Deviation Finder: An Elevator Anomaly Detection System

Keerthi Ningegowda
klnu@sfu.ca

Vipin Das
vdas@sfu.ca

Kenny Tony Chakola
kchakola@sfu.ca

Ria Thomas
riat@sfu.ca

Varnnitha Venugopal
vvenugop@sfu.ca

ABSTRACT

The objective of this project was to design a system that can predict anomalies in elevators using accelerometer data. We have created a data science pipeline that incorporates methodologies such as data cleaning, data pre-processing, exploratory data analysis, data modelling and model deployment to meet the objective. We have experimented various unsupervised models such as K-Means, DBSCAN, Isolation Forest, LSTM, and ANN Auto-encoders to capture deviations in normal patterns. LSTM Auto-encoder outperformed other models with an F1-score of 67%. A demonstration of model deployment on web using Kafka, AWS Dynamo DB, Flask and Plotly Dash is also described in this report.

MOTIVATION

Elevators have become an integral part of any building structure due to its ease of transporting people in minimal time. Elevators do not just help us get places a little faster – they make dense urban living possible, allowing the efficient construction of high-rise structures. Like every other machine, elevators also require periodic maintenance.

Periodic health check-up of lifts is done by elevator technicians through routine inspections. Increasing development of new infrastructures has resulted in exponential installation of lifts demanding more inspectors in this area. Acquiring new workforce and training them can be expensive and time consuming. Hence, it is important to setup an automated process that can effectively find anomalies in elevators using Internet of Things and Artificial Intelligence.

This motivated us to work with Technical Safety BC to analyze raw data collected using accelerometers installed in 15 different lifts across British Columbia. Accelerometer data can be valuable in identifying and predicting technical issues related to elevator brakes, elevator motor and elevator alignment.

PROBLEM STATEMENT

Following were some of the question we thought to find answers for:

- Can we find abrupt speed changes in lifts?
- Can we predict the point when brakes are worn out?
- Can we predict if elevator cable is torn?

To answer these questions, we decided to obtain accelerometer data from sensors installed in lifts. The data include vertical acceleration at a given point of time. Hence when a lift is about to break down due to any of the issues mentioned above, accelerometer will produce abnormal values enabling us to detect the deviations.

Some of the key challenges that identified in the data are listed below:

- Dataset had no labels about anomalies.
- Finding patterns in huge dataset
- Generalizing a model that can fit all types of lifts
- Minimal set of features
- Each lift had its accelerometer placed on different locations of lift resulting in axis and direction changes.

THE DATA

The 3-axis motion sensor was installed inside elevator to log data when the elevator moves vertically.

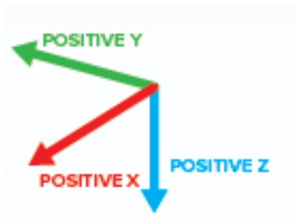


Fig 1: 3-axis Motion

The total data collected for 2 months across 15 lifts contributed to 75 GB of data. The data generated from accelerometer would consist only of a timestamp and a numeric acceleration value for each time point. Four columns were included in every file:

Table 1: Dataset Variables

Variable	**Description**	
Timestamp	(yyyy-MM-dd HH:mm:ss.SSS)	**Sampling frequency**: 25 Hertz which is 24 data points per second.
X-axis	Acceleration value (g-force = 9.8m/s2)	**Sensitivity:** 2g (The unit of data is in g).
Y-axis	Acceleration value (g-force = 9.8m/s2)	
Z-axis	Acceleration value (g-force = 9.8m/s2)	

The device is attached to the ceiling of each elevator and one of the axis values represents the movement of the elevator when it traveled up and down. Depending on the placement of accelerometer in lifts, the axis used for vertical movement tend to change. Hence, we use the vertical axis values and the timestamp columns for the analysis and machine learning section of this report.
Additionally, some of the lifts had their acceleration placed in inverted direction resulting in acceleration and deceleration values cited at opposite directions. The horizontal axis data are not analyzed in this project although those axes data could potentially be used to detect anomalous lateral (side-to-side) vibrations. The data also contain direction of the sensor and instructions about which axis to look for elevator vertical movements. For example:

- Positive (P): Sensor was placed towards positive axis. The bigger the value, the larger the acceleration
- Negative (N): Sensor was placed towards negative axis. The smaller the value, the large the acceleration.

When the sensor is not moving, the default value is 1g. Sensor data always has noises associated with it and needs calibration to make it consistent. A table depicting the accelerometer direction and its movement is shown below:

Name	**Axis catches ED vertical movement**	**Direction** Positive(P)/Negative(N)
EDS_1.csv	Z	P
EDS_2.csv	Z	P
EDS_3.csv	X	P
EDS_4.csv	Z	N
EDS_5.csv	X	P
EDS_6.csv	Z	N
EDS_7.csv	Z	P
EDS_8.csv	X	N
EDS_9.csv	Z	N
EDS_10.csv	Z	N
EDS_11.csv	X	P
EDS_12.csv	X	P
EDS_13.csv	Y	N
EDS_14.csv	Z	N
EDS_15.csv	Z	P

DATA SCIENCE PIPELINE

Data Cleaning

Since our questions were mostly related to the vertical movement of data, we focused our analysis on accelerometer signals depicting the vertical direction. Hence, we decided to remove other two axis that records the horizontal movements of the lift. Also, based on the above table, the sign of acceleration values for elevators were flipped where the direction is 'N'. This ensured us to maintain uniformity across all lifts which will further help in model training as well. We also used data only from 2018-07-09 12:00:00 to 2018-08-09 12:00:00 as data outside this range was not useful and had no patterns at all. This could probably be due to the initial hiccups of collecting data from sensor.

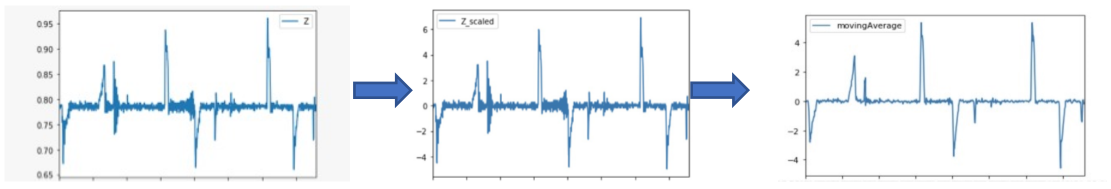
Data Preprocessing

First step to pre-process the data was to apply normalization. To achieve this, we used 'Standard Scaler' or otherwise known as 'Z-score normalization'. Standard scaler basically performs the below:

$$Z = \frac{(X - \mu)}{\sigma}$$

This method ensures to center the mean at 0 and standard deviation at 1. Second step was to remove noises in sensor data to find the hidden pattern. We used low pass filter called as 'Moving Average' on the data to smoothen the samples. The application of moving averages was done on a sliding window of size 6 without distorting the shape or suppressing anomalies.

$$MA = \frac{1}{n}\sum_{i=0}^{n-1}(a_{n-i})$$



Raw signal Z - Normalized Low pass - Filtered

Fig 3.1: Applying normalization and low-pass filter to raw data

Exploratory Data Analysis

Our next step in the process was to capture normal and abnormal trips of an elevator. To find this we first rounded off the pre-processed values to the nearest integer. Then, we plotted a histogram to understand the frequency distribution of values of each of the lifts. We could infer a normal distribution with majority of values centered near 0. We could also find certain maximum and minimum peaks from this distribution to find the acceleration patterns.

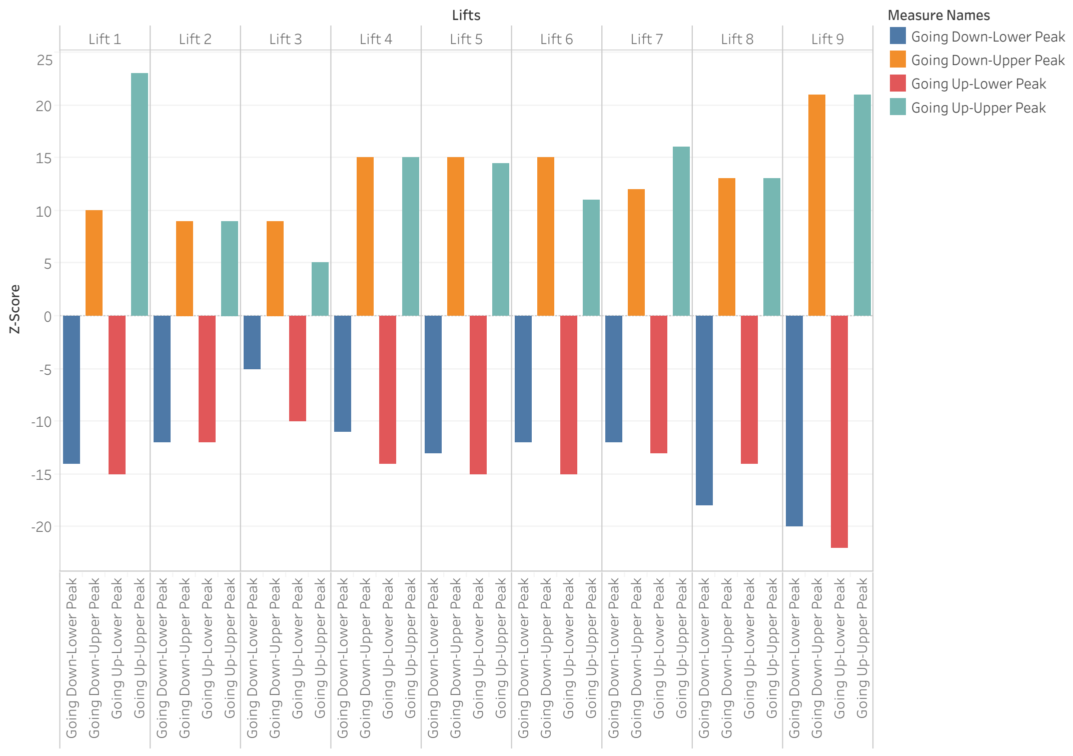


Fig 3.1: Frequency distribution of values per lift

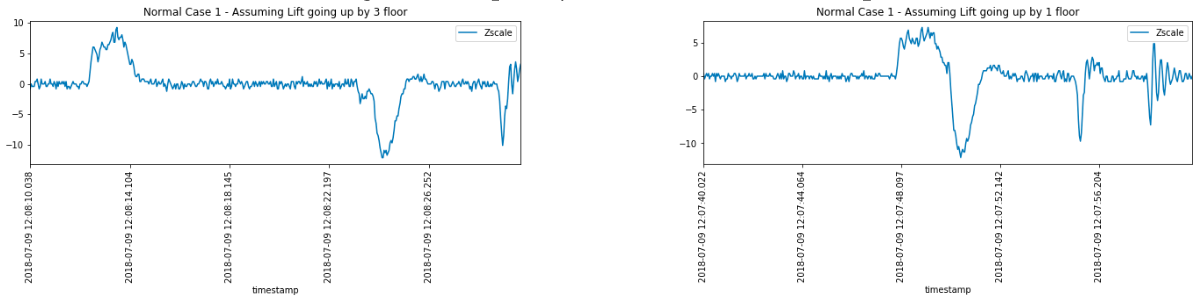


Fig 3.2: Normal Cases for lift going up

First plot tells us that the lift was going up by almost 3 floors. This can be understood by seeing a high acceleration at the beginning followed by a deceleration as it reaches the designated floor. Second case depicts the vertical motion of lift by exactly one floor. This can be understood by seeing the absence of flat line as compared to that in first plot.

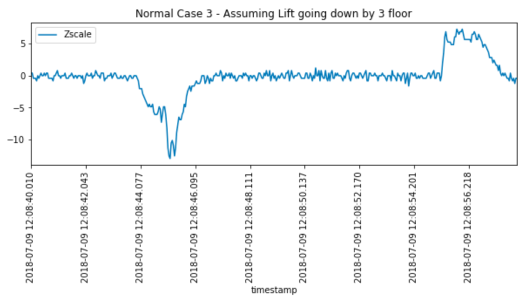


Fig 3.3: Normal Cases for lift going down

As you can see in this case, a negative acceleration is followed by an upward pull once it reaches the designated floor. It was not only normal patterns that we could find from this activity. By taking maximum and minimum values in dataset, we were able to capture some of the abnormal cases as well. The magnitude of such anomalies was easily visible by plotting those timestamps. Some visualizations of anomalies are shown in Fig 3.4.

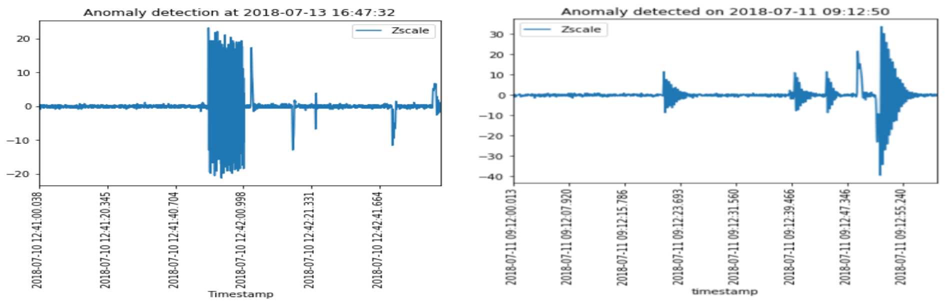


Fig 3.4: Anomalous Patterns

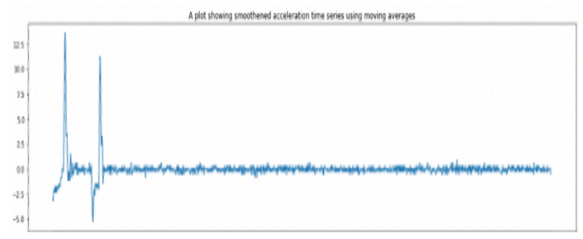
Feature Engineering

Since we had only one feature, we used python library **'tsfresh'** ("Time Series Feature Extraction based on Scalable Hypothesis tests") [2] to generate new features. The package can be used in different settings to generate features out of which we used two settings in our project.

Comprehensive feature generation: This setting involves extracting all the possible combinations of features from the time series dataset.

Efficient feature generation: This setting is like Comprehensive feature generation but does not generate features that need high computational cost. This setting is very helpful when working on streaming data. Both methods generated about 750 features out of which absolute energy and mode of acceleration had non-zero values. So, we decided to go ahead with these features.

Below are the sample plots of the de-noised signal and the absolute energy feature:



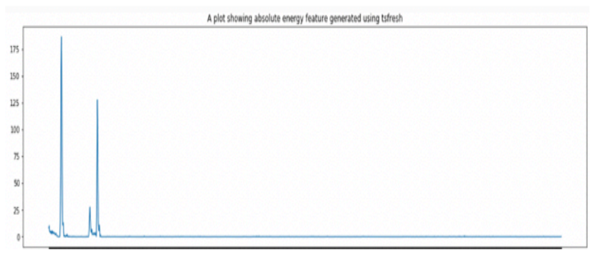
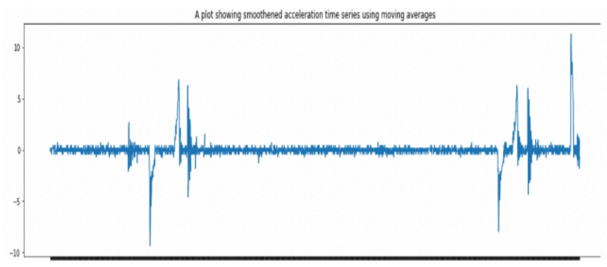


Fig 4.1: De-noised acceleration time series

Fig 4.2: Plot showing energy feature generated by tsfresh



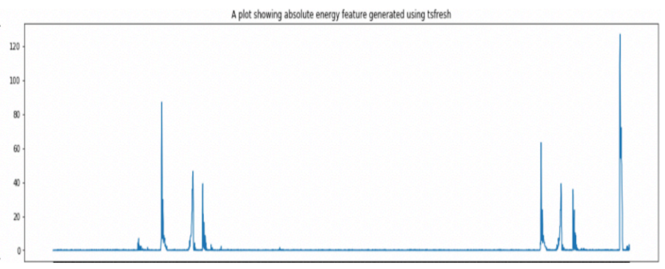


Fig 4.3: De-noised acceleration time series

Fig 4.4: Plot showing energy feature generated by tsfresh

**From the plots it is evident that anomalies have a sharper energy peaks than normal peaks.*

Machine Learning Models

Our project mainly uses unsupervised learning, where data has no labels and the algorithm often attempts to assign its own label as anomaly or not. We have experimented five different machine learning algorithms namely:

Standard Anomaly Detection:

A naïve or standard approach to identify anomalies from the numerical data is to classify data points that are 4 or more standard deviations away from the mean. A similar method is used for this approach where a sliding window moves over the data and searches for outliers (peaks or dips) in the data. If several peaks or dips are found within a window, then it is classified as an anomaly and timestamp is recorded. The basic intuition behind this technique was that after looking at some anomalous data, we found that anomalous elevator data had some spikes or dips than normal operations that might have caused due to over-speeding, over-loading, etc.
Observing this led to this method of anomaly detection by looking for large spikes or dips in data that occur close together. This method searches for anomalies within a 20 second time frame. i.e., the method starts out by moving a sliding window of size 20 over the entire data points. All the data points which lies 4 standard deviation away from the mean are considered as outliers and are tracked as peaks or dips (depending on whether the points lie above or below the standard deviation). The mean and standard deviation is found using the entire data set. If 5 peaks/dips are found within a window, then this is labeled as a cluster. And, if there are more than 1 such clusters within the window, then that is labeled as an anomaly**.**

Isolation Forest:

Isolation forest [5] [6] is an unsupervised learning algorithm that works on the principle of isolating anomalies, by building an ensemble of iTrees for a given data set. Anomalies are classified as those instances which have short average path lengths on the iTrees.

- During training, this model is fed with data containing normal data and a few instances of abnormal samples (configurable). In other words, it learns what normal data looks like to be able to distinguish the abnormal data.

- It works with the basic assumption that anomalies are few and easily distinguishable and has a linear time complexity with a low constant and memory requirement.

An external package of distributed iForest [7] on Spark is used (the usage is similar to the iForest sklearn implementation), which is trained via model-wise parallelism, and predicts a new dataset via data-wise parallelism. The contamination parameter plays an important role and represents the percentage of outliers in the data. In order to detect an anomaly (true negative) we have arrived at its value based on trial and error on validating its results with outliers in 2D plot [8].

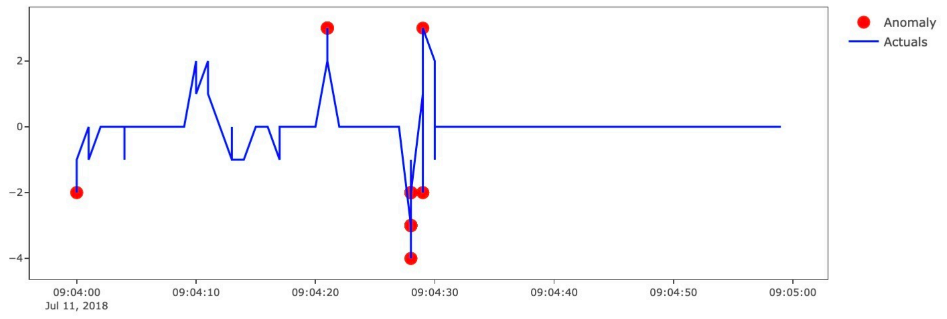
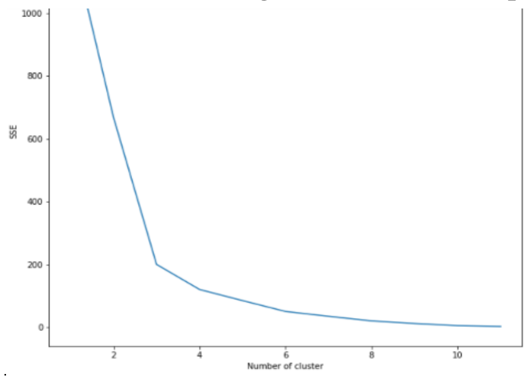


Fig 5.1: Plot on a time series with anomaly points highlighted (iForest) [9]

K-Means:

K-Means is a clustering algorithm that partitions n observations into k clusters. Here, each cluster formed would have a centroid. Every data point is identified to be in a cluster based on minimum distance from the centroid of each cluster. The distance metric followed here would be Euclidean distance. The identification of a point to a cluster is a convergence process with multiple iterations.

Considering we have three common states in our elevator data (flat, acceleration and deceleration), we decided to apply K-means on the data to see its effect. One of the hyper-parameters that we need to choose for this algorithm was the number of clusters to be used to efficiently find anomalies. To obtain this we used 'Elbow method'. This method uses the sum of squared errors obtained against different cluster ranges to arrive at an optimal number.



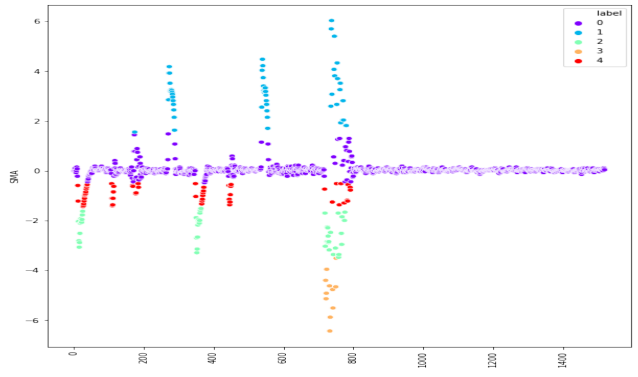


Fig 5.1: Elbow method to determine optimal clusters

Fig 5.2: K-means model classifying data into 5 clusters

Based on the above, it was decided to use 5 clusters and a threshold of 0.999 to classify whether a point was anomaly or not. While K-means was very successful in capturing anomalies related to abnormal acceleration, this method was not successful in capturing anomalies on streaming data. One of the primary reasons being that each window of streamed data would be converted to clusters of 5. Hence, whether an anomaly is present or not, this model would tend to produce a lot of false positives affecting the accuracy of the model.

DBSCAN:

DBSCAN[10](Density-Based Spatial Clustering of Applications with Noise) is another popular unsupervised learning method utilized in model building as part of this project. It requires two main hyper parameters: epsilon and minimum points to determine if a data point belongs to a cluster. If not, it is considered an anomaly. DBSCAN works by choosing one data point at a time and constructing a circle of radius epsilon around it. If data points greater than or equal to the minimum points are within the circle, the data point is called a core point. If there are at least one or more points inside the circle but not equal to minimum points, the data point is called border point. DBSCAN will continue this process until no other data points are nearby, and then it will look to form a second cluster. Any point which does not come under any cluster is represented by the cluster -1 and is considered an anomaly. In this project an epsilon of 1 and minimum points of 4 was used on a sample of data which provided promising results. However, there was not an apt way to evaluate the model on all the data. Apart from this, even though we could choose an optimal epsilon by considering the point in the histogram where the nearest neighbor distance converges, the minimum point's parameter depends on the distribution of data. Hence, this model was not quite effective for the problem.

ANN-Based Auto encoders:

The intuition of using auto encoders for anomaly detection was to train a model only on normal patterns, so when the model is exposed to a new instance that it has never seen before, the reconstruction error will be larger, and it will mark the sequence as an anomaly.

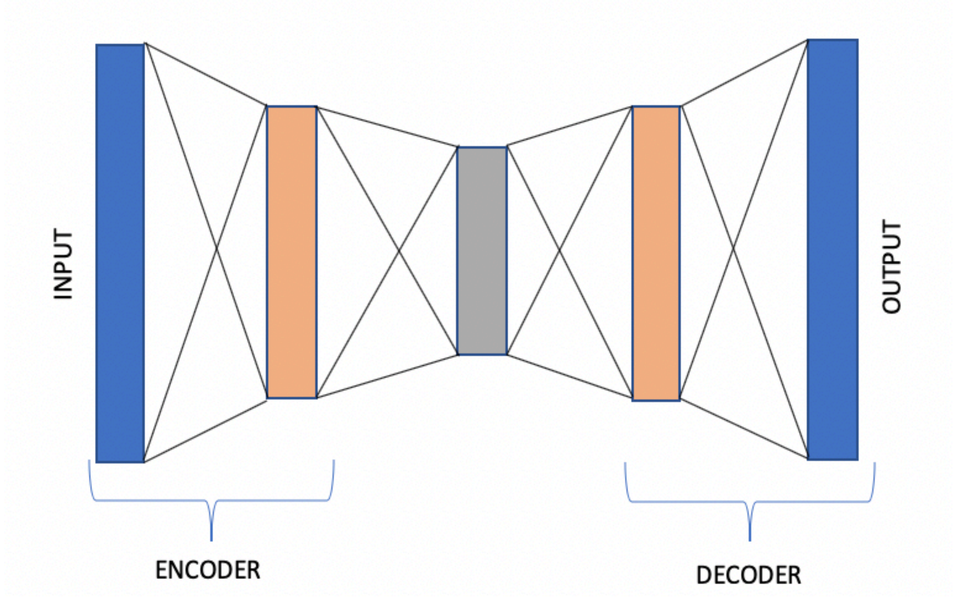


Fig 5.3 Structure of an ANN Autoencoder [12]

In our project, we have used a sparse auto encoder which has fully connected layers throughout the neural network. The neural network has been trained on batches of size 10 which was used to determine the number of neurons in the hidden layers.

The structure of the neural network is as follows

1. Input Layer: Takes acceleration values
2. Encoder: Two fully connected hidden layers of size equal to the batch size (10) and half of the batch size (5) with ReLu activations and L2 regularizations
3. Decoder: Two fully connected hidden layers of size equal to the half of batch size (5) and batch size (10) with ReLu activations and L2 regularizations
4. Output Layer: Outputs the reconstructed acceleration value

Since we are using fully connected layers, we wanted to make sure that weights that are not important are automatically set to zero. So, we used L2 regularizations on each layer.

Choosing the threshold for anomaly classification.

The threshold for anomaly detection was necessary to classify an acceleration value as normal or not. So, we used the distribution of mean absolute reconstruction error to fix the threshold. We chose the 95th percentile of mean absolute error on training set as threshold.

LSTM Auto encoder:

An LSTM Auto encoder [13] is an implementation of an auto encoder for sequence data using an Encoder-Decoder LSTM architecture. LSTM model is capable of dealing with sequential data as compared to regular auto encoders and might be a better choice for anomaly detection in sensor data. Normal patterns from accelerometer data across all the lifts are used to train the LSTM Auto encoder model. This is based on the premise that the encoder learns from the normal patterns and the decoder tries to reconstruct the same on the test data.

One LSTM layer each was chosen for encoder and decoder with 64 units each. Along with this feature, the memory of LSTM model helps in using the previous data points in predicting the current point. A time-step can be chosen to determine the number of previous data points to be considered for prediction. Based on trial and error, 5 time-steps were chosen for anomaly detection.

The given model is run on test data which had anomalies and the mean absolute error is calculated based on the prediction by the model and the actual test data values. If the error is greater than a certain threshold, the point is considered as an anomaly [12].

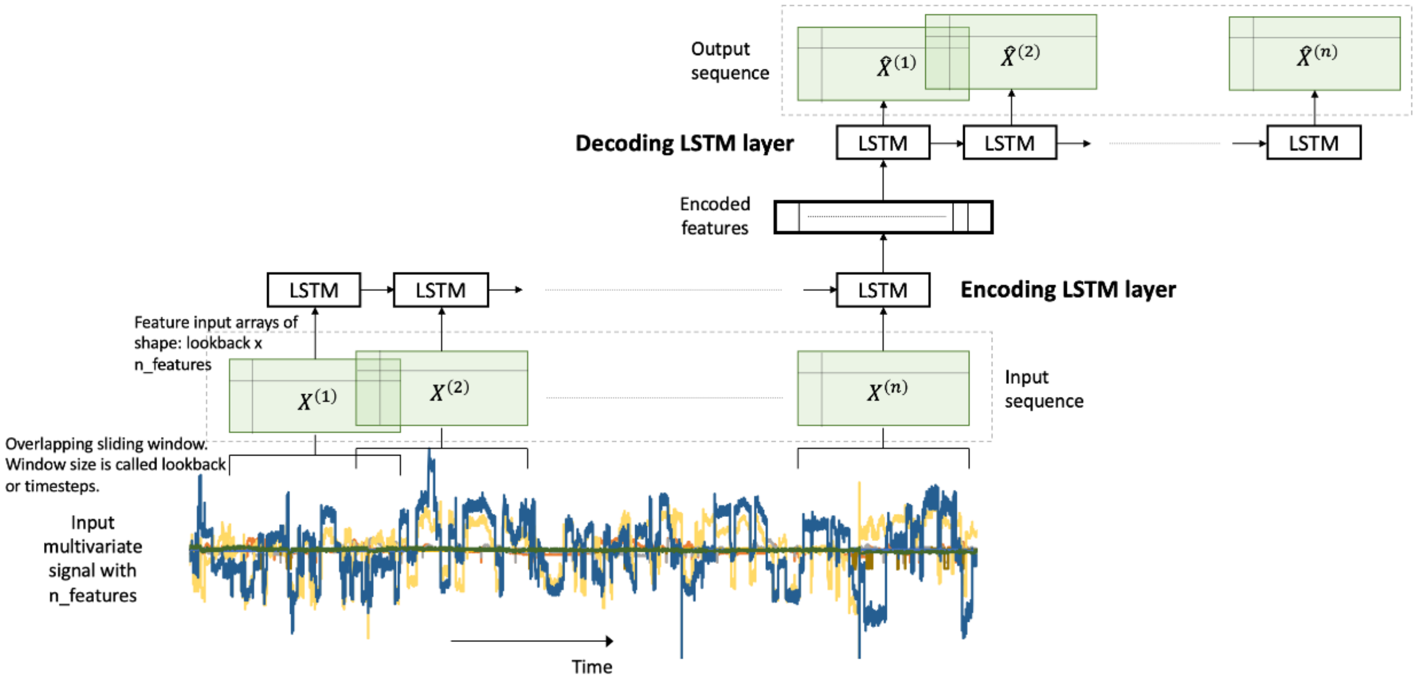

Output sequence
$\hat{X}^{(1)}$
$\hat{X}^{(2)}$
$\hat{X}^{(n)}$
Decoding LSTM layer
LSTM
Encoded features
Encoding LSTM layer
Feature input arrays of shape: lookback x n_features
$X^{(1)}$
$X^{(2)}$
$X^{(n)}$
Input sequence
Overlapping sliding window. Window size is called lookback or timesteps.
Input multivariate signal with n_features
Time

Fig 5.4: LSTM Auto encoder Architecture [14]

To set the threshold, the model is run on the training data and a histogram is plotted for the mean absolute error between the prediction on training data and the actual training data. This provides an intuition on the maximum limit of the error that the model can potentially make.

There are various methods to set a threshold such as considering a certain percentile value, using interquartile range or using z-score if the distribution is normal. However, for the project 95th percentile value of the error distribution was chosen after observing the data.

EVALUATION METRICS

As the 75GB of input data provided was unlabeled, there was not an infallible way to evaluate the machine learning models. To tackle this, we labelled a subset of the input sensor data manually with the help of a TSBC domain expert, public TSBC documentation and data visualization. For this a new Boolean field was added to the input data namely, ‘Anomaly’, which classified it as 1 if an anomaly and 0 if not.

This labelled data was used by the different machine learning models to make predictions. For each model, the confusion matrix was checked to find the true positives, true negatives, false positives and false negatives.

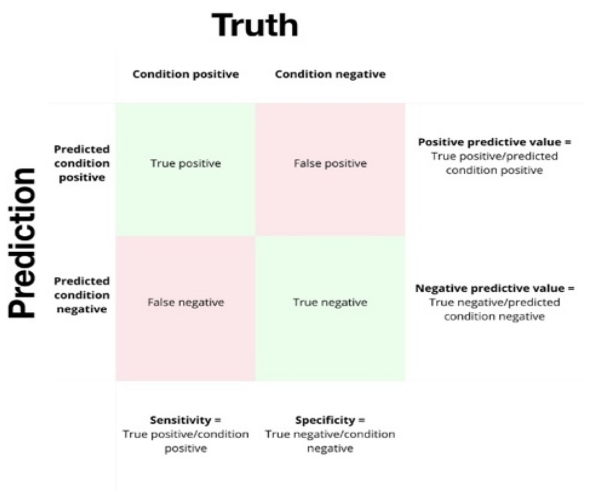

Truth
Condition positive
Condition negative
Prediction
Predicted condition positive
True positive
False positive
Positive predictive value = True positive/predicted condition positive
Predicted condition negative
False negative
True negative
Negative predictive value = True negative/predicted condition negative
Sensitivity = True positive/condition positive
Specificity = True negative/condition negative

Fig 5.4: Confusion matrix [15]

Precision and recall were calculated for each model using the below formulae.

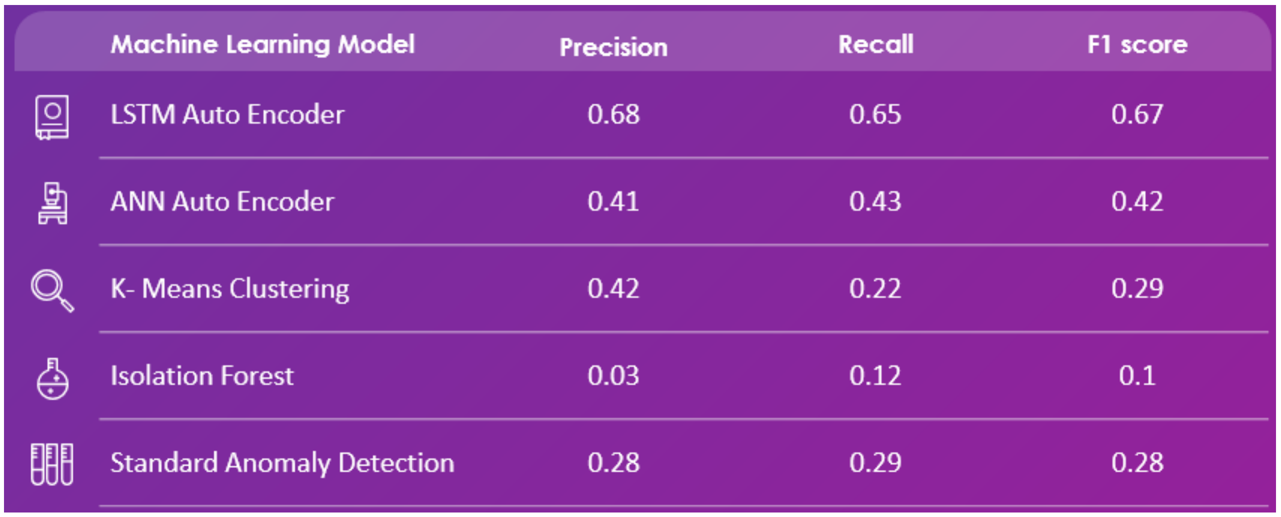
$$Precision = \frac{TP}{TP + \ FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Then, F1 score was calculated using the above metrics.

$$F1\ score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The model with highest F1 score was chosen for implementation. Below (**Fig 5.5)** are the results for the machine learning models.



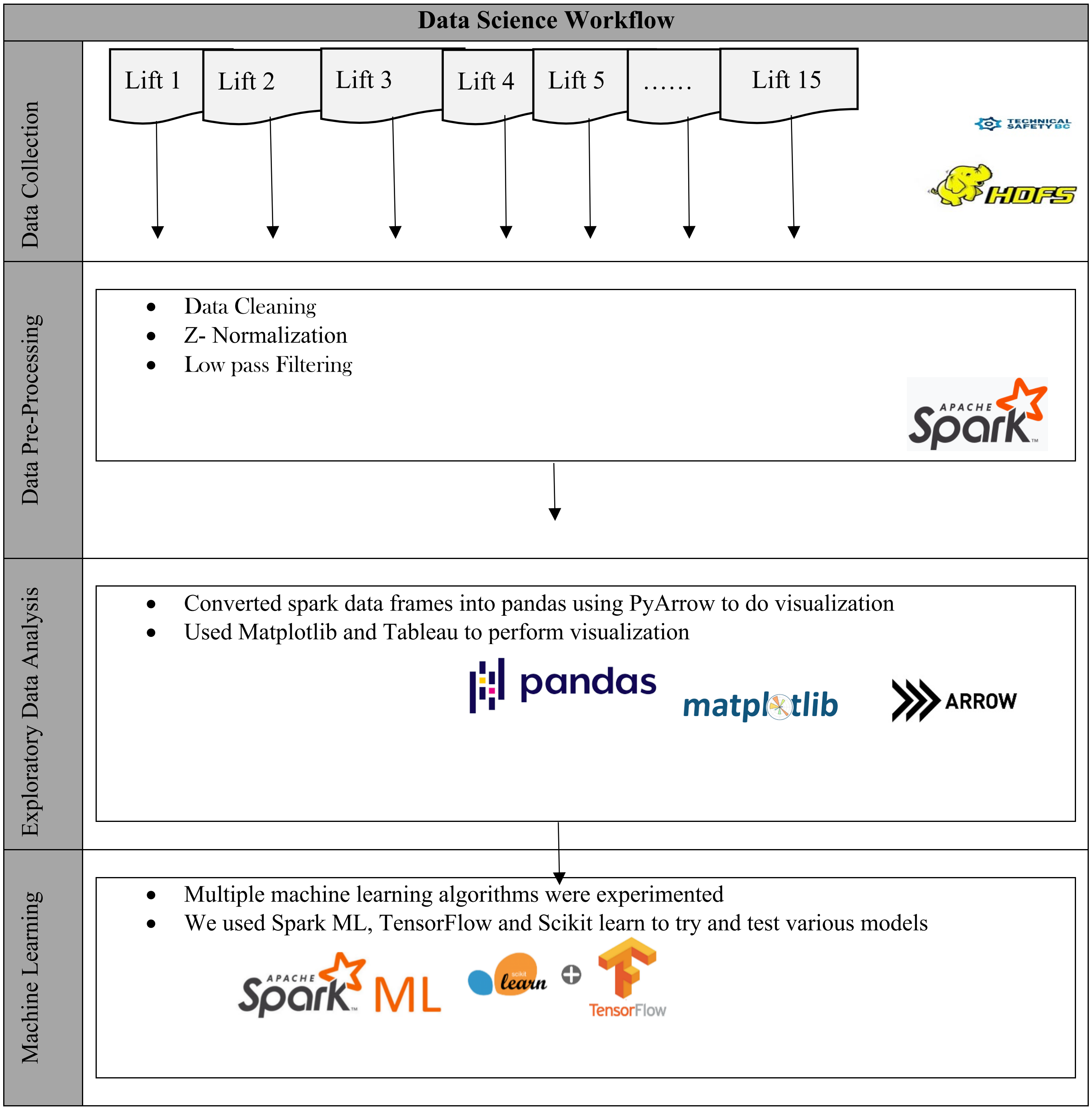
Machine Learning Model	Precision	Recall	F1 score
LSTM Auto Encoder	0.68	0.65	0.67
ANN Auto Encoder	0.41	0.43	0.42
K- Means Clustering	0.42	0.22	0.29
Isolation Forest	0.03	0.12	0.1
Standard Anomaly Detection	0.28	0.29	0.28

Fig 5.5: F1 Score

METHODOLOGY

Tools used in Data Science Workflow

A quick overview of our architecture diagram is provided below:



Data Science Workflow
Data Collection
Lift 1
Lift 2
Lift 3
Lift 4
Lift 5
……
Lift 15
TECHNICAL SAFETY BC
HDFS
Data Pre-Processing
Data Cleaning
Z- Normalization
Low pass Filtering
APACHE Spark
Exploratory Data Analysis
Converted spark data frames into pandas using PyArrow to do visualization
Used Matplotlib and Tableau to perform visualization
pandas
matplotlib
ARROW
Machine Learning
Multiple machine learning algorithms were experimented
We used Spark ML, TensorFlow and Scikit learn to try and test various models
APACHE Spark ML
scikit learn
TensorFlow

Fig 6.1: Architecture Diagram

DATA PRODUCT

Once we had the model ready, we needed to build a product that can serve the model. To accomplish that, we first created a Kafka service that consisted of 15 producers that simulates the data arriving from sensors. Similarly, 15 consumers were created as well to accept the data with all accelerometer information received as is. In-order to store the arrival of data we decided to use Amazon Dynamo DB. DynamoDB is a NoSQL database that is easily deployable in cloud services making it much easy to scale it to big data needs.

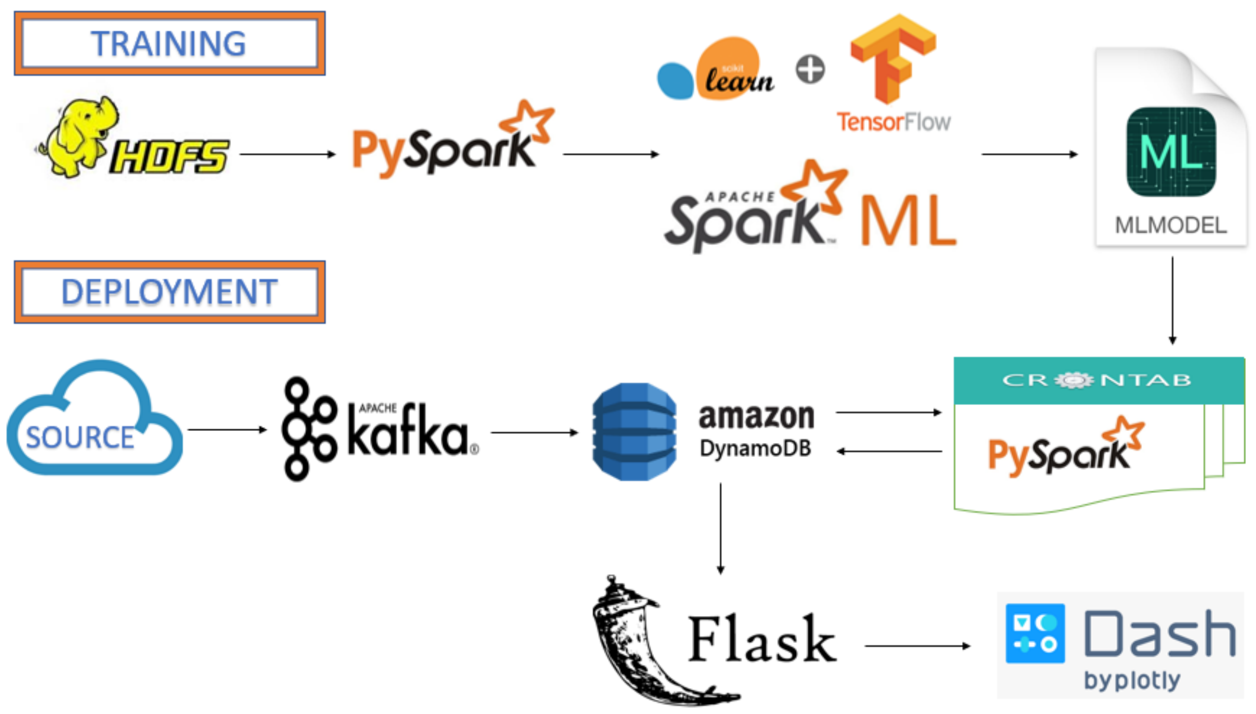


Fig 7.1 Data Product

Three tables were created in DynamoDB to handle the storage mechanism. The structure is depicted in the below diagram:

Table - Input	Table - Elevator	Table - Restart
• Hash Key - Elevator ID	• Hash Key - Elevator ID	• Hash Key - Elevator ID
• Sort Key - Index	• Sort Key - Index	• Attribute Item
• Attribute Items	• Attribute Items	• Restart Index
• X	• Z	
• Y	• timestamp	
• Z	• prediction	
• timestamp		

Fig 7.2 DynamoDB Tables

The input table (Fig **7.2)** described shall store the information as is from Kafka consumers. Once the data is stored in table - Input, we had setup a CronTab script that shall trigger every 5 minutes. This script triggers the prediction part of our product that loads the saved model and data from input table to classify a point as anomaly or not. Once the predictions are generated the data is loaded into Table named Elevator with up to data. Since this job runs on a cyclic fashion, we ensure to keep Restart table that stores the index of record until which the predictions have occurred. This setup guarantees that already processed records are not reprocessed again.

Additionally, a single prediction step would take 5 * 60 * 26 points for prediction ensuring all 5 minutes data are covered. The predictions that are loaded on the Elevator table is passed to Flask web app engine and then plotly dash to visualize the predictions. A snapshot of dashboard shall look like below

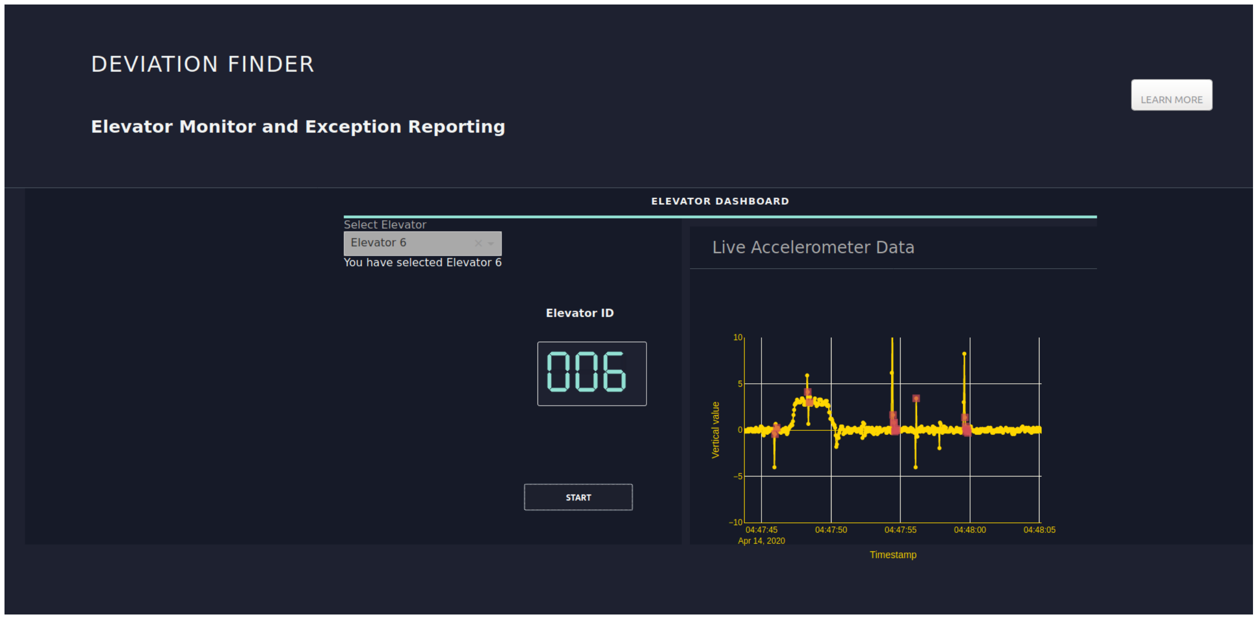


Fig 7.3: Snapshot of Dashboard

As depicted in the picture this dashboard supports the view of sensor data across 15 lifts. Additionally, you can also pause the streaming in between and switch to other lifts. You can also see a red point denoting the anomalies that has occurred in the stream.

LESSONS LEARNT

This project gave us an opportunity to design a predictive maintenance system using elevator sensor data which is challenging and interesting. From the very beginning of the project, we faced a lot of challenges because of the enormous dataset we had. However, the knowledge gained through previous courses helped us in solving them. We performed exploratory data analysis to understand the various normal and abnormal patterns in the dataset. During which we also learnt about de-noising data using different low pass filters.

This project gave us an opportunity to work on feature engineering in time series dataset. We also did a thorough research on solving the problem to come up with feasible unsupervised learning models. We gained an in-depth understanding of their working and brainstormed how to use them to solve our problem. These phases helped us improve our research and interpersonal skills. We also learnt how to evaluate unsupervised models. Considering the deployment stage of the project, we wanted to simulate a real-time streaming system and wanted to gain some exposure of cloud. So, we learnt about setting up a periodic Cron job to fetch data from DynamoDB and pass the results to Plotly dashboard via flask component to help elevator maintenance. Overall, the project prepared us in gaining technical expertise that are crucial for a data scientist.

SUMMARY

Summarizing the project, we aimed to build a real-time streaming system that can help elevator manufacturers or safety regulating companies like TSBC in predictive maintenance of elevators. In this regard, we were able to successfully develop a data product that can identify anomalies in real-time data. We were able to build models of varying performance, but LSTM Autoencoder performed with a F1-score of 67% which we have used in our data product. Secondly, our aim was to use a fast and scalable platform to accommodate the growing data. In this direction, we have used Amazon DynamoDB Services to make sure our product is scalable. All in all, this project created an awareness among us in understanding public safety in elevators.

There are a lot of future enhancements that can be done to this project. This product is aimed at alerting the user regarding the occurrence of an anomaly. But this can pave way to identify various causes for potential elevator breakdown in maintenance stage. The anomaly might have happened due to a shaking elevator, power failure, noisy bearings, worn sheaves and many others. Being more specific about what caused an anomaly will make the maintenance process efficient. Also, levelling is one of the important issues in elevators. This problem happens when the elevator car doesn't level perfectly with the building floor. So, predicting the levelling distance can be helpful in solving levelling issue.

REFERENCES

[1] Harnessing the Power of Artificial Intelligence (AI) and Internet Of Things (IoT) for Elevating Safety, TSBC [Online]. Available: https://www.technicalsafetybc.ca/safety-data/iot-elevating-safety.

[2] Christ, M., Braun, N., Neuffer, J. and Kempa-Liehr A.W. (2018). Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh -- A Python package). Neurocomputing 307 (2018) 72-77.

[3] "Mohammad Riazi et.al . "Detecting the Onset of Machine Failure Using Anomaly Detection Methods"".

[4] "Ivan Miguel Pires et.al. "Pattern Recognition Techniques for the Identification of Activities of Daily Living using Mobile Device Accelerometer"".

[5] Fei Tony Liu, Kai Ming Ting Gippsland School of Information Technology Monash University, Victoria, Australia {tony.liu},{kaiming.ting}@infotech.monash.edu.au.

[6] Zhi-Hua Zhou National Key Laboratory for Novel Software Technology Nanjing University, Nanjing 210093, China zhouzh@lamda.nju.edu.cn.

[7] SPARK-Iforest Package, [Online]. Available: https://github.com/titicaca/spark-iforest#references.

[8] [Online]. Available: https://towardsdatascience.com/isolation-forest-and-spark-b88ade6c63ff.

[9] [Online]. Available: https://towardsdatascience.com/anomaly-detection-with-isolation-forest-visualization-23cd75c281e2.

[10] "Hossein Saeedi Emadi et. al. "A Novel Anomaly Detection Algorithm Using DBSCAN and SVM in Wireless Sensor Networks"".

[11] "Krishna Mohan Mishra et. al. Elevator Fault Detection Using Profile Extraction and Deep Autoencoder Feature Extraction for Acceleration and Magnetic Signals".

[12] A. Borghesi, A. Libri, L. Benini and A. Bartolini, "Online Anomaly Detection in HPC Systems," *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Hsinchu, Taiwan, 2019, pp. 229-233.

[13] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, Gautam Shroff - LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection: https://arxiv.org/pdf/1607.00148 [2016].

[14] [Online]. Available: https://towardsdatascience.com/lstm-autoencoder-for-extreme-rare-event-classification-in-keras-ce209a224cfb.

[15] [Online]. Available: https://radiopaedia.org/cases/confusion-matrix-3.