Automatic extraction of relevant features from time series:

[http://tsfresh.readthedocs.io](http://tsfresh.readthedocs.io/)

https://github.com/blue-yonder/tsfresh

DFT,DWT一堆公式没概念，LPC, cepstrum，唯一能有点亲切感的是看到一个haar小波变换...]

特征提取的核心在于转换到频域空间进行分析。而基本技术，就是把时间序列分割成一个个时间窗口来分析，每个时间窗口对应到一个特征向量。

对于不同的数据特征，选择不同的算法组合...目前感觉技术研究比较成熟的主要是语音波形方面的特征提取。

[参考书目]

1. http://web.science.mq.edu.au/~cassidy/comp449/html/ch09.html

2. <Time series feature extraction for data mining using DWT and DFT> Fabian M¨orchen

#### Principle not to generate leaking feature (over-fitting)

**注意：交叉验证之前，需要将训练集与验证集分开进行特征工程，避免训练集中生成的特征包含验证集信息，尤其针对rolling aggregates方法！**

#### summary

We’re dealing with a time dimension. This is a very common problem. How to address it?

One can pretend the data is static and use feature engineering to account for time. As the features we could have, for example, binary indicators for the day of week (already provided), the month, perhaps the year.

Another possibility is to come up with a model inherently capable of dealing with time series.

Time series data always comes with time-stamps which makes it suitable for calculating lagging features.

For numerical values, a common method is to pick a window size for the lag features to be created and compute rolling aggregate measures such as mean, standard deviation, minimum, maximum, etc. to represent the short term history of the telemetry over the lag window.

For categorical values, aggregating methods such as averaging does not apply. Counting the different categories is a more viable approach where lagging counts of different types of category that occured in the lag window are calculated.

For event(特殊事件), calculate how long it has been since the last event take place.

#### feature engineering case

Before building predictive models, we expect the **business expert** to understand the data relevancy requirement and provide the domain knowledge that is needed to select relevant subsets of data for the analysis.

There are three essential data sources we look for when qualifying a business problem to be suitable for a predictive maintenance solution:

1. Failure History: Typically, in predictive maintenance applications, failure events are very rare. However, when building predictive models that predict failures, the algorithm needs to learn **the normal operation pattern** as well as **the failure pattern** through the training process. Hence, it is essential that the training data contains sufficient number of examples in both categories in order to learn these two different patterns. For that reason, we require that data has sufficient number of failure events. Failure events can be found in maintenance records and parts replacement history or anomalies in the training data can also be used as failures as identified by the domain experts.
2. Maintenance/Repair History: An essential source of data for predictive maintenance solutions is the detailed maintenance history of the asset containing information about the components replaced, preventive maintenance activates performed, etc. It is extremely important to capture these **events** as these affect the degradation patterns and absence of this information causes misleading results.
3. Machine Conditions: In order to predict how many more days (hours, miles, transactions, etc.) a machine lasts before it fails, we **assume the machine’s health status degrades over time during its operation.** Therefore, we expect the data to contain **time-varying features** that capture this aging pattern and **any anomalies** that leads to degradation. In IoT applications, the telemetry data from different sensors represent one good example. In order to predict if a machine is going to fail within a time frame, ideally the data should capture degrading trend during this time frame before the actual failure event.

Additionally, we require data that is directly related to the operating conditions of the target asset of prediction. The decision of target is based on both business needs and data availability.**For example:** Taking the train wheel failure prediction as an example, we may predict "if the wheel is going to have a failure" or "if the whole train is going have a failure". The first one targets a more specific component whereas the second one targets failure of the train. The second one is a more general question that requires a lot more dispersed data elements than the first one, making it harder to build a model. Conversely, trying to predict wheel failures just by looking at the high-level train condition data may not be feasible as it does not contain information at the component level. In general, it is more sensible to predict **specific** failure events than more general ones.

##### Data sources

The common data elements for predictive maintenance problems can be summarized as follows:

* Failure history: The failure history of a machine or component within the machine.
* Maintenance history: The repair history of a machine, e.g. error codes, previous maintenance activities or component replacements.
* Machine conditions and usage: The operating conditions of a machine e.g. data collected from sensors.
* Machine features: The features of a machine, e.g. engine size, make and model, location.
* Operator features: The features of the operator, e.g. gender, past experience.

It is possible and usually the case that failure history is contained in maintenance history such as in the form of special error codes or order dates for spare parts. In those cases, failures can be extracted from the maintenance data. Additionally, different business domains may have a variety of other data sources that influence failure patterns which are not listed here exhaustively. These should be identified by consulting the domain experts when building predictive models.

Given the above data sources, the two main data types we observe in predictive maintenance domain are **temporal data** and **static data**. Failure history, machine conditions, repair history, usage history almost always come with time-stamps indicating the time of collection for each piece of data. Machine features and operator features in general are static since they usually describe the technical specifications of machines or operator’s properties. It is possible for these features to change over time and if so they should be treated as time stamped data sources.

##### Merging data sources

Before getting into any type of **feature engineering** or **labeling** process, we need to first prepare our data in the form required to create features from. The ultimate goal is to generate a record for each **time unit** for each asset with its features and labels to be fed into the machine learning algorithm.Data collection can also be divided into other units such as actions, however for simplicity we use time units for the rest of the explanations.

The measurement unit for time can be in seconds, minutes, hours, days, months, cycles, miles or transactions depending on the efficiency of data preparation and the changes observed in the conditions of the asset from a time unit to the other or other factors specific to the domain. In other words, **the time unit does not have to be the same as the frequency of data collection as in many cases data may not show any difference from one unit to the other.** For example, if temperature values were being collected every 10 seconds, picking a time unit of 10 seconds for the whole analysis inflates the number of examples without providing any additional information. Better strategy would be to use average over an hour as an example.

##### Feature engineering

The feature engineering methods described below can be used as baseline for creating features.

Below, we discuss **lag features** that should be constructed from data sources that come with time-stamps and also **static features** created from static data sources and provide examples from the use cases.

###### decompose features

趋势特征：

周期特征：

Fourierterm / 正余弦特征

###### static features

While lag features are mostly numeric in nature, static features usually become categorical variables in the models.

**First and foremost**, **convert to user local time**. For example, if you are trying to predict something that occurs around 8pm for all users, if you look at UTC time, it will be harder to predict from.

**Second**, generate static features

**time-stamp features:**

**absolute\_time (例如设定基准时间，将所有时间转换为与基准时间的时间差)--即回归模型之时间自变量t.**

day\_of\_weekdays-of-the-month day\_of\_year

week\_of\_month week\_of\_year

month\_of\_year

**planning\_of\_year (5年为周期的国家发展规划)**

hour\_of\_day (对于特殊时点，需要单独作为特征，例如股票交易中9:30和13:00通常为高峰)

period\_of\_day(morning/noon/afternoon/evening/night/midnight, 根据数据特征确定合适区间)

minute\_of\_hour

**注意：如果该定性特征具有顺序，则保留其自然数编码；否则进行哑变量编码。**

**holiday**

**dummy feature:**

is\_holiday\_this\_week

is\_holiday\_previous\_week

is\_holiday\_next\_week

is\_weekends

is\_labour day

is\_national day

is\_easter

is\_superbowl

is\_national\_emergency

etc.

These often require that you cross reference your data against some external source that maps events to time.

**count feature:**

consecutive holiday counts, un-consecutive holiday counts

**注意：从基准时间开始计数，可以避免验证集信息泄露。**

Holidays ( before , on and after ) are important and heuristics ( i.e. not simply done ! ) are required to identify many of these structures.

Look for trends in all of these (visualize the time series plot), before it or after it always have strong influence in the data.

For example:

There seems a **peak/bell-shape** of weekly sales located around the holiday. This is easy to understand.

Before holiday comes, more and more people are going shopping, and thus the weekly sales before/during holiday weeks increase.

And after that, weekly sales begin to fall.

To capture this, we can use:

**What is the last/next holiday? How many days are there from/to that?**

And furthermore, to reflect the peak/bell-shape, we may want to use:

1) abs(days) (er...this is days itselt) or exp(-abs(days)/diversity)

(recall a laplace distribution)

2) days^2 or exp(-days^2/2\*sigma^2) (recall a gaussian distribution)

Herein, we set diversity=7 (i.e., one week) and sigma=7 (i.e., two week)

However, the raw days seem more helpful rather than these complex ones. But we just leave these in the final features as they are, and let gbm/rf to figure out how important they are for the forecasting task.

(https://github.com/ChenglongChen/Kaggle\_Walmart-Recruiting-Store-Sales-Forecasting/blob/master/preprocess\_data\_v2.R)

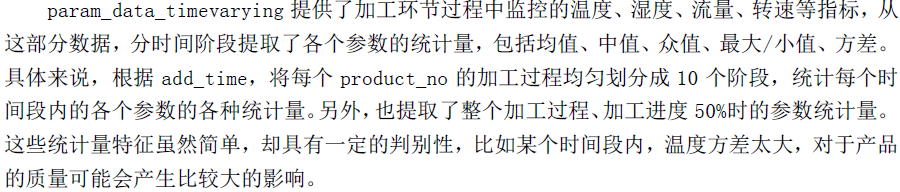
Furthermore there may be changes in daily patterns over time and different volatilities (uncertainties/variability) for different days of the week. To determine these factors requires searching for patterns not just fitting coefficients. Detecting level shifts and local time trends along with one-time unusual values is also critical beside correctly forming an appropriate structure ( i.e. ARIMA model identification) is also critical as one needs to craft together a number of competing model possibilities.

Finally changing error variance and/or changing model parameters over time need to be considered as they can come into play quite frequently.

**third**, convert dynamic features to static features, calculate the features for time interval

对于1小时的分钟数据，可计算每10分钟的均值特征，即得到6个特征，类似可计算中位数、方差等特征。

示例：



**fourth,** combine qualitative and quantitative features. Such as N1 and N2 are the quantitative features, C1 and C2 are qualitative features, then group by the qualitative features and calculate the quantitative features:

mean(N1)\_group\_by(C1)

median(N1)\_group\_by(C1)

max(N1)\_group\_by(C1)

min(N1)\_group\_by(C1)

var(N1)\_group\_by(C1)

sd(N1)\_group\_by(C1)

skewness(N1)\_group\_by(C1)

kurtosis(N1)\_group\_by(C1)

...

###### lag features

**基于decompose features和static features，进一步aggregate，构建用于训练的特征集。**

**注意：rolling aggregate构建特征要求数据量充足；当数据量不足时，可以选择aggregate后，重复利用aggregate值的方式构建特征。如根据2009年-2015年的月度数据预测2016年1-12月，则计算2009年-2015年该指标的平均值（12个平均值），训练集则2009年-2015年（7年）重复该指标7次(rep(12个平均值,7))，测试集则重复该指标1次。此外，将训练集平均值作为测试集的预测值，可用于比较不同模型的预测精度。**

There are many ways of creating features from the data that comes with timestamped data. In this section, we discuss some of these methods used for predictive maintenance. However, we are not limited by these methods alone.

last periodic

for example:

What is the weekly sales of last year?

These features are motivated as the times series of weekly sales appear highly periodic. For some depts, e.g., dept=1, you can simply predict the next year weekly sales using the corresponding weekly sales last year. (If you look at the ts plot of weekly sales, it sounds quite convincing to do so).

aggregates

It may prove most useful to aggregate all of the data itself rather than including any raw data. In other words, only include aggregated features for the time-varying features. **It is best to try different levels of aggregation and evaluate model performance in order to determine the optimal aggregation level.**

**Rolling aggregates/Sliding Window**

Rolling aggregates, also known as a **moving window aggregates** or **running aggregates**, aggregate statistics from observations in a window of observations that are before/after the current point. With this feature, you can compute:

* For each observation, an aggregate value (**counts, means, medians, sum, variance, standard deviations, outliers based on standard deviations, CUSUM measures, minimum and maximum,frequency domain, skewness and kurtosis featuresetc.**) of a fixed number of observations that occur before the current observation.

**注意：虽然means较常用，但medians对异常值更加稳健.**

* A similar aggregation operation on observations that occur after the current observation.
* A combination of the above two operations.
* **Other features such as change values within a window, change from the initial value, velocity of change, number over a defined threshold, autocorrelation coefficients, regression coefficients, coefficients of an AR model, the test statistic of the augmented dickey fuller hypothesis test, wavelets, etc. could be included as well.**

(rolling correlation/rolling regression)

For each record of an asset, we pick a rolling window of size "W" which is the number of units of time that we would like to compute historical aggregates for. We then compute rolling aggregate features using the W periods before the date of that record.

For demonstration, see Figure 1 where we represent sensor values recorded for an asset for each unit of time with the blue lines and mark the rolling average feature calculation for W=3 for the records at t1 and t2 which are indicated by orange and green groupings respectively.

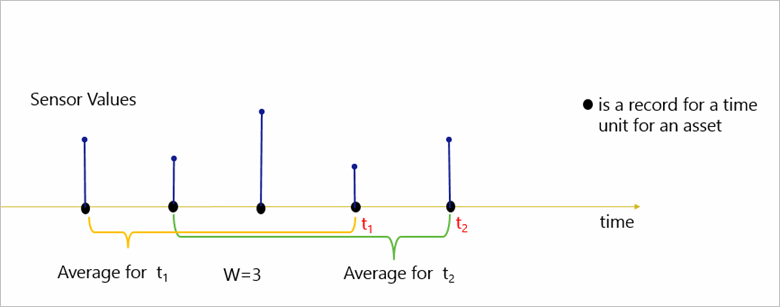
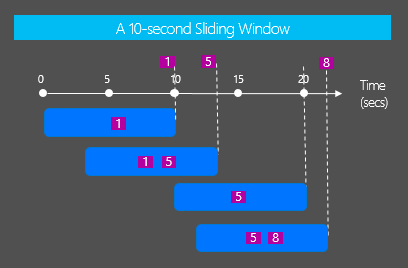
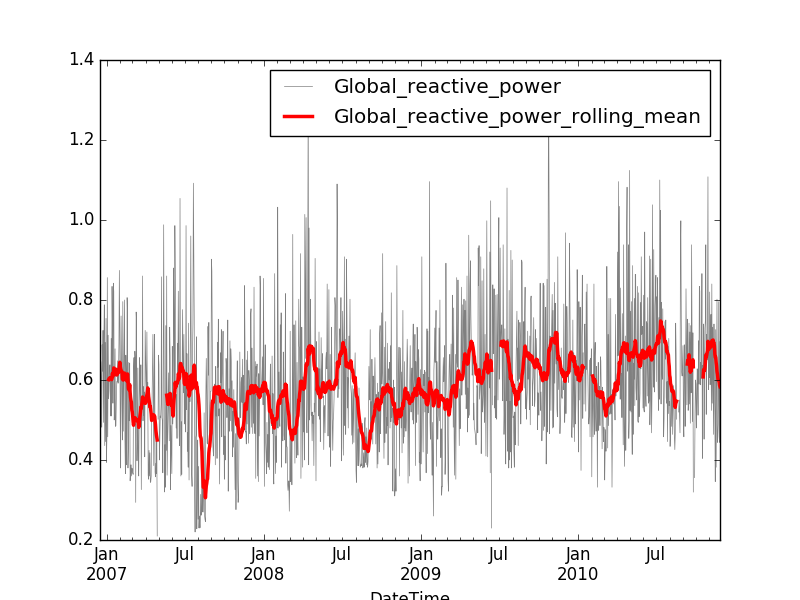


Figure 1. Rolling aggregate features

With a [Sliding Window](https://msdn.microsoft.com/en-us/library/azure/dn835051.aspx), the system is asked to logically consider all possible windows of a given length and output events for cases when the content of the window actually changes – that is, when an event entered or existed the window.



*Example: Generate an output event if the temperature is above 75 for a total of 5 seconds*



The result of the rolling mean is typically smoother than the original curve while still capturing some of the recent trends in the data.

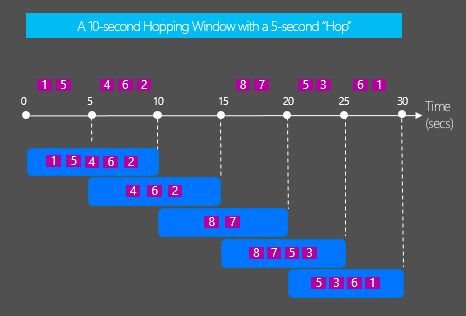
As **examples**, for aircraft component failure, sensor values from last week, last three days and last day were used to create rolling means, standard deviation and sum features. Similarly, for ATM failures, both raw sensor values and rolling means, median, range, standard deviations, number of outliers beyond three standard deviations, upper and lower CUMSUM features were used.

For flight delay prediction, counts of error codes from last week were used to create features. For train door failures, counts of the events on the last day, counts of events over the previous 2 weeks and variance of counts of events of the previous 15 days were used to create lag features. Same counting was used for maintenance-related events.

Additionally, by picking a W that is very large (ex. years), it is possible to look at the whole history of an asset such as counting all maintenance records, failures etc. up until the time of the record. This method was used for counting circuit breaker failures for the last three years. Also for train failures, all maintenance events were counted to create a feature to capture the long-term maintenance effects.

***Hopping Window***

Like Tumbling Windows, [Hopping Windows](https://msdn.microsoft.com/en-us/library/azure/dn835041.aspx) move forward in time by a fixed period but they can overlap with one another.



*Example: Every 5 seconds calculate the count of sensor readings and the average temperature over the last 10 seconds*

**Tumbling aggregates/Tumbling Window**

For each labeled record of an asset, we pick a window of size "W-k" where k is the number or windows of size "W" that we want to create lag features for. "k" can be picked as a large number to capture long-term degradation patterns or a small number to capture short-term effects. We then use k tumbling windows W-k , W-(k-1), …, W-2 , W-1 to create aggregate features for the periods before the record date and time (see Figure 2). These are also rolling windows at the record level for a time unit which is not captured in Figure 2 but the idea is the same as in Figure 1 where t2 is also used to demonstrate the rolling effect.

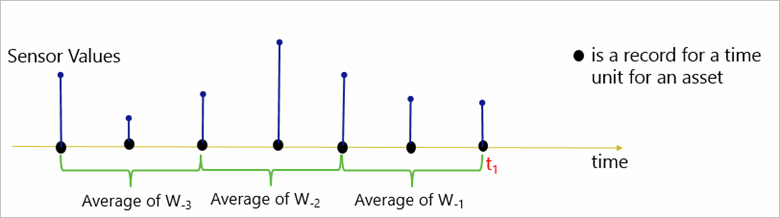
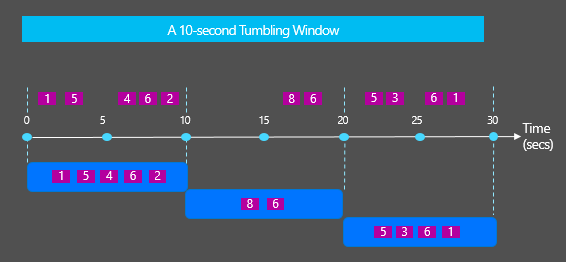


Figure 2. Tumbling aggregate features

[Tumbling Windows](https://msdn.microsoft.com/en-us/library/azure/dn835055.aspx) define a repeating, non-overlapping window of time.



*Example: Calculate the count of sensor readings per device every 10 seconds*

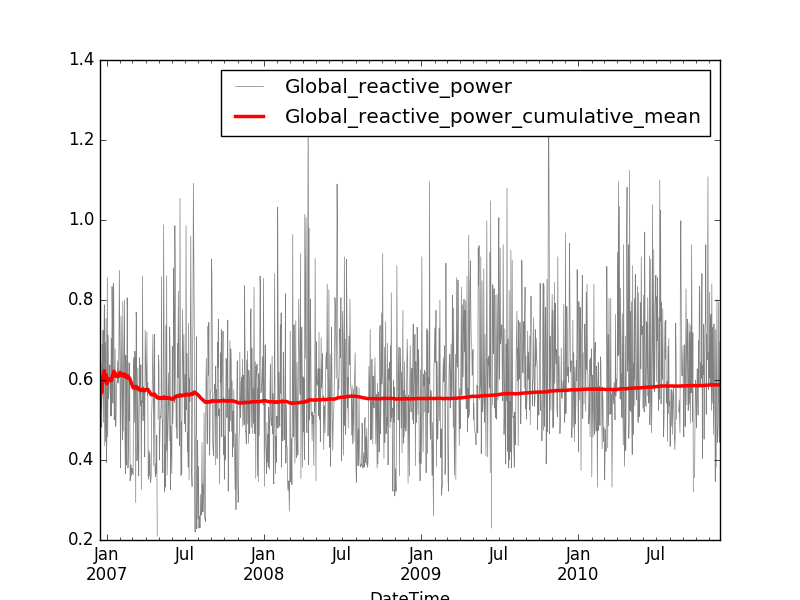
As an **example**, for wind turbines, W=1 and k=3 months were used to create lag features for each of the last 3 months using top and bottom outliers.

**Tumbling Window** and**Sliding Window**. The main difference between these windows is that Tumbling windows are non-overlapping whereas Sliding windows **can be**overlapping.

http://stackoverflow.com/questions/12602368/sliding-vs-tumbling-windows

**Cumulative aggregates**

In addition to rolling aggregates, we can also perform cumulative aggregates using all the observations before (and including) the current observation.



The result of the cumulative mean is typically much smoother than the original curve. Unlike the rolling means, the cumulative mean cannot capture recent trends in the data, but it can however spot global trends in the data.

Cumulative sum

Cumulative variance

Cumulative standard deviation

Cumulative counts

Cumulative min

Cumulative max

Capture trend/spikes/level changes

Another interesting technique is to capture trend changes, spikes and level changes using algorithms that detect anomalies in data using anomaly detection algorithms.

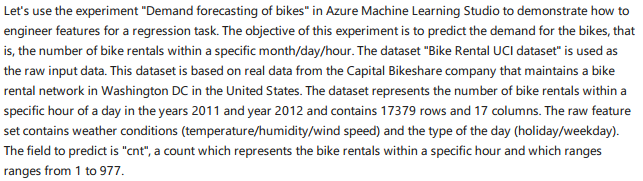
对于极端高或极端低的值，则用异常值检测算法，根据结果将相应异常点进行标记，即生成一个新的特征。

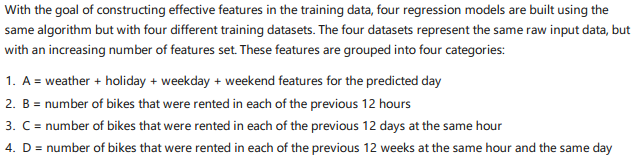
###### missing value\normalization and so on

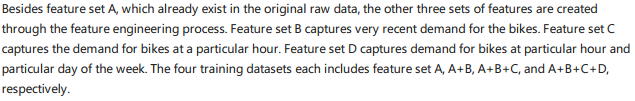
During feature generation, some other important steps such as handling missing values and normalization should be performed. There are numerous methods of **missing value imputation** and also **data normalization** which is not discussed here. However, **it is beneficial to try different methods to see if an increase in prediction performance is possible**.

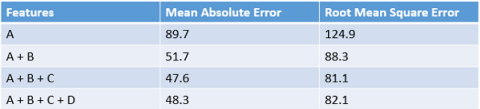
###### example

Adding Temporal Features for Regression Model









the comparison of the performance results of the four models

Prediction the total social electric consumption

The black points is the training data of electricity consumption from 1991 to 2015, prediction the following 5 years in the future.

I have generate three features:

the last 5 years electric consumption moving average

the last 5 years GDP moving average

the 5 cycles features (encoding to dummy variables)

中国的五年发展规划，可以看到每五年宏观指标(GDP)上升一个台阶.

1986-1990第七个五年规划

1991-1995第八个五年规划

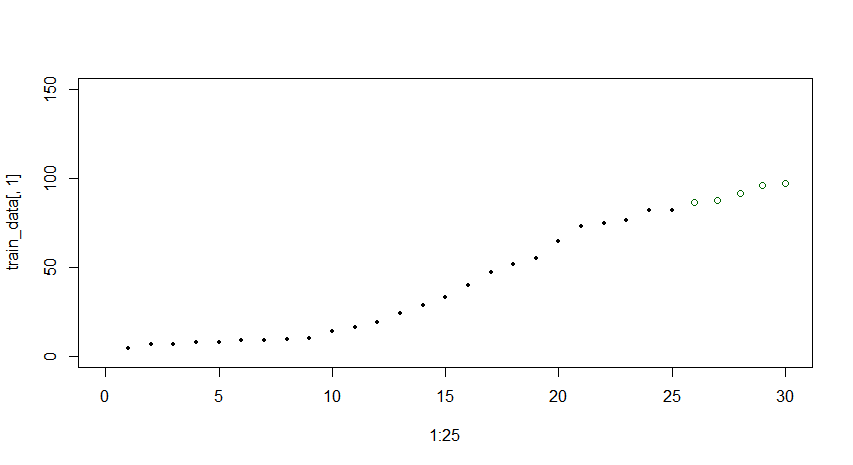
1996-2000第九个五年规划

2001-2005第十个五年规划

2006-2010第十一个五年规划

2011-2015第十二个五年规划

2016-2020第十三个五年规划



solid points is the electric consumption, green circles is the predictedvalue using cubist model

##### Modeling techniques

Predictive Maintenance is a very rich domain often employing business questions which may be approached from many different angles of the predictive modeling perspective.we provide main techniques that are used to model different business questions that can be answered with predictive maintenance solutions. Although there are similarities, **each model has its own way of constructing labels** which are described in detail.

###### Binary classification for predictive maintenance

Binary Classification for predictive maintenance is used to predict the probability that equipment fails within a future time period. The time period is determined by and based on business rules and the data at hand. Some common time periods are minimum lead time required to purchase spare parts to replace likely to damage components or time required to deploy maintenance resources to perform maintenance routines to fix the problem that is likely to occur within that time period. We call this future horizon period "X".

Label construction

In order to create a predictive model to answer the question "What is the probability that the asset fails in the next X units of time?", labeling is done by taking X records prior to the failure of an asset and labeling them as "**about to fail**" (label = 1) while labeling all other records as "**normal**" (label =0). In this method, labels are categorical variables (see Figure 3).

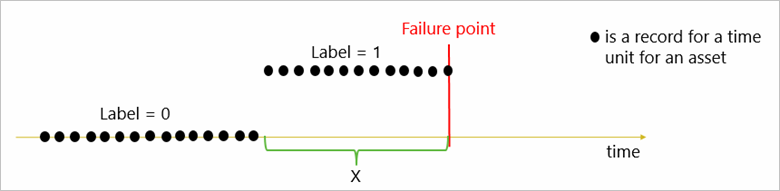


Figure 3. Labeling for binary classification

For flight delays and cancellations, X is picked as one day to predict delays in the next 24 hours. All flights that are within 24 hours before failures were labeled as 1s. For ATM cash dispense failures, two binary classification models were built to predict the failure probability of a transaction in the next 10 minutes and also to predict the probability of failure in the next 100 notes dispensed. All transactions that happened within the last 10 minutes of the failure are labeled as 1 for the first model. And all notes dispensed within the last 100 notes of a failure were labeled as 1 for the second model. For circuit breaker failures, the task is to predict the probability that the next circuit breaker command fails in which case X is chosen to be one future command. For train door failures, the binary classification model was built to predict failures within the next 7 days. For wind turbine failures, X was chosen as 3 months.

Wind turbine and train door cases are also used for regression analysis to predict remaining useful life using the same data but by utilizing a different labeling strategy which is explained in the next section.

###### Regression for predictive maintenance

Regression models in predictive maintenance are used to compute the remaining useful life (RUL) of an asset which is defined as the amount of time that the asset is operational before the next failure occurs. Same as binary classification, each example is a record that belongs to a time unit "Y" for an asset. However, in the context of regression, the goal is to find a model that calculates the remaining useful life of each new example as a continuous number which is the period of time remaining before the failure. We call this time period some multiple of Y.

Label construction

Given the question "What is the remaining useful life of the equipment? ", labels for the regression model can be constructed by taking each record prior to the failure and labeling them by calculating how many units of time remain before the next failure. In this method, labels are continuous variables (See Figure 4).

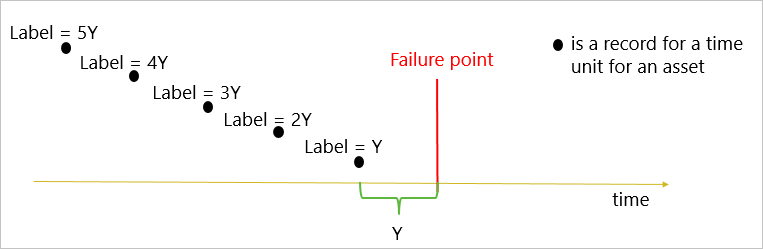


Figure 4. Labeling for regression

Different than binary classification, for regression, assets without any failures in the data cannot be used for modeling as labeling is done in reference to a failure point and its calculation is not possible without knowing how long the asset survived before failure. This issue is best addressed by another statistical technique called Survival Analysis.

###### Multi-class classification for predictive maintenance

Multi-class classification for predictive maintenance can be used to predict two future outcomes. The first one is to assign an asset to one of the multiple possible periods of time to give a range of time to failure for each asset. The second one is to identify the likelihood of failure in a future period due to one of the multiple root causes. **That allows maintenance personnel who are equipped with this knowledge to handle the problems in advance.** Another multi-class modeling technique focuses on determining the most likely root cause of a given a failure. **This allows recommendations to be given for the top maintenance actions to be taken in order to fix a failure.** By having a ranked list of root causes and associated repair actions,technicians can be more effective in taking their first repair actions after failures.

Label construction

Given the two questions which are "What is the probability that an asset fails in the next "aZ" units of time where "a" is the number of periods" and "What is the probability that the asset fails in the next X units of time due to problem "Pi" where "i" is the number of possible root causes, labeling is done in the following way for these to techniques.

For the first question, labeling is done by taking aZ records prior to the failure of an asset and labeling them using buckets of time (3Z, 2Z, Z) as their labels while labeling all other records as "normal" (label =0). In this method, label is categorical variable (See Figure 5).

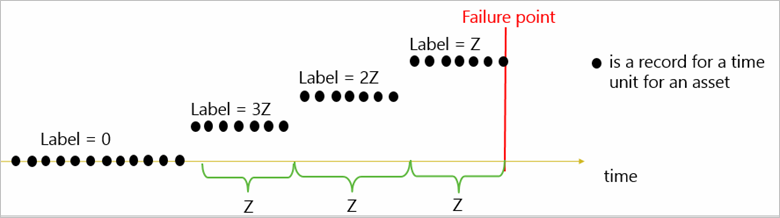


Figure 5. Labeling for multiclass classification for failure time prediction

For the second question, labeling is done by taking X records prior to the failure of an asset and labeling them as "about to fail due to problem Pi" (label = Pi) while labeling all other records as "normal" (label =0). In this method, labels are categorical variables (See Figure 6).

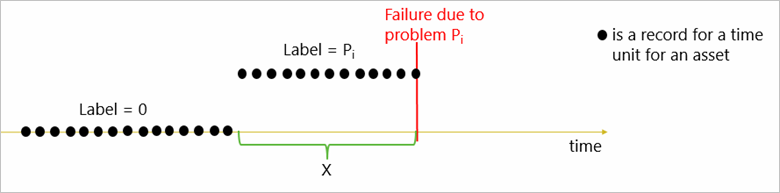


Figure 6. Labeling for multiclass classification for root cause prediction

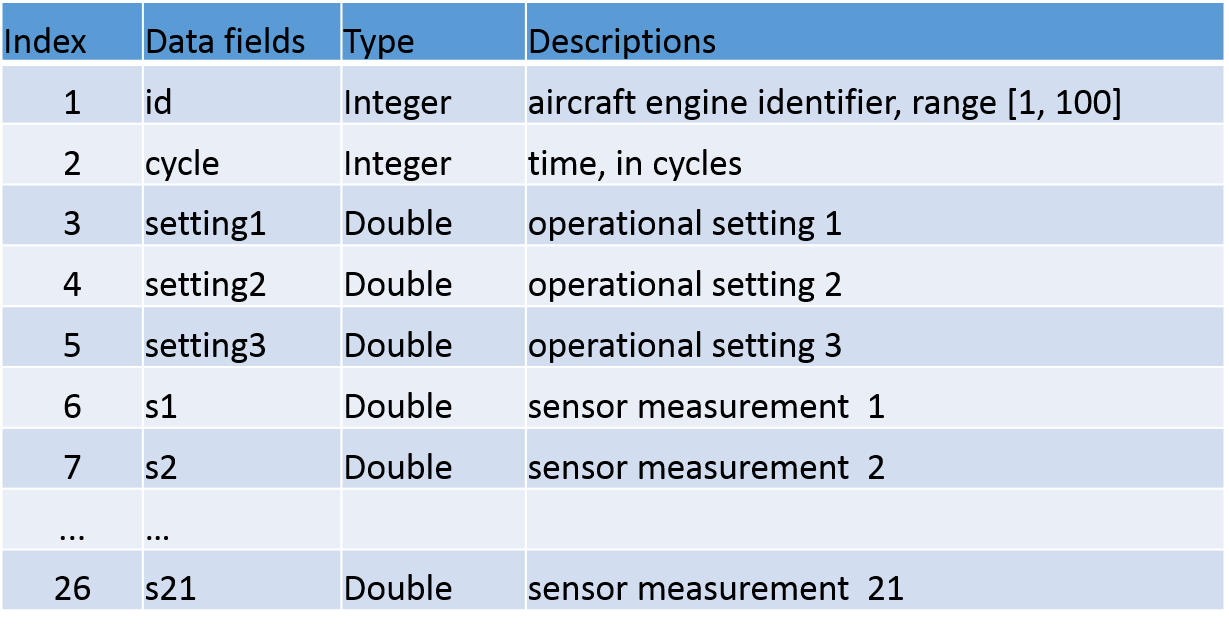
The model assigns a failure probability due to each Pi as well as the probability of no failure. These probabilities can be ordered by magnitude to allow prediction of the problems that are most likely to occur in the future.

Aircraft component failure use case was structured as a multiclass classification problem. This enables the prediction of the probabilities of failure due to two different pressure valve components occurring within the next month.

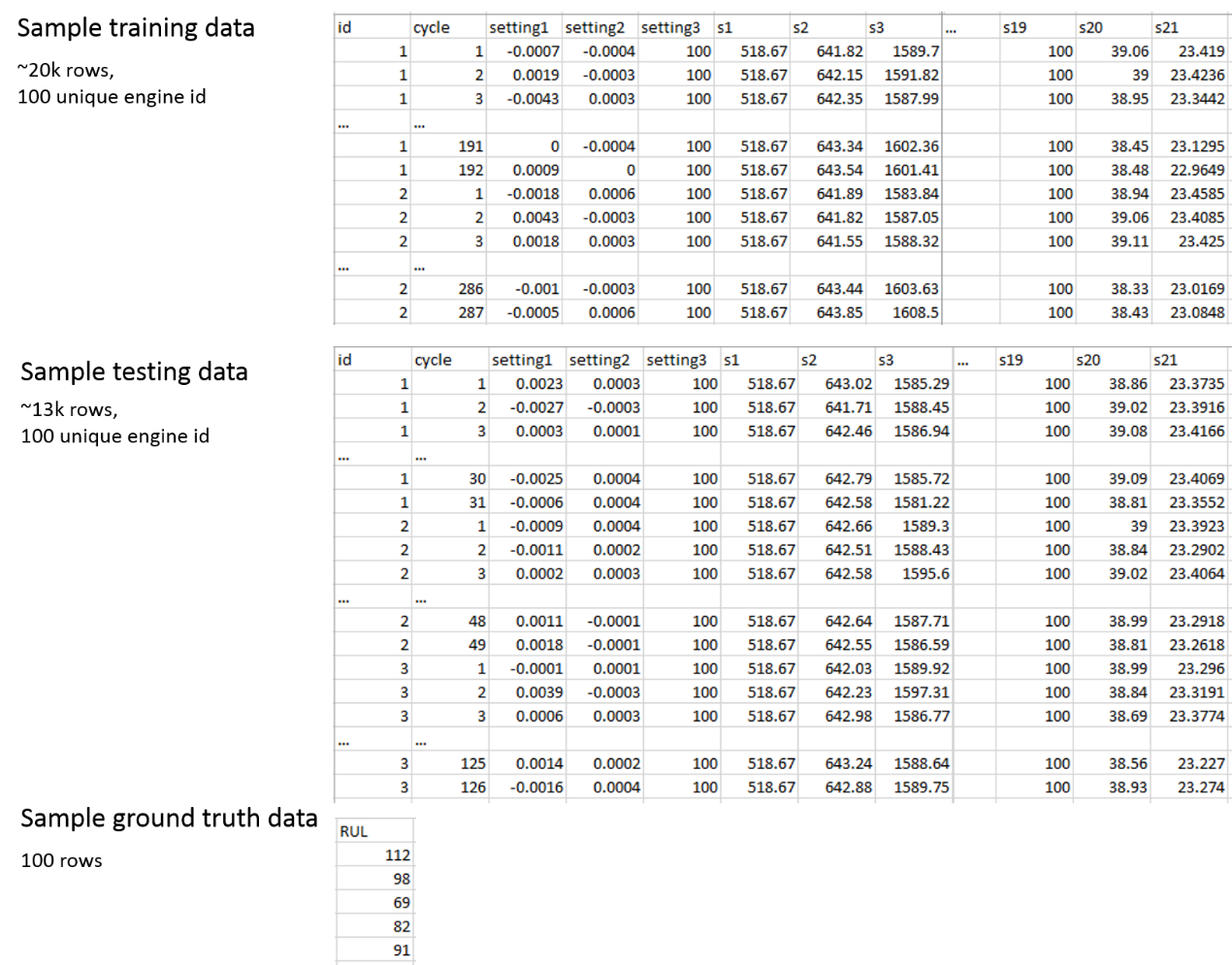
For recommending maintenance actions after failures, labeling does not require a future horizon to be picked. **This is because the model is not predicting failure in the future but it is just predicting the most likely root cause once the failure has already happened.** Elevator door failures fall into the third case where the goal is to predict the cause of the failure given historical data on operating conditions. This model is then used to predict the most likely root causes after a failure has occurred. One key benefit of this model is that it helps inexperienced technicians to easily diagnose and fix problems that would otherwise need years’ worth of experience.

###### Example

The data schema for the training and testing data is shown in the following table.

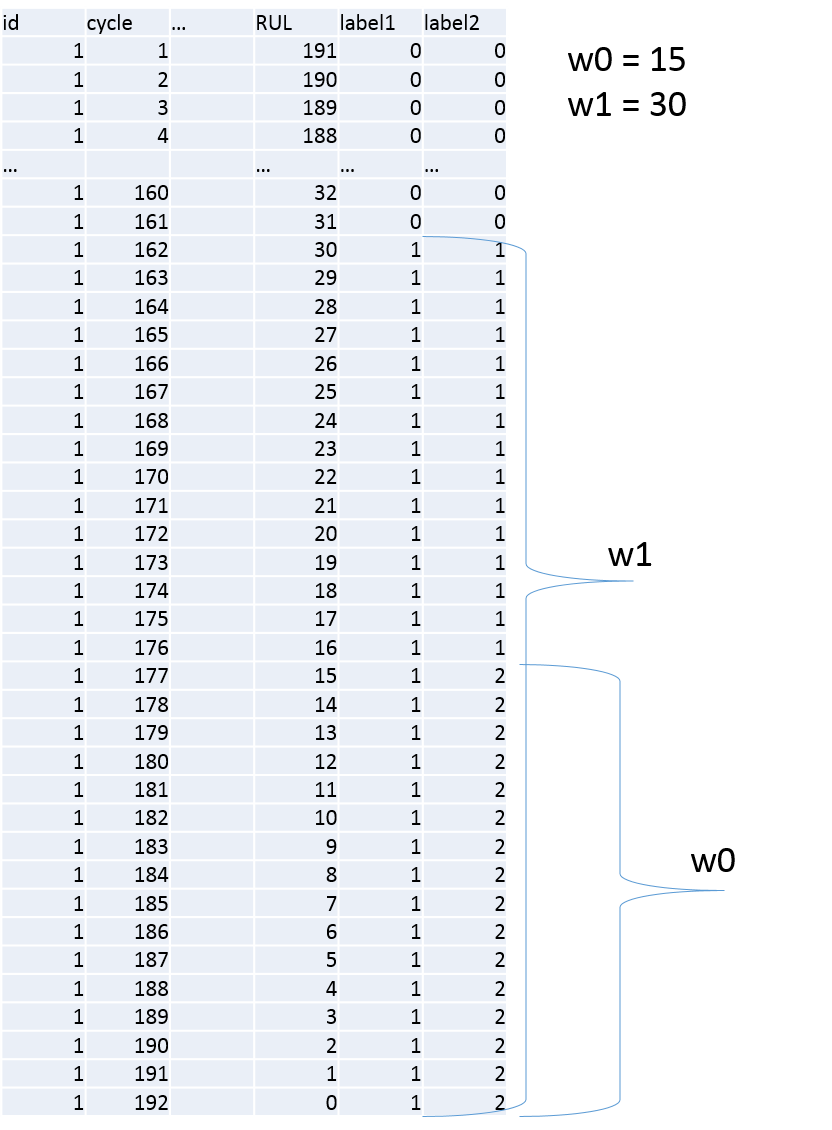


The training data consists of multiple multivariate time series with "cycle" as the time unit, together with 21 sensor readings for each cycle. Each time series can be assumed as being generated from a different engine of the same type. Each engine is assumed to start with different degrees of initial wear and manufacturing variation, and this information is unknown to the user. In this simulated data, the engine is assumed to be operating normally at the start of each time series. It starts to degrade at some point during the series of the operating cycles. The degradation progresses and grows in magnitude. When a predefined threshold is reached, then the engine is considered unsafe for further operation. **In other words, the last cycle in each time series can be considered as the failure point of the corresponding engine.** Taking the sample training data shown in the following table as an example, the engine with id=1 fails at cycle 192, and engine with id=2 fails at cycle 287.



* Regression models: How many more cycles an in-service engine will last before it fails?
* Binary classification: Is this engine going to fail within *w1* cycles?
* Multi-class classification: Is this engine going to fail within the window [1, *w0*] cycles or to fail within the window [*w0*+1, *w1*] cycles, or it will not fail within *w1* cycles?

Taking the example of engine with id=1, the following figure shows how the training data is labeled, where "RUL", label1", and "label2" are labels for regression, binary classification, and multi-class classification models respectively. Here w0 and w1 are predefined use case related parameters which are used to label the training data. The customer needs to decide how far ahead of time the alert of failure should trigger before the actual failure event.



reference:

<https://gallery.cortanaintelligence.com/Experiment/Predictive-Maintenance-Step-1-of-3-data-preparation-and-feature-engineering-2>

##### Training, validation and testing methods in predictive maintenance

In predictive maintenance, similar to any other solution space containing timestamped data, the typical training and testing routine needs to take account the time varying aspects to better generalize on unseen future data.

###### Cross validation

In predictive maintenance problems, data is recorded as a time series of events that come from several data sources. These records can be ordered according to the time of labeling a record or an example. Hence, if we split the dataset **randomly** into training and validation set, some of the training examples are later in time than some of validation examples. This results in estimating future performance of hyperparameter values based on the data that arrived before model was trained. These estimations might be overly optimistic, especially if time-series are **not stationary** and **change their behavior over time**. As a result, chosen hyperparameter values might be sub-optimal.

A better way of finding good values of hyperparameters is to split the examples into training and validation set **ina time-dependent way**, such that all validation examples are later in time than all training examples. Then, for each set of values of hyperparameters we train the algorithm over training set, measure model’s performance over the same validation set and choose hyperparameter values that show the best performance. When time-series data is not stationary and evolves over time, the hyperparameter values chosen by train/validation split lead to a better future "model's performance than with the values chosen randomly by cross-validation.

**The final model is generated by training a learning algorithm over entire data using the best hyperparameter values that are found by using training/validation split or cross-validation.**

###### Testing for model performance

One way is to split the data randomly into training, validation and test sets.

Another way which is relevant to predictive maintenance, is to split the examples into training, validation and test sets in a time-dependent way, such that all test examples are later in time than all training and validation examples. The training and validation sets are used to select values of hyperparameters and train the model with them. The performance of the model is measured over the test set.

When time-series are stationary and easy to predict both approaches generate similar estimations of future performance. But when time-series are non-stationary and/or hard to predict, the second approach will generate more realistic estimates of future performance than the first one.

###### Time-dependent split

For example, for binary classification, as described in Feature Engineering and Modeling Techniques sections, features are created based on the past events and labels are created based on future events within "X" units of time in the future. Thus, the labeling timeframe of an example comes later then the timeframe of its features. For **time-dependent split**, we pick a point in time at which we train a model with tuned hyperparameters by using historical data up to that point. **To prevent leakage of future labels** that are beyond the training cut-off into training data, we choose the latest timeframe to label training examples to be X units before the training cut-off date. In Figure 7, each solid circle represents a row in the final feature data set for which the features and labels are computed according to the method described above. Given that, the figure shows the records that should go into training and testing sets when implementing time-dependent split for X=2 and W=3:

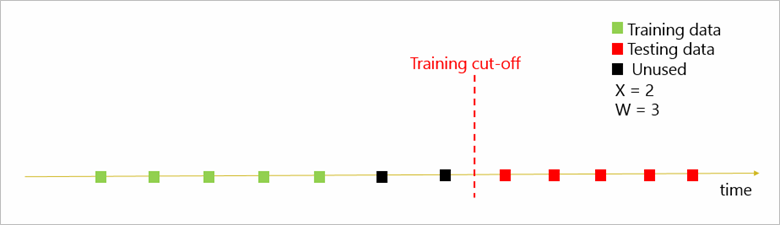


Figure 7. Time-dependent split for binary classification

The green squares represent the records belonging to the time units that can be used for training. As explained earlier, each training example in the final feature table is generated by looking at past 3 periods for feature generation and 2 future periods for labeling before the training day cut-off. We do not use examples in the training set when any part of the 2 future periods for that example is beyond the training cut-off since we assume that we do not have visibility beyond the training cut-off. Due to that constraint, black examples represent the records of the final labeled dataset that should not be used in the training data set. These records won’t be used in testing data either since they are before the training cut-off and their labeling timeframes partially depend on the training timeframe which should not be the case as we would like to completely separate labeling timeframes for training and testing to prevent label information leakage.

This technique allows for overlap in the data used for feature generation between training and testing examples that are close to the training cut-off. Depending on data availability, an even more severe separation can be accomplished by not using any of the examples in the test set that are within W time units of the training cut-off.

From our work, we found that regression models used for predicting remaining useful life are more severely affected by the leakage problem and using a random split leads to extreme overfitting. Similarly, in regression problems, the split should be such that records belonging to assets with failures before training cut off should be used for the training set and assets that have failures after the cut-off should be used for testing set.

As a general method, another important best practice for splitting data for training and testing is to use a split by asset ID so that none of the assets that were used in training are used for testing since the idea of testing is to make sure that when a new asset is used to make predictions on, the model provides realistic results.

##### Handling imbalanced data

If one class is less than 10% of the data, we can say that the data is imbalanced and we call the underrepresented dataset minority class. Drastically, in many cases we find imbalanced datasets where one class is severely underrepresented compared to others for example by only constituting 0.001% of the data points.

Class imbalance is a problem in many domains including fraud detection, network intrusion and predictive maintenance where failures are usually rare occurrences in the lifetime of the assets which make up the minority class examples.

Conventional evaluation metrics such as overall accuracy on error rate, are not sufficient in case of imbalanced learning. Other metrics, such as precision, recall, F1 scores and cost adjusted ROC curves are used for evaluations in case of imbalanced datasets which is discussed in the Evaluation Metrics section.

However, there are some methods that help remedy class imbalance problem. The two major ones are sampling techniques and cost sensitive learning.

###### Sampling methods

Although there are a lot of different sampling techniques, most straight forward ones are random oversampling and under sampling.

Random oversampling:

One danger of oversampling is that multiple instances of certain examples can cause the classifier to become too specific leading to overfitting. This would result in high training accuracy but performance on the unseen testing data may be very poor.

Random under sampling:

Removing examples from majority class may cause the classifier to miss important concepts pertaining to the majority class.

Hybrid sampling where minority class is oversampled and majority class is under sampled at the same time is another viable approach.

There are many other more sophisticated sampling techniques are available and effective sampling methods for class imbalance is a popular research area receiving constant attention and contributions from many channels. **Use of different techniques to decide on the most effective ones is usually left to the data scientist to research and experiment and are highly dependent on the data properties**. Additionally, it is important to make sure that **sampling methods are applied only to the training set but not the test set**.

###### Cost sensitive learning

In predictive maintenance, failures which constitute the minority class are of more interest than normal examples and thus the focus is on the performance of the algorithm on failures is usually the focus. This is commonly referred as **unequal loss** or **asymmetric costs** of misclassifying elements of different classes where incorrectly predicting a positive as negative can cost more than vice versa. The desired classifier should be able to give high prediction accuracy over the minority class, without severely compromising on the accuracy for the majority class.

There are several ways this can be achieved. **By assigning a high cost to misclassification of the minority class**, and trying to minimize the overall cost, the problem of unequal loses can be effectively dealt with. Some machine learning algorithms use this idea inherently such as SVMs (Support Vector Machines) where cost of positive and negative examples can be incorporated during training time. Similarly, boosting methods are used and usually show good performance in case of imbalanced data such as boosted decision tree algorithms.

##### Evaluation metrics

Accuracy is the most popular metric used for describing a classifier’s performance. However as explained above accuracy is ineffective and do not reflect the real performance of a classifier’s functionality as it is very sensitive to ***data distributions***. Instead, other evaluation metrics are used to assess imbalanced learning problems. In those cases, precision, recall and F1 scores should be the initial metrics to look at when evaluating predictive maintenance model performance.

In predictive maintenance, recall rates denote how many of the failures in the test set were correctly identified by the model. Higher recall rates mean the model is successful in catching the true failures. Precision metric relates to the rate of false alarms where lower precision rates correspond to higher false alarms. F1 score considers both precision and recall rates with best value being 1 and worst being 0.

Moreover, for binary classification, **decile tables** and **lift charts** are very informative when evaluating performance. They focus only on the positive class (failures) and provide a more complex picture of the algorithm performance than what is seen by looking at just a fixed operating point on the ROC (Receiver Operating Characteristic) curve. **Decile tables** are obtained by ordering the test examples according to their predicted probabilities of failures computed by the model before thresholding to decide on the final label. The ordered samples are then grouped in deciles (i.e. the 10% samples with largest probability and then 20%, 30% and so on). By computing the ratio between true positive rate of each decile and its random baseline (i.e. 0.1, 0.2 ..) one can estimate how the algorithm performance changes at each decile. **Lift charts** are used to plot decile values by plotting decile true positive rate versus random true positive rate pairs for all deciles. Usually, the first deciles are the focus of the results since here we see the largest gains. First deciles can also be seen as representative for "at risk" when used for predictive maintenance.

##### Sample solution architecture

When deploying a predictive maintenance solution, we are interested in an end to end solution that provides a continuous cycle of data ingestion, data storage for model training, feature generation, prediction and visualization of the results along with an alert generating mechanism such as an asset monitoring dashboard. We want a data pipeline that provides future insights to the user in a continuous automated manner. An example predictive maintenance architecture for such an IoT data pipeline is illustrated in Figure 8 below. In the architecture, real-time telemetry is collected into an Event Hub which stores streaming data. This data is ingested by stream analytics for real-time processing of data such as feature generation. The features are then used to call the predictive model web service and results are displayed on the dashboard. At the same time, ingested data is also stored in an historical database and merged with external data sources such as on-premise data bases to create training examples for modeling. Same data warehouses can be used for batch scoring of the examples and storing of the results which can again be used to provide predictive reports on the dashboard.

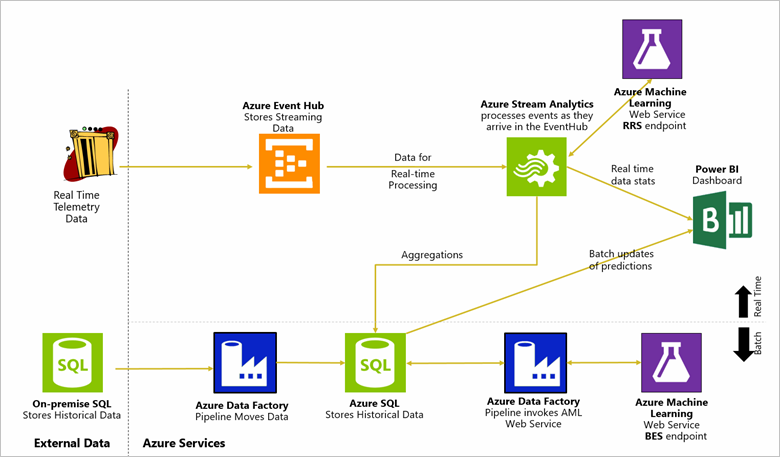


Figure 8. Example solution architecture for predictive maintenance

reference:

<https://docs.microsoft.com/en-us/azure/machine-learning/cortana-analytics-playbook-predictive-maintenance>

<https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-data-science-create-features> (Adding Temporal Features for Regression Model)

<https://opbuildstorageprod.blob.core.windows.net/output-pdf-files/en-us/Azure.azure-documents/live/machine-learning.pdf>

https://turi.com/learn/userguide/timeseries/timeseries-data.html

https://blogs.technet.microsoft.com/machinelearning/2015/06/01/the-azure-stream-analytics-query-language/

https://msdn.microsoft.com/en-us/library/azure/dn835041.aspx