PREDICTIVE ANALYTICS ON AIRCRAFT MAINTENANCE

Team Members:

19BDS0144 (Jyothi K C)
19BDS0147 (Usha Ranjani V A)

Report submitted for the
Final Project Review of Aircraft Maintenance Prediction

Course Code: CSE3045
Predictive Analysis

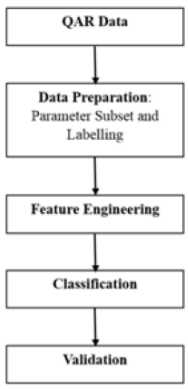
Slot: A2 Slot

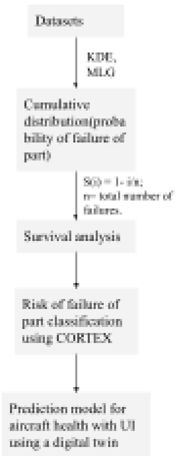
Professor: Dr.Ilanthenral Kandasamy

Introduction

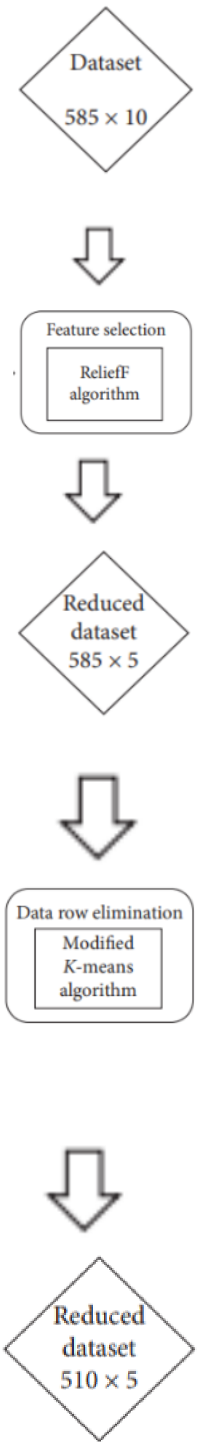
Reliability and availability of aircraft components have always been an important consideration in aviation. Accurate prediction of possible failures will increase the reliability of aircraft components and systems. The scheduling of maintenance operations help determine the overall maintenance and overhaul costs of aircraft components. Maintenance costs constitute a significant portion of the total operating expenditure of aircraft systems.There are three main types of maintenance for equipment: corrective maintenance, preventive maintenance, and predictive maintenance [1]. Corrective maintenance helps manage repair actions and unscheduled fault events, such as equipment and machine failures. When aircraft equipment fails while it is in use, it is repaired or replaced. Preventive maintenance can reduce the need for unplanned repair operations. It is implemented by periodic maintenance to avoid equipment failures or machinery breakdowns. Tasks for this type of maintenance are planned to prevent unexpected downtime and breakdown events that would lead to repair operations. Predictive maintenance, as the name suggests, uses some parameters which are measured while the equipment is in operation to guess when failures might happen. It intends to interfere with the system before faults occur [1, 2] and help reduce the number of unexpected failures by providing the maintenance personnel with more reliable scheduling options for preventive maintenance. Assessing system reliability is important to choose the right maintenance strategy.Failure prediction is a major topic in predictive maintenance in many industries. Airlines are interested in predicting engine failures in advance to enhance operations and reduce flight delays. Observing engine's health and condition through sensors and telemetry data is assumed to facilitate this type of maintenance by predicting Time-To-Failure (TTF) of in-service engines. Consequently, maintenance work could be planned according to TTF predictions instead of /to complement costly time based preventive maintenance.In this project, we have downloaded our dataset from Kaggle which contains simulated aircraft engine run-to-failure events, operational settings, and 21 sensors measurements .

Literature Review Summary

Authors and Year (Reference)	*Title (Study)*	*Concept / Theoretical model/ Framework*	*Methodology used/ Implementation*	*Dataset details/ Analysis*	*Relevant Finding*	*Limitations/ Future Research/ Gaps identified*
Savitha Ramasamy ,Yang Xue ,Royston Phoon , Richard Han ,Nelson Lowan ,Chee Siang Lim 2018	Predictive Maintenance of the Aircraft Engine Bleed Air System Component		(a) Pre-processing QAR data for obtaining a subset of parameters relevant to the bleed air system component, (b) Generating features from the parameter subset that are clearly discriminative of the health status of the component, (c) Deriving the health status of the component based on airlines maintenance records and observed QAR signals, (d) Training a machine learning classifier that learns the relationship between the	The data collected from the aircraft fleet from 2015-2016 to develop the predictive maintenance solution, and validate the solution using prospective data collected in 2017. In all these cases, the sensor data is collected from the QAR, and the health condition of the component is assessed from airline maintena	Predicting aircraft maintenance. Sensor data collection and preprocessing.	Need to investigate additional data sources, e.g., SAR data for improving the coverage of the model.

			generated features and the health status	nce records.		
Sam Heim, Jason Clemens, James E. Steck, Christopher Basic, David Timmons, Kourtney Zwiener 2020 .	Predictive Maintenance on Aircraft and Applications with Digital Twin		a) To establish the likelihood of failure based on past failures and the time since last repair- kernel density estimate(KDE) and maximum likelihood gaussian(MLG) used to generate the cumulative distribution(for probability of failure). b) To classify the risk of failure of each part at a particular point in its lifecycle -a survival function was used. c) for classification of risk(low,medium,high) , additional data was synthesized and predicted using a neural network.	a)The 2 datasets used contained information regarding maintenance events on a particular aircraft. b) Dataset 1's feature variable ,time, for each part was measured wrt total aircraft flight hours whereas in Dataset 2, the part time is wrt time elapsed, in days, between maintenance events on the same part. c) The time that the part was used the only feature	Neural network classification; aircraft engine failure prediction;	- More data was generated under the assumption that the part is healthy prior to fault occurrence.However,we can not be sure, as there can false positives as well(undetected failures). - availability of sensor data, flight conditions, etc. could make the dataset more robust.

			d) data was fed into CORTEX software for training the model(multilayer perceptron) e) An accuracy of 88% was achieved overall for all aircraft parts. f) It was also inferred that the only variables needed to accurately predict and classify were the age of the aircraft and age of the part,despite the additional data synthesised. g) The results obtained were then incorporated into a digital twin model to provide an UI for real time health of an aircraft.	considered for likelihood d) For the neural network classification 8 features were considered,representing overall aircraft age and usage, part specific usages, and lastly the operating stage and nature of condition. e) More data was required for classification and thus, was generated randomly assuming that no incident means the part was in a healthy state.		

Kadir Celikmih, Onur Inan , and Harun Uguz 2020	Failure Prediction of Aircraft Equipment Using Machine Learning with a Hybrid Data Preparation Method		In the first stage, ReliefF, a feature selection method for attribute evaluation, is used to find the most effective and ineffective parameters. In the second stage, a *K*-means algorithm is modified to eliminate noisy or inconsistent data. Performance of the hybrid data preparation model on the maintenance dataset of the equipment is evaluated by Multilayer Perceptron (MLP) as Artificial Neural network (ANN), Support Vector Regression (SVR), and Linear Regression (LR) as machine learning algorithms. Moreover, performance	The context in which the present case study was carried out was an avitation company in Ankara, Turkey. The maintenance data were collected from the records of the maintenance department. they included removal of equipment, repair activities, experience of the operators, flight hours of the equipment, and	Failure prediction of aircraft equipment.	The suggested hybrid system helped attain higher accuracy in prediction as it enabled us to select the most effective features and eliminate noisy or redundant data that could lower the accuracy of predictions

			criteria such as the Correlation Coefficient (CC), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE) are used to evaluate the models.	other information relevant to the case study. used. The dataset consists of nine input variables and an output variable (failure count). The input variables/ factors are operational and environmental parameters which could influence failure occurrence and the length of operation before failures occur. Input variables include		

				such parameters as flight hours, the number of removals of equipment,and the number of faults with planned/unplanned removals		
Maren David Dangut, Zakwan Skaf, Ian K. Jennions 2020	Rare Failure Prediction Using an Integrated Auto-encoder and Bidirectional Gated Recurrent Unit Network	a)Objective was to predict rare failures in aircraft engine maintenance using auto-encoder(AE) and BGRU(birectional gated recurrent unit network). b) The AE helps in training the model with only negatively labelled data to detect rare faults using the reconstruction error as a threshold. The output of AE is used as input to the BGRU network, to predict the	a) The performance metrics used are precision, recall, G-mean, and AUC. b) To determine the optimal threshold, an iterative function is constructed using precision, recall and G-mean. The threshold value is 0.4. c) Upon evaluation, the proposed approach was 25% better in precision, 14% in the recall, and	a) Obtained from a fleet of long-range (A330) aircraft. 2 datasets: 1) aircraft fault report records) and the flight deck effects (FDE) 2) logs of aircraft maintenance activities(w.r.t time)	historical data; Auto encoder for imbalanced dataset; Neural network implementation; Fault prediction and detection in aircraft engine	Future research can be conducted on other architectures of AE-CNN-BGRU to improve prediction performance on the aircraft log-based dataset to achieve predictive maintenance.

		occurrence of those faults in Next-N-step.	3% in G-mean. The result also shows robustness in predicting failure within a defined useful period.	b) Data from the year 2011 to 2016 was used for training, while the reaming from 2016 to 2018 is used for testing. c) The dataset was imbalanced d) LRUs for predictive modelling were: Electronic Engine Unit, Pressure Bleed Valve and Trim Air Valve.		
Ade Pitra Hermawan, Dong-Seong Kim 2020	Predictive Maintenance of Aircraft Engine using	a) convolutional layer was used for the first	The proposed model(CLSTM) is a	The aircraft engine's	Feature extraction; sensor	For future research, energy efficient based approaches will

	Deep Learning Technique	layer to process the input data. b) After that max pooling was used with kernel size of 1. Dropout layer with value 0.1 also used after the max pooling layer to avoid overfitting in the system. c) Rectified linear unit (ReLU) activation function was used transforms the output into 0 if the input value is less than 0, else transform to X. $f = \begin{Bmatrix} (x < 0) & f(x) = 0 \\ (x \geq 0) & f(x) = x \end{Bmatrix}.$ Lastly, the dense layer with three of hidden layers associated with the number of classes , was used. On the last dense layer, we	hybrid of CNN and LSTM. Upon comparison of the proposed model with CNN and LSTM the following results were produced for the accuracy metric: CNN=89.33 % LSTM= 97.32% CLSTM= 99.60%	dataset contains 21 sensor data. The sensor devices fed the informati on to the server periodical ly and the proposed system learnt the data behaviour and thus, predicted the maintena nce condition in the future. The train-test split was 70:30. Total data for training was 14849 samples, and 782 samples for validation .	data for aircraft engine;	be investigated.

		used a softmax activation function to interpret the output optimally. $S(y_i) = \frac{e^{y_i}}{\sum_j^{j=i} e^{y_i}}$.				

Objective:

The objective of our project is to create a prediction model to estimate failure for aircraft engines using the Microsoft aircraft engines dataset.

We aim to achieve the following goals:

a)To find the aircraft's remaining useful life(RUL).

b) To predict which engine will fail currently or in a given time period.

We executed these objectives using regression for RUL. The regression algorithms are: Polynomial regression, decision tree regression and random forest regression. We performed binary classification using LSTM for predicting failure in a given time frame(cycle) and multi- class classification using Random Forest , MLP and decision tree.

Innovative component:

We have chosen the official Azure code as a reference for our project. In this , they have implemented LSTM for binary classification of engine failure. Inspired by this, we aim to implement regression to find time to failure or remaining useful life(RUL) using Polynomial regression, decision tree regression and random forest regression. In addition, we plan to implement multi class classification to predict failure in a given time frame, using algorithms such as Random Forest , Decision tree and MLP and perform a comparative analysis to find the most optimal predictive model.

5. Work done and implementation

a. Methodology adapted:

Data Pre-processing:

Labels:

We generated training data labels for regression and classification purposes:

- Regression: Time-To-Failure TTF (no. of remaining cycle before failure) for each cycle/engine is the number of cycles between that cycle and the last cycle of the same engine.
- Binary Classification: Labels based on Remaining useful life(RUL), 2 labels defined:

  train_df['RUL'] = train_df['max'] - train_df['cycle'] -> RUL calculation

  - ➢ for time period, cycle window= 15: if failure occurs within 15 cycles(1) or not(0).
  - ➢ for time period , cycle window= 30: if failure occurs within 30 cycles(1) or not(0).
- Multiclass Classification: Segmenting the TTF into cycle bands (periods: 0-15(1), 16-30(2), 30+(0)), to identify in which period the engine will fail.

Assigning labels to the test data:

For test data, TTF was provided in a separate truth data file. Thus, the truth data containing the annotations was concatenated to the test dataset.

Data normalization:

Since a lot of the features were on varying scales, min-max normalisation was performed.

Data Wrangling:

Feature extraction was applied to the training and test data by introducing additional two columns for each of the 21 sensor columns: rolling mean and rolling standard deviation. This smoothing to the sensors' measurements over time improved the performance of some machine learning algorithms.

Checking for class imbalance:

The ratio of positive samples(failure detected) to the ratio of the negative samples(no failure) is 85:15 which indicates a high imbalance. Hence, we decided to use F-score, Precision and Recall in addition to Accuracy as our performance metrics for binary classification.

```
[ ]  # print stat for binary classification label

     print(df_tr_lbl['label_bnc'].value_counts())
     print('\nNegaitve samples =  {0:.0%}'.format(df_tr_lbl['label_bnc'].value_counts()[0]/df_tr_lbl['label_bnc'].count()))
     print('\nPosiitve samples =  {0:.0%}'.format(df_tr_lbl['label_bnc'].value_counts()[1]/df_tr_lbl['label_bnc'].count()))

     0    17531
     1     3100
     Name: label_bnc, dtype: int64

     Negaitve samples =  85%

     Posiitve samples =  15%
```

For multi-class classification(3 labels) the ratio was 85:7:8 which was also tells us that it highly imbalanced. To resolve this, we used macro and micro averages, macro and micro for precision and recall, AUC-ROC(area under curve for receiver operating characteristic) as performance metrics.

```
# print stat for multiclass classification label

print(df_tr_lbl['label_mcc'].value_counts())
print('\nClass 0 samples =  {0:.0%}'.format(df_tr_lbl['label_mcc'].value_counts()[0]/df_tr_lbl['label_mcc'].count()))
print('\nClass 1 samples =  {0:.0%}'.format(df_tr_lbl['label_mcc'].value_counts()[1]/df_tr_lbl['label_mcc'].count()))
print('\nClass 2 samples =  {0:.0%}'.format(df_tr_lbl['label_mcc'].value_counts()[2]/df_tr_lbl['label_mcc'].count()))

0    17531
2     1600
1     1500
Name: label_mcc, dtype: int64

Class 0 samples =  85%

Class 1 samples =  7%

Class 2 samples =  8%
```

Exploratory Data Analysis(EDA):

Feature variability, distribution, and correlation were examined to uncover underlying structure and extract important variables.

The features with high variability were checked for correlation with other features and regression label (TTF): structure and extract important variables.

Scatter matrix was also used to check the distribution and correlation of features.

Class imbalance:

Modelling:

LSTM:

We used LSTM to perform binary classification and created two class labels: to predict if the engine will fail within 30 and if it will fail within 15.

Regression:

We used regression models to predict Time-to-Failure (TTF), for each cycle/engine, is the number cycles between that cycle and last cycle of the engine in the training data. The models used are decision tree ,polynomial regression and random forest regression.

For decision tree regression, we performed feature selection via recursive feature elimination. We then validated the model with cross validation.

Dataset used:

The dataset we have chosen is the Turbofan engine degradation simulation dataset , which we downloaded from Kaggle. It has text files containing simulated aircraft engine run-to-failure events, operational settings, and 21 sensor measurements. We assume that the degradation of engine parts are reflected in the sensor measurements. The dataset is divided into three parts as follows:

Training Data: The aircraft engine run-to-failure data.It consists of 20,000+ cycle records for 100 engines.

Test Data: The aircraft engine operating data without failure events recorded.

Ground Truth Data: The true remaining cycles for each engine in the testing data.

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	sd15	sd16	sd17	sd18	sd19	sd20	sd21	ttf	label_bnc	label_mcc
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.000000	191	0	0
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	0.008697	0.0	0.000000	0.0	0.0	0.042426	0.003253	190	0	0
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	0.007640	0.0	1.154701	0.0	0.0	0.055076	0.044573	189	0	0
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	0.028117	0.0	1.000000	0.0	0.0	0.076322	0.037977	188	0	0
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	0.025953	0.0	1.095445	0.0	0.0	0.073621	0.033498	187	0	0

5 rows × 71 columns

Features:

id: EngineId. There Are 100 Engines. Range 1-100

cycle:sequence perengine, starts from 1 to the cycle number where the failure had happened.

setting 1 to setting 3:engine operational settings

s1 to s21:sensors measurements in each cycle

c. Tools used:

SOFTWARE REQUIREMENTS

- Python 3.6
- Google Colab

PACKAGES:

- numpy 1.13.3
- scipy 0.19.1
- matplotlib 2.0.2
- spyder 3.2.3
- scikit-learn 0.19.0
- h5py 2.7.0
- Pillow 4.2.1
- pandas 0.20.3
- TensorFlow 1.3.0
- Keras 2.1.1

HARDWARE REQUIREMENTS

Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

Windows 10 Home Single Language

Asus

Data Preprocessing

Importing the text files into a dataframe:

```python
df_train_raw = pd.read_csv('PM_train.txt', sep = ' ', header=None)
df_train_raw.head()
```

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26	27
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190	NaN	NaN
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236	NaN	NaN
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442	NaN	NaN
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739	NaN	NaN
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044	NaN	NaN

5 rows × 28 columns

```python
df_train_raw.drop([26,27], axis=1, inplace=True)
```

```python
df_train_raw.columns = col_names
df_train_raw.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442

Exploring dataset:

```python
df_train_raw.columns = col_names
df_train_raw.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044

5 rows × 26 columns

```python
df_train_raw.describe()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14
count	20631.000000	20631.000000	20631.000000	20631.000000	20631.0	2.063100e+04	20631.000000	20631.000000	20631.000000	2.063100e+04	...	20631.000000	20631.000000	20631.000000
mean	51.506568	108.807862	-0.000009	0.000002	100.0	5.186700e+02	642.680934	1590.523119	1408.933782	1.462000e+01	...	521.413470	2388.096152	8143.752722
std	29.227633	68.880990	0.002187	0.000293	0.0	6.537152e-11	0.500053	6.131150	9.000605	3.394700e-12	...	0.737553	0.071919	19.076176
min	1.000000	1.000000	-0.008700	-0.000600	100.0	5.186700e+02	641.210000	1571.040000	1382.250000	1.462000e+01	...	518.690000	2387.880000	8099.940000
25%	26.000000	52.000000	-0.001500	-0.000200	100.0	5.186700e+02	642.325000	1586.260000	1402.360000	1.462000e+01	...	520.960000	2388.040000	8133.245000
50%	52.000000	104.000000	0.000000	0.000000	100.0	5.186700e+02	642.640000	1590.100000	1408.040000	1.462000e+01	...	521.480000	2388.090000	8140.540000
75%	77.000000	156.000000	0.001500	0.000300	100.0	5.186700e+02	643.000000	1594.380000	1414.555000	1.462000e+01	...	521.950000	2388.140000	8148.310000

```python
df_train_raw.isnull().sum()
```

```
id          0
cycle       0
setting1    0
setting2    0
setting3    0
s1          0
s2          0
s3          0
s4          0
s5          0
s6          0
s7          0
s8          0
s9          0
s10         0
s11         0
s12         0
s13         0
s14         0
s15         0
s16         0
s17         0
s18         0
s19         0
s20         0
s21         0
dtype: int64
```

```python
df_truth = pd.read_csv('PM_truth.txt', sep = ' ', header=None)
df_truth.head()
```

	0	1
0	112	NaN
1	98	NaN
2	69	NaN
3	82	NaN
4	91	NaN

```python
df_truth.drop([1], axis=1, inplace=True)
df_truth.columns = ['ttf']
df_truth.head()
```

	ttf
0	112
1	98
2	69
3	82
4	91

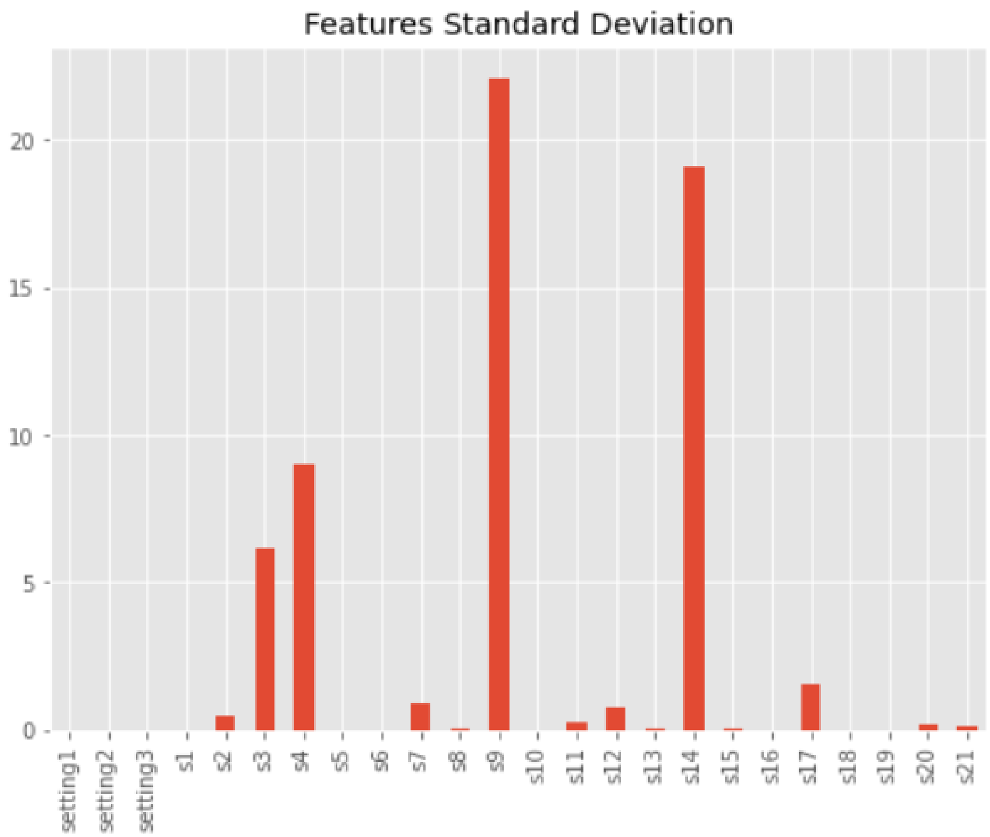
Feature extraction: Adding rolling mean and rolling standard deviation since we're tackling withtime series data.

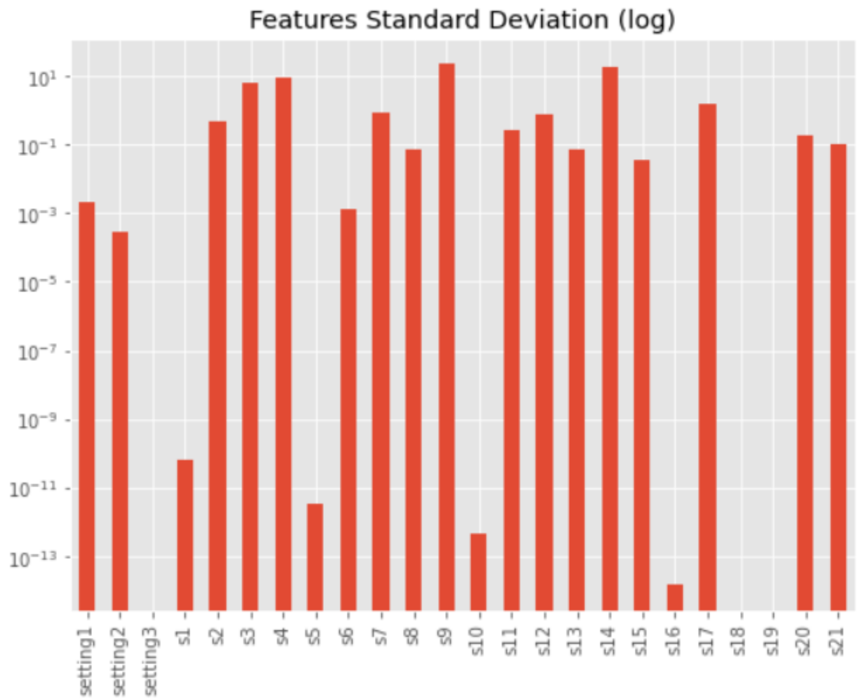
```python
[ ] def add_features(df_in, rolling_win_size):

        """Add rolling average and rolling standard deviation for sensors readings using fixed rolling window size.

        Args:
                df_in (dataframe)     : The input dataframe to be proccessed (training or test)
                rolling_win_size (int): The window size, number of cycles for applying the rolling function

        Reurns:
                dataframe: contains the input dataframe with additional rolling mean and std for each sensor

        """

        sensor_cols = ['s1','s2','s3','s4','s5','s6','s7','s8','s9','s10','s11','s12','s13','s14','s15','s16','s17','s18','s19','s20','s21']

        sensor_av_cols = [nm.replace('s', 'av') for nm in sensor_cols]
        sensor_sd_cols = [nm.replace('s', 'sd') for nm in sensor_cols]

        df_out = pd.DataFrame()

        ws = rolling_win_size

        #calculate rolling stats for each engine id

        for m_id in pd.unique(df_in.id):

            # get a subset for each engine sensors
```

```
sensor_cols = ['s1','s2','s3','s4','s5','s6','s7','s8','s9','s10','s11','s12','s13','s14','s15','s16','s17','s18','s19','s20','s21']

sensor_av_cols = [nm.replace('s', 'av') for nm in sensor_cols]
sensor_sd_cols = [nm.replace('s', 'sd') for nm in sensor_cols]

df_out = pd.DataFrame()

ws = rolling_win_size

#calculate rolling stats for each engine id

for m_id in pd.unique(df_in.id):

    # get a subset for each engine sensors
    df_engine = df_in[df_in['id'] == m_id]
    df_sub = df_engine[sensor_cols]
     # get rolling mean for the subset
    av = df_sub.rolling(ws, min_periods=1).mean()
    av.columns = sensor_av_cols

    # get the rolling standard deviation for the subset
    sd = df_sub.rolling(ws, min_periods=1).std().fillna(0)
    sd.columns = sensor_sd_cols

    # combine the two new subset dataframes columns to the engine subset
    new_ftrs = pd.concat([df_engine,av,sd], axis=1)

    # add the new features rows to the output dataframe
    df_out = pd.concat([df_out,new_ftrs])

return df_out
```

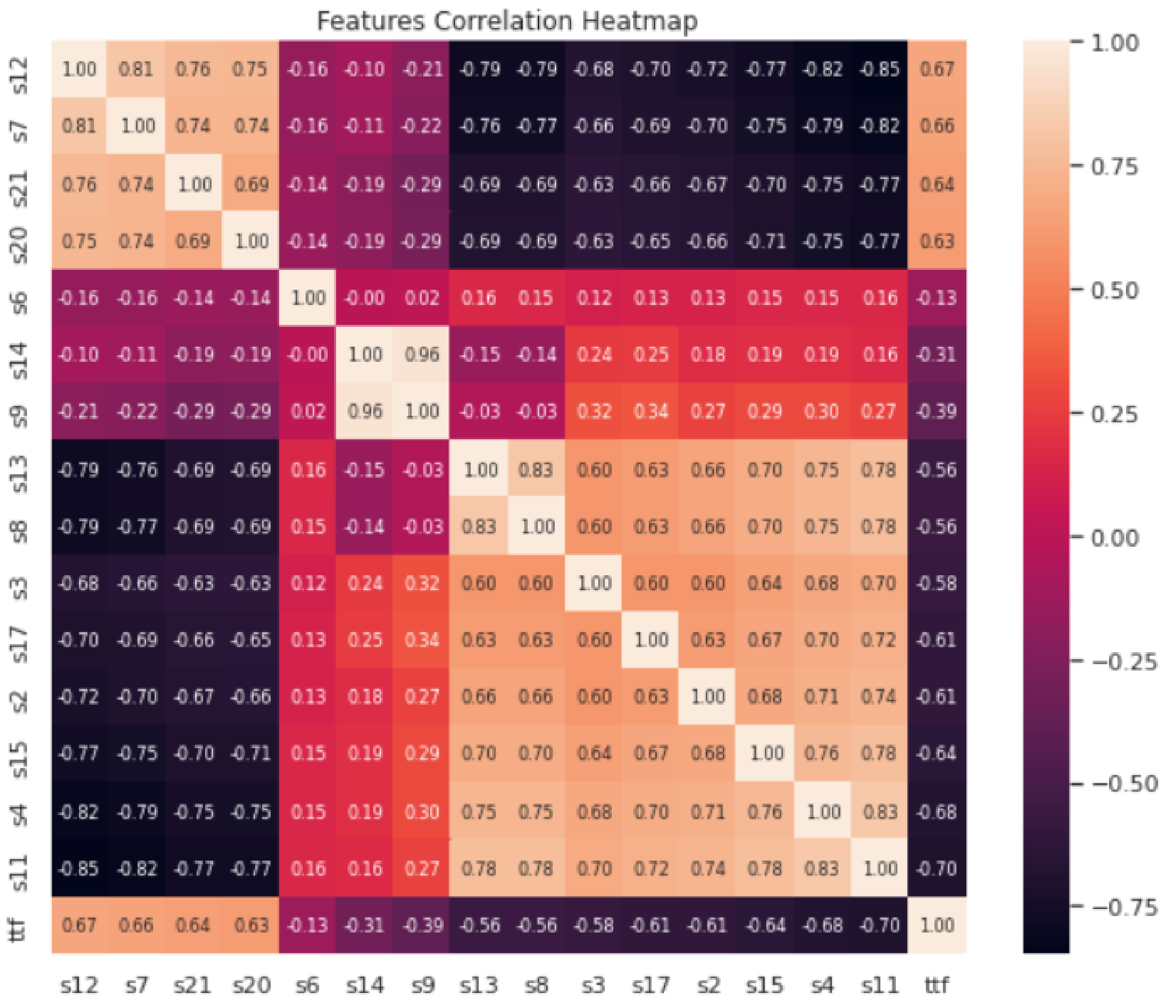
Exploratory Data Analysis

Feature variability, distribution, and correlation were examined to uncover underlying structure and extract important variables.

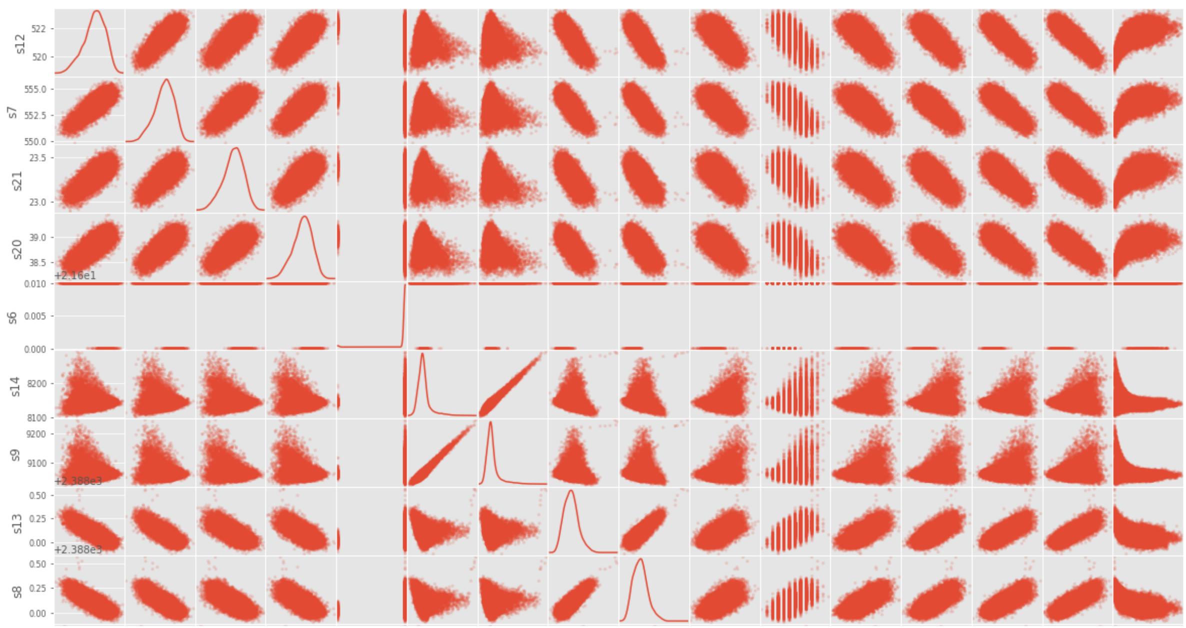


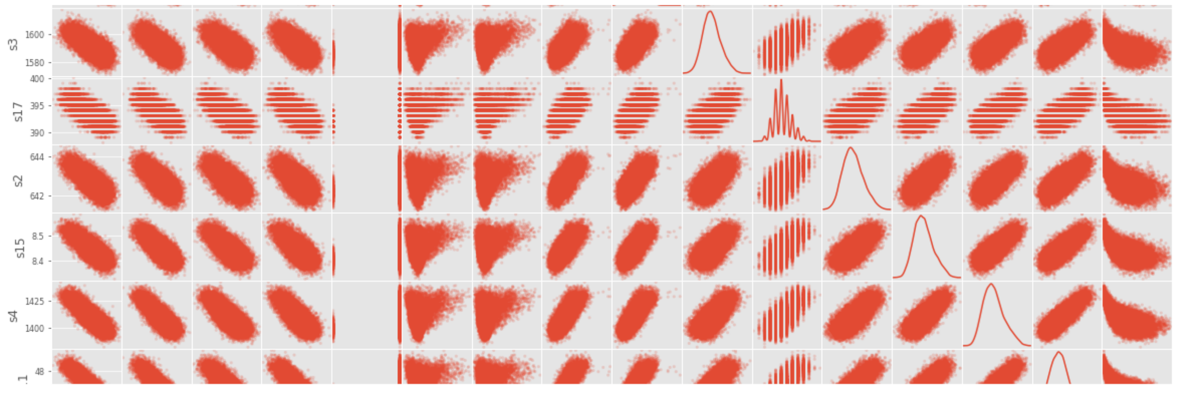


Features with high variability were checked for correlation with other features and regression label (TTF):



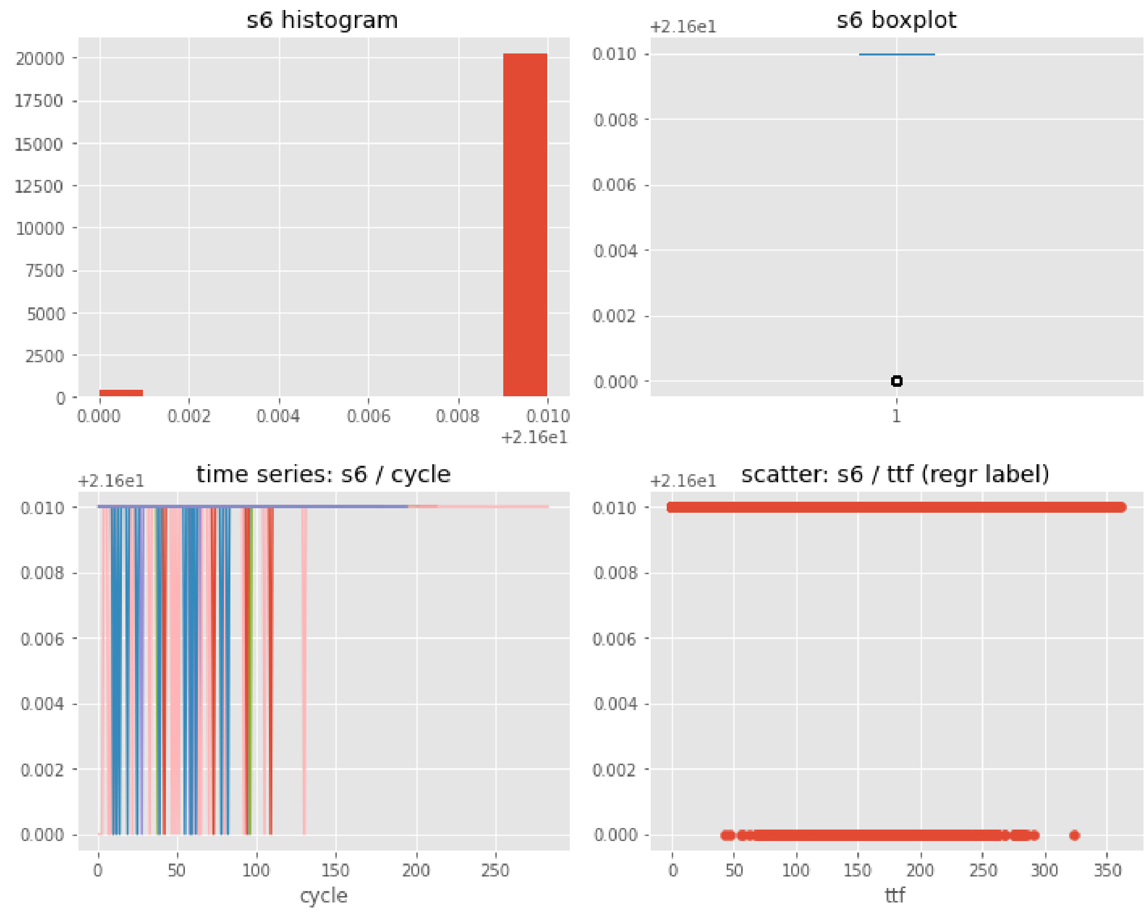
Scatter matrix was also used to check the distribution and correlation of features:



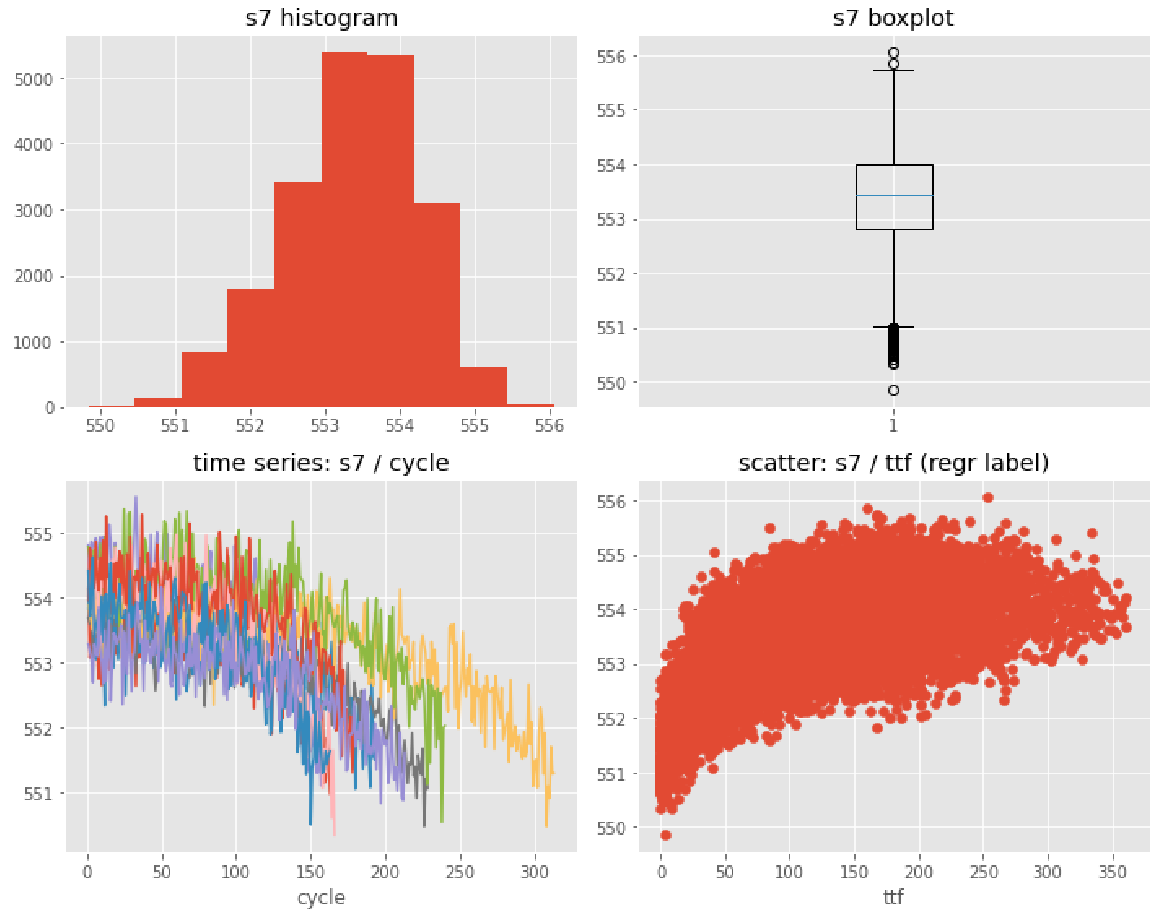


A number of EDA charts were also used to have more insights on each feature individually, eg:

s6)



s7)



There is a very high correlation (> 0.8) between some features e.g.: (s14 & s9), (s11 & s4), (s11 & s7), (s11 & s12), (s4 & s12), (s8 & s13), (s7 & s12).

This multicollinearity may hurt the performance of some machine learning algorithms. So, part of these features will be target for elimination in feature selection during the modeling phase.

Most features have nonlinear relation with the TTF, hence adding their polynomial transforms may enhance models performance.

Most features exhibit normal distribution which is likely to improve models' performance.

Models used:

Model 1:Random Forest(RF)

It can perform both regression and classification tasks. A random forest produces good predictions that can be understood easily. It can handle large datasets efficiently.

Random forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Model 2: Decision Tree

Decision Tree creates a training model that can be used to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

It builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed.

It learns in a hierarchical fashion by repeatedly splitting the dataset into separate branches that maximize the information gain of each split. In regression tree, the value obtained by terminal nodes in the training data is the mean response of observation falling in that region, whereas in the classification tree, the value (class) obtained by the terminal node in the training data is the mode of observations falling in that region.

Model 3: MLP

Multilayer Perceptrons, or MLPs for short, are the classical type of neural network. They are composed of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer. MLPs are suitable for classification prediction problems where inputs are assigned a class or label.

Model 4: LSTM

In LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things. The information at a particular cell state has three different dependencies.

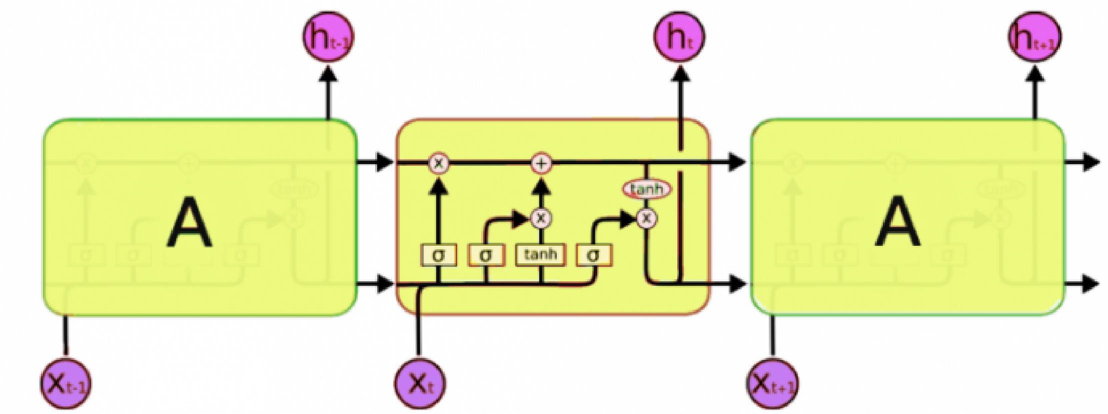
These dependencies can be generalized to any problem as:

The previous cell state (i.e. the information that was present in the memory after the previous time step)

The previous hidden state (i.e. this is the same as the output of the previous cell)

The input at the current time step (i.e. the new information that is being fed in at that moment).

A typical LSTM network is composed of different memory blocks called cells (the rectangles that we see in the image). There are two states that are being transferred to the next cell; the cell state and the hidden state. The memory blocks are responsible for remembering things and manipulations to this memory are done through three major mechanisms, called gates. The gates are: forget, input and output.

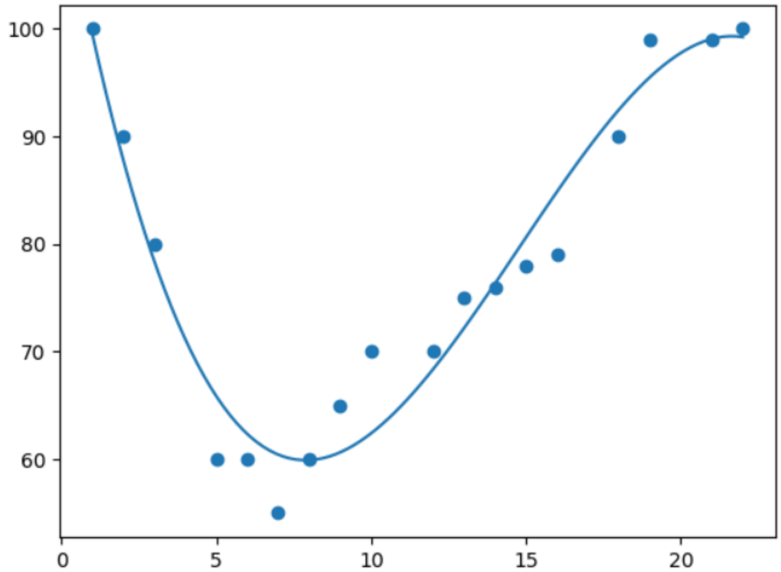


RNNs face the problem of Vanishing Gradient. Hence we implement LSTM. Among the deep learning methods, Long Short Term Memory LSTM networks are especially appealing to the predictive maintenance domain due to the fact that they are very good at learning from sequences. This fact lends itself to their applications using time series data by making it possible to look back for longer periods of time to detect failure patterns.

The first layer is an LSTM layer with 100 units followed by another LSTM layer with 50 units. Dropout is also applied after each LSTM layer to control overfitting. Final layer is a Dense output layer with single unit and sigmoid activation since this is a binary classification problem.

Model 5: Polynomial regression

Polynomial Regression: is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x.



Polynomial provides the best approximation of the relationship between the dependent and independent variable. A Broad range of functions can fit under it. When we observed the trend between the cycles and the sensor readings we noticed it is non-linear.

f. Screenshot and Demo along with Visualization (For results):

Predicting Engine's Time-To-Failure (TTF)

Key regression evaluation metrics calculated were Root Mean Squared Error (RMSE), R-squared (R2), Mean Absolute Error, and Explained Variance. The results on test dataset listed below:

```
import pandas as pd
import numpy as np



import matplotlib.pyplot as plt
plt.style.use('ggplot')
%matplotlib inline


from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor


from sklearn import model_selection #import cross_val_score, StratifiedKFold
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz
from sklearn import metrics  # mean_squared_error, mean_absolute_error, median_absolute_error, explained_variance_score, r2_score
from sklearn.feature_selection import SelectFromModel, RFECV
df_train = pd.read_csv('train.csv')
df_train.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	sd15	sd16	sd17	sd18	sd19	sd20	sd21	ttf	label_bnc	label_mcc
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	0.000000	0.0	0.000000	0.0	0.0	0.000000	0.000000	191	0	0
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	0.008697	0.0	0.000000	0.0	0.0	0.042426	0.003253	190	0	0
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	0.007640	0.0	1.154701	0.0	0.0	0.055076	0.044573	189	0	0
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	0.028117	0.0	1.000000	0.0	0.0	0.076322	0.037977	188	0	0
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	0.025953	0.0	1.095445	0.0	0.0	0.073621	0.033498	187	0	0

5 rows × 71 columns

df_test = pd.read_csv('test.csv')

df_test.head()

#Prepare data for regression model

 # original features

features_orig = ['setting1','setting2','setting3','s1','s2','s3','s4','s5','s6','s7','s8','s9','s10','s11','s12','s13','s14','s15','s16','s17','s18','s19','s20','s21']

 # original + extracted fetures

features_adxf = ['setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14', 's15', 's16', 's17', 's18', 's19', 's20', 's21', 'av1', 'av2', 'av3', 'av4', 'av5', 'av6', 'av7', 'av8', 'av9', 'av10', 'av11', 'av12', 'av13', 'av14', 'av15', 'av16', 'av17', 'av18', 'av19', 'av20', 'av21', 'sd1', 'sd2', 'sd3', 'sd4', 'sd5', 'sd6', 'sd7', 'sd8', 'sd9', 'sd10', 'sd11', 'sd12', 'sd13', 'sd14', 'sd15', 'sd16', 'sd17', 'sd18', 'sd19', 'sd20', 'sd21']

 # features with low or no correlation with regression label

features_lowcr = ['setting3', 's1', 's10', 's18','s19','s16','s5', 'setting1', 'setting2']

```
# features that have correlation with regression label

features_corrl = ['s2', 's3', 's4', 's6', 's7', 's8', 's9', 's11', 's12', 's13', 's14',
's15', 's17', 's20','s21']


# a variable to hold the set of features to experiment with

features = features_orig



X_train = df_train[features]

y_train = df_train['ttf']


X_test = df_test[features]

y_test = df_test['ttf']

def get_regression_metrics(model, actual, predicted):


    """Calculate main regression metrics.


    Args:

        model (str): The model name identifier

        actual (series): Contains the test label values

        predicted (series): Contains the predicted values
```

```
    Returns:
        dataframe: The combined metrics in single dataframe


    """
    regr_metrics = {
                        'Root Mean Squared Error' : metrics.mean_squared_error(actual, predicted)**0.5,
                        'Mean Absolute Error' : metrics.mean_absolute_error(actual, predicted),
                        'R^2' : metrics.r2_score(actual, predicted),
                        'Explained Variance' : metrics.explained_variance_score(actual, predicted)
                   }

    #return reg_metrics
    df_regr_metrics = pd.DataFrame.from_dict(regr_metrics, orient='index')
    df_regr_metrics.columns = [model]
    return df_regr_metrics
def plot_features_weights(model, weights, feature_names, weights_type='c'):
```

```
    """Plot regression coefficients weights or feature importance.

    Args:
        model (str): The model name identifier
        weights (array): Contains the regression coefficients weights or feature importance
        feature_names (list): Contains the corresponding features names
        weights_type (str): 'c' for 'coefficients weights', otherwise is 'feature importance'

    Returns:
        plot of either regression coefficients weights or feature importance


    """
    (px, py) = (8, 10) if len(weights) > 30 else (8, 5)
    W = pd.DataFrame({'Weights':weights}, feature_names)
    W.sort_values(by='Weights', ascending=True).plot(kind='barh', color='r', figsize=(px,py))
    label = ' Coefficients' if weights_type =='c' else ' Features Importance'
    plt.xlabel(model + label)
    plt.gca().legend_ = None
def plot_residual(model, y_train, y_train_pred, y_test, y_test_pred):
```

```
"""Print the regression residuals.

Args:
    model (str): The model name identifier
    y_train (series): The training labels
    y_train_pred (series): Predictions on training data
    y_test (series): The test labels
    y_test_pred (series): Predictions on test data

Returns:
    Plot of regression residuals

"""

plt.scatter(y_train_pred, y_train_pred - y_train, c='blue', marker='o', label='Training data')
plt.scatter(y_test_pred, y_test_pred - y_test, c='lightgreen', marker='s', label='Test data')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.legend(loc='upper left')
plt.hlines(y=0, xmin=-50, xmax=400, color='red', lw=2)
```

```
    plt.title(model + ' Residuals')
    plt.show()

#try Polynomial Regression

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)

X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.fit_transform(X_test)


polyreg = linear_model.LinearRegression()
polyreg.fit(X_train_poly, y_train)

y_test_predict = polyreg.predict(X_test_poly)
y_train_predict = polyreg.predict(X_train_poly)

print('R^2 training: %.3f, R^2 test: %.3f' % (
      (metrics.r2_score(y_train, y_train_predict)),
      (metrics.r2_score(y_test, y_test_predict))))
```

```
polyreg_metrics = get_regression_metrics('Polynomial Regression', y_test, y_test_predict)

polyreg_metrics

#try Decision Tree regressor


#dtrg = DecisionTreeRegressor(max_depth=8, max_features=5, random_state=123) # selected features

dtrg = DecisionTreeRegressor(max_depth=7, random_state=123)

dtrg.fit(X_train, y_train)


y_test_predict = dtrg.predict(X_test)

y_train_predict = dtrg.predict(X_train)


print('R^2 training: %.3f, R^2 test: %.3f' % (

    (metrics.r2_score(y_train, y_train_predict)),

    (metrics.r2_score(y_test, y_test_predict))))


dtrg_metrics = get_regression_metrics('Decision Tree Regression', y_test, y_test_predict)

dtrg_metrics

# try recursive feature elimination
```

```
kfold = model_selection.KFold(n_splits=5, random_state=10,shuffle=True)


dtrg = DecisionTreeRegressor(max_depth=7)


rfecv = RFECV(estimator=dtrg, step=1, cv=kfold, scoring='neg_mean_squared_error', n_jobs=-1)

rfecv.fit(X_train, y_train)


print("Optimal number of features : %d" % rfecv.n_features_)


sel_features = [f for f,s in zip(X_train.columns, rfecv.support_) if s]

print('The selected features are: {}'.format(sel_features))


# Plot number of features VS. cross-validation scores

plt.figure();

plt.xlabel("Number of features selected (RFE)")

plt.ylabel("Cross validation score (mse)")

plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)

plt.show()


#regression metrics comparison before feature engineering
```

```
reg_metrics_bfe = pd.concat([dtrg_metrics, polyreg_metrics, rf_metrics], axis=1)

reg_metrics_bfe
```

	Decision Tree Regression	Polynomial Regression	Random Forest Regression
Root Mean Squared Error	32.095349	31.956018	28.634253
Mean Absolute Error	24.319068	24.433672	23.167130
R^2	0.403480	0.408648	0.525198
Explained Variance	0.632767	0.633525	0.767320

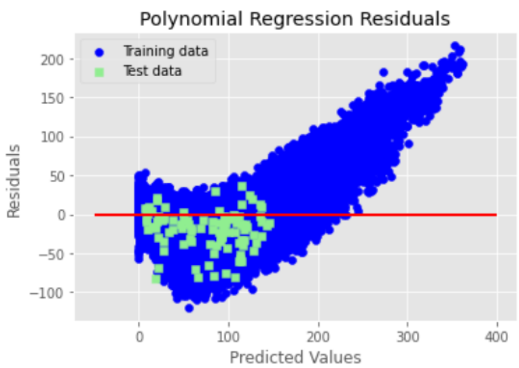
In accordance with our analysis in the data exploratory phase, non-linear regression models like Polynomial and Random Forest performed . Random Forest clearly outperformed other models scoring RMSE of 28.63 cycles, i.e. the model predicts TTF within average error range of ±28.63 cycles.

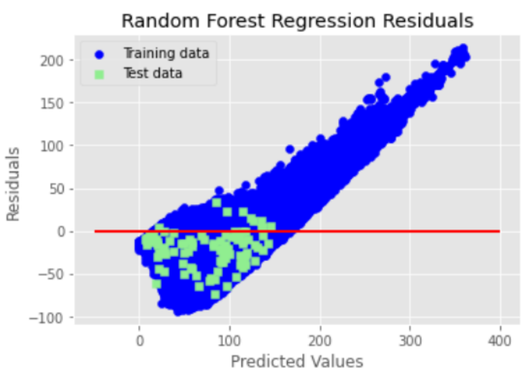
First 9 predictions of Random Forest model on test dataset were:

```
# view predictions vs actual

rf_pred_dict = {

            'Actual' : y_test,

            'Prediction' : y_test_predict

        }

rf_pred = pd.DataFrame.from_dict(rf_pred_dict).T

rf_pred
```

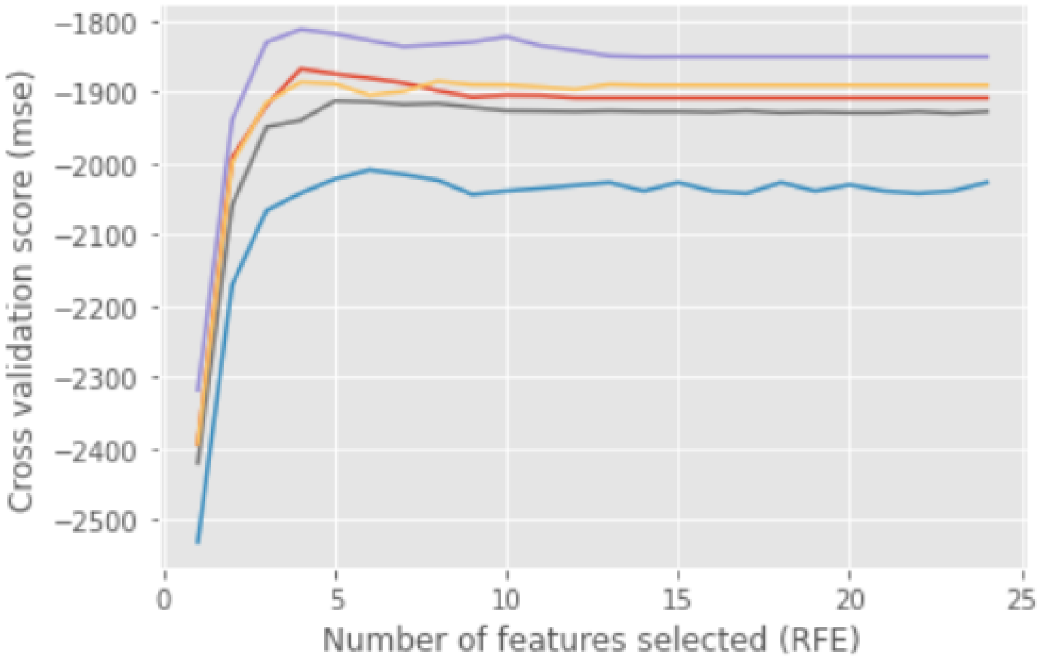
	0	1	2	3	4	5	6	7	8	9
Actual	112.000000	98.000000	69.000000	82.000000	91.00000	93.000000	91.000000	95.000000	111.00000	96.000000
Prediction	151.578408	119.268513	74.415647	96.470907	112.59338	130.279445	128.114019	100.692144	116.11839	127.369752

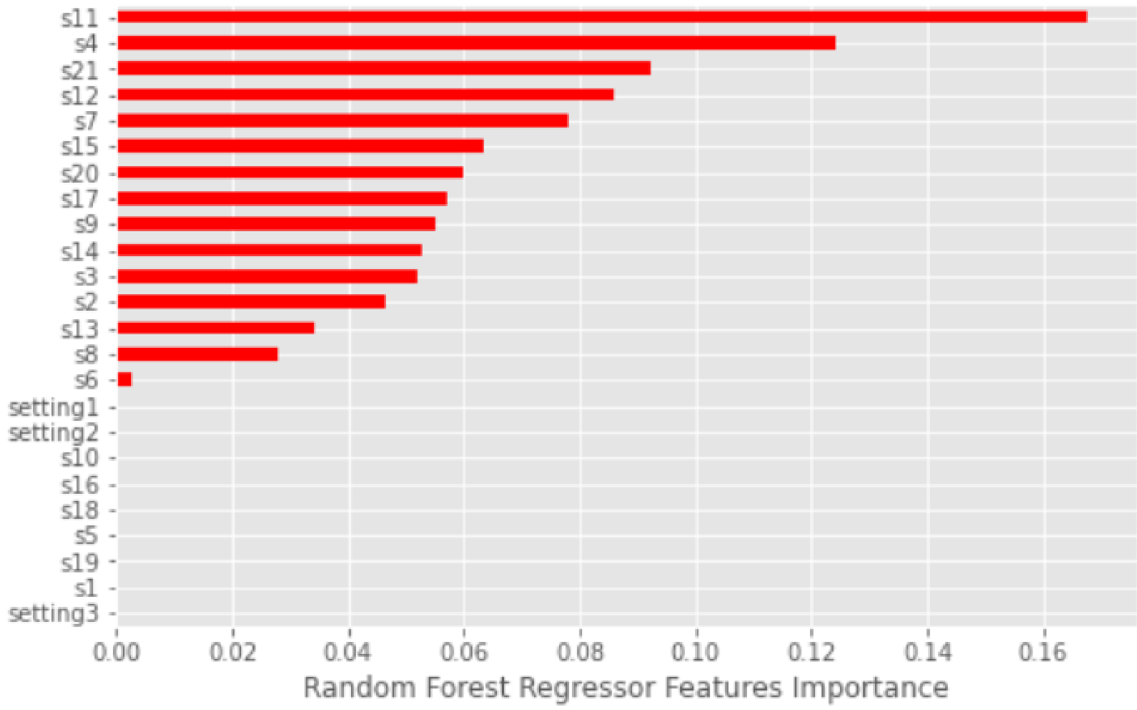
As per the regression residuals plot shown below, residuals were not randomly spread across the average value of the residuals. This could be improved by many methods including fixing the data (e.g. outliers), model parameters tuning, or trying other ML algorithms.





Key features were also examined through feature selection methods like Recursive Feature Elimination and through parameters returned by models like Random Forest feature importance as shown below:





Key binary classification performance metrics calculated include Area under Receiver Operating Characteristics Curve (AUC ROC), Recall,

Precision, F1 Score, and Accuracy. AUC ROC was the score used in Grid Search hyper-parameters tuning. As shown below, performance of various models was evaluated on the test dataset, where B stands for original features set (Before features extraction), and A stand for modified feature set (After feature extraction).

def multiclass_classify(model, clf, features, params=None, score=None, scale=False, OvR=True, prob='P'):

"""Perfor Grid Search hyper parameter tuning on a classifier.

Args:

model (str): The model name identifier

clf (clssifier object): The classifier to be tuned

features (list): The set of input features names

params (dict): Grid Search parameters

score (str): Grid Search score

OvR (bool): True if the classifier inherently support multiclass One-Vs-Rest

prob (str): For getting classification scores: 'P' for predict_proba, 'D' for decision_function

Returns:

Tuned Clssifier object

array: prediction values

array: prediction scores

```
"""

X_train = df_train[features]

X_test = df_test[features]

if scale:
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    grid_search = model_selection.GridSearchCV(estimator=clf, param_grid=params, cv=5, scoring=score, n_jobs=-1)

grid_search.fit(X_train, y_train)
y_pred = grid_search.predict(X_test)

if prob == 'P':
    y_score = grid_search.predict_proba(X_test)
    if OvR:
```

```
        y_score = [y_score[i][:,[1]] for i in range(len(y_score))]

        y_score = np.concatenate(y_score, axis=1)

    elif prob == 'D':

      y_score = grid_search.decision_function(X_test)

    else:

      y_score = y_pred



    return grid_search.best_estimator_, y_pred, y_score


#compare all models

metrics_mc = pd.concat([ metrics_dtr, metrics_rfc,metrics_nnc], axis=1).T

metrics_mc
```

	Accuracy	macro F1	micro F1	macro Precision	micro Precision	macro Recall	micro Recall	macro ROC AUC	micro ROC AUC
Decision Tree B	0.84	0.684053	0.861538	0.818970	0.884211	0.668889	0.84	0.905594	0.956625
Decision Tree A	0.84	0.607906	0.857143	0.852146	0.875000	0.651111	0.84	0.949857	0.973550
Random Forest B	0.82	0.612536	0.854167	0.776749	0.891304	0.573333	0.82	0.964340	0.978550
Random Forest A	0.85	0.705759	0.867347	0.800813	0.885417	0.662222	0.85	0.967744	0.980600
Neural Net MLP B	0.88	0.788832	0.894472	0.862860	0.898990	0.740000	0.89	0.971816	0.982550
Neural Net MLP A	0.87	0.752819	0.890000	0.845649	0.890000	0.755556	0.89	0.973144	0.981300

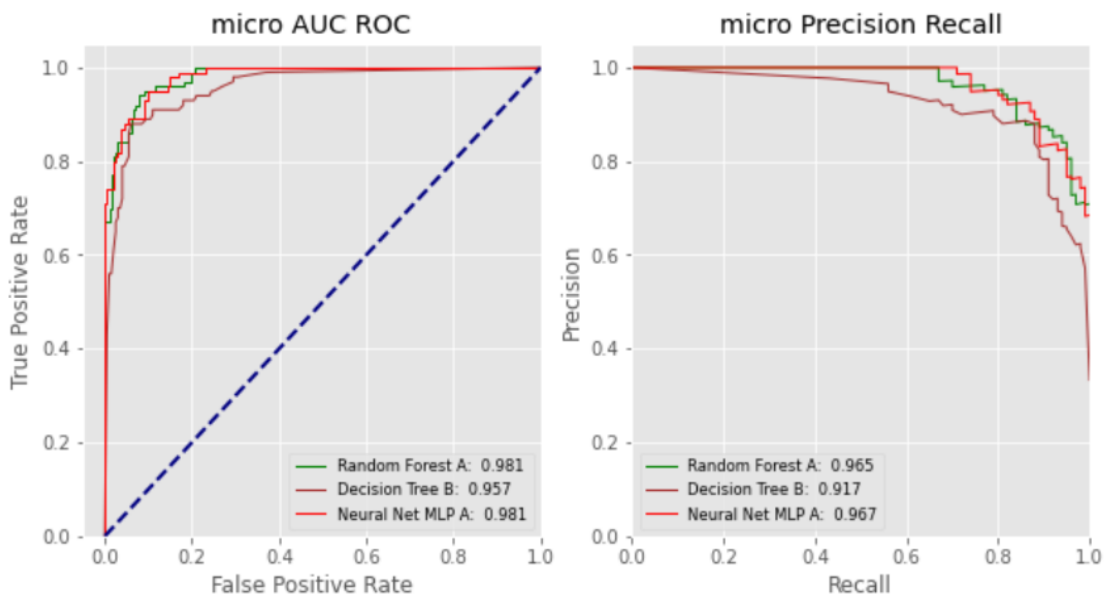
MLP and Random Forests scored best AUC ROC. It also noticed that feature extraction has improved most models performance metrics.

AUC for ROC and Precision-Recall curves were plotted for best models as shown below:

 # Plot AUC-ROC and precision-recall curves for best models

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharex=False, sharey=False
)

fig.set_size_inches(10,5)



ax1.plot(roc_rfca.FPR['micro'], roc_rfca.TPR['micro'], color='green',
lw=1, label= roc_rfca.Model['micro'] + ':  %.3f' % roc_rfca.AUC['micro'])


ax1.plot(roc_dtrb.FPR['micro'], roc_dtrb.TPR['micro'], color='brown',
lw=1, label= roc_dtrb.Model['micro'] + ':  %.3f' % roc_dtrb.AUC['micro'])


ax1.plot(roc_nnca.FPR['micro'], roc_nnca.TPR['micro'], color='red', lw=1,
label= roc_nnca.Model['micro'] + ':  %.3f' % roc_nnca.AUC['micro'])

ax1.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

ax1.set_xlim([-0.05, 1.0])

ax1.set_ylim([0.0, 1.05])

ax1.set_xlabel('False Positive Rate')

ax1.set_ylabel('True Positive Rate')

ax1.legend(loc="lower right", fontsize='small')

ax1.set_title('micro AUC ROC')
```

```
ax2.plot(prc_rfca.Recall['micro'], prc_rfca.Precision['micro'], color='green', lw=1, label= prc_rfca.Model['micro'] + ': %.3f' % prc_rfca['Avg Precision']['micro'])

ax2.plot(prc_dtrb.Recall['micro'], prc_dtrb.Precision['micro'], color='brown', lw=1, label= prc_dtrb.Model['micro'] + ': %.3f' % prc_dtrb['Avg Precision']['micro'])

ax2.plot(prc_nnca.Recall['micro'], prc_nnca.Precision['micro'], color='red', lw=1, label= prc_nnca.Model['micro'] + ': %.3f' % prc_nnca['Avg Precision']['micro'])
ax2.set_xlim([0.0, 1.0])
ax2.set_ylim([0.0, 1.05])
ax2.set_xlabel('Recall')
ax2.set_ylabel('Precision')
ax2.legend(loc="lower left", fontsize='small')
ax2.set_title('micro Precision Recall')
```



LSTM

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

 # Setting seed for reproducability

np.random.seed(1234)

PYTHONHASHSEED = 0

from sklearn import preprocessing

from sklearn.metrics import confusion_matrix, recall_score, precision_score

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM, Activation

%matplotlib inline

```
# Data ingestion - reading the datasets from Azure blob

!wget http://azuremlsamples.azureml.net/templatedata/PM_train.txt

!wget http://azuremlsamples.azureml.net/templatedata/PM_test.txt

!wget http://azuremlsamples.azureml.net/templatedata/PM_truth.txt

# Data Labeling - generate column RUL

rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()

rul.columns = ['id', 'max']

train_df = train_df.merge(rul, on=['id'], how='left')

train_df['RUL'] = train_df['max'] - train_df['cycle']

train_df.drop('max', axis=1, inplace=True)

train_df.head()

# MinMax normalization

train_df['cycle_norm'] = train_df['cycle']

cols_normalize =
train_df.columns.difference(['id','cycle','RUL','label1','label2'])

min_max_scaler = preprocessing.MinMaxScaler()

norm_train_df =
pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),

                            columns=cols_normalize,

                            index=train_df.index)

join_df =
train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)

train_df = join_df.reindex(columns = train_df.columns)
```

```
train_df.head()

# generate column max for test data

rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()

rul.columns = ['id', 'max']

truth_df.columns = ['more']

truth_df['id'] = truth_df.index + 1

truth_df['max'] = rul['max'] + truth_df['more']

truth_df.drop('more', axis=1, inplace=True)

# generate labels

label_gen = [gen_labels(train_df[train_df['id']==id], sequence_length, ['label1'])

            for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)

label_array.shape

# build the network

nb_features = seq_array.shape[2]

nb_out = label_array.shape[1]


model = Sequential()


model.add(LSTM(

        input_shape=(sequence_length, nb_features),
```

```
        units=100,
        return_sequences=True))
model.add(Dropout(0.2))


model.add(LSTM(
         units=50,
         return_sequences=False))
model.add(Dropout(0.2))


model.add(Dense(units=nb_out, activation='sigmoid'))
model.compile(loss='binary_crossentropy',                optimizer='adam',
metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 100)           50400

 dropout (Dropout)           (None, 50, 100)           0

 lstm_1 (LSTM)               (None, 50)                30200

 dropout_1 (Dropout)         (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 80,651
Trainable params: 80,651
Non-trainable params: 0
_________________________________________________________________
None
```

 # make predictions and compute confusion matrix

y_pred_test=(model.predict(seq_array_test_last) > 0.5).astype("int32")

#y_pred_test = model.predict_classes(seq_array_test_last)

y_true_test = label_array_test_last

print('Confusion matrix\n- x-axis is true labels.\n- y-axis is predicted labels')

cm = confusion_matrix(y_true_test, y_pred_test)

cm

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s16	s17	s18	s19	s20	s21	cycle_norm	RUL	label1	label2
0	1	1	0.632184	0.750000	0.0	0.0	0.545181	0.310661	0.269413	0.0	...	0.0	0.333333	0.0	0.0	0.558140	0.661834	0.00000	142	0	0
1	1	2	0.344828	0.250000	0.0	0.0	0.150602	0.379551	0.222316	0.0	...	0.0	0.416667	0.0	0.0	0.682171	0.686827	0.00277	141	0	0
2	1	3	0.517241	0.583333	0.0	0.0	0.376506	0.346632	0.322248	0.0	...	0.0	0.416667	0.0	0.0	0.728682	0.721348	0.00554	140	0	0
3	1	4	0.741379	0.500000	0.0	0.0	0.370482	0.285154	0.408001	0.0	...	0.0	0.250000	0.0	0.0	0.666667	0.662110	0.00831	139	0	0
4	1	5	0.580460	0.500000	0.0	0.0	0.391566	0.352082	0.332039	0.0	...	0.0	0.166667	0.0	0.0	0.658915	0.716377	0.01108	138	0	0

5 rows × 30 columns

values= [scores_test[1],precision_test,recall_test,f1_test]

series_values = pd.Series(values)

results_df = pd.DataFrame({'accuracy':series_values[0],'precision':series_values[1],'recall':series_values[2],'f1score':series_values[3]},index=[0])

results_df

```
values= [scores_test[1],precision_test,recall_test,f1_test]
series_values = pd.Series(values)

results_df = pd.DataFrame({'accuracy':series_values[0],'precision':series_values[1],'recall':series_values[2],'f1score':series_values[3]},index=[0])
results_df
```

	accuracy	precision	recall	f1score
0	0.967742	0.923077	0.96	0.941176

Comparison, Results and discussion along with Visualization

Neural Net Multi-layer Perceptron classifier clearly outperformed other models in all metrics, followed by Random Forests classifier.

Performance metrics for regression:

RMSE: used to measure the differences between values (sample or population values) predicte

MAE: mean absolute error is a measure of errors between paired observations expressing the same phenomenon by a model or an estimator and the values observed.

R^2: represents the coefficient of how well the values fit compared to the original values. The value from 0 to 1 is interpreted as percentages. The higher the value is, the better the model is.

explained variance: explained variation measures the proportion to which a mathematical model accounts for the variation of a given data set.

Performance metrics for classification:

False negatives and false positives are samples that were incorrectly classified

True negatives and true positives are samples that were correctly classified

Accuracy is the percentage of examples correctly classified .

Precision is the percentage of predicted positives that were correctly classified.

Recall is the percentage of actual positives that were correctly classified.

AUC refers to the Area Under the Curve of a Receiver Operating Characteristic curve (ROC-AUC). This metric is equal to the probability that a classifier will rank a random positive sample higher than a random negative sample.

The support is the number of samples of the true response that lie in that class

7. References

i) Heim, S., Clemens, J., Steck, J. E., Basic, C., Timmons, D., & Zwiener, K. (2020). Predictive maintenance on aircraft and applications with digital twin. Paper presented at the Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020, 4122-4127. doi:10.1109/BigData50022.2020.9378433

ii)A. P. Hermawan, D. -S. Kim and J. -M. Lee, "Predictive Maintenance of Aircraft Engine using Deep Learning Technique," 2020 International Conference on Information and Communication Technology Convergence (ICTC), 2020, pp. 1296-1298, doi: 10.1109/ICTC49870.2020.9289466.

iii)Y. Liu, J. Zeng, L. Xie, S. Luo, and H. Su, “Structured joint sparse principal component analysis for fault detection and isolation,” IEEE Transactions on Industrial Informatics, vol. 15, no. 5, pp. 2721–2731, 2019.

iv)Kadir Celikmih, Onur Inan , and Harun Uguz “Failure Prediction of Aircraft Equipment Using Machine Learning with a Hybrid Data Preparation Method”,Hindawi Scientific Programming Volume 2020, Article ID 8616039, 10 pages https://doi.org/10.1155/2020/8616039