Toumetis Operational Mode Classifier

Data Cleaning

During the data cleaning process, I noticed an issue with the time section of the dataset. Initially, I needed to rename and split the section into several atomic categories. My initial approach was to create a timestamp value that gradually increased for each row, capturing the values for month, year, day, hour, and minute. However, upon further consideration, I realized that machine sensor readings are partially dependent on the weather, which is loosely correlated with the time of day and day of the year. Therefore, I added columns to the dataframe to represent these values.

To handle the target values for the "operational mode", a one-hot encoding was necessary to maintain numeric values for the problem. Additionally, I observed that the sensor values were highly dependent on the machine's asset_id. To address this, I calculated the mean and standard deviation of the sensor values for each machine. For each row in the dataset, I normalized the sensor values by comparing them to sensor statistics relating to the asset. This normalization involved calculating the number of standard deviations that each sensor value deviated from the mean. These are the values that I would plan on feeding into the neural network.

The last task in data cleaning was to eliminate outliers. I simply created a function to determine if any sensor values in a given row were 3 standard deviations away from the mean value for that sensor and asset id. If I were given my time, I would have done a better job at fitting the dataset to a distribution and created a more accurate criteria for identifying outliers.

Statistical Analysis

Outside of data preparation, I performed several tasks to gain a better understanding of the data. Initially, I assumed that the sensor values were related and were all intended to measure the same thing. For each asset, I calculated the average variance of these sensor values and concluded that they were unrelated. I used Matplotlib to visualize asset usage and also created a frequency chart of operational modes across the entire dataset. In addition, I defined a function that listed the amount of time each asset spent in each operational mode. Finally, I implemented a function that listed all the indices for each machine where the mode was changed. I expected this to be helpful for debugging purposes later on.

Limitations

This was by far the largest dataset I had ever worked with. I was forced to practice efficient coding techniques. Despite this, each run of data cleaning took close to 3 minutes, making it challenging to point out problems in my code. Analyzing the data in the statistical analysis phase helped me to understand the dependencies and overall distribution of this data set, but I believe that If I had more time and a faster computer, I would be able to more effectively identify tasks I needed to complete in the data processing section. I tested my data with the normalized sensor values, and concluded that the operational mode classification was much better with these normalized sensor values. There were plenty of dependencies that I couldn't address in my code due to runtime and time constraints. These include the possible dependencies between sensor values and time intervals, and perhaps even the sensor values on each other. Since every run of the neural network took about 30 minutes, I was unable to use trial and error to improve my model.

One of the big things I feel would have greatly improved my model was access to weather data for a current time period. If I was given more time, I believe I could have developed a process to import weather data from an outside source based on the timestamp input, and used the weather as a way to strengthen the effectiveness of my classification.

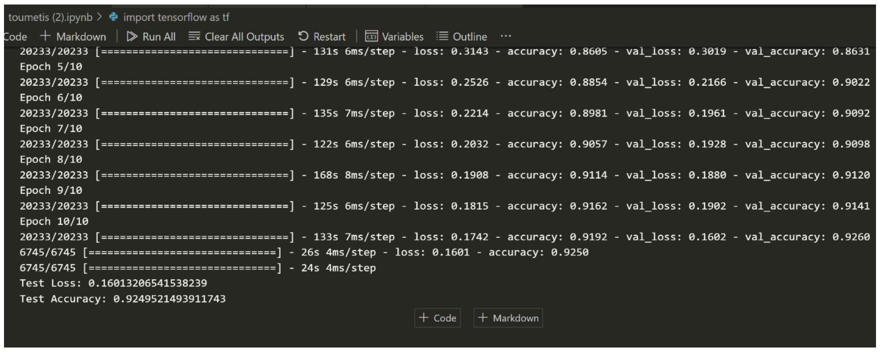
Lastly, I feel as If I could have done a better job, given more time, to classify special events that occurred. Perhaps I could have found a way to effectively identify sensor failures or incorrect values. Given more time for research and testing, I believe I could have found ways to effectively identify anomolies.

Design Choices

After a few hours of research, I concluded that the most fitting machine learning model for this time series problem was a Recurrent Neural Network. I realized that data points related to nearby data points, and the only way to truly capture this was with a Recurrent Neural Network. My neural network was simple, containing only two layers. The input layer was an LSTM. For this particular time series problem, this seemed like the right choice. After fiddling around with the train test split, I ultimately decided to make the training data occupy 40% of the entire set, while the test and validation sets occupied 30% each. Much like this trial and error based decision, I came to the conclusion that the accuracy and loss stopped significantly improving around the 10th epoch. 10 epochs also took around 30 minutes to run, so any more time would have been unrealistic. For the input layer, I used a simple Relu activation function, and a softmax activation function for the output layer. I tried various things in my design to try and improve my accuracy. One of the things I tried was normalizing the sensor values based on the hour of the day. Unfortunately, this only increased the accuracy by 1%. I chose not to include this as an input to my neural network as its small effect on the accuracy leads me to believe that it was somewhat unimportant and could even result in overfitting if more data points were introduced. The greatest design choice that I made was increasing the amount of LSTM units in the beginning layer. The difficulty in finding the right choice was between running the risk of underfitting and overfitting. Ultimately, after trial and error, I landed on the size of 64 which I believe was just enough to avoid overfitting. This decision came from trial and error, as I observed 128 caused a minor change in accuracy and is far more subject to overfitting.

Performance

While I am happy to have designed a functioning neural network for this project, I was somewhat disappointed by the 92.5% accuracy and 0.16 loss I received from my neural network in classifying the operational mode. I believe the main thing causing this poor accuracy was my statistical elimination of outliers. If I was able to do more research on the data and how the sensors truly work, I believe I could have classified and eliminated highly skewed values to better classify the operational mode. See below for the results of my neural network.



```
20233/20233 [==============================] - 131s 6ms/step - loss: 0.3143 - accuracy: 0.8605 - val_loss: 0.3019 - val_accuracy: 0.8631
Epoch 5/10
20233/20233 [==============================] - 129s 6ms/step - loss: 0.2526 - accuracy: 0.8854 - val_loss: 0.2166 - val_accuracy: 0.9022
Epoch 6/10
20233/20233 [==============================] - 135s 7ms/step - loss: 0.2214 - accuracy: 0.8981 - val_loss: 0.1961 - val_accuracy: 0.9092
Epoch 7/10
20233/20233 [==============================] - 122s 6ms/step - loss: 0.2032 - accuracy: 0.9057 - val_loss: 0.1928 - val_accuracy: 0.9098
Epoch 8/10
20233/20233 [==============================] - 168s 8ms/step - loss: 0.1908 - accuracy: 0.9114 - val_loss: 0.1880 - val_accuracy: 0.9120
Epoch 9/10
20233/20233 [==============================] - 125s 6ms/step - loss: 0.1815 - accuracy: 0.9162 - val_loss: 0.1902 - val_accuracy: 0.9141
Epoch 10/10
20233/20233 [==============================] - 133s 7ms/step - loss: 0.1742 - accuracy: 0.9192 - val_loss: 0.1602 - val_accuracy: 0.9260
6745/6745 [==============================] - 26s 4ms/step - loss: 0.1601 - accuracy: 0.9250
6745/6745 [==============================] - 24s 4ms/step
Test Loss: 0.16013206541538239
Test Accuracy: 0.9249521493911743
```