Analysis and Enhancement of Supervised Failure Models for industrial Time-series data with limited labels: Final Report

Date: 7/7/2019

Team Composition

Name	Role	Contacts
Uptake		
Felipe Parages	Senior Data Scientist	felipe.parages@uptake.com
Brian Silva	Senior Data Science Manager	brian.silva@uptake.com
Illinois Institute of Technology		
Divya Soopal	Data Scientist	bsoopal@hawk.iit.edu
Shikha Saxena	Data Scientist	ssaxena13@hawk.iit.edu
Eric Ballam	Data Scientist	eballam@hawk.iit.edu

Project Background

Business Overview

Uptake provides SaaS (Software as a Service) platform using industrial AI to deliver fast financial outcomes that impact their client's business. This includes lowering client's Operation & Maintenance costs, improving their asset reliability, and providing new findings and efficiencies from client's operational data. Uptake deals with machine-level time series data such as sensor data, maintenance records, mechanical data and contextual data for industrial applications.

Problem Statement

In industrial applications, unplanned equipment downtime and service interruptions can have a large financial impact on business performance. Supervised machine learning is an effective tool in predicting and preventing these failure instances.

There are two types of data that are primarily used for this task:

- sensor and performance data that represents the normal operation of the machinery.
- maintenance records that cover how and when failure occurs.

With full access to well-maintained dataset, effective models can be built to predict these failure instances. However, such datasets are not the norm. Companies are often unwilling to release all their operational data for analysis. Maintenance records are often poorly maintained. Additionally, failure instances are rare which creates a large class imbalance and makes building models problematic.

Problem Objective

In the project below two objectives were delivered:

Objective 1

Standard supervised models for the dataset were build. Once created we then explored how the performance of the models degraded as failure labels were removed.

Objective 2

Unlabelled signal data was leveraged by using it to train Autoencoders in order to generate estimated features and use them to improve the performance of a supervised model trained with labels.

Data Source

The data being used by us in project is openly available at Backblaze . Backblaze data center daily collects snapshots of each operational drive.

Data Description

Uptake deals with two kinds of time-series data when it comes to prediction models:

- **Signal Data**
  It is time stamped data from sensors, used as inputs or features for failure models. In our case, signal data is overall data collected over 6 years from 2013 to 2018.
- **Maintenance Data**
  These are the records when machine malfunctioned in the past. Currently, the failure column is interpreted as maintenance records available for labelling in Supervised ML classifiers.

Data Size

Approximately 36 GB of storage capacity, our dataset therefore contains around 165 million records with each year divided into four quarters. Each quarter holds data for 3 months time period.

Data Type

These snapshots are then processed and stored in stats file in form of CSVs. Each day CSV file contains 85-129 columns and 40,000 to 1,00,000 records which indicates the stats related to operational drives that day.

The label data for this dataset is called Failure. It's represented as a 0 or 1, where 1 indicates the last day a drive was operational before failure. There are 4 other identification fields (data, serial number, model and capacity in bytes).

The remaining fields represent SMART stats data. SMART in this instance stands for self-monitoring, analysis and reporting technology. They are a standard set of data points collected for hard drive monitoring. For each SMART stat there is a raw and normalized field.

For validation/hold-out set, we will be using 2018(all quarters) and 2019(Q1- first quarter) data which contains approximately 44 million records. This will leave the bulk of our data, 2013 - 2017, for modeling. To create the train / test split we will use stratified sampling to ensure that an appropriate number of failure events exists in both our training and testing datasets. We will investigate multiple methods of carrying out this sampling including simple sampling on the labels and a time-based sampling.

Project Complications and Constraints

Project Constraints

For this project one major limitation was our programming environment. The RAM space available to us was simply not large enough, as such the amount of data we could work with at one time was severely constrained.

We submitted initial request with writeup to Uptake regarding cloud resource but unfortunately they did not have any resource free, then we also submitted request with detailed project description for Chameleon cloud resource to IIT, we did get access during the second month of project but it had no VM with good amount of space and computation power free.

Therefore, our computational power limited the amount of work we could accomplish. Running the models, even with our limited datasets, was very time consuming and as such we were only able to run a few trials. To improve this project a cloud computing resource would be necessary.

These constraints cost us a lot of time and we simply were not able to connect Objective 1 (Supervised Modeling) with Objective 2 (Anomaly detection using Autoencoders) and produce collaborated results for the two objectives. Any further work on this project should start by connecting the work of the two objectives.

Project Complications

We had availability of large dataset for the project from year 2013-2019, with each month containing around million of records and in total 36 GB of data but were continuously facing lack of computation resource. Therefore in the beginning of the project we decided to split our data year wise with two years each member for EDA and compare the relevant features we obtain.

Moreover, while we were performing data exploration we realized that we may not have enough time to complete both objectives. Hence we decided that Eric and Divya will work on Objective-1(Quantify degradation of Supervised Model performance) and Shikha will work on

Objective-2(Leverage Unlabelled data to improve Supervised Model performance) in order to achieve substantial results with respect to both the objectives by the end of project.

Again under the mentioned project constraints, we had to proceed with our objectives using a subset of around two years of data for each objective instead of whole data of all years.

Project Deliverables

Description of Deliverables

The deliverables contain the following :

- Project Scope Document
- Github Repository containing the code
- Reports of EDA, Supervised Models and Autoencoders
- Project Report

All the above deliverables and results are shared with Felipe Parages through mail as well as explained verbally in weekly meetings and presentations at Uptake.

Availability of Deliverable

The GitHub repository where all the code related to our project is located is as follows:- https://github.com/fparages/iit-uptake-capstone

Project Execution

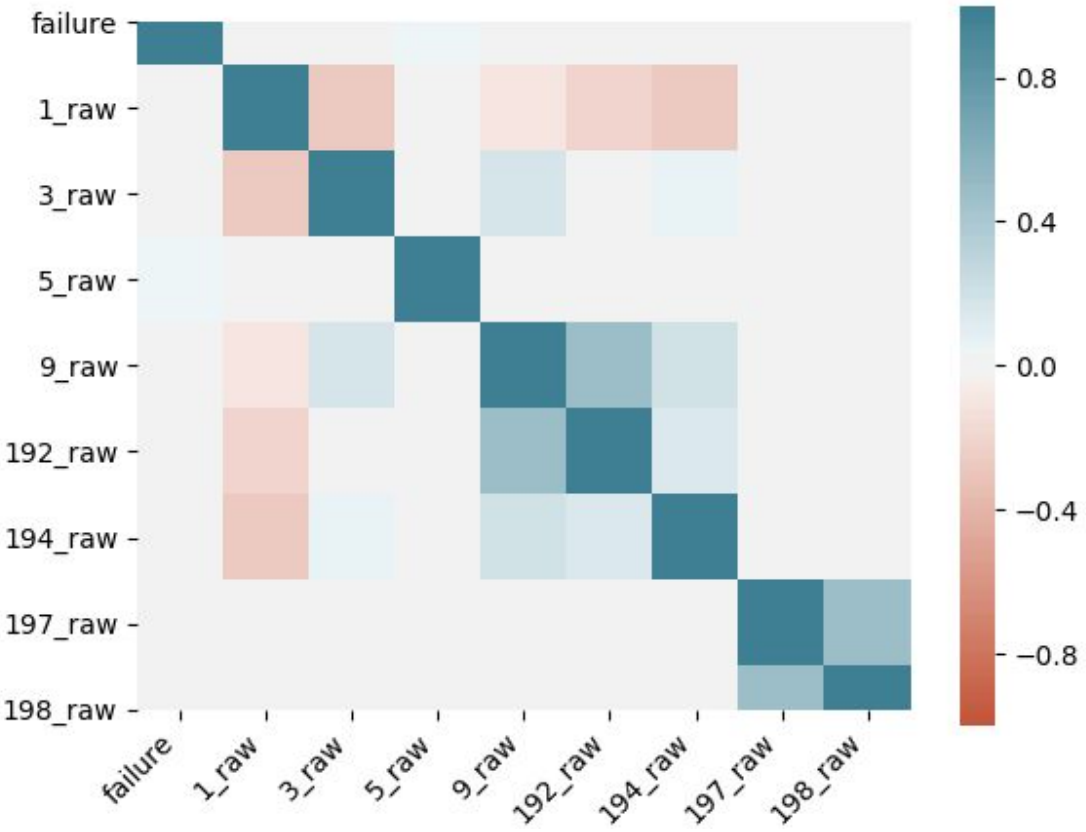
Supervised Modeling(Objective 1)

EDA

We use 2015 dataset for initial exploratory data analysis. We had 95 columns in the original dataset. After dropping all the columns with empty values, we had 43 columns left. Cleaning the dataset further, we were left with 9 smart stats values alongside the failure value that were significant when results were compared from other datasets.

```
Index(['failure', '1_raw', '3_raw', '5_raw', '9_raw', '192_raw', '194_raw',
       '197_raw', '198_raw'],
      dtype='object')
```

To understand the relationship between the different smart stat values with the failure, we plot a correlation matrix as follows:



We explore the correlation of each of these smart stats against failure. We do notice a closer relationship among the smart stats: 9, 192, 194, and 197. Also, the statistical significance of all these smart stats are clearly positive (blue squares outnumbered the red squares) from the correlation matrix. Hence, these features were used for further model predictions.

Tasks Completed

Initial Modeling:

Standard machine learning algorithms make the assumption that the data points fed into them are independent from each other. For most use cases this is a good assumption but not for our

project. The Backblaze data tracked a daily snapshot of each hard drive SMART stats. The possibility of failures being dependent on a previous days value wasn't something we could ignore. To investigate this, we came up with two different time series approaches to compare against a Naive model (one where we assumed the data didn't have a time series relationship) and a Labeled model.
For all the following exploratory models we used a date based train/test split of 6 months and the following one month from the 2014 data was used. The 2014 data was chosen because it had the fewest tracked SMART stats and the fewest hard drives while still being reasonably complete. This allowed us to run the models without having to worry about reaching memory cap on our local clients. To this end we attempted to use two simple models, logistic regression and Random Forest (at 15 trees). However logistic regression was unable to produce meaningful models so Random Forest was the chosen model.

The first time series approach we tried was Back Labeling. In this approach we relabeled a number of days before a failure as failures. This method dramatically increased the number of failure events in our dataset and allowed our models to look for trends in the days before a failure. The other time series method we attempted was Feature Extraction. Instead of labeling the days before as failures we simply created new features using the previous days values. For this method we used a rolling average of the previous days values and an average of the rate of change between days.

Once the models and labeling / feature extraction code were set up, we then tested out different methods for under and oversampling with our labels. We tested out random sampling to remove the normal class as well as some more involved methods such as SMOTE and Tomek Links. For our purposes SMOTE and Tomek Links were used in conjunction. Tomek Links helps create a larger class separation by identifying nearest neighbor pairs of differing classes and removing the item from the majority class where SMOTE is a method of creating artificial data from the minority class.

Initial model KPIs:

In order to compare the initial models, we used Precision and Recall however we needed to make an important distinction between our pointwise and eventwise values. The pointwise values are simply the ones generated from the standard confusion matrix. These values didn't capture exactly what is happening in our models. Because of the way we are approaching our time series data we ended up adding or removing labeled points from our dataset. To account for this we needed to compare our predicted labels from our model to our data original labels. If our model flagged a failure within the given time window before a failure in our original data then we consider that failure successfully predicted. Summing all the failures correctly predicted in this way gave us our Eventwise Accuracy. The following table details the results of our initial modeling:

Model (Random Forest 15 trees)	**Precision**	**Recall**	**Eventwise Accuracy**
Naive	0.989	0.627	0.627
Naive SMOTE Tomek	0.928	0.758	0.758
Naive Random Sampling	0.990	0.654	0.65
Labeled 3 days	0.898	0.501	0.881
Labeled 3 days SMOTE Tomek	0.851	0.544	0.920
Labeled 3 days Random Sampling	0.636	0.575	0.850
Labeled 5 days	0.900	0.511	0.718
Labeled 5 days SMOTE Tomek	0.825	0.583	0.797
Labeled 5 days Random Sampling	0.669	0.495	0.670
Features 3 days	1	0.738	0.823
Features 3 days SMOTE Tomek	0.978	0.706	0.854
Features 3 days Random Sampling	0.926	0.698	0.775
Features 5 days	0.989	0.715	0.848
Features 5 days SMOTE Tomek	1	0.754	0.860
Features 5 days Random Sampling	0.942	0.754	0.793

A lot of trials were run making the above table a little hard to read but there are a few things we can immediately takeaway from it. First is that both Back Labeling and Feature Extraction out performed our Naive model which is encouraging. Also it is pretty clear that SMOTE Tomek sampling did improve our models. However running these methods were very time consuming.

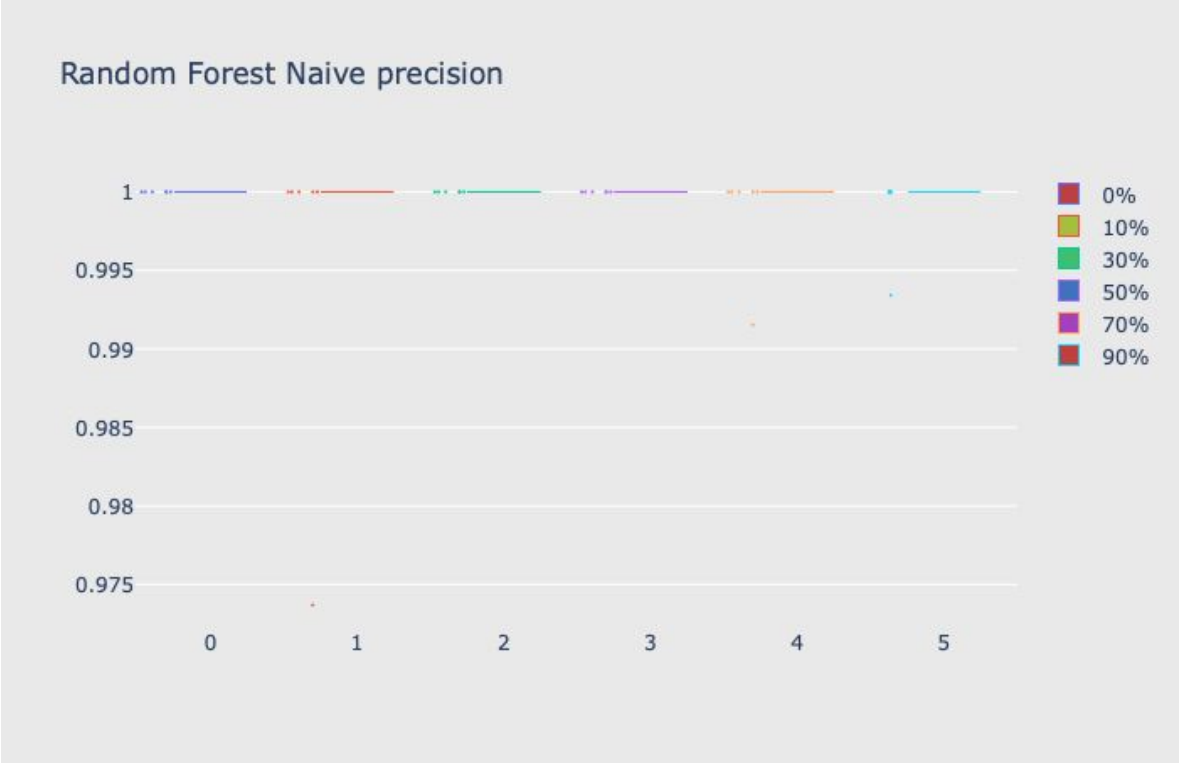
Finally it also appears that our 3 days back methods out performed the 5 day models. It should be noted that models were also ran at 1, 7 and 9 days back but they performed significantly worse than the models listed here.

Modeling

Using the lessons learned from the initial modeling we decided to set up our trials using a 3 day window for both our Back Labeling and Feature Extraction models with SMOTE Tomek. For the final modeling we would be running each of our models across several train / test splits, for each train test split we would be removing failure data points to see how the model performance changed as a result. One trial would be run using all the failure data points, then 90%, 70%, 50%, 30% and finally 10%. The data points to remove were selected at random and then the failure as well as the 3 days before they were removed to ensure all signal from that failure even was removed.

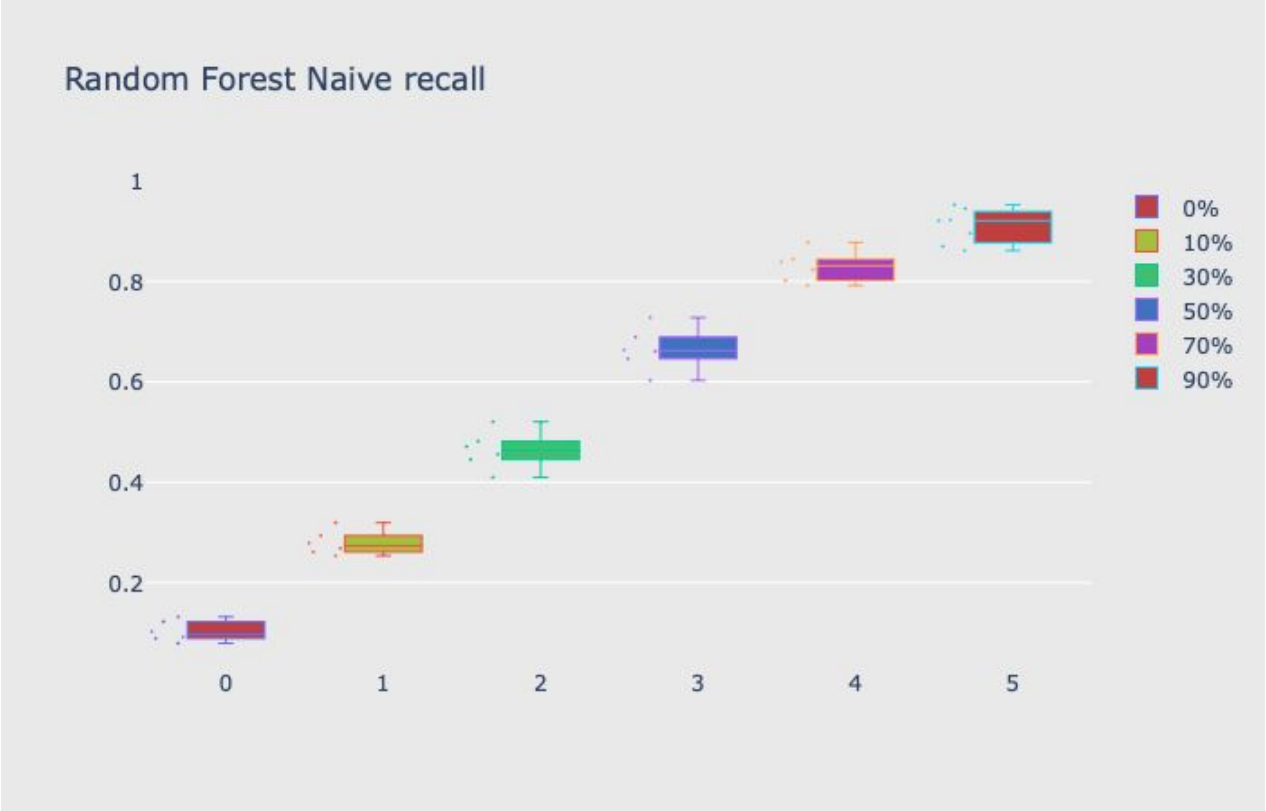
The following three charts cover the results for the Naive Random forest method. The charts show the Precision, Recall and Eventwise Accuracy for all of our test sets as we removed the failure points. From left to right the bar charts show 90% removal, 70%, 50%, 30%, 10% and finally the results with all the failure points included.

Random Forest with Naive Implementation



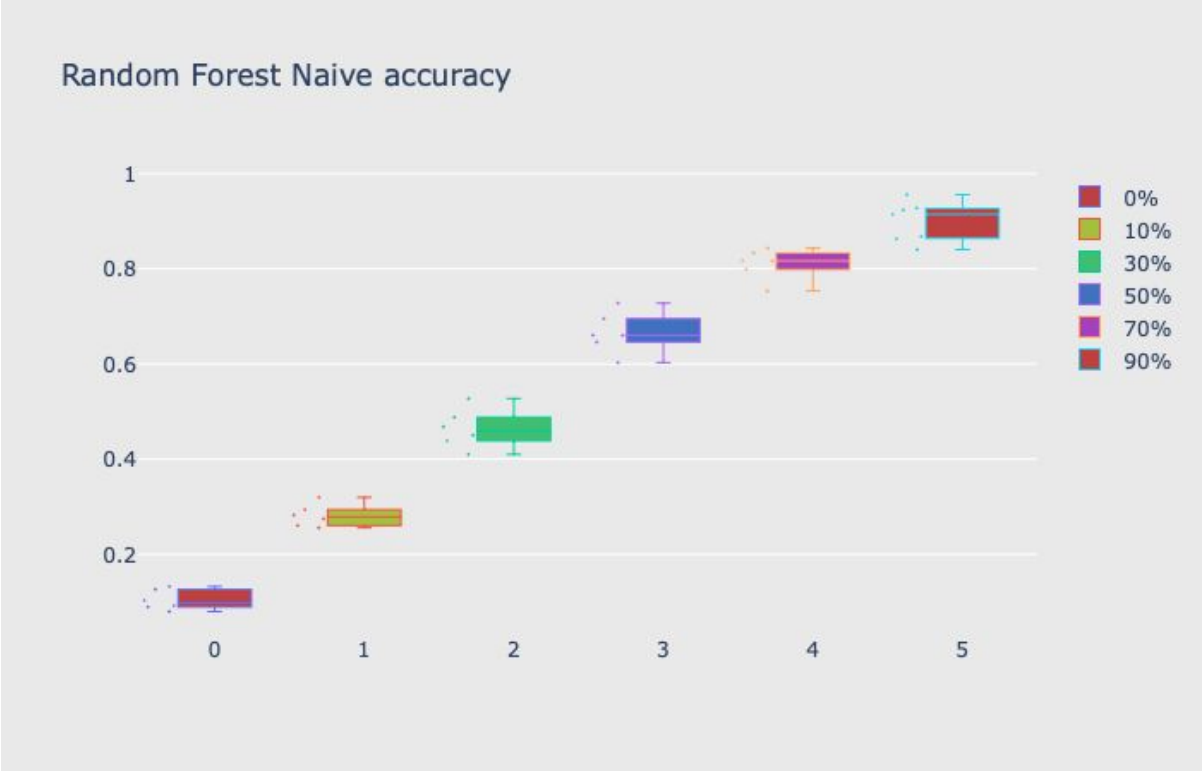
Random Forest Naive Precision

Here we can see that the Naive Random Forest method returned exceedly high Precision. This is because the method was not returning any false negatives. Thus, our Precision was nearly 100%.



Random Forest Naive Recall

Recall was a better indicator of our models performance. Here we can see how the Recall changed as labels were removed. Again the bar chart labeled 5 shows the results with all labels. Recall was very high with all labels and fell off steadily when labels were removed. The largest drop is between 3 and 2 (30% and 50% of labels removed). The lowest point is at 1 (90% of labels removed) where the Recall for most Train / Test splits was near 0.

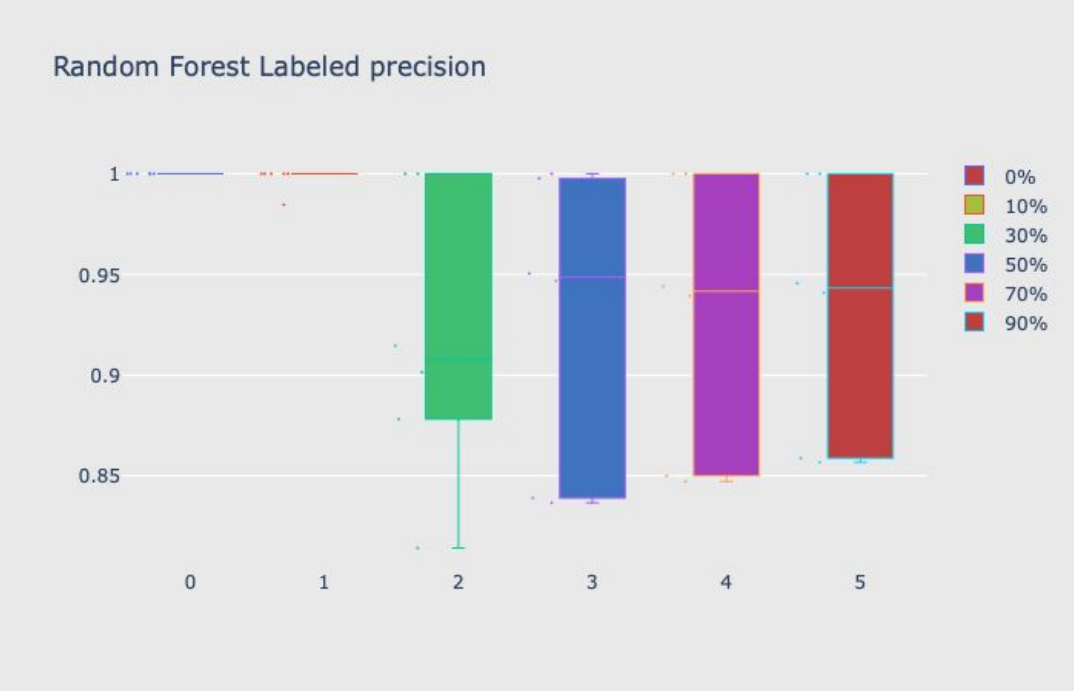


Random Forest Naive Accuracy

For the Naive method, Recall and the Eventwise Accuracy shown here were virtually identical. This is because without applying either our Backing Labeling or Feature Extraction methods for the pointwise Recall and Eventwise Accuracy are measuring the same parameters.

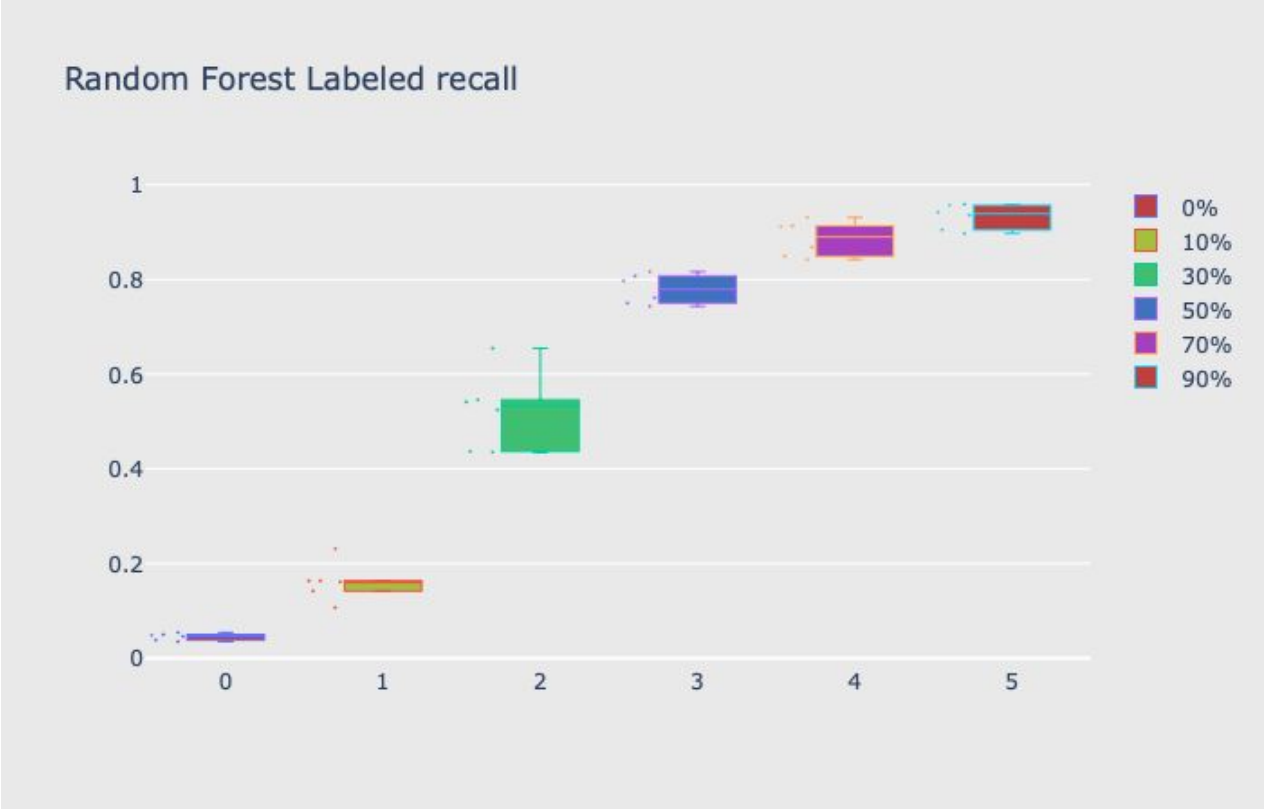
Random Forest with Back Labeled Implementation

The next three charts cover the results for our Back Labeled Random forest method. For this method we labeled the three days before each failure as failures. The charts show the Precision, Recall and Eventwise Accuracy for all of our test sets as we removed the failure points. From left to right the bar charts show 90% removal, 70%, 50%, 30%, 10% and finally the results with all the failure points included.



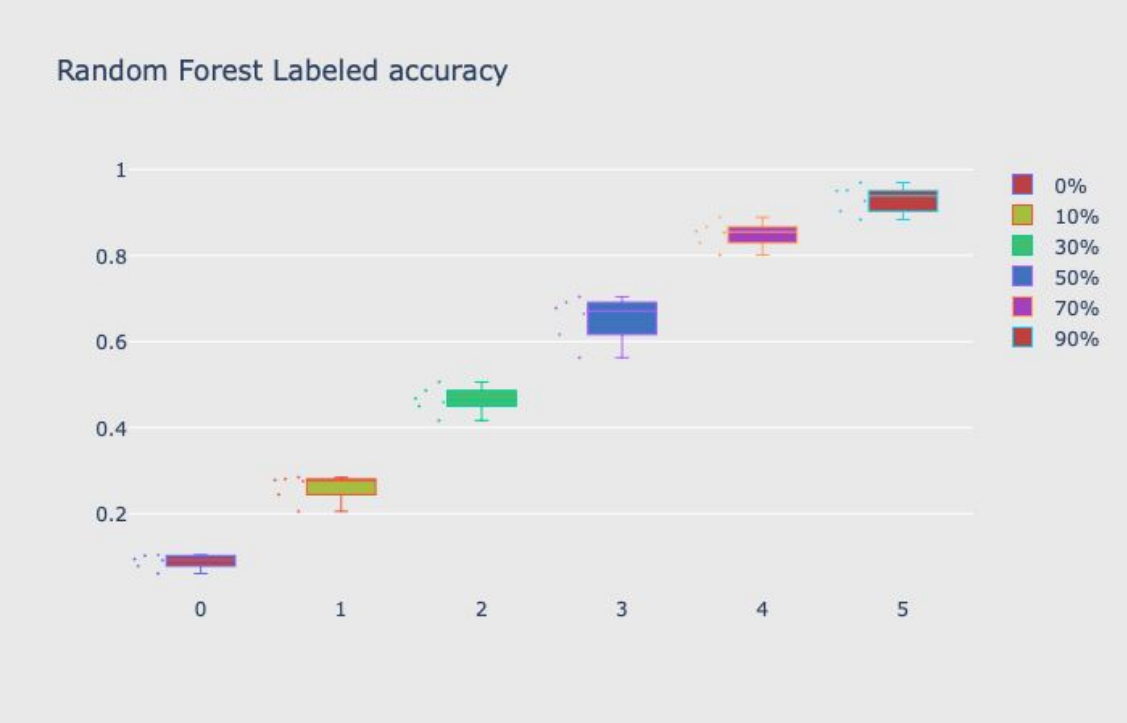
Random Forest Labeled Precision

This chart shows the Precision for our Back Labeled method. Once again we see that Precision was very high. This method produced more false negatives then Naive Random Forest but only when most of the labels were left in the data set. As we can see the method had perfect Precision once 70% or more of the labels were removed.



Random Forest Labeled Recall

Again Recall proved to be a better metric then Precision. As in our Naive method we can see Recall dropped as labels were removed. Once again, we see a significant decrease between 30% and 50% removed points, however unlike the Naive method the Labeled Recall decreased slower.

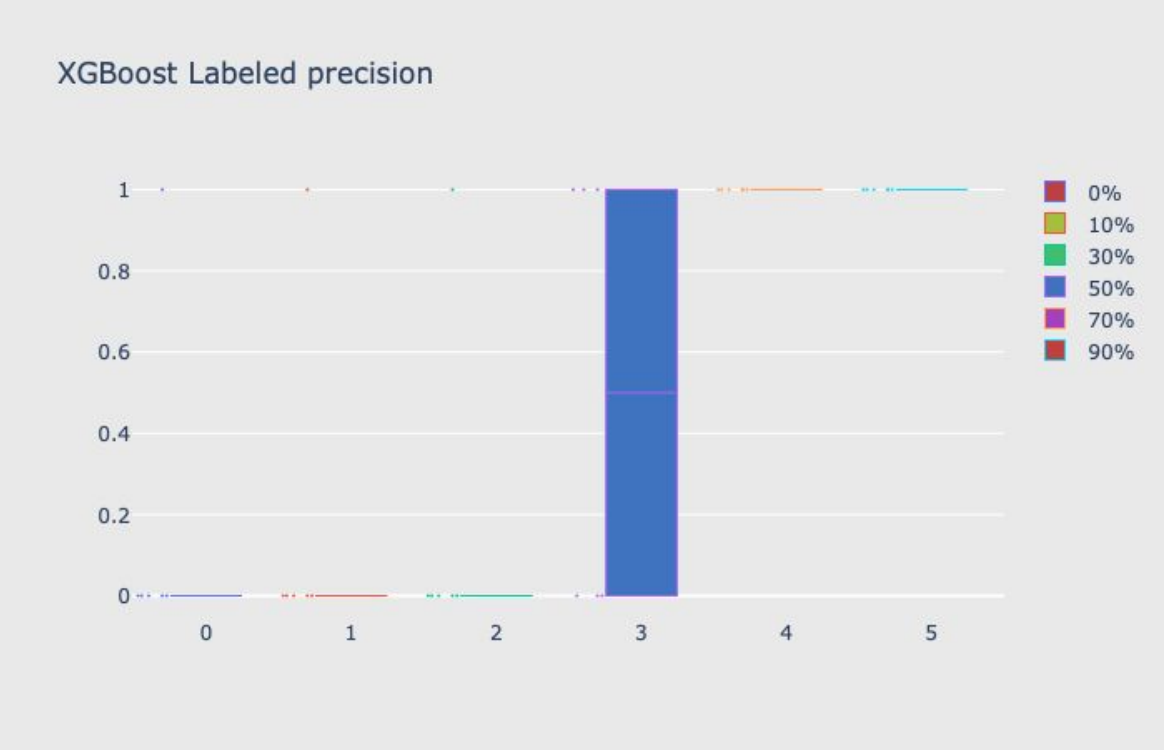


Random Forest Labeled Accuracy

For this method Pointwise Recall and Eventwise Accuracy were not the same thing and we can see the differences in this chart. While the Recall of this method didn't fall off linearly the Eventwise Accuracy did. This shows that while the method was able to predict individual points well was failures were removed it's ability to predict actual failure events did suffer.

XGBOOST with Back Labeled Implementation

The next three charts cover the results for our Back Labeled XGBoost method. For this method we labeled the three days before each failure as failures. The charts show the Precision, Recall and Eventwise Accuracy for all of our test sets as we removed the failure points. From left to right the bar charts show 90% removal, 70%, 50%, 30%, 10% and finally the results with all the failure points included.

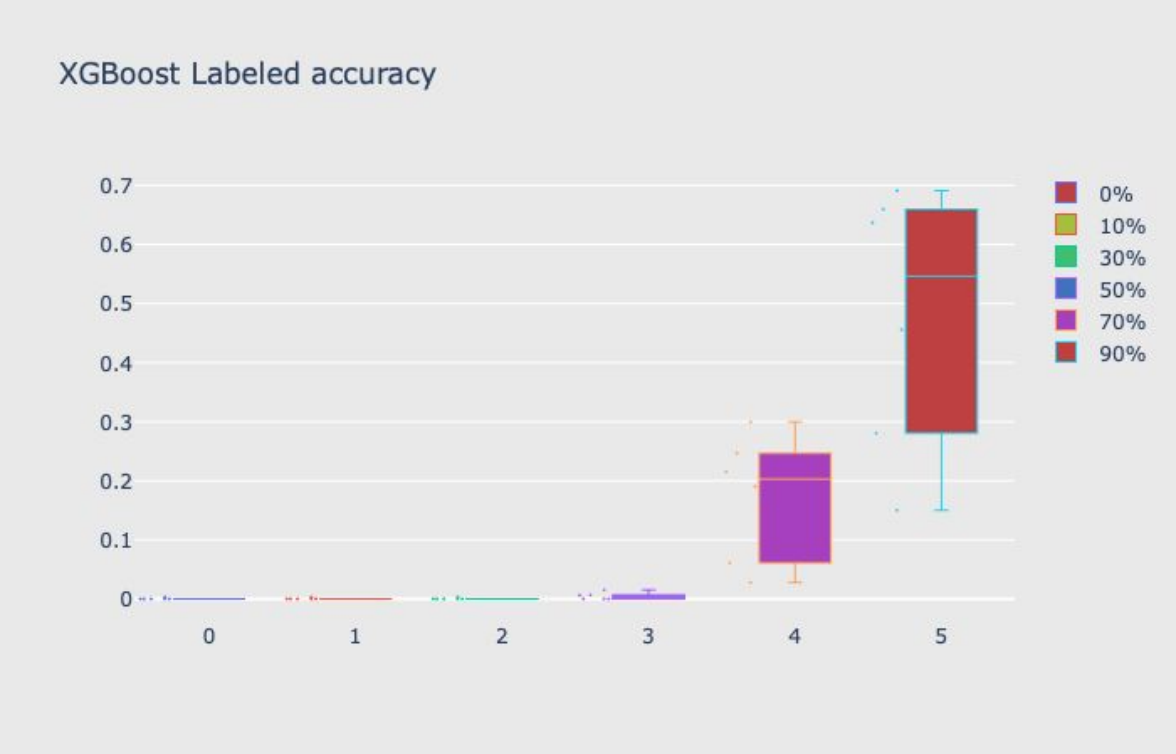


XGBoost Labeled Precision

Here we can see Precision for the Back Labeled XGBoost method. For bars 5 and 4 (0% and 10% of labels removed), the XGBoost method had near perfect Precisions. However once 30% or more of the labels were removed the Precision fell drastically. The three bar charts on the right are showing 0 Precision. This is because this method didn't predict any failure events under those conditions.

XGBoost Labeled Recall

Here we can see the Recall for Back Labeling. Once again we can see a clear divide between where the method was performing better and when it failed to predict any failure events. Once more then 10% of the failure labels were removed this method simply wasn't able to make any correct predictions. Even with most of the labels in the dataset the Recall for this method was quite low.



XGBoost Labeled Accuracy

As expected with a low Pointwise Recall the Eventwise Accuracy was also low. However it should be noted that Eventwise Accuracy of this method is considerably higher then the Recall. This is because of the Back Labeling. Remember that if the model predicts a failure upto 3 days before the actual failure event then we consider that failure event correctly predicted. Thus the method is clearly missing most of the labeled failure events but it is still getting a few and thus predicting a fair number of failure events.

Due to computational constraints we were not able to run our Feature Extraction method. We also didn't display the Naive XGBoost results because of how poor they turned out to be. In all trails across all methods Naive XGBoost was only able to predict a few failure events.

As such, the Back Labeled Random Forest Method clearly performed the best. It had the highest Recall and Eventwise Accuracy with all failures included and those values dropped the slowest as failures were removed.

Autoencoders (Objective 2 - Leverage Unlabelled data to improve Supervised Model performance)

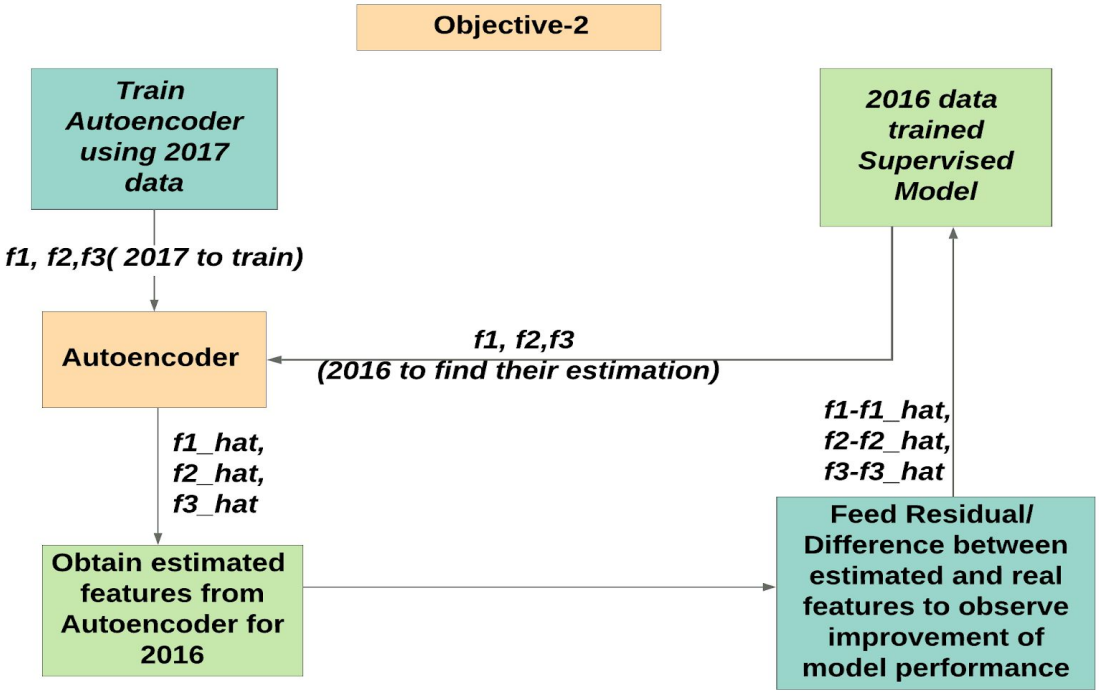
Problem and Data Pipeline

The problem was that signal data is available of equipment but maintenance data is not. Hence we need to leverage the left unlabeled signal data to improve performance of supervised model classifier predicting failure of equipment.

In our project, maintenance data is Failure column in drive data, and that is assumed to be limited for this objective. First Autoencoder is trained to over normal behaviour of operational drive data by dropping the rows with label of Failure (label 1). This data which only represents normal behaviour of drive is used to train Autoencoder.

Then Supervised Classifier (Random Forest) is trained over features on other dataset to predict failure of drives. Actual features used in Supervised Classifier is then fed to trained Autoencoder to estimate features, calculate residuals by subtracting actual features and estimated features. Now, new dataset which contains actual features along with all the residuals is fed to Supervised Classifier and changes in its performance is observed.

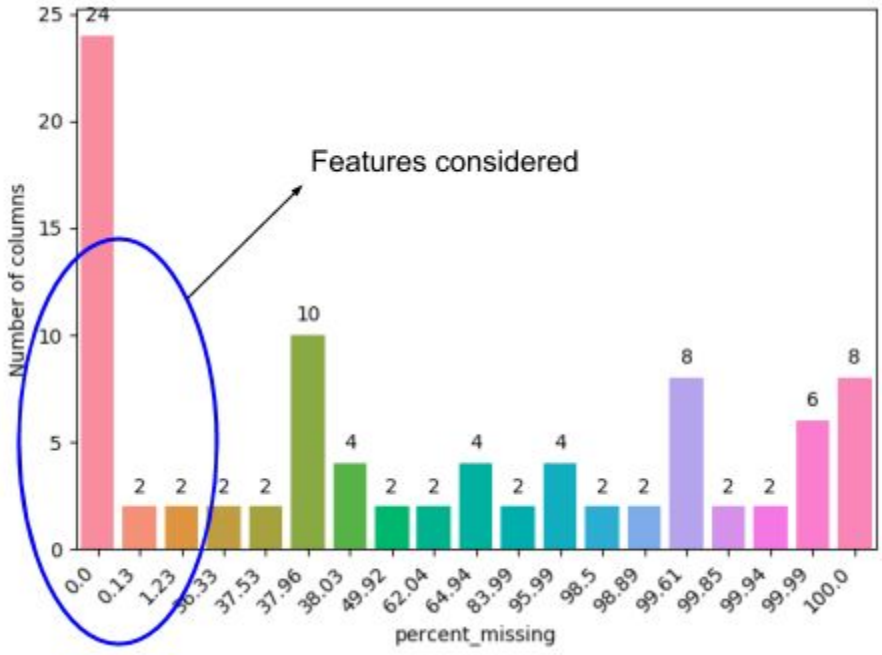
The procedure used is explained visually in below lucid chart -



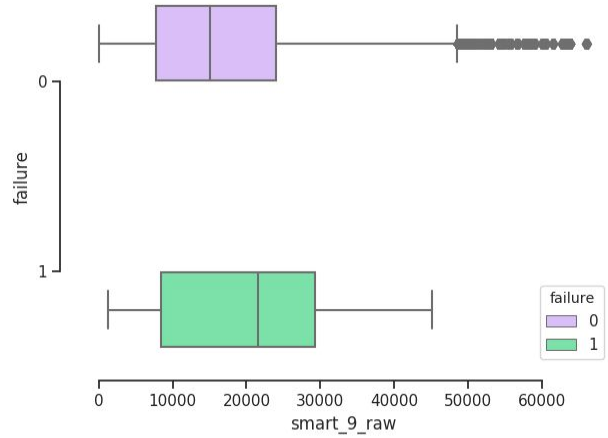
Step 1: EDA for year 2017 data

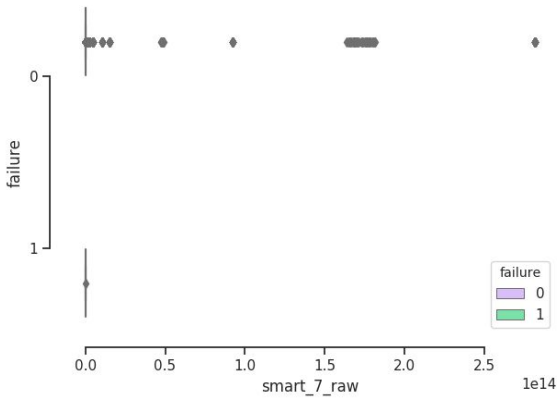
Tasks Completed

The data is highly imbalanced, for the first month of 2017 there are 1989462 - 1 million 989 thousand and 462 records, with 73,885 drives data, failure cases are just 101.
Then percentage of emptiness of all the columns with each smart stat represented with two columns of raw and normalised, were plotted as shown in below bar plot indicating 14 smart stats (28/2) with a threshold of less than 30% emptiness .

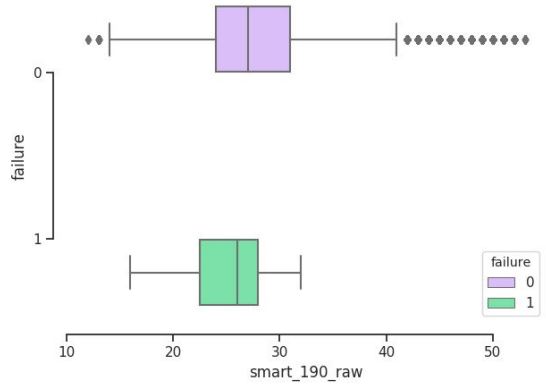


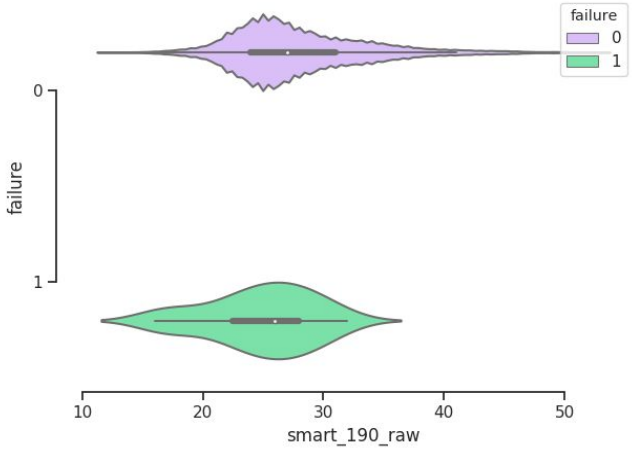
Further EDA was carried by plotting whisker plots representing medians and ranges for Smart Stats. As we can see in below two whisker plots, some features showed different variations, ranges and median values for failure and non-failure data. While for many features such as Smart Stat 7, Smart Stat 188, Smart Stat 189, Smart 191, etc. no variations were found indicating only constant values or outliers for these features.



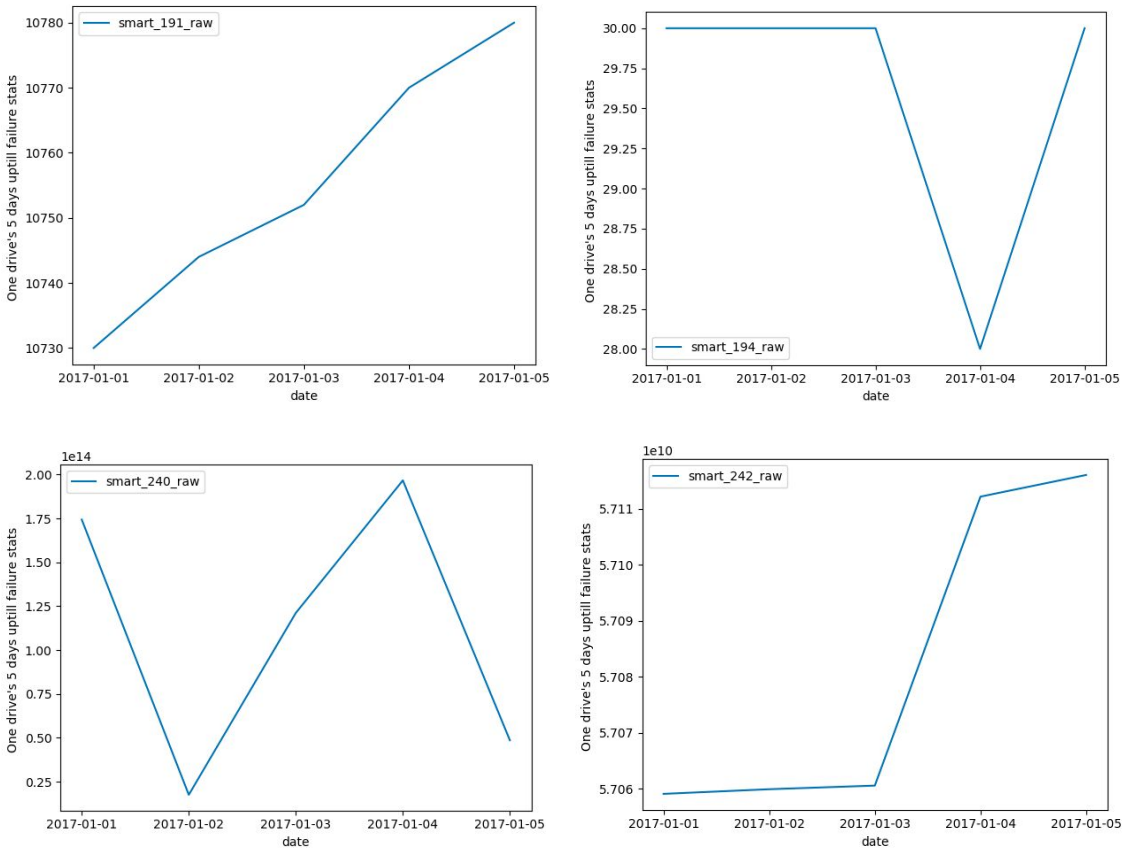


There were few features such as Smart Stat 190 showing overlapping variations in feature values for failure and non-failure for which violin plots were also used in order to find out the data distribution and density estimation over the feature values. Through the violin plot we can observe that data distribution for failure has lower values of Smart Stat 190 while higher for operational drives.



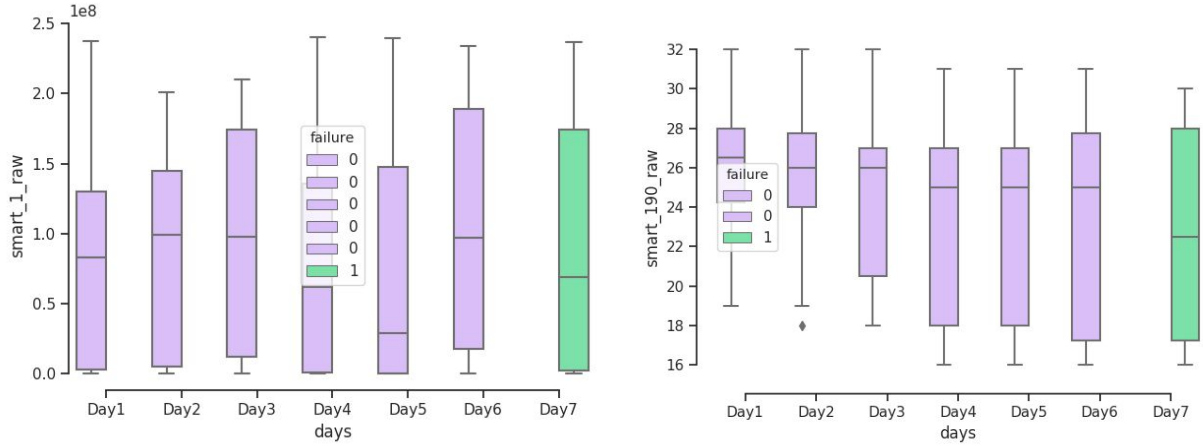


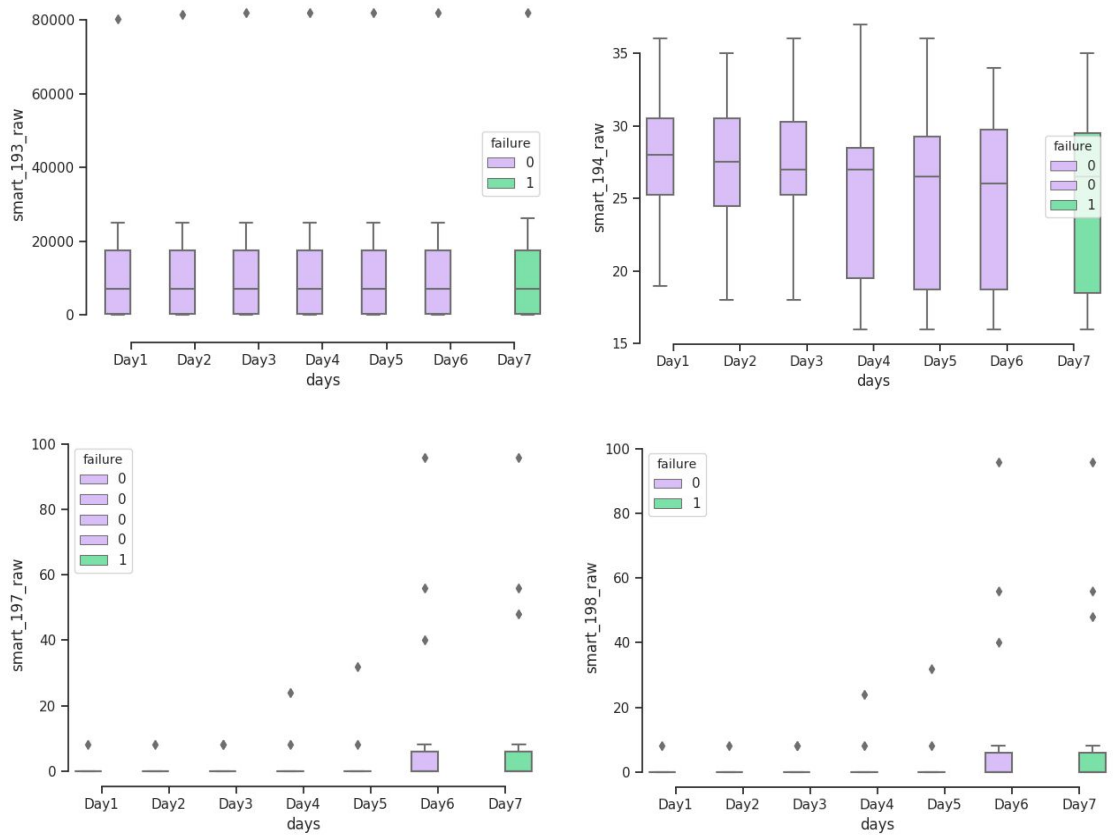
Then, analysis of smart stat values for a fixed time frame before failure happened was done using 5 days prior data for one particular drive which failed on 5th January, 2017 and observing its values of various features for the first 5 days of January.



We can notice in the above plots that values of these features changed before drives failure, for some features the value was continuously rising or decreasing before failure, for some features it remained constant indicating not helpful in failure prediction. For many features the slope change was quite significant in plots such as for Smart Stat 242 in above figures.

Lastly group of drives were plotted by using data of 14 drives together for 7 days in order to observe changes in Smart stats values and changes before their failure on 7th day.





Few features such as Smart Stat 194, 197, 198 proved to be relevant by showing that median value increases/decreases every day before failure. Some features were discarded as in above plots they show no change in entire range of values for complete time frame of 7 days. While tracking behaviour for a group of drives before failure in the above plot of Smart Stat 1 and 194, we can notice that the average of the first 3 days are different than the average values of 3 days before failure which indicates that back labelling of data with 3 days as label 1 is a good choice.

Results

Summarizing the results from all the plots of EDA, it was concluded that below 7 features[1] are relevant in order to carry out further steps.

Smart 1 Read Error Rate
Smart 3 Spin Up Time
Smart 9 Power On Hours
Smart 192 Unsafe Shutdown Count
Smart 194 Temperature or Temperature Celsius
Smart 197 Current Pending Sector Count
Smart 198 Uncorrectable Sector Count

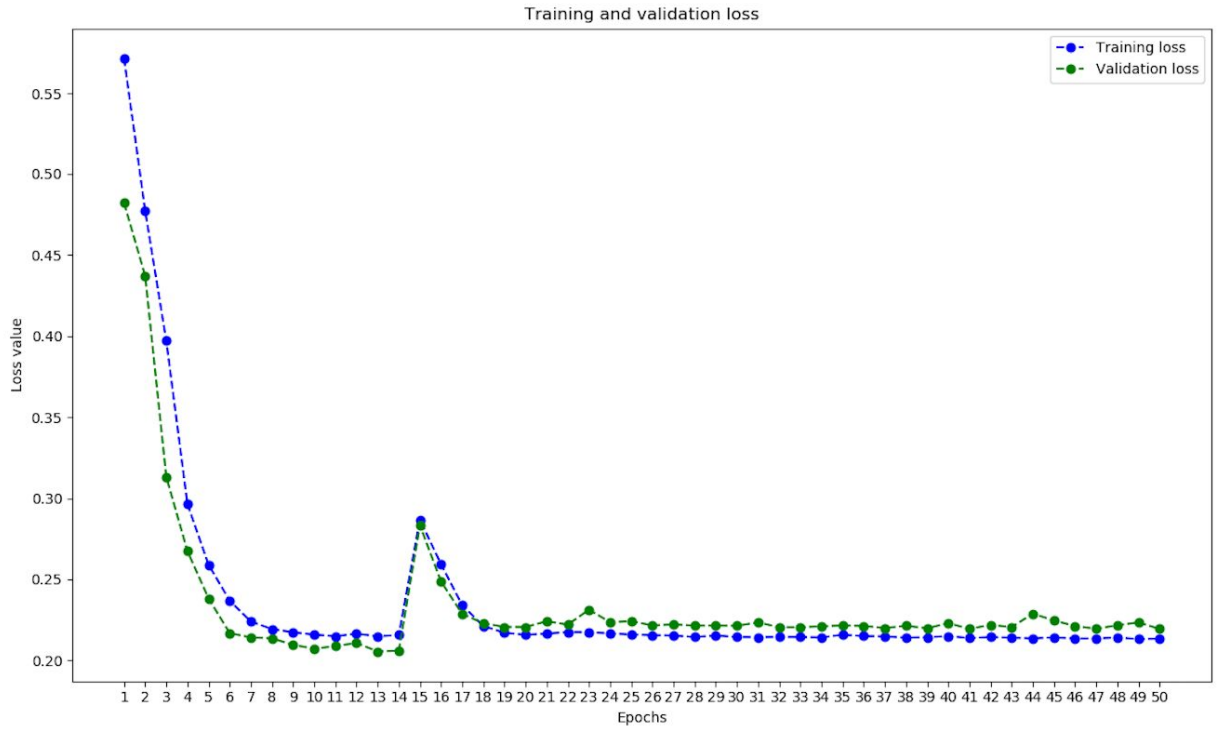
Availability of Deliverables

All the plots related to EDA of 2017 data can be found at below github link - https://github.com/fparages/iit-uptake-capstone/blob/master/failure_modelling/reports/reports_eda_2017.pdf

Step 2: Training Autoencoders

Tasks Completed

Autoencoder was implemented with two layers of encoder, one hidden layer and two layers of decoder. Now using the above 7 features and back labelled data with 3 days before failure was used (around 1,40,000 data points from the first month of 2017) to train and tune hyperparameters of Autoencoder for normal behaviour of drives. But, the Autoencoder's validation data was producing lower loss value than training data's loss value, due to which it was decided to retrain Autoencoder with cross-validation.
The average loss for training data is - 0.3164
The average loss for validation data is - 0.3128



Results

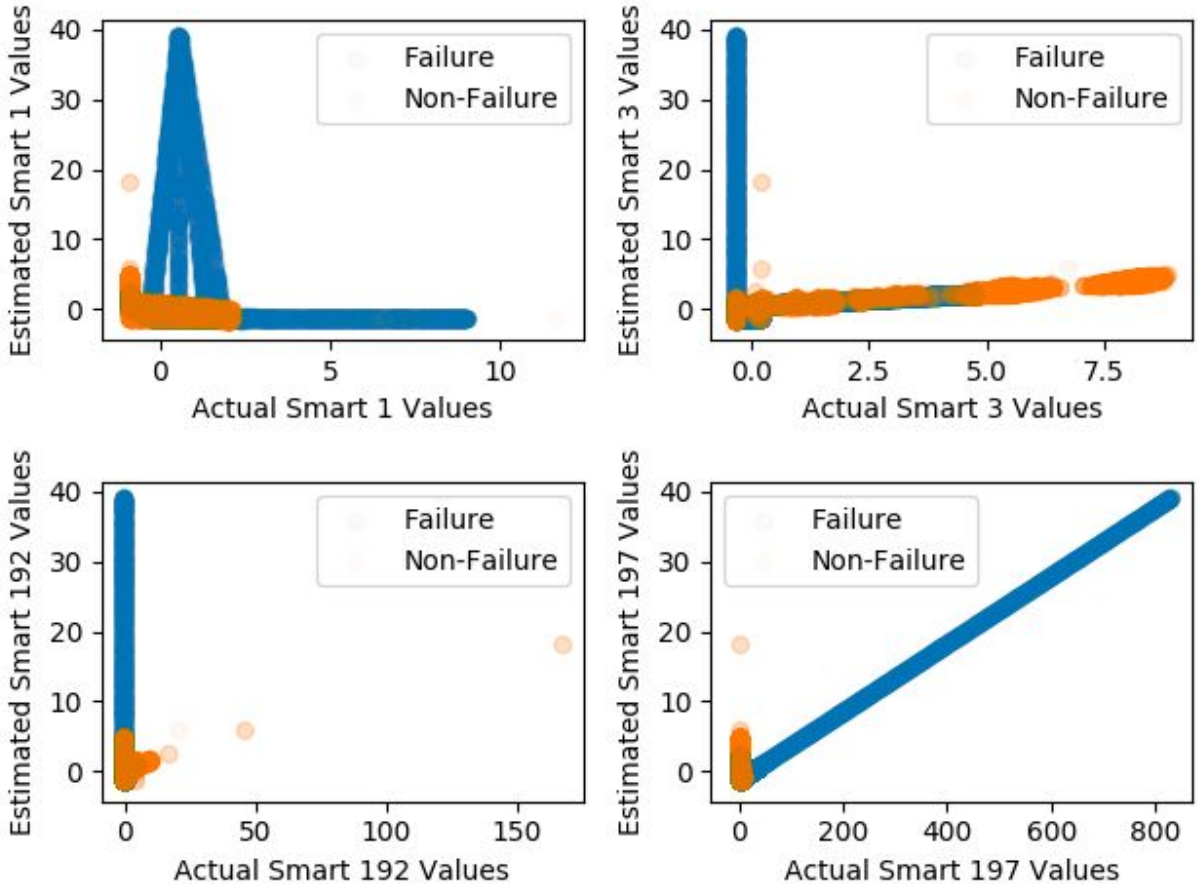
The Autoencoder was successfully trained and saved in file along with its weights so that it can be used to estimate or predict features without retraining again to recognize normal behaviour of drives. The validation data and training data loss value decreases as epoch value increases Moreover, as shown by spike in the above plot, it can be considered to use lower learning rate for future work.

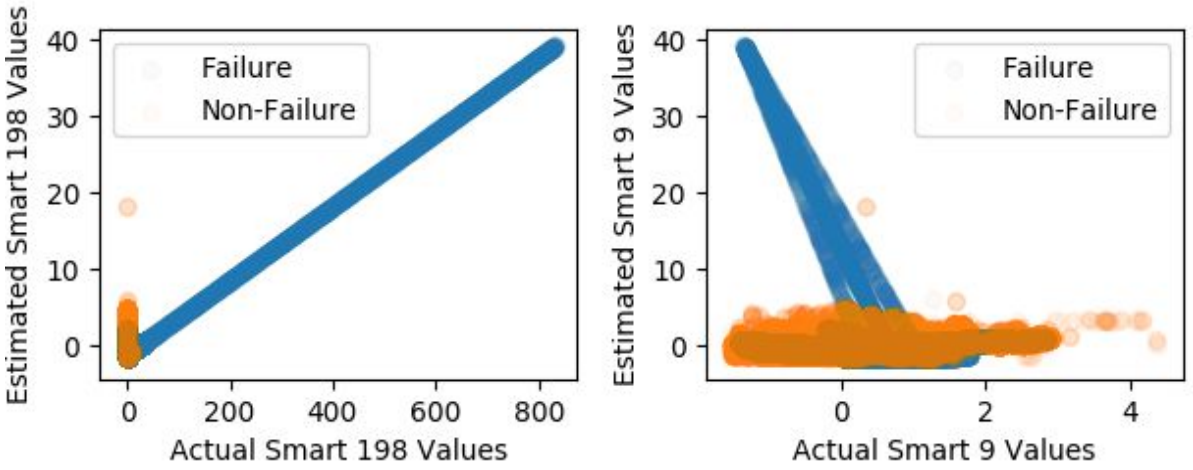
Step 3: Exploring estimated features from Autoencoders

Tasks Completed

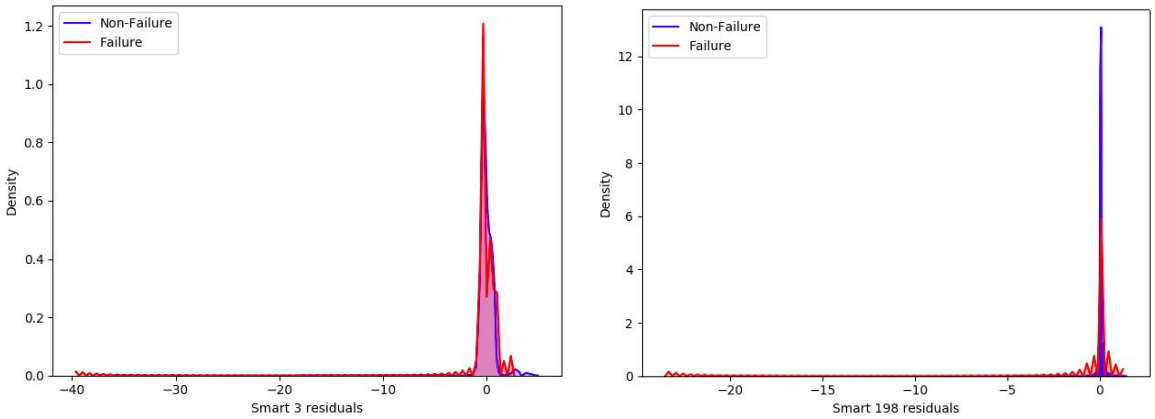
For exploring whether Autoencoders is recognizing abnormalities for failure data points three kinds of plots were used.

1. **Estimated versus Actual features** plots were explored by creating scatter plot to establish how each relevant features for failure drives and non failure drives differ in estimated values by autoencoders from actual feature values. Incase not clear in figures below orange represents non-failure data points and blue failure ones.

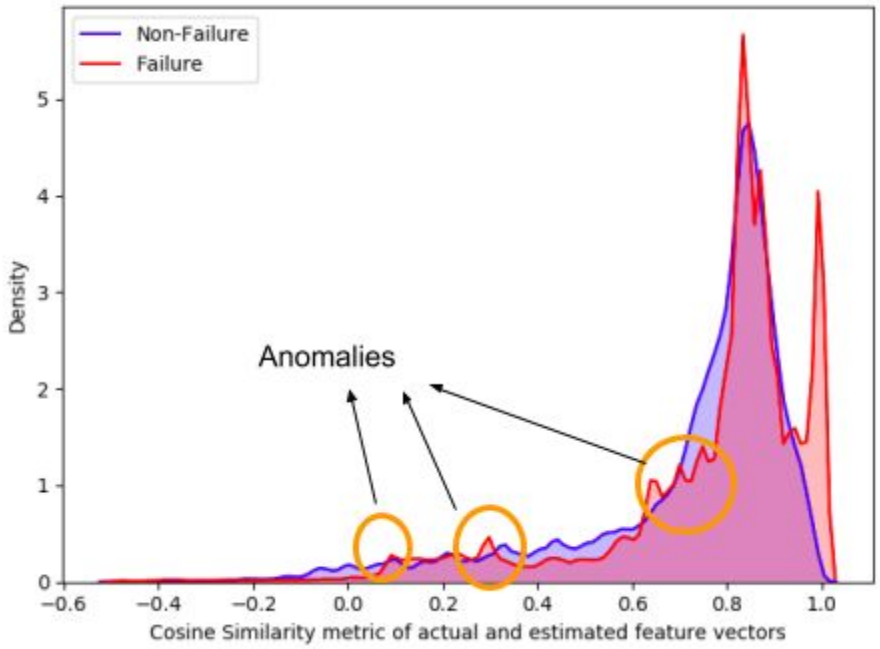




2. **Residual Plot** was explored by plotting density curve for residuals of failure and non failure drive data points of each feature. In below residual plots we can see there are spikes for failure data points in tail which represent abnormal feature value indicating different values of residuals than the one in blue curve.



3. **Cosine Similarity plot** was built treating estimated features and actual features as vectors and then calculating cosine similarity metric between those vectors. In below plot we can notice than many failure values have cosine similarity and spikes in negative scale as for failure drives, estimated and actual features must have less similarity hence cosine angle with angle 36 degree and more.



Results

We can see spikes in residual plots and cosine similarities where anomalies are recognized by Autoencoders.

Availability of Deliverables

All plots of autoencoders explained in Step 2 and 3 in detail are available at the below git link- https://github.com/fparages/iit-uptake-capstone/blob/feature/anomaly_detection_using_autoencoders/failure_modelling/reports/final_plots_for_autoencoders.pdf

Step 4: Using residuals to improve Supervised Model Performance

Tasks Completed

Random Forest performance was observed without residuals with SMOTE oversampled 1million and 400,000 data points and its performance changes were noted after feeding residuals (estimated features - actual features) to Random Forest classifier as shown in below two confusion matrix. Important thing to note is that the classifiers with and without residuals is each tuned separately for hyperparameters as features fed to them are different and also it supports to ignore any noise.

Random Forest without residuals

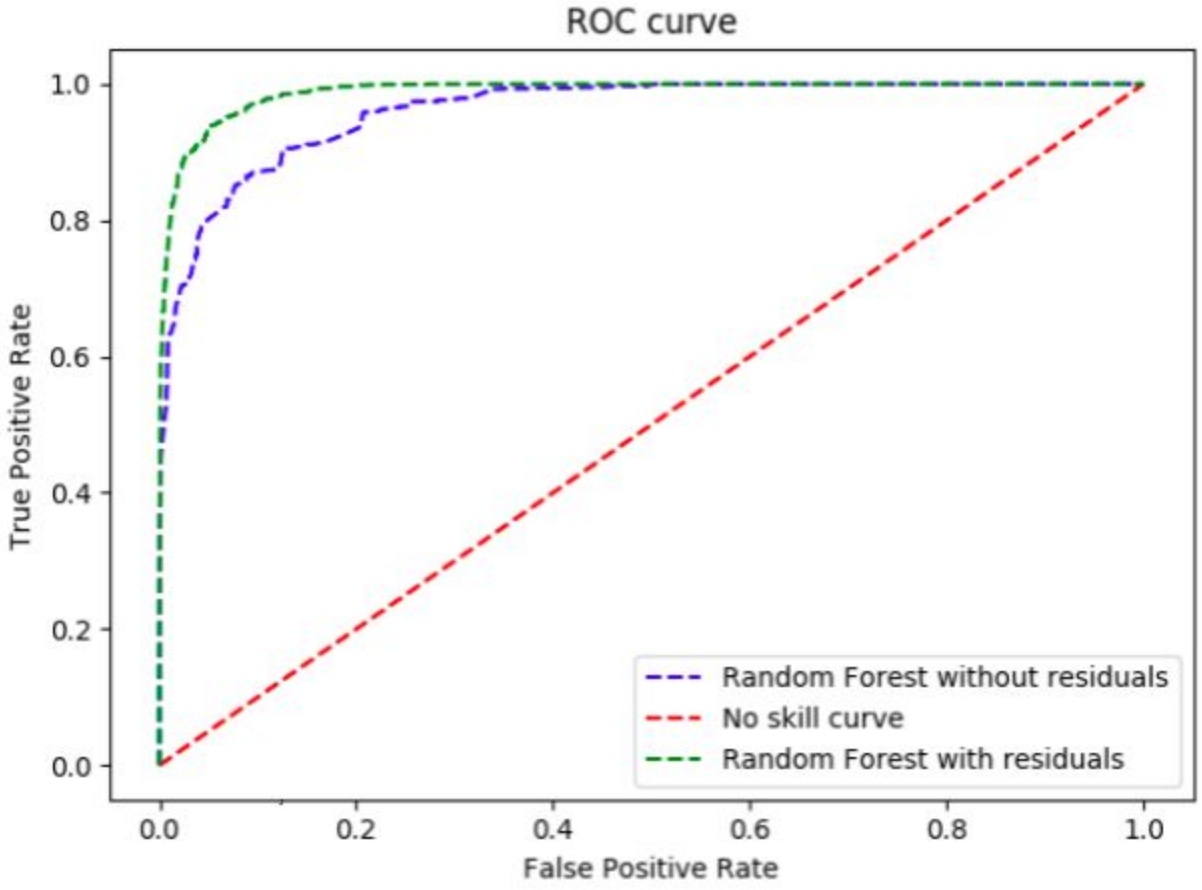
Actual Predicted	Class-0/ Negative	Class-1/ Positive (Failure)	**Precision :-** TP/(TP+FP) **0.9079** **Recall:-** TP/(TP+FN) **0.8554**
Class-0/ Negative	134184 (TN)	21291 (FN / Missed Alarms)	
Class-1/ Positive (Failure)	12772 (FP/ False Alarms)	125967 (TP)	

Random Forest with residuals

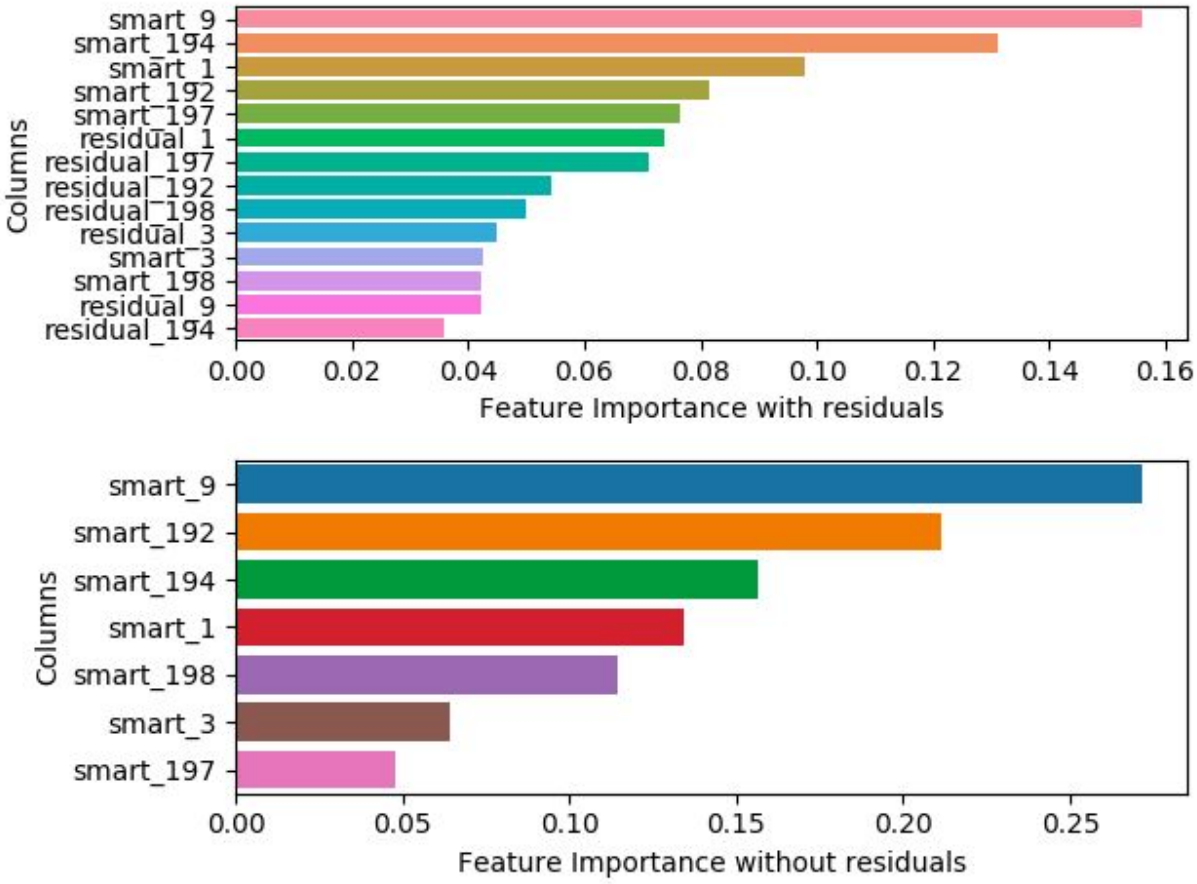
Actual Predicted	Class-0/ Negative	Class-1/ Positive (Failure)	**Precision** :- TP/(TP+FP) **0.9464** **Recall:-** TP/(TP+FN) **0.9382**
Class-0/ Negative	139134 (TN)	9097(FN / Missed Alarms)	
Class-1/ Positive (Failure)	7822 (FP/ False Alarms)	138161 (TP)	

From these two confusion matrices we can notice that after feeding residuals to our classifier the Precision, Recall [2]and True Positive counts have increased and False Negative (missed alarms) which are very important to be less to save clients from monetary loss has also decreased.

Below plot contains ROC curves of Random Forest before and after feeding residuals which also shows that for ROC curve with residuals in green has True Positive Rate is at higher than ROC curve without residuals in blue at lower value of False Positive rate.



Moreover, feature importance was taken out for Random Forest with and without residuals. We can notice than for feature importance few of residuals have even surpassed actual features for example residual 3 and residual 198, and others are also proving to be significant such as residual corresponding to features Smart Stat 1, 197 and 198 for classification with improved results.



Results

Random Forest performance was improved with residuals as recall and precision is increased with model recognizing True alarms better and ignoring false ones. Moreover, the false negatives (missed alarms) were decreased which was very important as missed alarms will lead to monetary loss of client costing due to failure without an alarm.

Availability of Deliverables

Github link for overall code of autoencoder part - https://github.com/fparages/iit-uptake-capstone/tree/master/failure_modelling/

Step 5: Possible Improvements for Autoencoders

For future work, all the plots shown above and results related to trained Autoencoders have room for improvement by using Autoencoders with more layers and even trying with LSTM Autoencoders [3,4,5] and including different features as well as experimenting with different year data.

Quality Assurance

Throughout this project we were in constant contact with Felipe Parages, our Uptake contact. In our weekly meetings we discussed the progress we were making and discussed any possible problems and solutions for issues as they arose. This ensured that we were working toward the project goals and that our client understood the progress we were making. It also kept us focused on the goals that the client had for our project.

Project Support

As this project would require substantial cloud computing resources, continued support for it is unlikely. The focus of this project was research, looking at how supervised models degraded as failure labels were removed and how using autoencoders could be used to improve performance. While we were not able to accomplish everything we set out to do, still we were able to obtain useful results for both the objectives in short span of time of two and a half months.

Project Retrospective

Clearly, we learned during this project the difficulty in working with expansive dataset. When we first began this project we were happy to have an expansive and relatively complete dataset. However it soon became clear that the size of our dataset wasn't an asset but a hindrance. This was especially problematic because of the large class imbalance. We were forced to work with as much data as possible to get any meaningful results. Throughout the project we explored different ways to approach this issue and while we weren't able to solve it, we did realize that large datasets can be dealt with access to the right resources.

Another lesson learned was how to deal with time series data. Before this project none of us had ever considered running models where the data points were dependant on previous values. We learned several different techniques for handling time series data and showed how they improved the performance of our models. In the future we will be able to apply these same techniques to other projects.

Moreover, none of us have worked with Neural Networks before, so it was new learning again to implement and use Autoencoders and also read about LSTM for classification.

References

1. "S.M.A.R.T. - Wikipedia." https://en.wikipedia.org/wiki/S.M.A.R.T.. Accessed 10 Aug. 2019.
2. "How and When to Use ROC Curves and Precision-Recall Curves for ...." 31 Aug. 2018, https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/. Accessed 10 Aug. 2019.
3. "Engineering Extreme Event Forecasting at ...." 9 Jun. 2017, https://eng.uber.com/neural-networks/. Accessed 10 Aug. 2019.
4. "Deep and Confident Prediction for Time Series at Uber." 6 Sep. 2017, https://arxiv.org/abs/1709.01907. Accessed 10 Aug. 2019.
5. "Dataset: Rare Event Classification in Multivariate Time Series." 27 Sep. 2018, https://arxiv.org/abs/1809.10717. Accessed 10 Aug. 2019.

Document Revision History

Initial Version - August 7th, 2019
First Revision - August 9th, 2019
Final Revision - August 10th, 2019