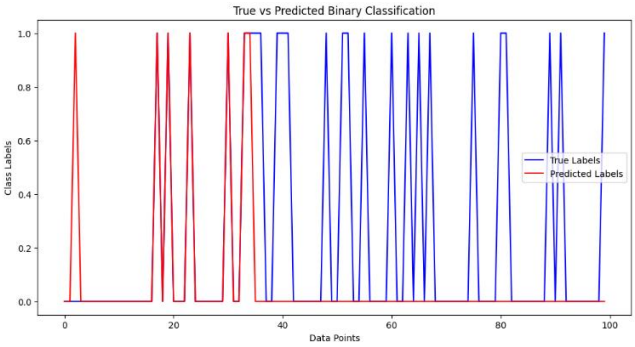

Figure 4: LSTM Classification

	Accuracy	Precision	Recall
LSTM	0.984663	0.971867	0.766129
Template Best Model	0.940000	0.952381	0.800000

Figure 5: Full Data Results

	Accuracy	Precision	Recall	F1-score
LSTM	0.80	0.857143	0.24	0.375000
Template Best Model	0.94	0.952381	0.80	0.869565

Figure 6: Subsection Results

GRU Model Results

The accuracy score of this model is 99%, the precision is 77%, and the recall is 97%. These metrics decline when accessing a subsection of the data.

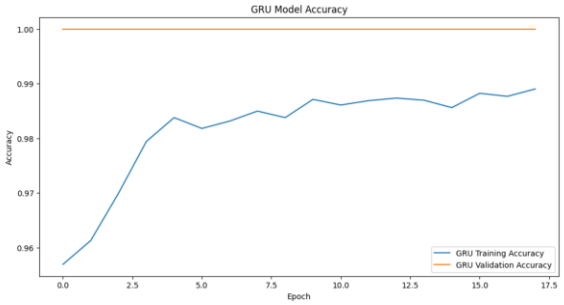

Figure 7: GRU Model Accuracy

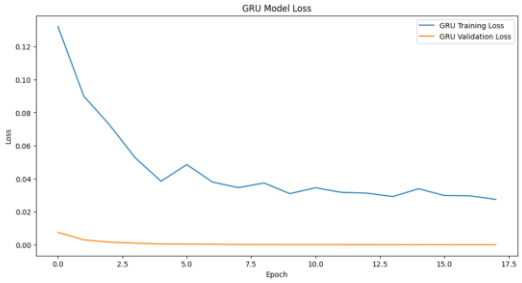

Figure 8: GRU Model Loss

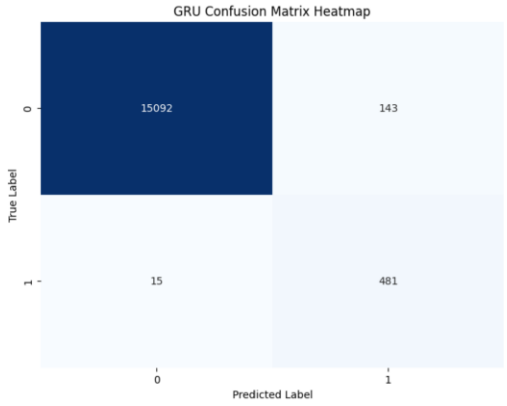

Figure 9: GRU Confusion Matrix

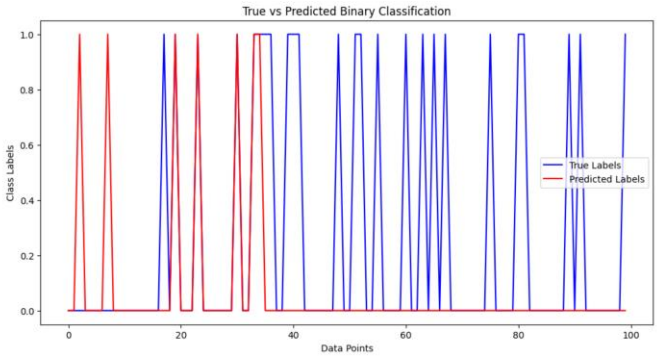

Figure 10: GRU Classification

	Accuracy	Precision	Recall
GRU	0.989034	0.770833	0.969758
Template Best Model	0.940000	0.952381	0.800000

Figure 11: Full Data Results

	Accuracy	Precision	Recall	F1-score
GRU	0.78	0.714286	0.2	0.312500
Template Best Model	0.94	0.952381	0.8	0.869565

Figure 12: Subsection Results

Conclusion/Insights

The results of this project yield some meaningful insights. The first thing that I noticed via the model accuracy plots was that for both models the training accuracy was just under 100% while the validation accuracy was consistently 100%. Although this raises slight concern, I reason that this can be an indicator that both models are not overfitted to the training data and generalize to the validation data. Another point of concern was there was a large decline in recall, or true positive rate, when the model was evaluated on just a validation set. It is quite possible that the specific subsection used in this evaluation was potentially not representative of the whole dataset, meaning that the differences in the distribution of the validation data set was large enough to through off the recall.

In comparing the LSTM and GRU models, I recognize that LSTM can retain more long-term dependencies due to its cell state and can handle more complex data while GRU is more efficient and trains faster but does not handle long term dependencies as well as LSTM. When it comes to predictive maintenance LSTM can be useful because it can remember long term patterns and nuances in the data to yield better results, but it can be argued that the data must be large and complex enough to justify the computational cost and longer training time. On the other hand, GRU is appropriate in times when the data does not require such a robust model. Based on the output of the jet engine models, I argue that the GRU model is better than the LSTM model. This makes sense because the accuracy of the GRU model was higher. Another convincing factor in preferring the GRU model is the false positive and false negative rates. In this case a false positive is saying an engine will fail when it will not, and a false negative is saying an engine will not fail and it does. It is way more detrimental to have a false negative because an engine that should be reliable is not and it can lead to costly losses to the organization. The false negative rate in the GRU model is much

lower, leading me to recommend GRU over LSTM. Lastly, it can be argued that GRU is more appropriate for the data set because it is not overwhelmingly complex, and the sequences are not too long to where a quality model cannot be produced using GRU.