Diesel Particulate Filter Failure Prediction for Vnomics

Matthew Evan Taruno, Daniel He, James Sastrawan, Vatsal Agarwal, Linzan Ye

Team 4, DSC 383W

Goergen Institute for Data Science

University of Rochester

1. Introduction / Motivation

Diesel Particulate Filters (DPF) are essential particulate emission control devices for large vehicles and having a DPF failure is a significant problem that the truck industry is facing. It can be costly, or it can make the truck have a sudden breakdown in the middle of its journey. DPF failure can be caused by the driver's behavior, such as how much the driver is accelerating, shifting, speeding, and idling or the trip context that the truck has, such as the environment, routing, truck configuration, and load capacity it's taking. Hence, our goal is to help Vnomic to build a model to predict the DPF failures in trucks for at least two weeks in advance, so we can help to avoid DPF failures in advance.

2. Dataset

2.1 Dataset Description

Our sponsor provided us with 161 trucks' data over 2 years. 69 of the trucks do not have any DPF failure. 92 trucks have one or more DPF failure during 2 years. The file "vehicle_list.csv" contains information regarding the vehicles (vehicle_model, vehicle_year, dpf_failure, etc.). The individual files for each platform id contain 16 columns of features for the truck over 2 years. Each entry in the file represents a single day. If the truck was not active on that day, the data should be filled with zeroes. Table 2.1 is an example of the individual files. The column "vehicle_serviced" refers to whether or not the truck encounters a dpf_failure.

	date	id	mile	fuel	..	..	dpf...	..	..	vehicle_serviced	Diagnostic code
0	2019 1/2	30 94 50	48.1	7.9			268			0	[]

Table 2.1

2.2 Preprocessing and Imputation

The dataset was relatively clean and preprocessed without any missing values. For our assignment, we planned on predicting a DPF failure 14 days before a failure occurs, so therefore, all entries 1-14 days before the service date were marked as a failure before training. When there were missing values, we would impute them with the mean values. Given the raw dataset and features, it was noticeable from the EDA that we would run into problems if the models were built using these features. These problems were solved later on in the feature engineering section (Section 4 below).

3. Exploratory Analysis

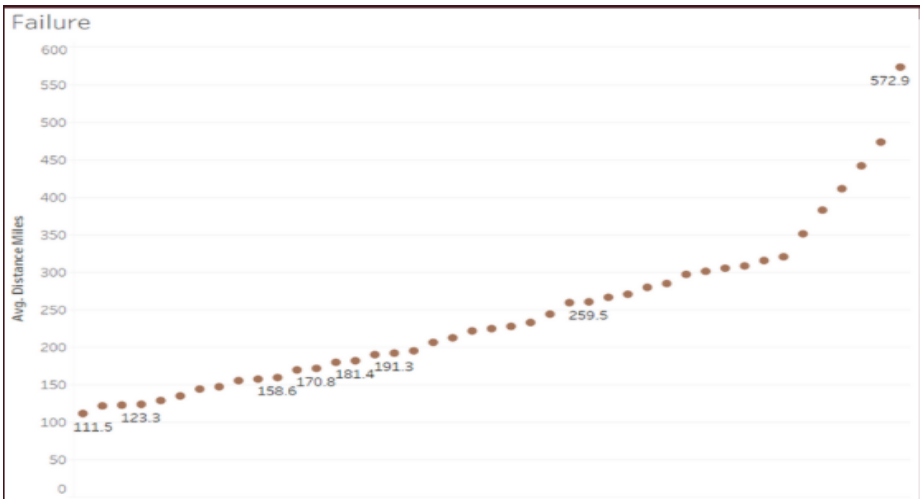

Figure 3.1.1

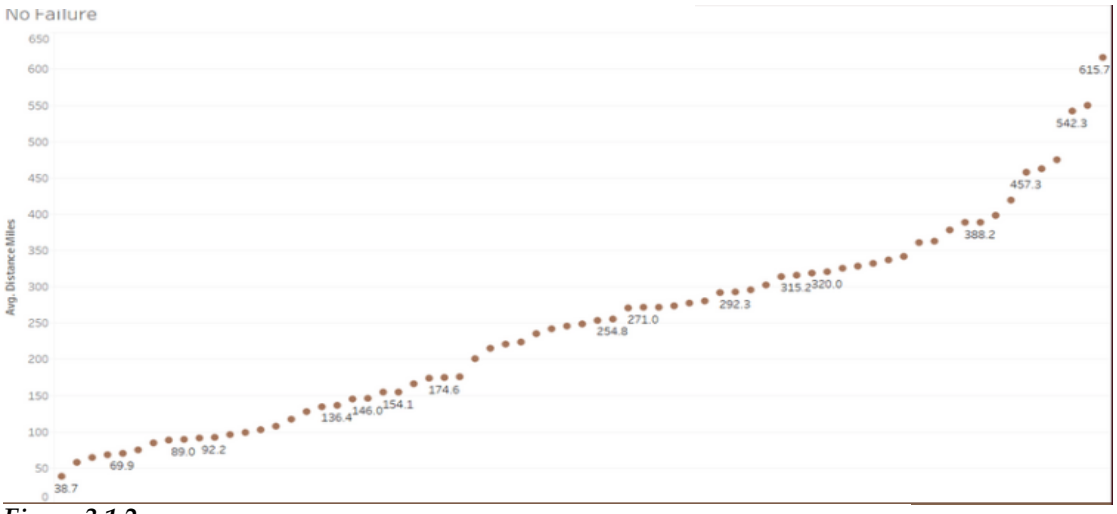

Figure 3.1.2

Fig 3.1.1 and 3.1.2 and clearly shows the variance distance data set on where the truck has no failure and failure, ranging from an average distance of 90 to 600. We think the average distance that a truck travels is a significant feature as distance relates to the truck's speedometer, where we can see if the truck does or does not have problems before the given life span. If it breaks before its life span, it shows that the driver or the mechanic is not maintaining the truck properly.

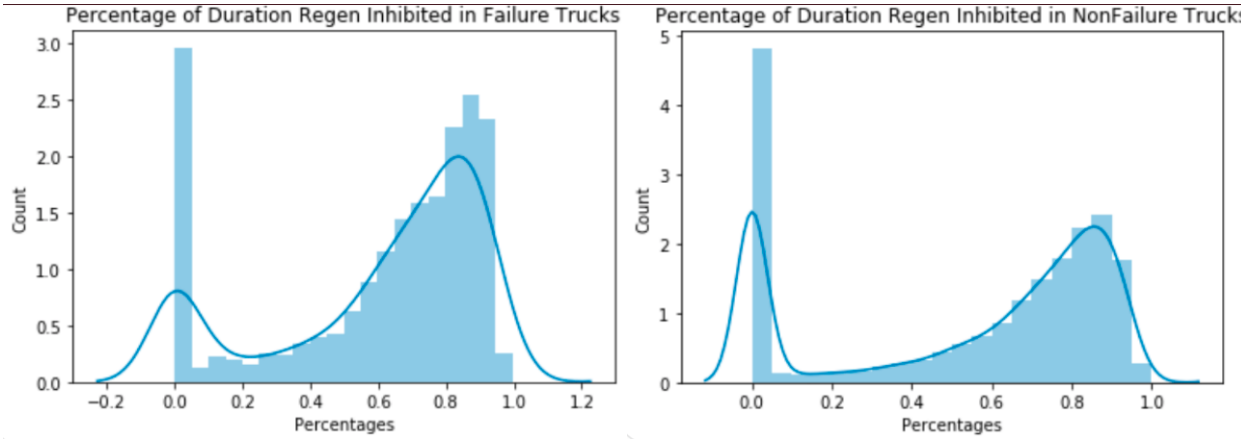

Figure 3.2

Figure 3.2 displays the percentages of duration regen inhibited in failure trucks. The chart showed a stronger left skew compared to the nonfailure truck on the right chart in Figure 3.2. This showed that the failure trucks had more regen blocked compared to the nonfailure chart.

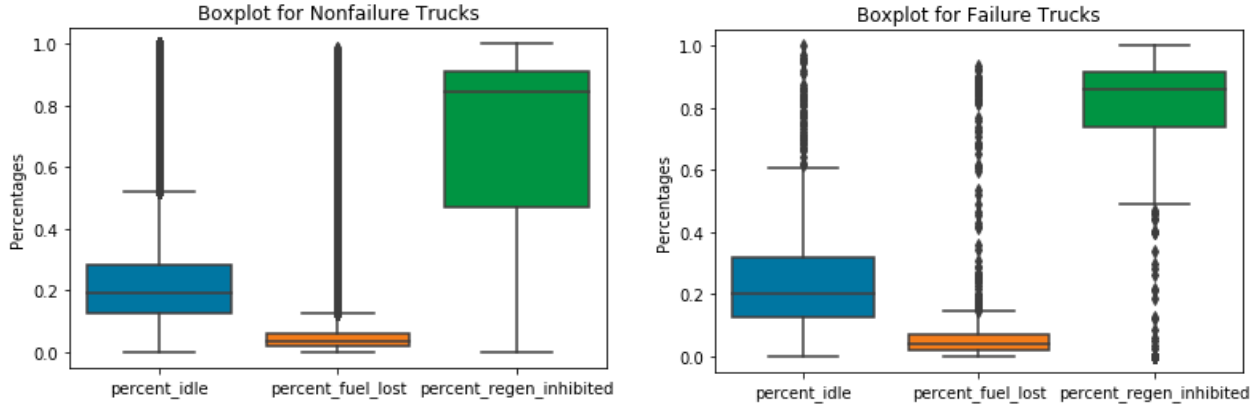

Figure 3.3

Figure 3.3 displays a few features comparing both nonfailure and failure trucks. Noticeable is that the percent regen inhibited for failure has a much tighter gap closer to the upper range.

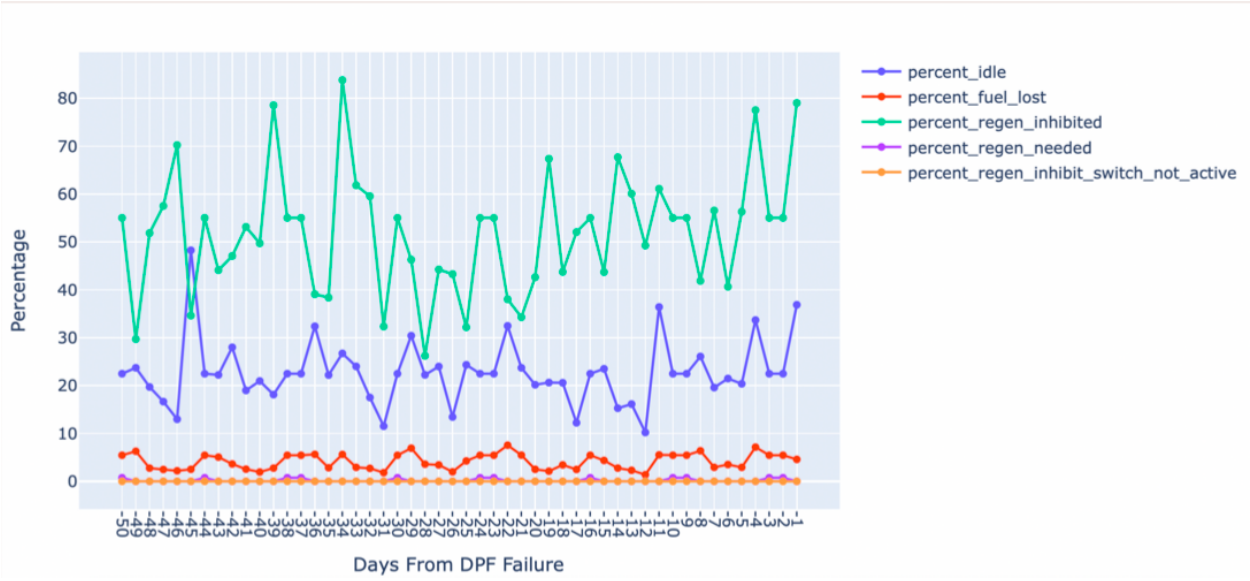

Figure 3.4

Figure 3.4 is a visualization of all five normalized percent features, precisely 50 days before failure. This is specifically from platform ID 300490. We made the same plots for all of the other platforms and found that percent regen inhibited and percent fuel lost looked like better candidates than percent_regen_inhibit_switch_not_active that did not have much variance at all.

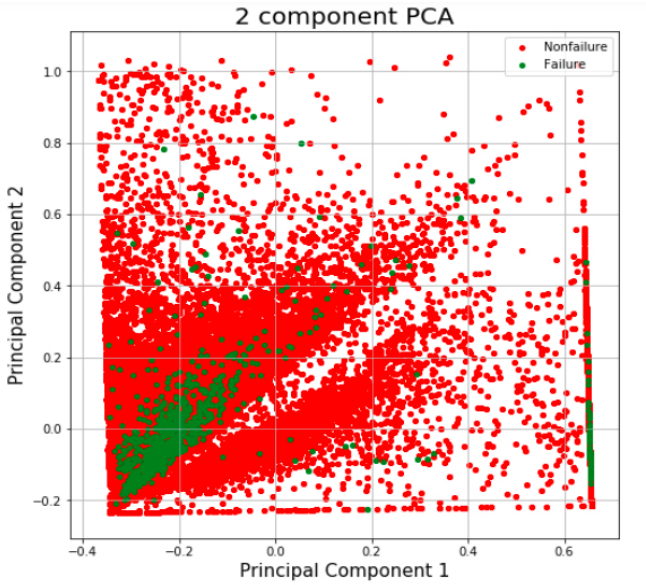

Figure 3.5

Figure 3.5 is a PCA plot using the top two principal components. It is clear that PCA did not linearly separate the failure vs nonfailure trucks due to the sheer number of nonfailures.

3.1 EDA Key Findings

From our EDA, it is evident that we can clearly distinguish between failure and nonfailure trucks for a majority of the features. We also see that much predictive power can be obtained from a relatively small feature set. Lastly, we also found some unimportant features.

On the other hand, there were some features with not much difference in the failure vs the nonfailure case. This makes it difficult to use that feature in our predictive model. All of this information was used to determine feature selection later on in the process. There is an obvious class imbalance at hand since the ratio of failure trucks to nonfailure trucks are 100:1. As seen in Figure 3.5, it is very difficult

to linearly separate the two classes due to the class imbalance problem. Moreover, there are some clear outliers for certain features.

4. Feature Engineering

Following EDA, we got a much better sense of the data we are working with and the features. We tackled our problems faced from EDA one by one. For the features that did not result in much difference between the two classes, we removed them or modified them. The outliers were due to features at varying scales. We replaced raw numbers into percentages. For example, instead of using the original features showing total duration and minutes idle, a new feature would be aggregated as the percentage of total time idle. This would make all entries on the same percentage scale.

New features were also created on the basis of existing features. For example, "Fuels Used Per Mile" was created from the Fuels Used and Mile Traveled features. The benefit of this is it leads to a different way of aggregating the data which could potentially be a more effective feature in the model.

4.1 Principal Component Analysis (PCA)

Once we got the suitable features extracted from the raw data, our goal was to determine which features to use in our final model. We used a variety of methods, including PCA anomaly detection, random forest feature importances, and TSFresh feature selection.

To reduce the class imbalance, not all of the nonfailures are included in the model. The entries 15 days before a service date would still be considered a failure. However, only the entries 15-30 days after a service date would be counted as a nonfailure. This would lead to a balanced dataset. Next, we ran a random forest model and determined the top 5 most important features. Those top 5 features were miles_per_minute, fuels_used_per_mile, percent_fuel_lost, percent_regen_inhibited, and fuel_lost_per_mile.

Principal Components Analysis (PCA) was performed on the original feature set to reduce the dimensions to include only the top 2 principal components. Training data includes only the nonfailure trucks to calculate the covariance matrix of the normal class. Calculated the Mahalanobis distance (a metric used in cluster analysis) for each of the failure trucks to the nonfailure trucks. If the distance is above a certain threshold (3 standard deviations from mean), that entry would be marked as an anomaly. The process was then repeated for varying starting features. The feature set that most effectively distinguished the failure class from the nonfailure class was consistent with the set from the random forest.

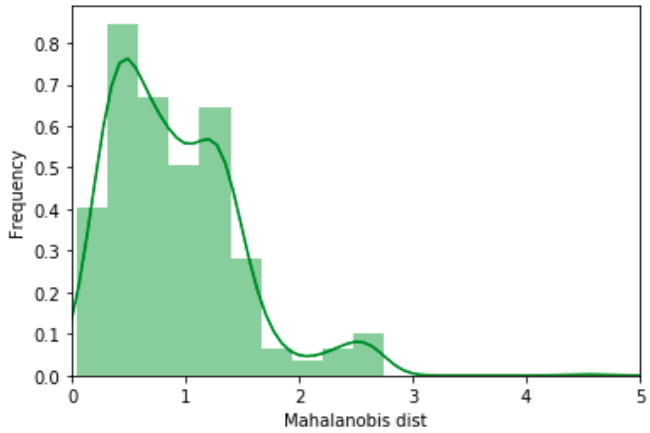

Figure 4.1

Figure 4.1 depicts a sample distribution of Mahalanobis distances for the nonfailure class. As you can see, the threshold is around 3 standard deviations from the mean. Any record with a higher value would be considered an outlier.

4.2 TSFresh Feature Selection

TSFresh can transform a given feature into thousands of different signals. Because of the high number of potential features that it transforms into, a good feature selection method is required so that by the time the pipeline reaches the modeling phase, the model iterates better and the Spark cluster will be under less loadThere are three possible transformation sets that TSFresh provides that are embodied within three "settings." These settings are efficient, comprehensive, and minimal. Respectively, there are 72, 74, and 9 transformations in each set. Furthermore, in these 74 transformations in the case of the comprehensive setting, there are sub transformations that are contained within them. An example of a sub transformation would be standard deviation. The sub transformations within this is with r = 0.05 and r = 1. As you can see, standard deviation may have two sub transformations, but some of the transformations have a lot of sub transformations. The distribution of this is shown in figure 4.2.1.

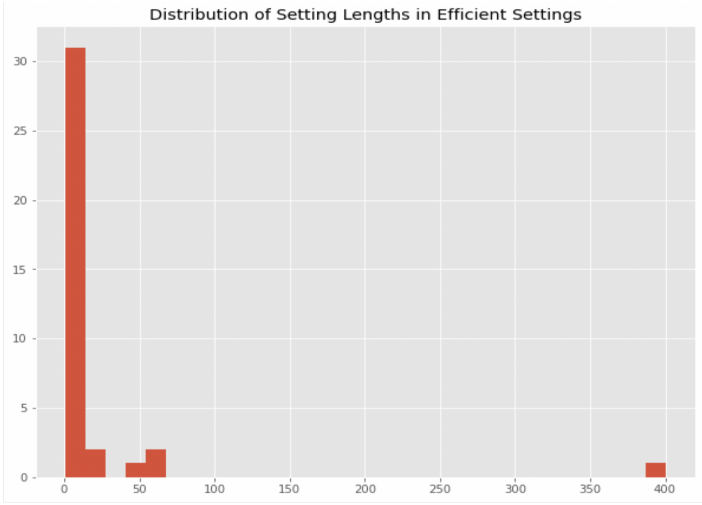

Figure 4.2.1

Within TS Fresh, different window configurations are built into the package. We used this capability to create rolling windows on the data at this stage of this pipeline. This increases the size of the data by a lot. Depending on the window size, this will scale the data by different factors. If we have a window size of 50 and shift of 1, and let's assume that the data is 1000 rows, the number of windows will be 1000 - 50. And to get the total number of rows of data after this rolling transformation, we will have the product of the total number of windows and the window size, which in this scenario will be 950 * 50 = 47500 rows. When this is further scaled by the number of transformations in the settings, the data quickly grows at a scale where computational runtime becomes an issue.

To combat this, our group initially wanted to use the minimal set proposed. However, the tradeoff for fewer transformations is less usable transformations in such a way that the desired signals of the data will not be fully captured. We needed to find a middle ground for not having enough transformations to cause runtime issues but have enough to extract useful signals. This motivated us to put a threshold for the number sub transformations that a single transformation should have. Imagine a vertical line at x = 5, and any transformation to the right will be taken out of the transformation set. We called this the "medium" setting, and this led to a good balance of the tradeoff mentioned above.

As for the actual feature selection method, we employed two methods. The first is through the use of a random forest (figure 4.2.2). Since traditional classification methods don't bode well with class imbalance, we did both an undersampling and an oversampling (with SMOTE) of the TSFresh transformed features (both in terms of the rolling transforms and feature extractions) to narrow down the feature set. Respectively, we converted a class imbalance of only 0.0909% containing DPF failures to 33% in the undersampled dataset and 50% in the oversampled dataset. We ran the random forest with 5-fold cross validation and hyperparameter tuning over 6 different n-estimators configurations. Figure 4.2.2 shows the confusion matrix of the random forest results on the unbalanced and balanced dataset, alongside the feature importances ranked by importance for the balanced dataset.

Figure 4.2.2

However, this turned out to be more an exploratory technique. It turns out that for the nature of time series data such as this, TSFresh employed a better technique for feature selection that minimizes spurious correlations in importance using a Benjamini Hochberg procedure. Figure 4.2.3 shows the results employing TSFresh's relevance table module to find the top 20 features to feed into the autoencoder.

Figure 4.2.3 displays the results of the TS Fresh feature selection. We did this on the oversampled data, and you can see the resulting relevance tables on the right that came out of this method. This method employed the Benjamini Hochberg procedure testing a feature relevance null hypothesis, and what we are looking for are very low p values because that means that the feature is relevant for our

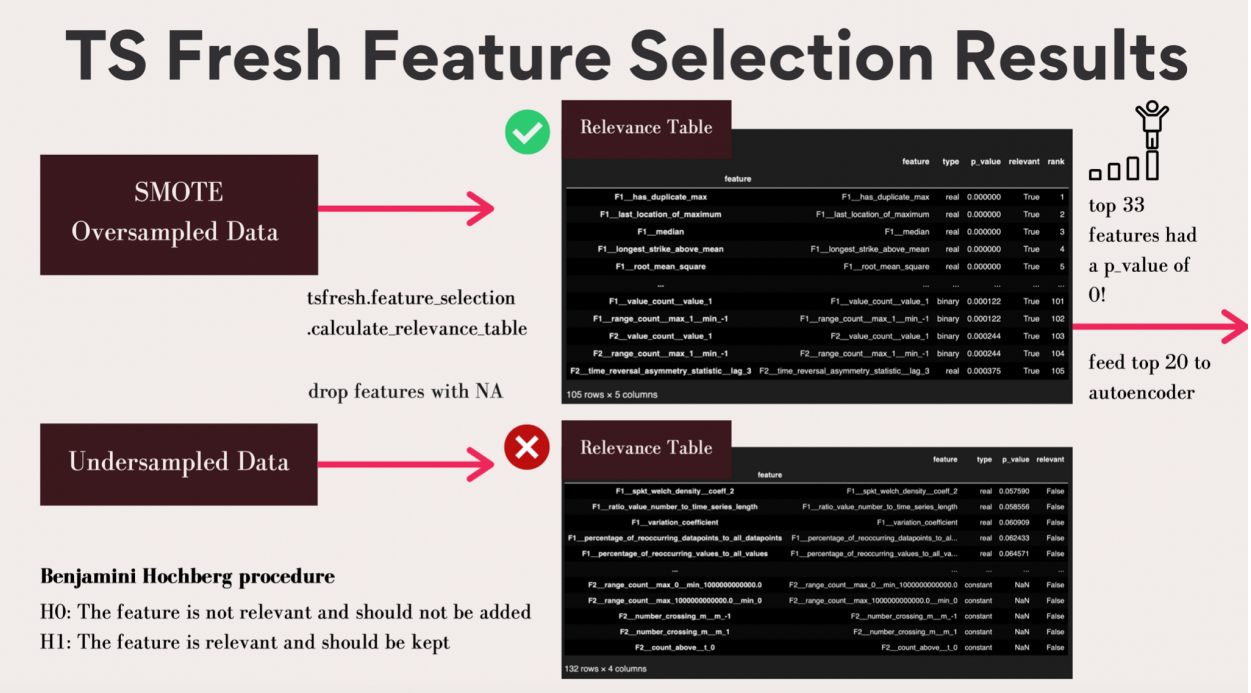

Figure 4.2.3

specific prediction problem. The oversampled data performed better because there was simply more data available to train to utilize it, and the top 33 features had a p-value of 0. The top 20 features are what we ultimately used to feed into the autoencoder.

5. Model Development

Since we want to predict DPF failure 2 weeks in advance and deal with the heavy class imbalance, we decide to use autoencoders and treat the prediction as an anomaly detection problem. We did not oversample nor reduce the data for the training and testing of autoencoders because we believe that heavy class imbalance will always be an issue in the case of DPF failure prediction - DPF failure will almost always be an extremely rare event. We hope that, through our realistic approach, the autoencoders can learn some of the trends and temporal relationships in DPF nonfailure trucks. So far, we have built three autoencoder models - one with only dense layers, one convolutional autoencoder, and one LSTM autoencoder. We expect that the autoencoder with only dense layers will not perform as well as the other ones because of the linearity in dense layers.

5.1 Modelling Setup

This section describes the methodology and setup for our models.

5.1.1 Data Preparation

After selecting the most relevant features through the aforementioned feature engineering, we window the data again so that it can be processed by the autoencoders. Here, the window size and overlapping percentage are hyperparameters to tune with. We started with window size 50 shifting by 1, because we are worried that a big overlapping percentage will skip the window representing DPF failure.

5.1.2 Data Labeling

We need to relabel the TSFresh-processed data because we windowed the data again. We started by only labeling the window exactly 14 days before the DPF failure as 1. Then, we removed the data immediately after the window until there were no more 1s in the window. This is because the autoencoder requires only 0s in the window in order to correctly predict again. This approach resulted in a heavily imbalanced dataset as we had expected - 88 DPF failure trucks vs over 53000 DPF nonfailure trucks. We then experimented with augmenting the DPF failure by labeling any record within 14 days before the DPF failure as 1. This approach boosted the number of DPF failure records to 1326.

5.1.3 Training, Validation, and Testing

We trained the model only with the normal trucks in order to have the autoencoders learn a compact representation of only the non-failures. The input to the autoencoder will be the same as the output. The training set contains 80% of the DPF nonfailure data. The validation set contains 20% of the DPF nonfailure data. We will test our model on the data of DPF failure trucks, which contain both DPF nonfailure and failure records. This testing process will also enable us to determine the optimized threshold for the prediction. Finally, we will test our model and threshold on the unseen dataset provided by Vnomics.

5.1.4 Evaluation Metric

The most important evaluation metric we use is the recall score. As aforementioned, missing out on a DPF failure will be more costly than misclassifying a DPF nonfailure truck. However, we will only be able to get the recall score after setting the threshold and tuning the hyperparameters. Therefore, to evaluate the training process we used mean-square error(MSE). Then we use the recall score to evaluate the testing set.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative} \qquad \text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

5.1.5 Determining the Threshold for Prediction

Figure 5.1 is an example of how the normal trucks (DPF nonfailure) and DPF failure trucks are classified. The horizontal red line represents the threshold at which any data points beyond this line are classified as DPF failures. After training the models, we will determine this threshold based on its maximum reconstruction error (mean-square error). If the threshold equals the error, it means that the threshold will probably successfully classify most of the normal trucks. The threshold can also be adjusted in order to maximize the recall score for DPF failure trucks at the cost of lowering the recall score for DPF nonfailure trucks. We select the best threshold that maximizes the recall score for DPF failure trucks and the normal trucks.

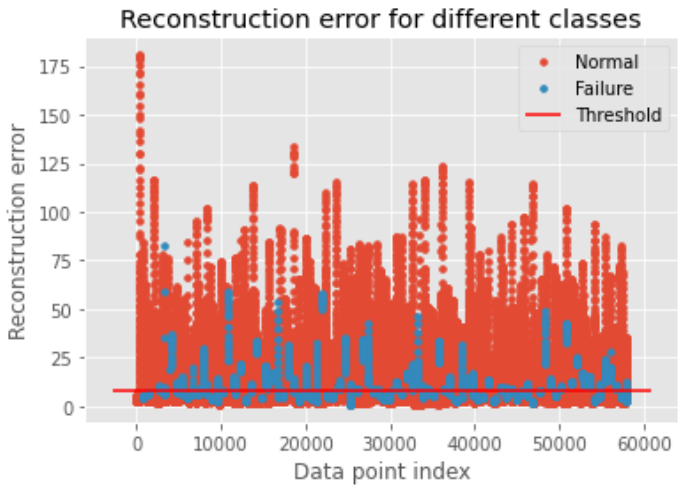

Figure 5.1

5.2 Hyperparameter Tuning

After setting up the baseline models, we thoroughly examine all aspects relating to their performance. They can be mainly broken down into three parts: relevance of data, windowing and model hyperparameters, and labeling methods.

```
Hyperopt with SparkTrials will automatically track trials in MLflow. To view the MLflow experiment as
in the notebook context bar on the upper right. There, you can view all runs.
To view logs from trials, please check the Spark executor logs. To view executor logs, expand 'Spark
e stage from the trial job. Click it and find the list of tasks. Click the 'stderr' link for a task t
100%|██████████| 30/30 [32:39<00:00, 65.32s/trial, best loss: 0.05790967494249344]
Total Trials: 30: 30 succeeded, 0 failed, 0 cancelled.
```

Figure 5.2.1

5.2.1 Relevance of Data

This pertains to the feature engineering part of this project. It is crucial that the generated features capture essential temporal information in the dataset. As aforementioned, we have tried various approaches to select the most important features. So far, our PCA, TSFresh windowing, and random forest approach have led to an increase in the recall score for our models with similar model parameters.

5.2.2 Windowing

Here, windowing refers to the window size in both TSFresh time series and autoencoder windowing. We have experimented with TSFresh window size 50 shifting by 1 and 20 shifting by 1. Respectively, we have autoencoder window sizes 8 shifting by 1 and 30 shifting by 1. Due to the limited time, we had less opportunity to test with different overlapping percentages but they should be a meaningful parameter to tune with in future research.

5.2.3 Model Hyperparameters

For the autoencoder models, we have the dropout rate, optimizer, epochs, and learning rate as the hyperparameters to tune with. We utilized MLFlow from Databricks to help the tuning process. The final choice of hyperparameters is based on the MSE loss of the models. Figure 5.2.1 is an example output from MLFlow on LSTM autoencoder hyperparameter tuning.

5.2.4 Labeling

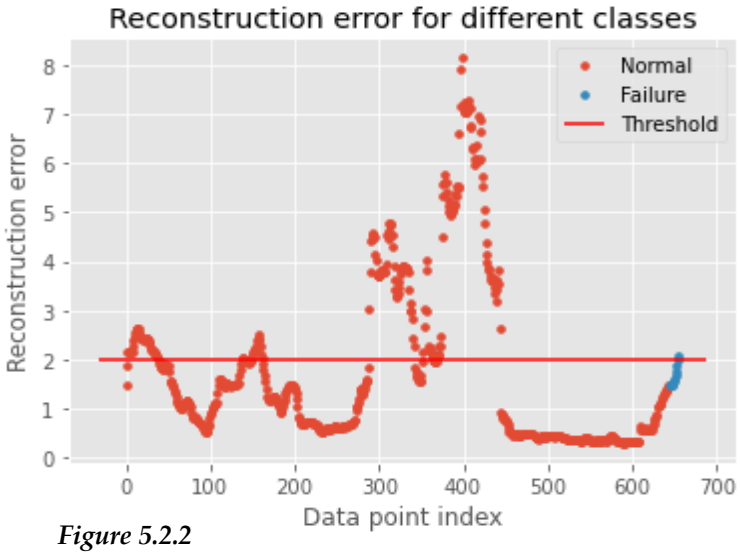

Figure 5.2.2

In section 5.1.2, we discussed the problem of the heavy class imbalance (88 vs 53000+) from labeling only the window exactly 14 days before as DPF failure. We took the new approach to label all windows 14 days before the actual DPF failure as a failure, which augmented the number of DPF failure records to 1326. In future studies, labeling windows between 140 and 300 days before the actual DPF failure should be tested. It would also be interesting to compare how well the model performs by predicting 300 days in advance vs 14 days in advance.

So far, we think this approach can be promising. For example, figure 5.2.2 demonstrates how our LSTM autoencoder classifies one single platform, with blue dots representing the DPF failure 14 days before the actual failure. However, most of the variations in the reconstruction error take place about 300 rows before the actual DPF failure. One possible explanation is that the driver performs abnormal driving behaviors which lead to a DPF failure 300 days later. This may suggest that the true indicator for DPF failure might be many more days before the DPF failure, and there could exist some larger-scale temporal relationships that lead to a DPF failure. In addition, less overlapping between the windows should be considered to learn and distinguish the trends.

6. Performance and results (10%)

After hyperparameter tuning for each model, we evaluated their performance on the test set. The table below shows the MSE loss and recall score for the three models on both DPF normal and DPF failure records. Figure 6.1 demonstrates an example of the training on the autoencoder with only dense layers. Figure 6.2 illustrates how the LSTM autoencoder classified failures. Figure 6.3, 6.4, 6.5 are the classification reports for each model.

Model	**Final MSE Loss (validation)**	**Recall (DPF normal)**	**Recall (DPF failure)**
Autoencoder with only dense layers	0.105	0.5	0.58
Convolutional autoencoder	0.389	0.5	0.55
LSTM autoencoder	0.117	0.5	0.58

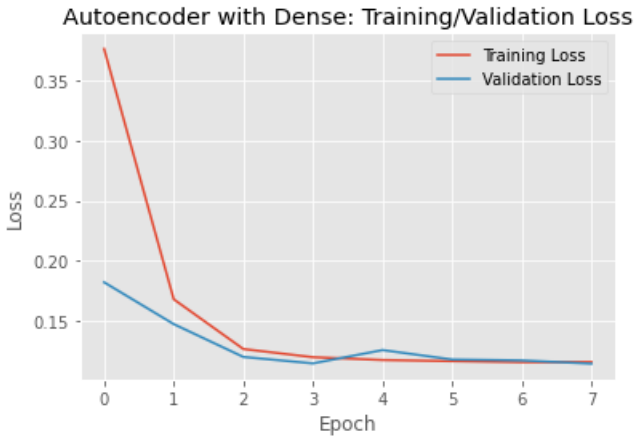

Fig 6.1

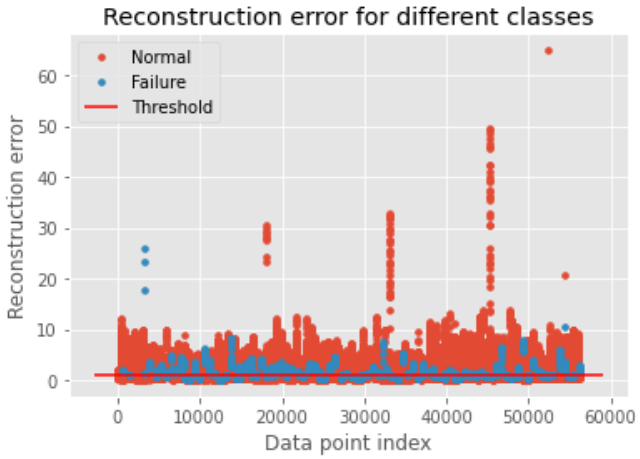

Fig 6.2

Surprisingly, the performance of the dense layer autoencoder matches the performance of the LSTM autoencoder. One possible implication is that there exists certain linearity in the temporal relationship between the records. Another reason could be the relatively small window size. As mentioned in the previous section, such window size can fail to capture some larger-scale trends in the data.

Classification Report for Autoencoder with Dense Layers:

	precision	recall	f1-score	support
0	0.98	0.50	0.66	54835
1	0.03	0.58	0.05	1322
accuracy			0.50	56157
macro avg	0.50	0.54	0.35	56157
weighted avg	0.96	0.50	0.64	56157

Fig 6.3

Classification Report for Convolutional Autoencoder:

	precision	recall	f1-score	support
0	0.98	0.50	0.67	54835
1	0.03	0.55	0.05	1322
accuracy			0.50	56157
macro avg	0.50	0.52	0.36	56157
weighted avg	0.96	0.50	0.65	56157

Fig 6.4

Classification Report for LSTM Autoencoder:

	precision	recall	f1-score	support
0	0.98	0.50	0.67	54835
1	0.03	0.58	0.05	1322
accuracy			0.51	56157
macro avg	0.50	0.54	0.36	56157
weighted avg	0.96	0.51	0.65	56157

Fig 6.5

Besides, we did run an experiment on the LSTM autoencoder with window size equals 200, shifting by 1 row. The final recall score is 0.5 for DPF normal and 0.59 for DPF failure. Although not a huge improvement, we think that increasing the overlapping percentage with a different labeling method can further boost the performance.

7. Conclusion and Next Steps

In this project, we conducted EDA and found significant predictive power from a small set of features. Then we conducted feature engineering with PCA, random forest, and TSFresh to obtain more effective features to feed to the autoencoder. We built three autoencoder models and tuned their performance with various window sizes and model parameters. The autoencoder with only dense layers and LSTM autoencoder achieved the best performance.

If we predict on a synthetic and balanced dataset, we get an over 90% recall score. In the full fledged realistic scenario of predicting two weeks in advance and with unseen data, we were able to achieve a recall score of 50% for the DPF non-failure trucks and a 58% for the DPF failure trucks.

It is reassuring that the developed pipeline and infrastructure is comprehensive. Many exploratory methods were used in the work so far, which leaves just the higher quality transformations in our current pipeline. The path to achieving even higher recall scores will simply involve tuning the many different parameters that are customizable in our modelling as our next step.

For example, we should experiment with bigger window size, most notably in the TSFresh windowing, with a larger overlapping percentage. Also, we have a hypothesis that bigger picture or aggregate trends in the records will be more helpful in classifying DPF failure. We should also try different labeling methods, for example, predicting 100 days in advance before the failure.