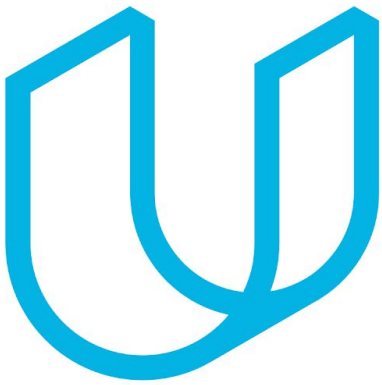
Machine Learning Engineer Nanodegree

Predicting Device Failures with Ensemble Models

Michael Lin

July, 2019



UDACITY

I. Definition

Project Overview

Modern industrial economies run on machines. From machineries seen on production floors, to tractors used in farms, from cars running on highways, to internet of things deployed in the field, machines are everywhere.

Overtime, machines do wear out, however. The ability to identify which machines become faulty before they actually break down can pay great dividends. A joint study by Wall Street Journal and Emerson found that unplanned downtime costs industrial manufacturers an estimated $50 Billion per year. Equipment failure is the cause of 42% of this unplanned downtime[1].

Taking preemptive actions before device failures, or predictive maintenance, is an area of active investigation by practitioners of machine learning and AI.

The use case for predictive maintenance is numerous[2]. In automotive, data from sensors in vehicles helps alert drivers of issues that need servicing. In utility, data from smart meters helps detect early signs of issues on the grid. In the internet of things, data from sensors in factories and products in combination with algorithms help uncover warnings of costly failures before they occur. And in insurance, data helps predict likelihood of extreme weather conditions.

On modeling front, predictive maintenance does present a unique challenge, namely that of imbalanced data distribution[3]. Failure cases are considered rare events, as most of the time machines function properly. The ability to discern truly failure cases in the training data is what motivates me to select this topic for capstone.

Moreover, we may require more sophisticated modeling approach to pick up the underlying complex relationship between sensor reading and machine failure. Ensemble models[4], combining feedback from many models to yield more accurate prediction is desirable in predictive maintenance. In such case, ability to accurately predict and take preemptive actions may be more important than to have a clear explainability of cause and effect.

Problem Statement

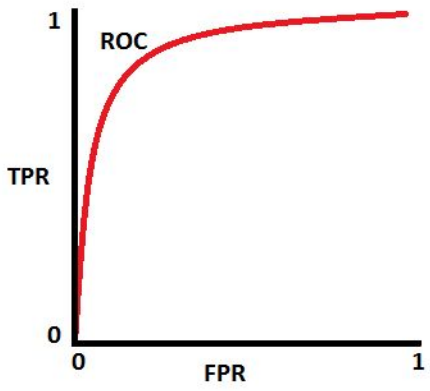
The objective is to accurately predict devices that require maintenance, while at the same time, minimize false positive cases (predicted failures that are not faulty) and false negative cases (predicted non-failures that are actually faulty). Wrongful classification of either type is not desirable. False positives mean unnecessary down time when we take devices out of service rotation, and incur cost of applying maintenance. False negatives translate to liability when machines break down unexpectedly.

Metrics

We envision the assignment as a classification challenge. We use devices' sensor reading to predict whether a particular device will be faulty the next working day.

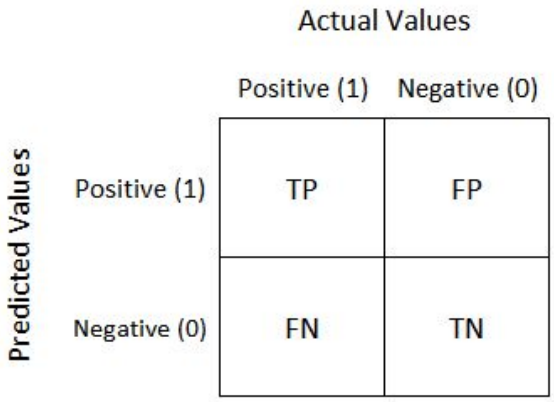
We propose to correctly identify as many true failure and non-failure cases as possible, while at the same time, minimize incidences of false positives and false negatives. We use area under the curve (AUC) as the scoring criteria to guide model search.

Intuitively, as we lower decision threshold, moving along the lower left of receiver operating characteristic curve (ROC curve), we want to increase more of true positives but less of false negatives. Conversely, when increase decision threshold, moving along the upper right of the ROC curve, we want to increase more of true negatives but less of false negatives. The area under the curve, AUC, gives us this precise measure. The larger the area, the more performant our model.



We also use confusion matrix to provide diagnostics of how well our model is performing in validation, balancing cases of false positives vs. false negatives. Intuitively we want to minimize the sum of false positives (FP) and false negatives (FN) while we strive to maximize the number of true positives (TP) and true negatives (TN).

Comparing FP and FN predictions across different models give us the hint of the better performing model specifications.



II. Analysis

Data Exploration

The dataset for the capstone is sourced from a github repository (https://github.com/dsdaveh/device-failure-analysis/blob/master/device_failure.csv).

The fields in the dataset include the following: date, device, failure, attribute1, attribute2, attribute3, attribute4, attribute5, attribute7, attribute8, attribute9.

```
# load in sample dataset

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

url = 'https://udacity-data-michlin0825.s3-ap-northeast-1.amazonaws.com/device_failure.csv'
df = pd.read_csv(url, header = 0, delimiter = ',', encoding = "ISO-8859-1")
df.head()
```

	date	device	failure	attribute1	attribute2	attribute3	attribute4	attribute5	attribute6	attribute7	attribute8	attribute9
0	2015-01-01	S1F01085	0	215630672	56	0	52	6	407438	0	0	7
1	2015-01-01	S1F0166B	0	61370680	0	3	0	6	403174	0	0	0
2	2015-01-01	S1F01E6Y	0	173295968	0	0	0	12	237394	0	0	0
3	2015-01-01	S1F01JE0	0	79694024	0	0	0	6	410186	0	0	0
4	2015-01-01	S1F01R2B	0	135970480	0	0	0	15	313173	0	0	3

The name of the sensor readings are concealed from us. We will not be able to leverage understanding of the sensor types to gauge its impact on device failures.

Initial analysis of the dataset reveals the following. The dataset is telemetry logs of 124,494 records, collected from 1,169 devices. The device failure rate is 9.07%.

```
# percentage of device with failure

len(df[df['failure']==1]) / len(df.groupby('device')) * 100
```

```
9.067579127459368
```

The data possibly comes from 3 different manufacturers (or device types).

```
# device types

df['device'].str[:3].value_counts(normalize = True)
```

```
S1F    0.440648
W1F    0.347551
Z1F    0.211801
Name: device, dtype: float64
```

The data spans period of 2015/1/1 to 2015/11/2.

```
# dataset timeframe

df['date'].head(1)
```

```
0    2015-01-01
Name: date, dtype: object
```

```
df['date'].tail(1)
```

```
124493    2015-11-02
Name: date, dtype: object
```

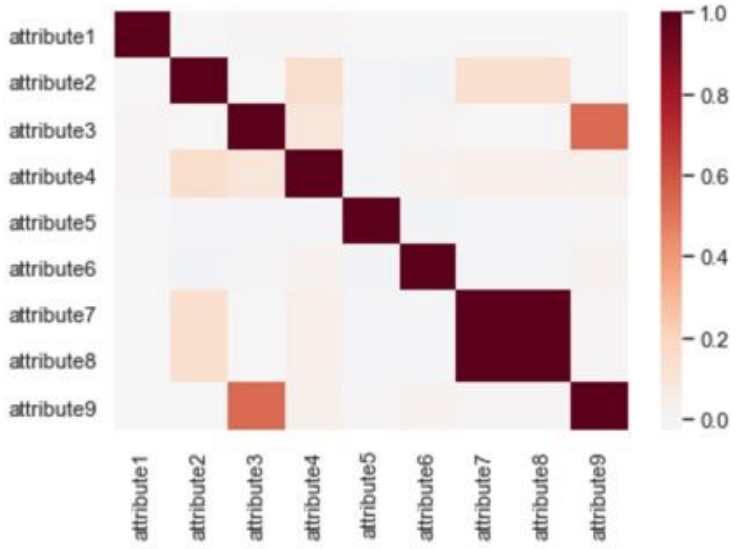
The duration of device in service varies widely, ranging from 1 day to 304 days.

```
# device service days

print(df.groupby('device')['date'].count().max())
print(df.groupby('device')['date'].count().mean())
print(df.groupby('device')['date'].count().min())
```

```
304
106.4961505560308
1
```

Exploratory Visualization

The heatmap of sensor reading is particularly critical in identifying perfect collinearity between attribute8 and attribute9. Dropping one of the duplicates helps model fitting with greater accuracy and stability.



Algorithms and Techniques

Chosen models are all pertinent to classification challenges, with model prediction in the range of 0 and 1, with 1 predicting device failure and 0 non-failure.

We first fit KNN and SVM. Both are distanced based models that historically yield reasonable results for classification challenges.

KNN: It stands for k nearest neighbors. The algorithm classifies a new case based on the distance of the case to its nearest neighbors in a hyperplane. If the majority of the surrounding cases are of one class, the new case will be assigned that class. The key is to find the optimum number of neighbors to work out the solution.

SVM: It is a classifier that tries to find a dividing hyperplane such that one class of the cases fall on one side and the other class the other side. The dividing hyperplane can be linear or non-linear depending on the characteristics of the data we encounter. The choice of the ‘kernel’, which decides the shape of the hyperplane, is a key hyperparameter we need to tune.

We also explore ensemble models of AdaBoost, Gradient Boosting, Random Forest, Extra Trees, XGBoost, LightGBM, and a voting model.

AdaBoost: Boosting is a technique where models are added sequentially until no further improvements can be made. AdaBoost weights data points that are difficult to predict so model specification can adapt and accommodate those difficult data points.

Gradient Boosting: New models are created to predict the residuals of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent to minimize the loss when adding new models.

Random Forest: The idea is to fit many models to the same problem, taking repeated samples of features and observations. A large number of relatively uncorrelated trees operate as a committee to vote on the outcome.

Extra Trees: It stands for extremely randomized trees. The algorithm fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy.

XGBoost: It stands for eXtreme Gradient Boosting. It is also the winner of many Kaggle competitions. The speed of the model is fast partly because it is written in C++, and partly it makes use of parallelization in CPUs and distributed computing across clusters.

LightGBM: It is decision-tree based algorithm. However, it splits the tree leaf wise, instead of depth wise or level wise. And it is also very fast, hence the label ‘Light’.

Voting model: Essentially we take majority votes of many model predictions, across different approaches, as our final predictions. By experimenting with the kind of models to include in the voting process, and weights we give to each individual model, we find a winning combination.

Benchmark

Baseline: The naive model uses the majority class of the training to predict outcome in validation. As over 90% of cases in training is non-failure, baseline model predicts all cases in validation to be non-faulty.

```
# predict class using majority class from training dataset

from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix

dummy_majority = DummyClassifier(random_state = seed, strategy = 'most_frequent').fit(X_train, Y_train)
Y_majority_predicted = dummy_majority.predict(X_validation)
confusion = confusion_matrix(Y_validation, Y_majority_predicted)

print('Most frequent class (dummy classifier)\n', confusion)
```

```
Most frequent class (dummy classifier)
 [[214   0]
 [ 20   0]]
```

```
# generate AUC

from sklearn.metrics import roc_auc_score

print('AUC')
print(roc_auc_score(Y_validation, Y_majority_predicted))
print()
```

```
AUC
0.5
```

III. Methodology

Data Preprocessing

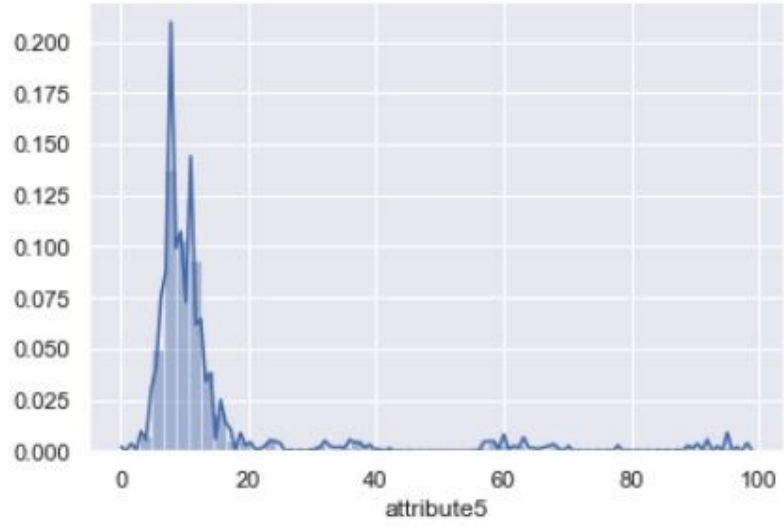
We identify a duplicate record in the device log, and decide to drop it.

We find attribute8 and attribute9 with perfect correlation, reflecting possibly duplicate entries. We opt to drop attribute9 to avoid perfect collinearity.

Features attribute1 to attribute9 do not conform to normal distribution, and varies widely in terms of scale. For example attribute5 is highly skewed towards the right. We plan to standardize the scale constrainting to unit variance of one. This step is essential as some of the distance based models (e.g. KNN, SVM) are sensitive to scale. However, we postpone the treatment to after splitting the data into training and validation to avoid data leakage.

```
sns.distplot(df['attribute5'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a25623c50>
```



For feature engineering, we extract day_of_week, month_of_year, service_days, and device_types.

```
df1_cut.head()
```

device	date	failure	day_of_week	month_of_year	service_days	device_type
S1F01085	2015-01-06	0	1	1	5	S1F
S1F013BB	2015-05-11	0	0	5	5	S1F
S1F0166B	2015-01-06	0	1	1	5	S1F
S1F01E6Y	2015-02-17	0	1	2	47	S1F
S1F01JE0	2015-01-06	0	1	1	5	S1F

In terms of attributes, we use sensor reading of the previous day, the previous 2 days, and previous 3 days as predictors. If we use reading of last day to predict possible device failure, the setup leaves us with no lead time for intervention, and won't qualify as a practical model.

device	date	att1_pre1day	att2_pre1day	att3_pre1day	att4_pre1day	att5_pre1day	att6_pre1day	att7_pre1day	att8_pre1day
S1F01085	2015-01-06	97393448.0	56.0	0.0	52.0	6.0	408114.0	0.0	0.0
S1F013BB	2015-05-11	85127128.0	0.0	0.0	0.0	5.0	689062.0	0.0	0.0
S1F0166B	2015-01-06	224339296.0	0.0	3.0	0.0	6.0	403812.0	0.0	0.0
S1F01E6Y	2015-02-17	182876688.0	0.0	0.0	0.0	12.0	259486.0	0.0	0.0
S1F01JE0	2015-01-06	158246712.0	0.0	0.0	0.0	6.0	410888.0	0.0	0.0

After dummy coding all categorical variables, and merging with numerical variables, including those 8 sensor readings across past 3 days, we are left with a working dataset of 43 variables, 1 label, and 1,167 records. This is the cleaned dataset before any case weighting to deal with imbalanced data issue.

```
df.shape
```

```
(1167, 44)
```

```
df.columns
```

```
Index(['dfw_1', 'dfw_2', 'dfw_3', 'dfw_4', 'dfw_5', 'dfw_6', 'mfy_2', 'mfy_3',
       'mfy_4', 'mfy_5', 'mfy_6', 'mfy_7', 'mfy_8', 'mfy_9', 'mfy_10',
       'mfy_11', 'device_W1F', 'device_Z1F', 'service_days', 'att1_pre1day',
       'att2_pre1day', 'att3_pre1day', 'att4_pre1day', 'att5_pre1day',
       'att6_pre1day', 'att7_pre1day', 'att8_pre1day', 'att1_pre2day',
       'att2_pre2day', 'att3_pre2day', 'att4_pre2day', 'att5_pre2day',
       'att6_pre2day', 'att7_pre2day', 'att8_pre2day', 'att1_pre3day',
       'att2_pre3day', 'att3_pre3day', 'att4_pre3day', 'att5_pre3day',
       'att6_pre3day', 'att7_pre3day', 'att8_pre3day', 'failure'],
      dtype='object')
```

Implementation

We purposely set aside 20% of data as hold-out. We use this pool to validate our model performance. The remaining 80% of data is for training of model parameters.

```
#Splitting validation dataset

X = df.drop(['failure'], axis = 1).values
Y = df['failure'].values

validation_size = 0.20
seed = 7
np.random.seed(7)

from sklearn.model_selection import train_test_split
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,
                                                                test_size=validation_size,
                                                                random_state=seed,
                                                                stratify=Y)
```

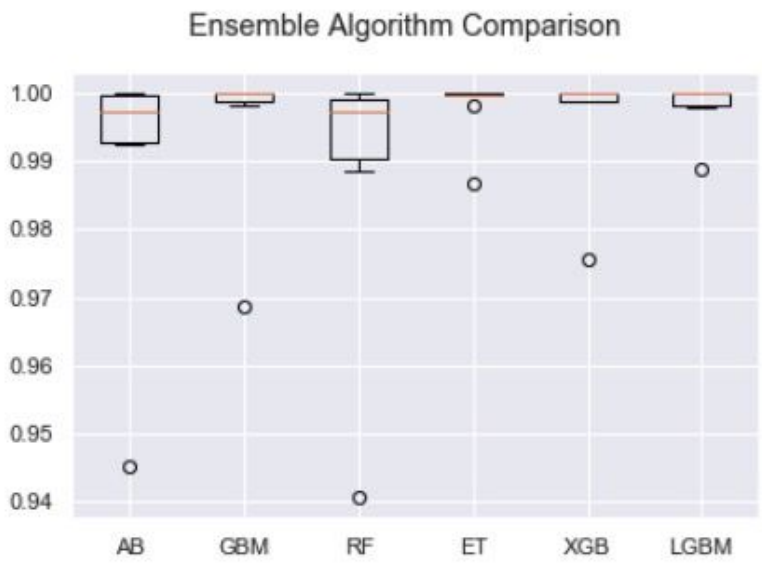
And for each round of model training, we propose over-sampling of minority class using SMOTE[5], keeping mix of failure vs. non-failure cases fixed throughout training cycle.

```
# over-sampling minority class to deal with imbalanced dataset

# option1: SMOTE

from imblearn.over_sampling import SMOTE
from collections import Counter

smote = SMOTE(ratio = 'minority', random_state = seed)
X_train, Y_train = smote.fit_sample(X_train, Y_train)
X_train.shape, Y_train.shape
```

```
((1704, 43), (1704,))
```

Refinement

Together we fit 10 model specifications. We start with simple baseline, using majority class of the training to predict outcome in validation. We then move to fitting 2 variants of classification models, namely KNN and SVM, before trying out 6 different ensemble models, namely AdaBoost, Gradient Boosting, Random Forest, Extra Trees, XGBoost[6], and LightGBM[7]. And finally we venture into voting model, combining predictions of the best three ensembles, e.g. AdaBoost, Extra Trees, and LightGBM, to yield prediction.



During each model fitting, we also grid search optimum values of selective hyperparameters to perfect model performance.

	Hyperparameters tuned	Range of value tried
AB	n_estimators learning_rate	[300, 400, 500, 600] [1,0.1, 0.01]
GBM	n_estimators max_depth	[200, 300, 400] [10, 20, 30]
RF	n_estimators	[400, 500, 600, 700]
ET	n_estimators max_depth	[400, 500, 600] [30,40,50]
XGB	n_estimators max_depth	[400, 500, 600] [5, 10, 15]
LGBM	n_estimators max_depth	[400, 500, 600] [5, 10, 15]

As a final calibration, we adjust the prediction cut-off point from typical 0.5 to 0.31.

IV. Results

Model Evaluation and Validation

During each iteration of the model fitting, we are able to reduce the number of false positives and false negatives in confusion matrix, and improve designated score of AUC. The AUC score increased from 0.5 in baseline model to 0.90+ in models of classifications, to 0.98+ in ensembles.

	TP	TN	FP	FN	FP+FN	AUC
Baseline	0	214	0	20	20	0.500
KNN	12	205	9	8	17	0.900
SVM	11	212	2	9	11	0.920
AB	14	214	0	6	6	0.984
GBM	12	212	2	8	10	0.982
RF	11	213	1	9	10	0.980
ET	16	213	1	4	5	0.996
XGB	12	214	0	8	8	0.988
LGBM	14	214	0	6	6	0.987
Voting	14	214	0	6	6	0.993
Voting + custom cut-point	14	214	2	2	4	0.993

Justification

Procedure wise, we set up 80% training and 20% validation, model performance is evaluated in data it has not seen before. The setup helps reduce risk of over-fitting. Performance results should be generalizable to future datasets.

Substantively, AUC of final voting is 0.993, compared to baseline model of 0.5. It is a 99% improvement. As for confusion matrix, voting model with cut-point calibration only

miss 4 cases (2 FP and 2 FN) in validation dataset, where as baseline model miss 20 cases (20 FN). The improvement is 5 folds. If we scale the result to production, the savings can be quite substantial.

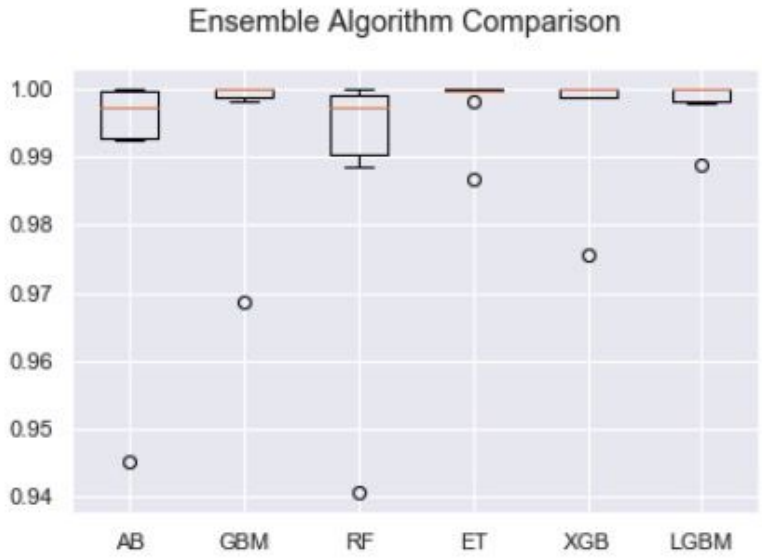
```
custom_cut = np.where(predictions_proba > 0.31, 1, 0)
print(confusion_matrix(Y_validation, custom_cut[:,1]))
```

```
[[212   2]
 [  2  18]]
```

V. Conclusion

Free-Form Visualization

Box-plots of the ensemble models, with all default parameters, yields clear visual comparison of the model performance. The plot provides spread of the estimates across training folds. Ideally we want models to have high scores on designated criteria of AUC, but also tighter spreads across training folds.



Reflection

All in all, we successfully build a model to predict device failure reasonably well. We pull in most of the data science toolsets to arrive at the milestone. We relentlessly cleanse the data. We extract and build new features. We normalize the data to be comparable across devices with different length of historical information. We fit a series of models from naive baseline, to families of classification models including KNN and SVM, to ensembles of averaging and boosting schools. We even experiment with voting model taking in best of all ensembles. We go the extra mile adjusting the prediction cut-off point to further optimize. In the end, we use 1,167 device records to train our model, and correctly classify 18 out of the 20 actual failure cases from validation pool of 234. The model precision is 90%, and recall also 90%.

I am particularly engaged with the oversampling of the minority class with SMOTE. I think this approach gives the model a good exposure of both positive and negative cases to estimate model parameters with confidence.

Another learning I found useful is the ability to extract features from time series and align with the outcome of interest. In this capstone we use sensor reading of the past to predict potential failure in the future. Data wrangling is key in getting dataset ready for this exercise.

Improvement

Time permitting, there are always things to try in a modeling exercise. Iteration towards most optimal solution is the essence of doing data science.

Talk to domain expert, and understand the meaning of the 8 telemetry attributes. Knowing what those attributes mean, and why they are collected in the first place should give insight on how to construct the features to capture the failure dynamics.

Collect more data, especially failure data points with diverse range across the measured attributes should enable us to train the model with more confidence, instead of relying on resampling or synthetic data points for the purpose.

Try other resampling methods. We have not exhausted all possible options. We might get mileage from using both under-sampling of majority class, and oversampling of minority class.

We can experiment with other ideas of feature engineering. For instance, creating difference between sensor reading of past 1 day and past 3 days, or variance of the past 3 days.

In model scoring, we treat the cost of making false positives and false negatives equal. If we know better the relative cost of making false positives vs. false negatives, we can factor this into optimization. The revised prediction should make more economic sense in a production setting.

References

[1] IndustryWeek, “How Manufacturers Achieve Top Quartile Performance” https://partners.wsj.com/emerson/unlocking-performance/how-manufacturers-can-achieve-top-quartile-performance/

[2] William Trotman, “5 Use Cases for Predictive Maintenance and Big Data” https://blogs.oracle.com/bigdata/predictive-maintenance-big-data-use-cases

[3] John Brennan, “Dealing with Imbalanced Data’ https://www.migarage.ai/intelligence/imbalanced-data/

[4] Joseph Rocca, “Ensemble methods: bagging, boosting and stacking” https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[5] Nick Becker, “The Right Way to Oversample in Predictive Modeling” https://beckernick.github.io/oversampling-modeling/

[6] Jason Brownlee, “How to Develop Your First XGBoost Model in Python with scikit-learn” https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

[7] Microsoft, “LightGBM” https://github.com/Microsoft/LightGBM/blob/master/examples/python-guide/simple_example.py