Predictive Maintenance with Conformal and Probabilistic Prediction: A Commercial Case Study

James Gammerman

Submitted for the Degree of Master of Science in

Machine Learning

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

September 13, 2018

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 11,020

Student Name: James Gammerman

Date of Submission: 12/09/2018

Signature: JAMES GAMMERMAN

Abstract

Industrial companies usually own assets which are prone to breaking down. Each year vast quantities of money are spent on maintenance to fix these assets after they have failed. This study presents a set of predictive maintenance techniques which could help combat this problem.

A large set of sensor data taken from a subsystem of a commercial gas terminal in the UK was analysed. We first used a recently developed machine learning technique called conformal clustering to show that there are patterns in this data, indicating predictability, and also to identify anomalies in the data. We then proposed a way to identify which metrics may explain why the subsystem has failed.

Finally we built a model to predict failure of the subsystem in probabilistic form, and ran experiments to optimise it.

We believe that the methodology developed in this project could be generalised to many such cases when large data sets from industrial sensors are available.

Contents

Acknowledgements

I would like to thank Dr Ilia Nouretdinov, Prof Zhiyuan Luo, Prof Alex Gammerman and Giovanni Cherubin from Royal Holloway for useful conversations and advice throughout this project. The same goes for my colleague Timothy Wong who introduced me to this project and gave me background information on it.

I would also like to thank my employer Centrica - firstly for allowing me to use one of their data sets, and secondly for letting me spend some company time on writing up this report.

1 Introduction

1.1 Predictive Maintenance

One of the main trends of the early 21st century has been the rapid shift of services and infrastructure from being physical in nature to digital. Indeed many technology companies now operate a business model which relies on barely any physical infrastructure. However, the majority of businesses still own physical assets, particularly industrial players. These often break down, at vast financial cost to the company.

To combat this, many businesses now perform *preventive maintenance*, which involves routine replacement of worn components based on their life expectancy. However, this is a suboptimal approach as it means that i) many replacements are made which actually don't need to be for a long time and ii) many failures still happen because some pieces of equipment break before their life expectancy is up.

Predictive maintenance (PM) is a set of techniques to improve this situation, which involve predicting when a piece of equipment will fail by monitoring its condition over time. This kind of knowledge makes it easier to plan ahead (e.g. engineer schedules) and to turn unplanned outages into planned ones, which increases productivity of the asset. Other advantages include increased lifetime of equipment, better safety records and optimisation of spare part management.

PM is a fast-growing field. Its popularity is understandable given the savings it has the potential to make - according to one study, PM will help companies save $630bn by 2025 [1].

This sets the scene for the project. Centrica plc is multinational energy company based in the UK (as well as the employer of the author of this report) with many physical assets to its name. Every year large sums of money are spent by the company on maintenance - most notably at the enormous Rough gas storage facility in the east of England, which it recently announced would be closed due to safety concerns. But until recently the company had not made any attempt to predict breakage of any of its assets ahead of time by monitoring equipment condition.

This report describes one of the first such projects undertaken by Centrica, investigating the potential for PM at one of its largest assets in the UK.

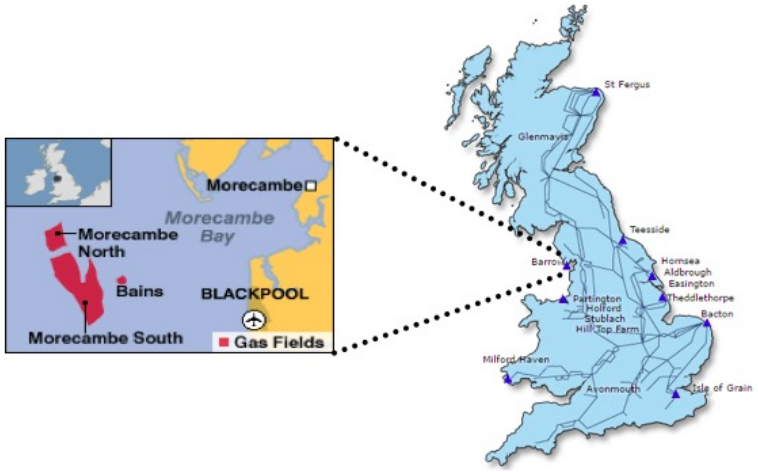

Figure 1: Location of Morecambe Bay Gas terminal

1.2 Background to the problem

The asset in question is a two-stage gas compressor train at a natural gas terminal belonging to Centrica at Morecambe Bay in the northwest of England - see Fig. 1.

The terminal receives unprocessed gas from an offshore platform via a pipeline. But the gas is not at a pressure appropriate for consumption in the UK gas grid (the National Transmission System) - it needs to be increased. For this purpose a gas compressor subsystem is used, with a simplified process diagram shown in Fig. 2, taken from Wong *et al* [2].

In brief, the process is as follows. When the gas arrives from offshore, it first passes through a low pressure (LP) suction scrubber which removes condensates such as propane, butane and impurities like hydrogen sulfide and carbon dioxide. It then enters a LP compressor which raises the pressure of the gas to an intermediate level, before being passed to an intercooler which reduces its temperature. The gas then goes through a high pressure (HP) stage consisting of another scrubber which further strips out impurities, followed by an HP compressor which raises its pressure to the level required for use in the NTS. The gas is then passed to other parts of the terminal downstream of the compressor train.

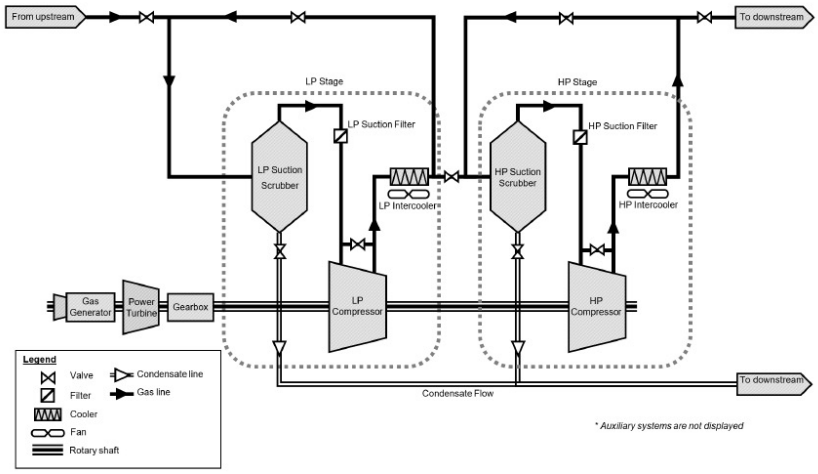

Figure 2: Process diagram of gas compressor in Morecambe Bay terminal

The power required to drive both the LP and HP stages comes from a gas generator and power turbine, also shown in the figure.

Sensors are located throughout the subsystem and measure various physical features of the system, such as temperature, pressure and rotary speed. It is known from discussions with the Morecambe Bay engineers that the terminal often breaks (once every few days), and that analysis of these sensor readings could provide insight on what caused a given failure and whether another one may occur in a given time frame.

Currently the engineers work on an ad hoc basis to fix problems in the terminal, after which they write short reports which explain what caused the failure and what they did to resolve it. This data is too messy and disorganised to be connected to the sensor data, which is currently not analysed at all. No preventive maintenance is performed.

And so we come to the problem that this project seeks to address. We would like to know if we can use machine learning on the sensor data to perform:

1) Failure explanation - i.e. shed some light on the causes of failure of the compressor in the gas terminal.

2) Predictive maintenance - i.e. predict future failures of this asset in a given time scale (which we will refer to as a time window), in order that maintenance could in principle be performed pro-actively rather than waiting for

breakdown to occur.

This report covers both of these topics in turn. We begin, however, with a summary of how the data was pre-processed.

2 Data Pre-Processing

We start with a short description of the data set before explaining how it was processed to facilitate analysis.

2.1 Data description

The data was obtained in tabular format as a batch extract from a computer system called Alerts. It covers the period March 2011 - March 2017, where each row gives a snapshot of the system at a moment in time, and is separated from the next snapshot by 10 minutes. There are 340,105 rows and 185 columns. A sample of the data can be found in Fig. 3.

	runclockactive	pressambient	lp_pressdisch	hp_pressdisch	gg_tempairinlet	gg_powershaft
1	1	0.9859012	18.03429	70.69764	19.67878	16.87461
2	1	0.9854003	18.90110	71.19204	21.85107	16.88424
3	1	0.9857899	18.79194	71.99892	21.12664	16.88424
4	1	0.9849552	18.41312	72.52115	21.02314	16.88424
5	1	0.9857064	19.38908	72.16586	20.81616	16.88424

Figure 3: First few rows and columns from Alerts data set

We can summarise the columns of the data as follows:

- A *runclockactive* column - a binary code where `1` means that the compressor was running at that moment, and `0` means it wasn't.
- 183 other columns containing measurements from sensors located in the compressor. For example, *pressambient* gives us the ambient pressure (measured in atmospheres), *lp_pressdisch* tells us about the discharge pressure in the LP stage of the compressor (just before the gas enters the HP stage), and *gg_tempairinlet* tells us about the temperature (in degrees) of the air inlet of the generator. Another common prefix for these metrics is *pt* which stands for power turbine.

Rather than having the *runclockactive* column, it would be easier to make predictions if we knew, at a given snapshot, how much time is left until the next failure. We therefore convert it into a *countdown* column which shows time until breakage measured in units of snapshots. Throughout this report we treat this value as a kind of label for each observation.

2.2 Observation Selection

In its original order, the data consists of a sequence of periods where the compressor was working, then not working, then working, then not working, and so on. Due to the lengthy and frequent non-working periods (there are 599 of them), by far the most common *countdown* value is zero, as we can see in figure 4.

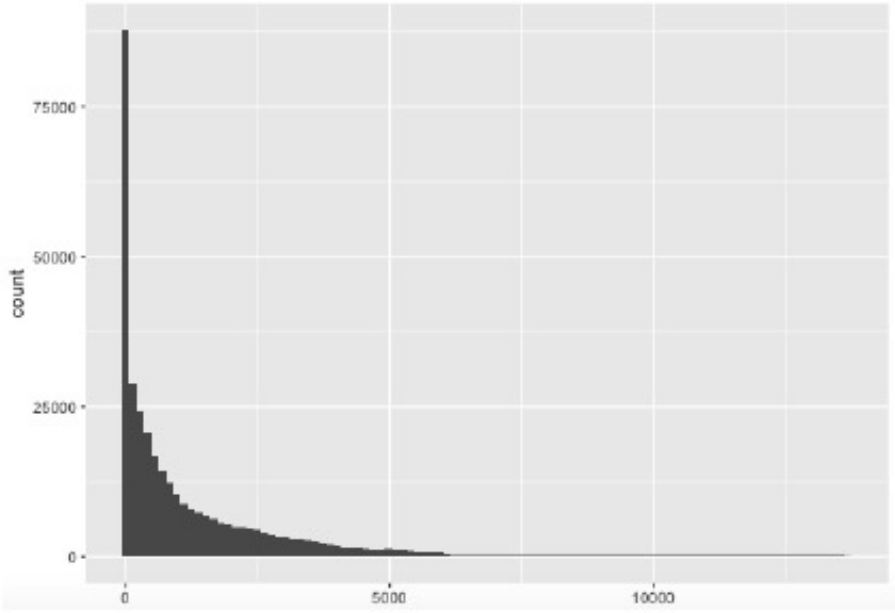

Figure 4: Histogram of countdown values in original data set

We consider these non-working periods to be of little value in telling us what caused the most recent failure. Hence we delete all rows in the table pertaining to these periods, reducing our dataset from ∼340,000 rows to ∼275,000.

This is still a sizeable number. At this point we choose to take a sample in order to i) improve computational efficiency and more importantly ii) get a dataset that is close to being independent and identically distributed (i.i.d), for reasons that will become clear later in this report. The sample is taken by drawing one random observation from each of the 599 total working periods.

2.3 Feature Selection

A high number of features not only makes training slow, but can also make it much harder to find a good solution. This problem is often referred to as the *curse of dimensionality*. We currently have 183 features and would therefore like to reduce this number.

2.3.1 Missing values

There are 65 columns with missing values, nearly all of which contain many of them. The two most common ways to deal with this problem are imputation and deletion of data [3]. We can kill two birds with one stone by removing all columns containing at least one missing value. This deals with the missing values problem and reduces dimensionality. We are left with 118 features.

2.3.2 Wilcoxon rank sum test

However, this is still too many. We need to perform further feature selection, and here we chose to use a popular statistical test called the *Wilcoxon rank sum test* (also known as the *Wilcoxon-Mann-Whitney* or *Mann-Whitney U test*) . This is a non-parametric approach used to test the null hypothesis that two independent samples drawn from the same population have the same distribution (but not necessarily the normal distribution, as in the student's t-test). It is useful for us because we can use it to test whether two samples of the same feature which belong to two different classes are drawn from the same i.i.d distribution, and hence whether that feature is informative for explaining the difference between the two classes.

A good explanation of the test is given by both [4] and [5]. It involves the calculation of a statistic called U, which has a known distribution under the null hypothesis. Let's say that we have samples of observations from each of two populations $A = \{a_1, a_2, ..., a_{n_A}\}$ and $B = \{b_1, b_2, ..., b_{n_B}\}$ containing n_A and n_B observations respectively. We wish to test the null hypothesis that the distribution of sample A is the same as that of sample B:

$$H_0 : A = B$$

The alternative hypothesis is that there is a so-called *location shift* between the two distributions:

$$H_1 : A \neq B, \text{ assuming a two-sided test}$$

See Fig. 5 for an illustration of this.

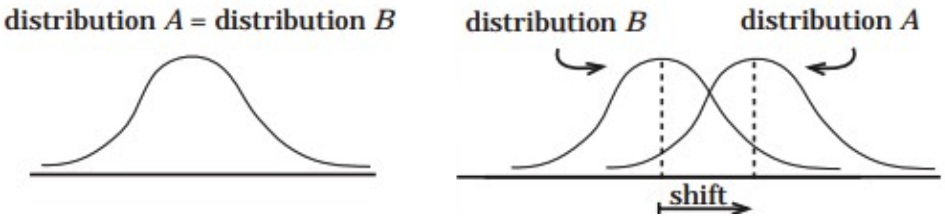

Figure 5: LHS: Null hypothesis - same distributions. RHS: Alternative hypothesis - distributions differ by a location shift. Taken from [5].

To perform this test, we rank each of the $n_A + n_B$ observations in order. For example, the smallest has rank 1, the second smallest rank 2, etc. In case of ties we assign each observation its average rank. For each observation a_i we sum up the number of bs that are smaller than it, and vice versa for each b.

We then sum up the total number of observations $b \in B$ that all the observations $a \in A$ are greater than, denoting this U_A, and vice versa, denoting this U_B.

We check that $U_A + U_B = n_A n_B$. Our $U-$statistic is then $min(U_A, U_B)$. Statistical tables can then be consulted to find the probability of observing a value of U or lower. For a two-sided test, we double this probability to obtain the p-value.

We can illustrate this with a short example taken from [5]. Say we have the following set of data showing age at diagnosis of type II diabetes for a set of young people. We want to test the null hypothesis that the age at diagnosis is the same for males and females.

Males: 19, 22, 16, 29, 24
Females: 20, 11, 17, 12

1. Arrange all observations in order of magnitude:

Age	11	12	16	17	19	20	22	24	29
M/F	F	F	M	F	M	F	M	M	M
M/F			2		3		4	4	4
F>M	0	0		1		2			

2. Assign each observation as being M or F.
3. Under each M write the number of Fs which are ranked smaller than it; under each F write the number of Ms which are ranked smaller than it (see above).
4. $U_M = 2 + 3 + 4 + 4 + 4 = 17$
   $U_F = 0 + 0 + 1 + 2 = 3$

   Check that $U_M + U_F = n_M n_F : 20 = 20$
5. $U = min(U_M, U_F) = 3$
6. Using tables for the Wilcoxon rank sum test, we find that the two-sided p-value is $p = 0.11$.

   If we had chosen a significance level of, for example, 0.1, then this p-value would not be enough to reject the null hypothesis.

2.3.3 Using the Wilcoxon rank sum test on our data

To perform the test we first label each row as being "safe" or "dangerous" depending on whether it is less or greater than the median countdown value for the sample data. This gives us two distributions for each feature. We then loop through all features applying the Wilcoxon rank sum test to each one, producing a p-value for each feature.

As we will see below, many of the features have tiny p-values, so selecting a significance level is not needed here. The features with the smallest p-values show the highest difference in distribution between the "safe" and "dangerous" classes, and are therefore most likely to be the cause of failure of the compressor (or at least one of them).

The twenty most important features by this definition are shown in Table 1, in ascending order of p-value.

Table 1: The twenty selected features

Number	**Feature**	**p-value**
1	gg-speedhp	8.6×10^{-57}
2	pt-tempbrgthrust1	3.8×10^{-56}
3	pt-tempexh	4.0×10^{-56}
4	gg-flowfuel	2.8×10^{-54}
5	gg-tempexhtc3	3.5×10^{-54}
6	gg-speedip	4.2×10^{-54}
7	gg-presscompdelip	4.0×10^{-53}
8	pt-tempcoolingair1	1.3×10^{-51}
9	gg-tempcompdel	3.5×10^{-51}
. 10	gg-tempcompdelhp	4.1×10^{-51}
11	hp-headantisurge	5.1×10^{-51}
12	gg-presscompdel	1.4×10^{-50}
13	gg-tempexhtc6	1.7×10^{-50}
14	gg-tempexhtc5	2.0×10^{-50}
15	lp-pressdifantisurge	2.1×10^{-50}
16	gg-tempexhtc2	2.1×10^{-50}
17	gg-tempcompdelip	3.8×10^{-50}
18	hp-pressdifantisurge	1.3×10^{-49}
19	lp-speed	1.3×10^{-49}
20	pt-templuboiltank	1.3×10^{-49}

We remove the other 98 features from our sample, leaving us with a final sample data set of 599 rows by 20 columns. This is the data set we will use in the next section of this report.

3 Pattern recognition and Anomaly Detection

Having obtained an i.i.d sample data set and performed feature selection, the first task was to look for patterns in the data that might be indicative of predictability. The popular dimensionality reduction technique t-SNE was used for this, producing clusters of similar observations.

After establishing predictability, a technique recently developed in [6] called *conformal clustering (CC)* was applied to formally identify each cluster and also allow for detection of individual anomalies that do not lie within the clusters. A key feature of CC is the ability to tune the proportion of anomalies detected by changing the significance level, which we explore here.

This section outlines the theory behind these techniques followed by an analysis of the results obtained by applying them to the data. We also propose a way to identify the most important features for each cluster, which corresponds to the most likely causes of system failure.

3.1 t-SNE

3.1.1 Intuition

This explanation is based on [7], [8] and [9].

t-SNE is a non-linear dimensionality reduction algorithm used for exploring high-dimensional data, developed by van der Maaten *et al* [9]. It usually transforms the data set into 2 or 3 dimensions which enables visualisation.

Traditionally, principal components analysis (PCA) has been a more popular technique for dimensionality reduction and visualisation. However, PCA is a linear technique and hence is not able to interpret complex polynomial relationships between features. Furthermore, it focusses on placing data points which are dissimilar in the original feature space far away from each other in the new dimensions.

By contrast, t-SNE is a non-linear technique which focusses more on data point *similarity*: similar points in the original space are placed close together in the t-SNE space. An explanation of the algorithm follows in the next section.

3.1.2 Algorithm details

t-SNE was inspired by the *Stochastic Neighbour Embedding (SNE) technique* developed by Hinton et al [10], but makes a few adjustments. We start with an explanation of SNE.

SNE starts by converting the Euclidean distances between data points in the original feature space into a conditional probability which represents their similarities. Formally, the similarity of data point x_i to data point x_j is $p_{j|i}$:

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2{\sigma_i}^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2{\sigma_i}^2)} \tag{1}$$

where $p_{j|i}$ represents the conditional probability that x_i would choose x_j as its neighbour if neighbours were chosen in proportion to their probability density under a Gaussian distribution centered at x_i. σ_i is the variance of that Gaussian. Note that $i \neq j$ and $p_{j|i} = p_{i|j}$.

Let's say that y_i and y_j are the low-dimensional counterparts to x_i and x_j. We can compute an analagous conditional probability for those counterparts, denoted as $q_{j|i}$:

$$q_{j|i} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq i} \exp(-||y_i - y_k||^2)} \tag{2}$$

where $j \neq i$ and $q_{j|i} = q_{i|j}$.

Logically, for the low-dimensionality feature space to be an accurate representation of the original feature space, the difference between $p_{j|i}$ in the high-dimensional data and $q_{j|i}$ in the low dimensional data should be as small as possible. This is the goal of the SNE algorithm - to minimise the mismatch.

It does so by using an asymmetric cost function known as the *Kullback-Leibler (KL) divergence*. This is a measure of how different one probability distribution is from a second probability distribution:

$$D_{KL}(P||Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)} \tag{3}$$

where $D_{KL}(q||p)$ is the KL divergence, often called the *information gain* that would occur if P were used instead of Q.

SNE minimises this cost function over all the data points by gradient descent. However, it is hampered by two problems. Firstly by fact that the cost function is difficult to optimise due to its complex gradients, and secondly by the so-called "crowding problem". This refers to the fact that the area of the low-dimensional map available to accommodate nearby data points is often not large enough to accommodate distant ones.

t-SNE provides a solution to these problems. Like SNE it minimises the difference between the conditional probabilities, but does so using a symmetric form of the cost function which has simpler gradients, proposed by Cook *et al* [11]. It has the following form:

$$C = D_{KL}(P||Q) = \sum_i \sum_j p_{i|j} \log \frac{p_{i|j}}{q_{i|j}} \tag{4}$$

where $p_{j|i} = p_{i|j}$ and $q_{j|i} = q_{i|j}$.

To address the crowding problem, t-SNE uses a Student t-distribution rather than a Gaussian to compute the similarity between two points in the low-dimensional space. Using this distribution with one degree of freedom, the joint probabilities q_{ij} are defined as:

$$q_{i|j} = \frac{\left(1 + ||y_i - y_j||^2\right)^{-1}}{\sum_{k \neq l} \left(1 + ||y_k - y_l||^2\right)^{-1}} \tag{5}$$

This heavy-tailed distribution has the property that $(1 + ||y_k - y_l||^2)^{-1}$ approaches an inverse square law for large pairwise distances $||y_i - y_j||$ in the low-dimensional map. This results in a bigger distance between mapped points of dissimliar points.

3.2 Conformal Prediction

This explanation follows that of [12] and [13].

3.2.1 Intuition

Traditional ML algorithms do not provide confidence information in association with their predictions - their only output is *simple predictions*. But often we would like to have such confidence information in order to answer the question "How good is our prediction?"

Some statistical learning theories such as PAC and VC theory do give us bounds on prediction errors, but these bounds are often too loose to tell us anything useful unless the data set being studied is particularly clean, which is rarely the case. Bayesian methods can produce predictions with probabilistic measures of their accuracy, but involve making prior assumptions about the distribution generating the data. When these assumptions are wrong, the resulting confidence intervals are also incorrect.

Conformal prediction is a probabilistic framework for making predictions with confidence. It is makes no assumptions as to the nature of distributions, and has theoretically proven guarantees of maximum error rate, with the only assumption being that the training and test data are i.i.d.

The price to pay is that predictions are no longer single-valued. Instead, the predictions consist of a set of label values. In this sense the prediction can be considered as "hedged", and is considered correct if the prediction set contains the actual label. If it does not, then we consider this an error.

Rather than being a self-contained ML method, conformal predictors operate on top of another ML algorithm (known as the 'underlying' algorithm). This can be any classification or regression algorithm - the only requirement is that we can extract a "score" from it, as opposed to a simple prediction with no accompanying metric. This is because in order to apply conformal prediction to a traditional ML algorithm, we need to develop a *nonconformity measure (NCM)* based on that underlying algorithm. The NCM evaluates how different a new example (i.e. a feature-label relationship) is from a set of previous examples. It can be thought of as a measure of how "strange" the new example is given what came before, and lets us decide whether it is too strange to be included in our prediction set.

3.2.2 Formal explanation

We can formalise conformal prediction following the explanation in [12] .

Say that we have a training set $\{z_1, ..., z_l\}$ of observations where each $z_i \in Z$ is a pair (x_i, y_i). Note that $x_i \in \mathbb{R}$ is the vector of attributes for example i and $y_i \in \mathbb{R}$ is the label for that example.

Let's say we have a new unlabelled example x_{l+1} and we want to give some kind of confidence in the various $\bar{y}$ values which are candidates for the true label y_{l+1}. Our only assumption is that all $(x_i, y_i), i = 1, 2, ...$ are i.i.d. That is, they are generated independently from the same probability distribution.

In Section 3.2.1 we introduced the concept of a nonconformity measure. Formally, this is a function $A : Z^{(*)} \times Z \to \mathbb{R}$ where Z is the set of all possible labelled examples and $Z^{(*)}$ is the set of all bags of examples $\{z_1, ..., z_{i-1}, z_{i+1}, ..., z_n\}$. A tells us how different an example z_i is from bag $Z^{(*)}$ by assigning it a *nonconformity score (NCS):*

$$\alpha_i = A(\{z_1, ..., z_{i-1}, z_{i+1}, ..., z_n\}, z_i) \tag{6}$$

Suppose we are interested in some particular candidate label $\bar{y}$ for the example x_{l+1}. Our null hypothesis is that $(x_{l+1}, \bar{y})$ was drawn i.i.d from the same distribution as the training examples and so we can include it in our prediction set Γ^ϵ. Adding this new example $(x_{l+1}, \bar{y})$ to our training set $\{(x_1, y_1), ..., (x_l, y_l)\}$ gives us the extended set

$$\{z_1, ..., z_{l+1}\} = \{(x_1, y_1), ..., (x_{l+1}, \bar{y})\} \tag{7}$$

Now we can use an NCM A_{l+1} to compute the NCS

$$A_{l+1}(\{z_1, ..., z_{i-1}, z_{i+1}, ..., z_{l+1}\}, z_i) \tag{8}$$

of each example $z_i, i = 1, ..., l + 1$.

What might such a NCM be for an underlying algorithm like k-nearest neighbours (see section 4.1.1 for details of this algorithm)? For a bag $Z^{(*)} = \wr(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\int$ and $z_i = (x_i, y_i)$ it is as follows:

$$A(B, z) = \frac{min_{j:y_j=y_i} d(x_j, x_i)}{min_{j:y_j \neq y_i} d(x_j, x_i)} \tag{9}$$

In words it is the ratio:

$$\frac{\text{distance to z's nearest neighbour in } Z^{(*)} \text{ with the same label}}{\text{distance to z's nearest neighbour in } Z^{(*)} \text{ with a different label}}$$

Returning to our example, on its own the NCS α_{l+1} is not particularly useful. But if we compare α_{l+1} for example z_{l+1} against that of all other examples then we can quantify how unusual it is. To do this we use the formula

$$p(\bar{y}) = \frac{\#\{i = 1, ..., l+1 : \alpha_i \geq \alpha_{l+1}\}}{l+1} \tag{10}$$

where $p(\bar{y})$ is the *p-value* of $\bar{y}$. In words we can express this as:

$$\frac{\text{number of examples that conform worse or the same as } \alpha_{l+1}}{\text{total number of examples}}$$

We can reject the null hypothesis if the p-value is less than a chosen *significance level* ε, meaning that this candidate label is not selected for our prediction set. We then repeat this process for all other candidate labels. It is also common to talk in terms of a *confidence level* δ, where

$$\delta = 1 - \varepsilon \tag{11}$$

We can illustrate the meaning of ε and δ with the following example. Imagine that we have a training set of observations, each with a label from 0-9. After training a conformal predictor with k-nearest neighbours as described above, we obtain the following p-values for each label:

0	1	2	3	4	5	6	7	8	9
0.8	0.3	0.2	0.7	0.9	0.4	0.6	0.7	0.8	0.5

If we choose a significance level of 85%, we select only those labels with p-values > 0.85 for our prediction set $\Gamma^{\epsilon} : \Gamma^{0.85} = \{4\}$. Correspondingly, this is our prediction set at a confidence level of 15%.

Likewise, if we choose a significance level of 5%, our prediction set at a confidence level of 95% is $\Gamma^{0.05} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Conformal prediction has two desiderata:

1. **Validity:** The long run frequency of error does not exceed the significance level ε at each chosen confidence level δ. One of the key advantages of CPs is that they are automatically valid under the assumption of *exchangeability* [14]. Note that the exchangeability property is weaker than i.i.d, since

$$iid \Rightarrow exchangeable$$

   There is also the notion of **strict validity** which means that the probability of error *equals* ε, rather than being less than or equal to it. To achieve this, we adjust the formula for the p-value:

$$\tilde{p}(\bar{y}) = \frac{\#\{i = 1, ..., l+1 : \alpha_i > \alpha_{l+1}\} + \tau\{i \in \{1, ..., l+1\} : \alpha_i = \alpha_n\}}{l+1} \tag{12}$$

   where τ is a random number sampled in $Uni(0, 1)$. This is known as a smoothed conformal predictor [15].

2. **Efficiency:** The prediction set should be as small as possible. This depends on the quality of the NCM based on the underlying algorithm.

3.3 Conformal clustering

3.3.1 Intuition

Conformal clustering (CC) is a technique based on conformal prediction, in which clusters are identified from unlabelled training examples. It was developed by Cherubin *et al* in [6] and [7], and this explanation is based on those publications.

The intuition behind CC is that unlabelled data is used to train a conformal predictor which then outputs a region of "normality" (with respect to the data) on a 2D grid. This prediction region (or prediction set) varies in size depending on the value of ε, and can be divided into clusters, which we call "predicted clusters".

The original data points are then projected onto the grid and themselves grouped into clusters corresponding to the predicted clusters they lie in. Those points which do not lie in any cluster are classed as *anomalies*.

As an additional step to the original algorithm in [6], for purposes of failure explanation the data is projected back into the original feature space. We then repeat the Wilcoxon test from Section 2.3.3, but this time we compare clusters rather than different classes of the same feature. The idea is that this tells us which features explain the difference between the clusters we are interested in.

3.3.2 Formal explanation

A d-dimensional grid of equally-spaced points per side is created, where d is the number of features (in our case, $d = 2$). The range of each side of the grid corresponds to the minimum and maximum of the d features. The space between each grid point is a constant of 1 unit of the t-SNE projection space.

Now we train our conformal predictor. To do this we use the whole data set as an input to the k-nearest neighbour algorithm to find the distances to the k-nearest neighbours for all observations (each of which is located somewhere within our 2D grid). This gives us a table of distances with n rows and k columns.

Then we consider each grid point to be a kind of pseudo-test set, and loop through each one. During each loop we extend the table of distances by one row and one column to account for this test observation, and update the values in the table. We compute a nonconformity score for all $n + 1$ rows (that is, the whole data set plus grid point i) using the following nonconformity measure:

$$\text{A} = \text{Sum of the distances to the nearest k-neighbours}$$

We can then compute a p-value for each grid point by finding the proportion of observations which have an NCS at least as high as the test object, as explained in the previous section.

As in conformal prediction, our null hypothesis is that the test object (our grid point) is from the same distribution as the training set. If a grid point's p-value is below the significance level that we choose, then this null hypothesis is rejected.

Correspondingly, those grid points with a p-value above the chosen significance level ε are included in the prediction region, shown in yellow in Fig.6.

We start by assuming that each grid point is its own cluster (to differentiate this from a cluster of actual data points we will refer to these as *predicted clusters*). But intuitively clusters which are very close together should be classed as the same cluster. Hence we merge the clusters using a neighbouring rule, whereby two points z_i and z_j are in the same predicted cluster if they are within one grid point of each other on the grid.

Finally, having established the prediction clusters, we project the training data onto the grid and assign all points to data clusters corresponding to the predicted cluster that they lie in. Different clusters are denoted by different symbols in the upcoming figures. Any points that do not lie in a predicted cluster are considered anomalies.

An illustration of CC is shown in Fig.6. We start with a 2D grid (not to scale - the actual grid lines would be finer than this). We then train a conformal predictor on our training data as described above such that all grid points with a p-value above ε are included in our prediction region - these points are coloured yellow in the figure. The prediction region is then split into clusters according to the neighbouring rule. Finally we project the training data (in blue) onto the grid, with the data points shown in different shapes depending on which predicted cluster they lie in. Note, for example, the diamonds, triangles and crosses in the clusters on the left of Fig.6. Any points remaining outside the clusters are considered as anomalies. These data points break the i.i.d assumption so are not included in the prediction region.

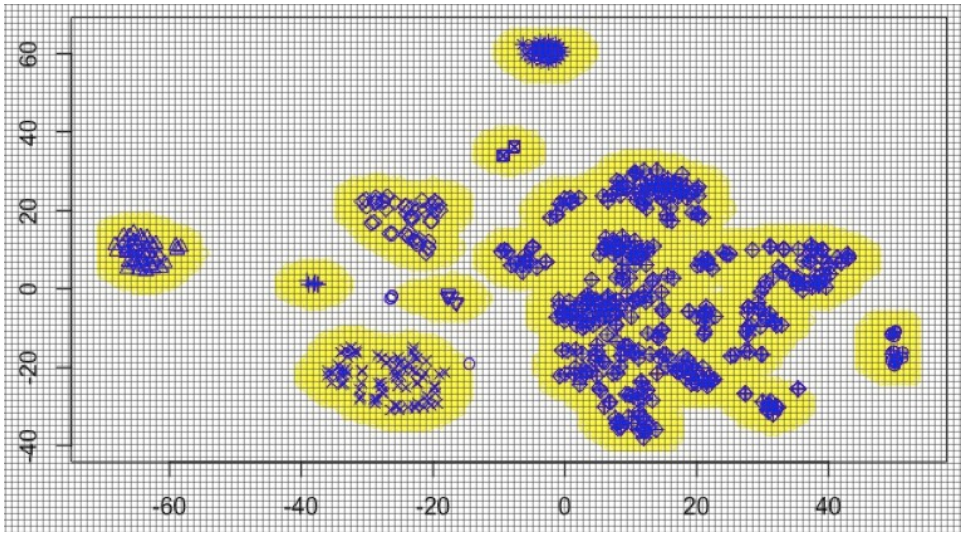

Figure 6: Two-dimensional prediction region. Units are 1% of maximum value for each feature

The significance level ε can be used to vary the proportion of observations classed as anomalies. A higher ε means that less grid points exceed the p-value necessary to be included in the yellow prediction region, hence our whole prediction region is smaller, and when we project the data onto the grid less points end up within clusters.

3.3.3 Methodology

To summarise, the steps to implement this algorithm were as follows:

1. Reduce dimensionality using t-SNE.
2. Conformal clustering to produce prediction region. Significance level $\varepsilon = 0.1$ and underlying method is k-nearest neighbours with $k = 5$.
3. Divide the prediction region into predicted clusters according to the neighbouring rule.
4. Project actual data onto the grid. Identify data clusters and anomalies in actual data.
5. De-project data back into original feature space.

6. Attempt to identify features which explain the differences between clusters using the Wilcoxon rank sum test.

Steps 1-4 follow from [6] while steps 5 and 6 have been added to enable explanation of the clusters. Steps 1 and 5 are only required when dealing with high-dimensional data.

3.4 Results and Discussion

3.4.1 Two-dimensional example

To demonstrate the technique, let's start by only applying it to the two most important features as measured by the Wilcoxon rank sum test in section 2.3. These are **gg-speedhp** (the speed of the gas generator) and **pt-tempbrgthrust1** (the temperature of the thrust bearings in the power turbine). Both features have been rescaled by dividing by 1% of its maximal absolute value to put them on the same scale.

We start with a plot of the CC prediction region, shown (Step 2) in yellow in Figure 7. We can consider this as the set of points that are *not* abnormal given the training set (which is the full data set). Unconnected yellow regions can be considered as *prediction clusters* - analagous to clusters of actual data. Correspondingly, the white area is the region of abnormality: any point located in this region is "strange", or anomalous, with respect to the training data.

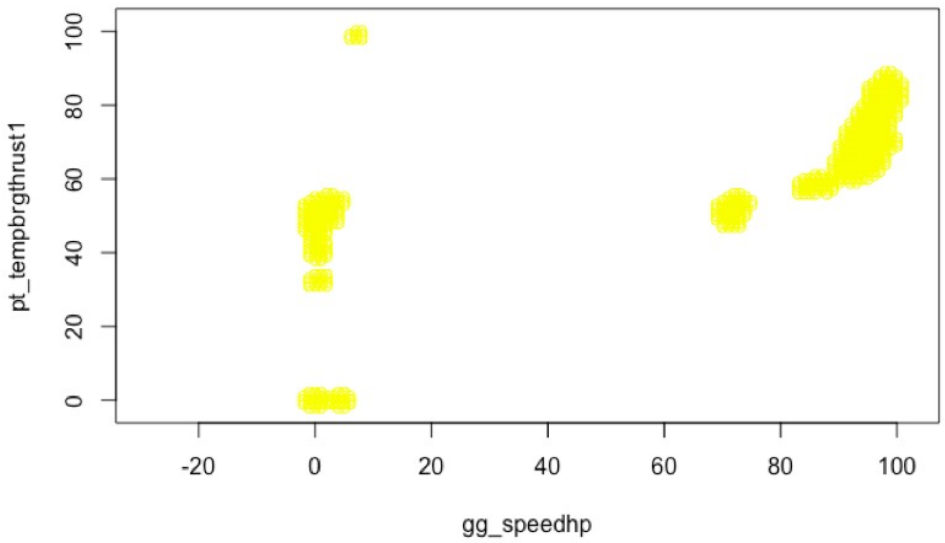

Figure 7: Two-dimensional prediction region. Units are 1% of maximum value for each feature

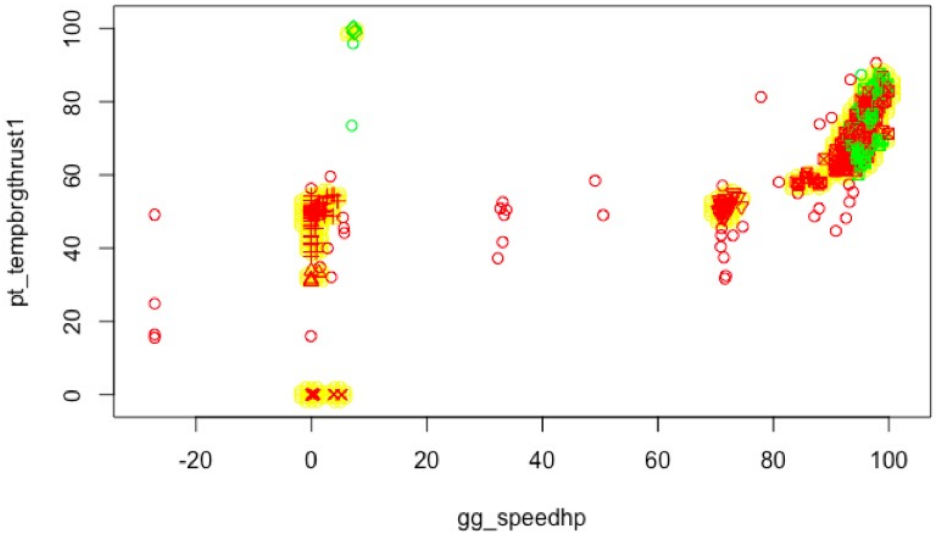

Figure 8: Two-dimensional prediction region and training data. Units are 1% of maximum value for each feature.

Figure 8 shows the training data superimposed on the prediction region. For each data point we can draw information from i) its x-position, ii) its y-position, iii) its shape and iv) its colour.

The shape of the data points correspond to their cluster number. For example, close inspection of the plot reveals triangles, crosses, circles with crosses inside, etc. A data point belongs to a different cluster than another data point if it is located in a different predicted cluster.

The colour of the data points corresponds to their **countdown** value. A green point means that that at that snapshot/moment the system was more than 24 hours from failure ($countdown > 144$: a "safe moment") and a red point means that it was less than 24 hours from failure ($countdown < 144$: a "dagnerous moment"). This colouring scheme is applied only at the end. It is noteworthy that the largest cluster centered at (7, 99) contains practically all the green observations.

A more quantitative analysis of the clusters is given in Table 2. It includes:

- The *location* as determined by the coordinates of the centre of the predicted cluster.
- The *volume* - a measure of space occupied by the predicted cluster.
- The *size* as determined by the number of actual data points in a the predicted cluster.
- The *risk* level - the proportion of red data points in the cluster.
- An *explanation* of the cluster in terms of its location on the axes.

It is also split into three sections: the largest cluster, which is also the safest by risk level; very dangerous clusters; and one cluster which is similar in risk level to the largest one.

The size and volume metrics are two different ways of measuring how large a cluster is, of which the volume is probably more valuable. A large and mainly red cluster would be of most interest to us since that would denote a dangerous state of the system that is frequently observed.

The final column is an attempt to explain the difference between a given cluster and the largest one in terms of its coordinates. We note that safer states of the system tend to be found at higher values of the features, apart from a very green data cluster at (7,99). This would indicate that when the

speed of the gas generator and the temperature of the thrust bearing fall, there is a high probability that the compressor will fail.

All anomalies are shown as circles on the plot, and we note that nearly all of them are coloured red. This means that any such anomalous moment usually precedes a break down of the compressor.

Table 2: Clusters explained in 2 features sorted by risk level

Cluster location	**Vol.**	**Size**	**Risk level**	**Explaining features (vs. majority)**
LARGEST CLUSTER				
(95, 71)	270	373	0.61	feature 1: very high 80–100 feature 2: high 55-90
VERY DANGEROUS CLUSTERS				
(0,32)	15	5	1	feature 1: very low -2–3 feature 2: 30-35
(1, 49)	97	63	1	feature 1: low -3–7 feature 2: 35–58
(1, 0)	37	74	1	feature 1: -3–7 feature 2: -2–3
(72, 51)	50	33	1	feature 1: 67-77 feature 2: 45-55
SAFEST CLUSTER				
(7, 99)	8	4	0.25	feature 1: low 5-10 feature 2: very high 95-100

3.4.2 Multi-dimensional example

Having demonstrated the CC technique with two features, lets us now look at all twenty features from the pre-processed data set. Unlike in the two-dimensional example, t-SNE has been applied to reduce each observation into two dimensions. Conformal clustering has then been applied to the t-SNE dimensions. As with two dimensions, we have used a significance level $\varepsilon = 0.1$ and k-nearest neighbours as the underlying method for the NCM calculation with $k = 5$.

Fig.9 shows the resulting prediction region in yellow. The four corners of the grid are marked with blue circles. Fig.10 shows the training data superimposed on the prediction region and coloured according to their danger

level.

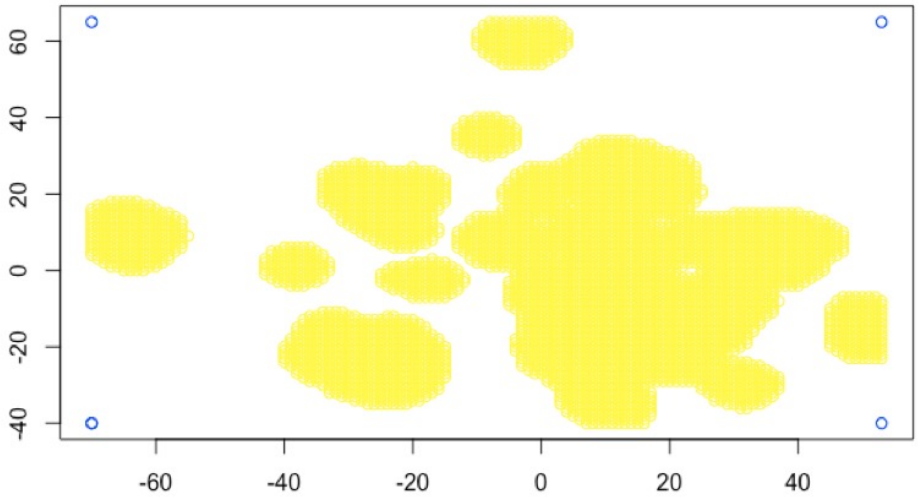

Figure 9: Prediction region for 20 features after t-SNE. Axes have artificial units in the t-SNE projection space

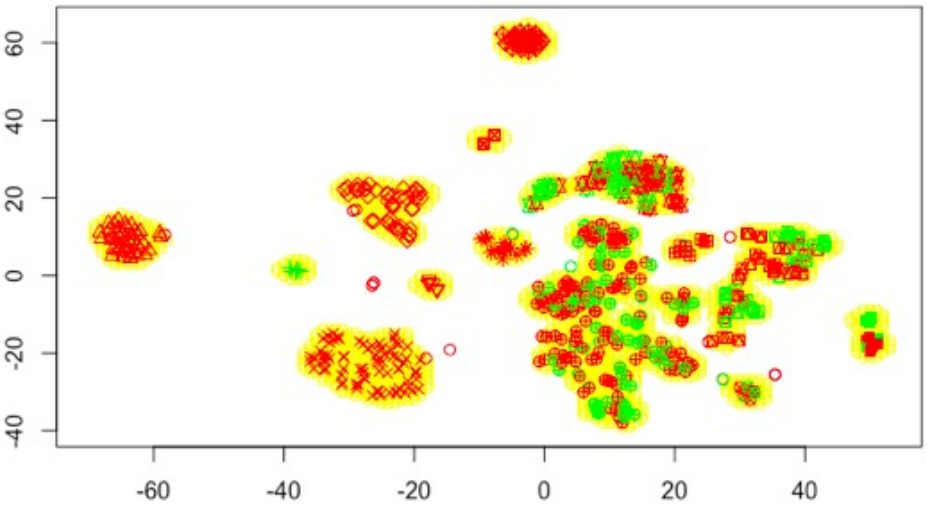

Figure 10: Prediction region and actual data for 20 features after t-SNE. Axes have artificial units in the t-SNE projection space

We can see that this plots contains many more clusters than in the two-dimensional case. The green observations are also more spread out among the clusters, rather than mainly being located in one. That said, there are several clusters - the largest at (-27,-23), (-24,18), (-64,9) and (-3, 60) - which are completely red and hence correspond to very dangerous states of the system.

These clusters can also be found in Table 3, which is analagous to Table 2. However, the last column now seeks to identify the features which may explain the cluster in a slightly different way. Simply looking at its coordinates does not tell us much since t-SNE dimensions do not have a physical meaning. Instead, we redo the Wilcoxon rank sum test to compare each feature from a given cluster with the largest cluster, which is also one of the safest ones, and may be interpreted as the most common state of the system. This produces a p-value for each feature, and we select those features with the smallest p-values as being the most likely causes of differentiation between the two clusters.

Table 3: Analysis of clusters produced from 20 features, sorted by risk level

Cluster	Vol.	Size	Risk	Explaining features
(-27,-23)	295	76	1	ALL $p < 10^{-35}$ smallest: 5,17,20
(-24,18)	168	42	1	ALL $p < 10^{-23}$ smallest: 5,10,12
(-64,9)	117	41	1	ALL $p < 10^{-22}$ smallest: 18,4,19
(-6,7)	55	14	1	ALL $p < 10^{-8}$ smallest: 12,6,13
(-17,-2)	29	6	1	ALL $p < 10^{-4}$ smallest: 1,5,17,20
(-9,35)	25	6	1	ALL $p < 10^{-4}$ smallest: 5,7,8,13,14
(-3,60)	71	25	1	ALL $p < 10^{-15}$ smallest: 14,5,20
(22,7)	40	10	0.9	1–15,17–18,20 ($p < 10^{-6}$) smallest: 8,15,4
(31,-30)	43	9	0.66	1,10,16 ($p < 10^{-5}$)
(10,12)	786	198	0.62	**THE LARGEST**
(34,0)	287	68	0.57	1–2,4–9, 11–15,17,19–20 ($p < 10^{-8}$)
(12,24)	236	66	0.57	12 ($p < 10^{-11}$)
(50,-15)	64	14	0.5	8 ($p < 10^{-9}$)
(-38,1)	28	6	0.16	ALL $p < 10^{-4}$ smallest: 7,8,10,13–15,17

3.4.3 Varying the significance level

As explained in section 3.3, the significance level ε can be used to vary the proportion of observations that are classed as anomalies. This is illustrated in Figs 11 - 14. We can see that the as ε increases, the prediction region contracts which reduces the size of the clusters causing some to split, and increasing the number of observations classed as anomalies. For example,

note how the area of the grid centered at approximately (18,2) goes from being part of the prediction region to being outside it by the time that ε has risen to 0.2, causing the surrounding data points to be classed as anomalies.

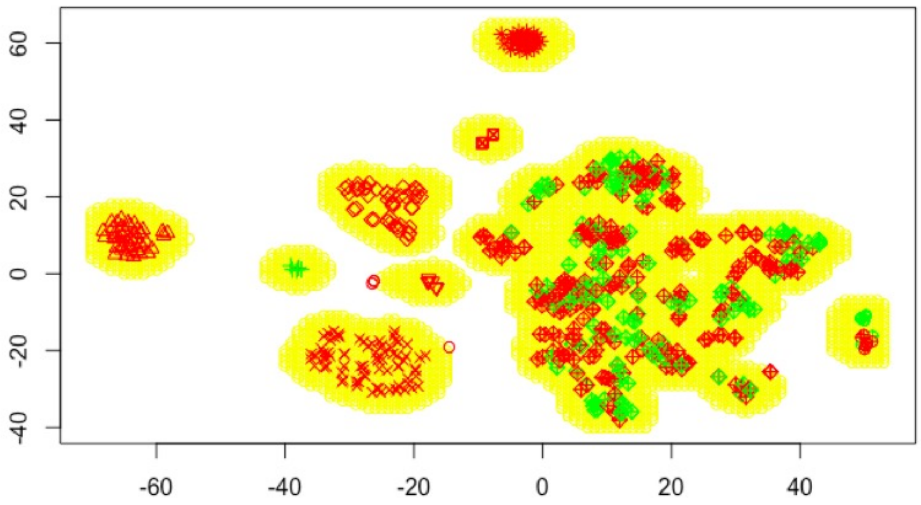

Figure 11: Prediction region and actual data with $\varepsilon = 0.01$. Axes have artificial units in the t-SNE projection space

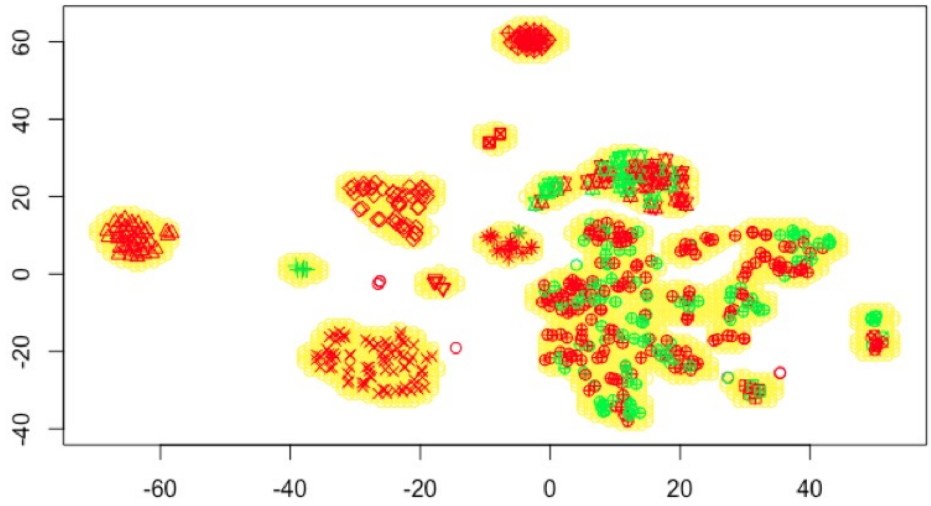

Figure 12: Prediction region and actual data with $\varepsilon = 0.05$. Axes have artificial units in the t-SNE projection space

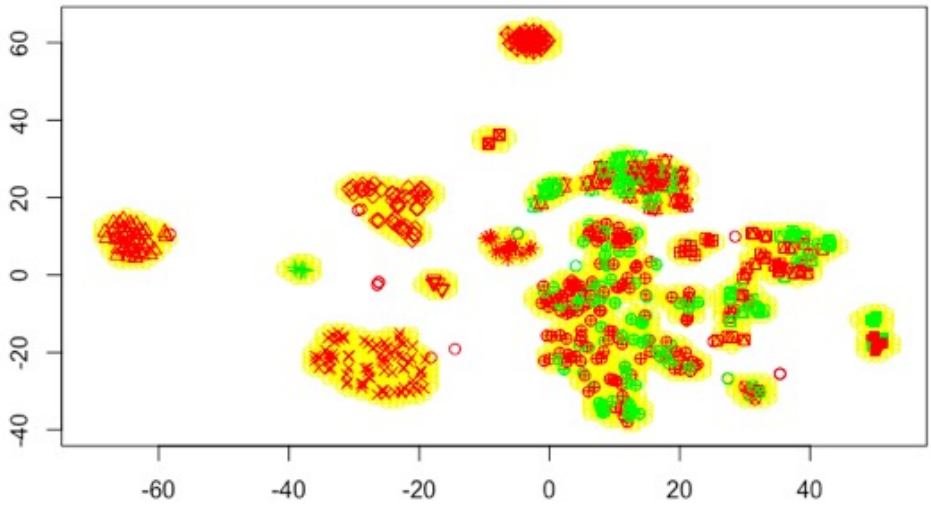

Figure 13: Prediction region and actual data with $\varepsilon = 0.1$.

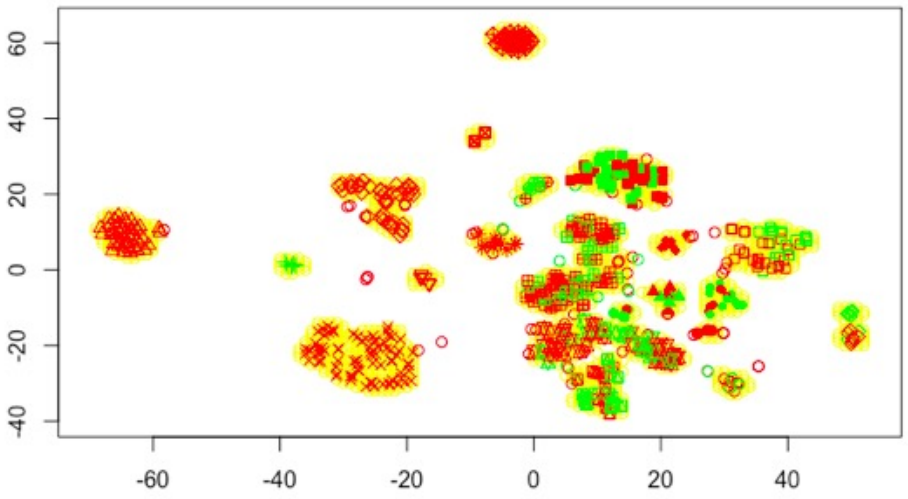

Figure 14: Prediction region and actual data with $\varepsilon = 0.2$.

4 Failure Prediction

In the previous section, we used conformal clustering to show that there are indeed patterns in the data, suggesting good predictability. As an extension, we also proposed a way to explain which features most likely explain the difference between smaller clusters and the largest one.

In this section we turn our attention to building a predictive model capable of predicting whether our system will break within a given time frame. We know from the CC analysis that some clusters are more predictable than others, as they have higher scores on the risk metric. This means that sometimes we can make predictions with higher confidence than other times. To capture this insight, we therefore choose to make predictions in a *probabilistic* form rather than providing a simple "Yes/No" prediction.

4.1 Probabilistic prediction

We start with an overview of the k-nearest neighbours algorithm, which is the basis of the predictive model. We then describe the full methodology, followed by a discussion of the results obtained.

4.1.1 k-Nearest Neighbours algorithm

This explanation is based on [16] and [3]. The k-nearest neighbours algorithm (k-NN) is a non-parametric technique for both classification and regression. It takes as input the k closest training examples to a test example in the feature space. The output depends on whether what kind of task we are doing:

- In classification, an object is assigned to a particular class by a majority vote of its nearest k neighbours. If k=1 it is simply assigned to the class of that single nearest neighbour.
- in regression and probabilistic prediction, we are seeking a numerical value for the object, which is usually the average of the values of its nearest k neighbours.

The key part of the calculation is calculating the distance between a test observation and its surrounding training observations. Euclidean distance

(i.e. the straight-line distance between two objects) is the most commonly used metric for this:

$$\left(\sum_{j=1}^{P}(x_{aj} - x_{bj})^2\right)^{1/2}$$

where x_a and x_b are two objects with features $j = 1, 2, ...P$.

In this approach to probabilistic prediction, we adapt the k-nearest neighbours classification algorithm such that instead of making a binary classification based on a majority vote, we produce a probability by taking the average of the classes of the surrounding k neighbours. For example, if $k = 50$ and a test object has 40 neighbours from class A out of those 50, we would predict class A for the object with a probability of 80%. In this sense it is a classification technique with an element of regression (but not a regression technique *per se*).

4.1.2 Batch setting vs Online setting

Any machine learning system can be said to learn in *batch mode*, where it is first trained using all available data and then runs without learning anymore, or *on-line mode*, where the system is fed data incrementally and learns on the fly. Here we explore both approaches.

Method in batch setting

1. Split i.i.d data set 50:50 into training and test set.
2. Feature selection using Wilcoxon rank sum test on training set.

for *each test observation* i **do**

3. Apply k-nearest neighbour algorithm:
   - Calculate Euclidean distance between all training observations and test observation i.
   - Find k nearest neighbours to i.
   - Make a prediction for i based on the average of its neighbours

end

Method in on-line setting

In on-line mode the method is slightly different:

for *each test observation* i **do**

1. Define the training set (all previous observations before the test observation) and test observation i.
2. Feature selection using Wilcoxon rank sum test on training set.
3. Apply k-nearest neighbour algorithm:
- Calculate Euclidean distance between all training observations and test observation i.
- Find k nearest neighbours to i.
- Make a prediction for i based on the average of its neighbours

end

Of the two, the on-line setting is more realistic in this scenario (sensor data is a time series) so we will use that one for our experiments later on.

4.2 Results and Discussion

4.2.1 Analysis of batch and on-line models

Figures 15 and 16 show the results for the batch and online approaches, with $k = 50$ and a time window of 1 day. The predicted probability is on the x-axis, while the upper part of the plot shows observations that were within 24 hours of failure and the lower part shows observations that were not.

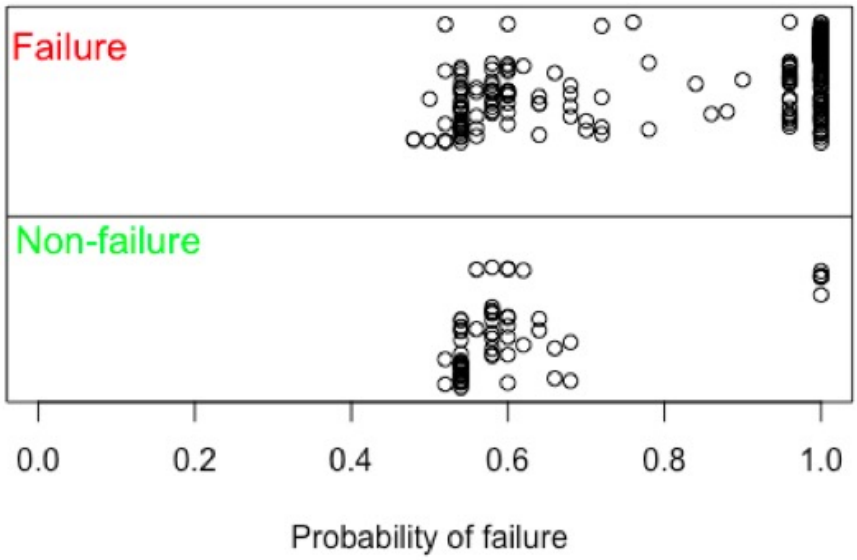

Figure 15: Predicted probability of failure within 1 day vs outcome: Batch setting

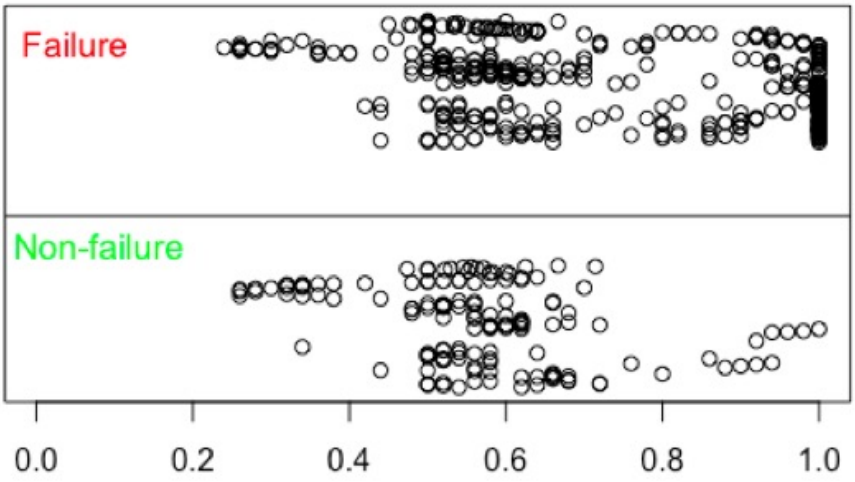

Figure 16: Predicted probability of failure within 1 day vs outcome: On-line setting

Both models tend to give high probability to realised failures rather than non-failures, which is positive. The predictions of 100% (some of which are wrong, especially for the batch model) clearly need to be adjusted. In future work this could be accomplished using a Venn-Abers predictor, which produces calibrated predictions [17].

There is a larger variance in the predictions made by the on-line model, as might be expected given that due to the nature of its learning process it is better tuned at the end of its training than at the beginning. It would be useful to see how its predictions change over time.

We therefore replot Fig.16 with time (in moments) on the x-axis and predicted probability on the y-axis - see Fig.17. As in the conformal clustering part of this report, we colour an observation red if it was within 1 day of failure or not and green if it wasn't.

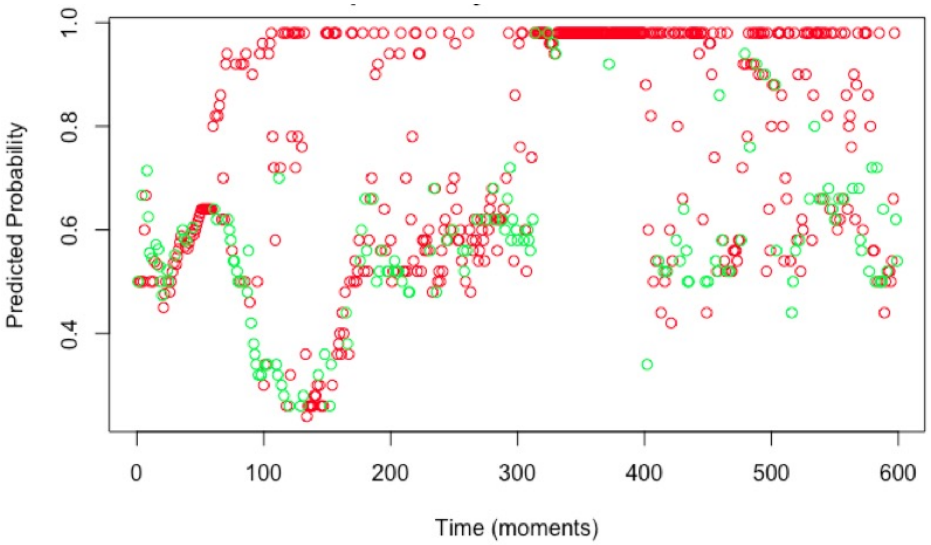

Figure 17: Time vs Predicted probability of failure within 1 day: On-line setting. Red = Realised failure, green = realised non-failure

In the plot we can observe improved performance of the model over time. Given that a false negative (i.e. incorrectly indicating no danger) is worse than a false positive (incorrectly indicating danger), we would hope to see that over time less red observations are given a low probability of failure, which is the case.

It is notable that nearly all observations with a low predicted probability ($< 40\%$) appear in a small region between ∼100-170 on the x-axis, with a high concentration of red observations around the 130 mark. This suggests that a new kind of breakage has appeared for which the algorithm has not yet been trained, and implies deviation from i.i.d.

After ∼170 on the x-axis, however, the model appears to have learnt about this kind of breakage and makes no more predictions $< 40\%$ for red observations. After ∼300, the vast majority of red observations are given a high predicted probability of failure, many of them around the 100% mark.

Visualisations can only give us limited insight on model performance. Table 4 shows a way in which we can summarise its accuracy at different intervals of probability:

Table 4: Proportion of actual failures in probability intervals from Fig. 17

Interval	**No. Predictions**	**Proportion of actual failures**
0.0-0.1	0	NaN
0.1-0.2	0	NaN
0.2-0.3	22	0.73
0.3-0.4	23	0.39
0.4-0.5	57	0.60
0.5-0.6	158	0.61
0.6-0.7	84	0.63
0.7-0.8	26	0.77
0.8-0.9	27	0.89
0.9-1.0	202	0.97

The third column of the table shows that (apart from the 20-30% interval discussed above) the proportion of actual failures increases with each probability interval. Most notably, the model puts 202 test observations in the 90-100% probability interval, and 97% of these resulted in failure within a day. This means that when our model is highly confident of failure, it is very often correct.

4.2.2 Parameter tuning

Maybe k=50 isn't an optimal value for the number of nearest neighbours? Maybe we would like to know how our model performs when trying to predict

failure within 12 hours rather than 24? To do so we need a single-value metric to evaluate the performance of our model. We therefore introduce a loss function at this point.

For probabilistic models such as ours, a common loss function is the *log loss* or *cross entropy*. This takes the predictions of the model and the true target, and computes a distance score between them which captures how well the model has done on this test example.

The log loss calculated for different values of k, and the results are shown in Table 5. The result is marginal, but the lowest log loss comes when $k = 25$.

Table 5: Log loss for different values of k neighbours

k	**Log loss**
5	0.52
10	0.48
25	0.47
50	0.48
100	0.50

Having established that k=25 is optimal, the next task was to investigate how our model performs for different time windows. Would it perform worse when asked to predict failure within 6 or 12 rather than 24 hours? See table 6 for the results.

Table 6: Log loss for different time windows

Time window (hours)	**Log loss**
1	0.45
6	0.51
12	0.51
24	0.47
48	0.43

It is interesting to note that model performance seems to perform best on short and long time windows (1 and 48 hours) rather than the medium ones.

5 Conclusions and Further Work

To conclude, we have managed to:

- Pre-process the data and take an (almost) i.i.d sample.
- Use the recently-developed conformal clustering technique to show that there are patterns in the data which imply predictability. The clusters refer to frequently observed states of the system.
- Formally identify the clusters programmatically (as opposed to just visual inspection), as well as the anomalous data points which sit outside these clusters. We have also shown that we can vary the proportion of data points classified as anomalies using the significance level ε, which is a feature of conformal clustering.
- Proposed a methodology for identifying which metrics are the most likely causes of failure in the subsystem.
- Built a predictive model using k-nearest neighbours that gives probabilistic predictions of whether the system will fail within a given time frame, and run experiments to find out the optimal value of k (25) and the optimal time window (48 hours). The model shows best performance when it makes predictions with high probabilities.

In terms of further work, having developed this methodology, it would be interesting to see how the algorithms work on the full data set, as opposed to just the i.i.d sample. In the case of the probabilistic prediction model, this would be a more realistic reflection of how data would be input into the model if were to be productionised in a commercial setting.

We only used the k-nearest neighbours algorithm for probabilistic prediction because it is a simple approach which has a proportion-based voting system and hence lends itself to producing probabilities. But there is no reason why other algorithms couldn't be used, such as random forests, neural networks or support vector machines.

It would also be worth adjusting the model to make sure that predictions with 100% probabilities are not made. Even when confidence of failure is high it is unwise to make such confident predictions, and as we saw in Section 4.2 they can sometimes prove to be wrong anyway. Venn-Abers prediction provides a framework for doing this [17]. In that section we also saw what

appeared to be deviation from i.i.d behaviour in Fig.17, which would be worth investigating.

Finally, we used t-SNE to reduce the high-dimensional data to two dimensions, but it can also output three dimensions. It would be interesting to see what kind of effect this might have on the number and purity of clusters produced by conformal clustering.

6 Professional Issues

Considerable time spent at the start of this project getting to grips with what the data set was actually describing. That is, understanding what the column names meant, where did the compressor fit into the rest of the terminal, was there potentially another data set that would be more useful to analyse, exactly what time period was covered by the data, etc. This was largely due to the fact that the engineers who know the systems best were based in Aberdeen and were not easy to reach.

Furthermore, the original data itself had to undergo some cleaning before being entered into Centrica's data lake, from where I was able to access it. This cleaning included the removal of the date/time feature due to data type constraints in the database. This in turn meant that it was impossible to compare the presumed failures in the Alerts data set with an analagous data set consisting of engineering text notes. These notes contained the engineer's diagnosis of every fault in the compressor and the solution s/he implemented to fix it. The value of this project would have been greatly increased if it had been possible to compare the main causes of failure we have proposed in Table 3 with the actual causes as identified by the engineer.

Another issue was that the Morecambe Bay gas terminal was built in 1992, and over its lifetime has undergone several upgrades and refurbishments. As a result, not all of the sensor data necessarily reflects the actual state of the system. That is, some "failures" in our original dataset may have actually been planned outages to allow for an upgrade. However, the patterns in the data as revealed by conformal clustering suggests that there were not many such cases.

Perhaps most importantly of all, soon after the commencement of this project it was announced that the subsidiary of Centrica that owned the data (Centrica Exploration & Production) was to be closed down and replaced with a joint venture between Centrica and the German utility Bayerngas Norge. This joint venture is under new management with different priorities to those of the previous management, and as a result there is (as yet) not much interest in using the results from this project for starting a predictive maintenance program. This author hopes that, despite this, the techniques developed here will be applied in other parts of the company. For example, Centrica Storage Limited is another subsidiary with several large assets which could benefit greatly from such a predictive maintenance program.

Regarding data privacy, since this data set pertained to sensor data from machines (and from a company that technically doesn't exist anymore) as opposed to customer data, there were no General Data Protection Regulation (GDPR) related issues in this project.

Regarding plagiarism, naturally I obtained permission to use the Cherubin *et al's* paper on conformal clustering [6] as a basis for the pattern recognition part of this report, and indeed benefited from the advice of the main authors of that paper as to how proceed throughout this project.

This proprietary dataset was provided by Centrica PLC (Registered office address: Millstream, Maidenhead Road, Windsor, Berkshire, SL4 5GD), all rights reserved.

7 Self assessment

To complete this section I will answer the questions posed in the Project Handbook in Q&A format:

- How did the project go?

Overall, very well. Plenty has been achieved in a relatively short space of time: recognising patterns in the data, detecting anomalies, proposing the main causes of failure for different states of the system, and building a model to predict failure of the system in various time frames. I'm pleased with how much this project covers.

- What did you do right/wrong?

The right things I did included talking at length to the main authors of the Cherubin *et al* paper for advice, as well as consulting with other data scientists and engineers at Centrica for getting more insight on the data set and the gas terminal.

In terms of things I would have done differently, I think that the project could have gone even better if I had asked the Morecambe Bay engineers more questions about the terminal and the Alerts data set at the very beginning, rather than waiting until I'd had a chance to look thoroughly at the data before doing so. This would have made planning the project easier (see next question).

Also, although my code works, I think it could perhaps be a bit more efficient. For example, to run the algorithms with different parameters you need to manually change a part of the script and then run it all from scratch. This could have been improved by writing formal functions which execute the algorithms, such that all the user needs to do is run the function while changing the values of the arguments provided to it.

- What have you learnt about planning and executing a project?

A great deal. In particular, as mentioned above I learnt how important it is to ask as much as possible about the data set and underlying physical systems before starting to analyse it yourself. This saves a lot of time further down the road.

From the point of productivity, I also grew to appreciate the *minimal viable product (MVP)* approach taken by many software companies, whereby you build a basic prototype of a model as quickly as possible and then iteratively

improve it, rather than spending a long time building that prototype such that it contains as few errors as possible first time round. The same applies to the writing of this report.

- *Where might you go next?*

See the Conclusions section for proposed next steps in this project. I am hopeful that a paper will be published based on some of the results in this thesis.

8 Appendices

8.1 How to use my project

This report has been submitted as a PDF along with four files of R code and one data file.

8.1.1 Code files in Code/ folder

Note that these files are in R Markdown (.Rmd) format, *which should be opened in the RStudio IDE, not the base R program.* This means that rather than being simple .R scripts that you execute in one go, they are more like notebooks containing chunks of R code that you run one chunk at a time. That said, there is a "Run All" option that lets you execute the whole script.

I chose to do this project in R markdown as the notebook-form facilitates analysis more easily than the traditional script-only form. This is analagous to the popular Jupyter Notebooks used in Python programming.

The four files are as follows:

1. *0_data_preprocessing.Rmd* - this file shows how the original data set (with 7 years' worth of data) was preprocessed to get the sample data set with dimensions of [599 x 119] which was the basis for the rest of this project. Note that the original data set has not been provided in the Data/ folder due to its large size (nearly 1 GB). Therefore this script cannot be run by the marker, it is just to illustrate how the data was preprocessed to produce the Alert_representatives data frame which formed the basis for the analysis in this report.

2. *1_clusters_for_2_fts_v2.Rmd* - this starts from the tb_alerts_representatives sample and runs conformal clustering in two dimensions. I recommend looking at the next file as this similar and contains more comments to explain how the script works.

3. *2_cluster_for_20_fts_and_tsne.Rmd* - this file contains the code for running conformal clustering on 20 features. Note the key lines for changing parameters (the value of k and the significance level ε) are lines 91 and 92.

4. *3_probabilistic_predictions.Rmd* - this file contains the code for the probabilistic prediction part of this project. Note that the key lines for changing parameters (the time window, referred to in the script as "danger_period", and the value of k) are lines 38 and 40.

All files contain extensive comments to explain what is happening in each part of the script.

8.1.2 Data files in Data/ folder

This folder contains the *Alert_representatives.RData* data frame. This contains the sample of dimensions [599x119] descibed above i.e. after processing the original data set, but before feature selection has been performed using the Wilcoxon rank sum test. This is the starting point for all the analysis in this report (i.e. for the last three scripts described in the previous section)

References

[1] Https://www.anodot.com/blog/predictive maintenance-whats-the-economic value/. Predictive Maintenance: What's the Economic Value? URL `https://www.anodot.com/blog/predictive-maintenance-whats-the-economic-value/`.

[2] Timothy Wong. Sequence-to-Sequence Model for Signal Analysis. 2017.

[3] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling [Hardcover]*. 2013. ISBN 1461468485. doi: 10.1007/978-1-4614-6849-3.

[4] Rosie Shier. The Mann-Whitney U Test. Technical report, Mathematics Learning Suport Centre, 2004. URL `http://statstutor.ac.uk/resources/uploaded/mannwhitney.pdf`.

[5] Chris Wild. The Wilcoxon Rank-Sum Test. URL https://www.stat.auckland.ac.nz/ wild/ChanceEnc/Ch10.wilcoxon.pdf.

[6] Giovanni Cherubin, Ilia Nouretdinov, Alexander Gammerman, Roberto Jordaney, Zhi Wang, Davide Papini, and Lorenzo Cavallaro. Conformal Clustering and Its Application to Botnet Traffic. In *Statistical Learning and Data Sciences*, volume 9047, pages 313–322. Springer, 2015. ISBN 978-3-319-17090-9. doi: 10.1007/978-3-319-17091-6. URL http://link.springer.com/10.1007/978-3-319-17091-6.

[7] Giovanni Cherubin. *Bots detection by Conformal Clustering*. PhD thesis, Royal Holloway, 2014. URL https://giocher.com/files/docs/bdcc-msc-thesis.pdf.

[8] Https://www.analyticsvidhya.com/blog/2017/01/t sne-implementation-r python/. Comprehensive Guide to t-SNE algorithm.

[9] L J P Van Der Maaten and G E Hinton. *Visualizing high-dimensional data using t-sne*, volume 9. 2008. ISBN 1532-4435. doi: 10.1007/s10479-011-0841-3.

[10] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, pages 833–840, 2002. ISSN 10495258. doi: http://books.nips.cc/papers/files/nips15/AA45.pdf.

[11] James Cook, Ilya Sutskever, Andriy Mnih, and Geoffrey Hinton. Visualizing Similarity Data with a Mixture of Maps. *International Conference on Artificial Intelligence and Statistics*, (1):67—-74, 2007. ISSN 15324435.

[12] Harris Papadopoulos, Vladimir Vovk, and Alex Gammerman. Regression conformal prediction with nearest neighbours. *Journal of Artificial Intelligence Research*, 40:815–840, 2011. ISSN 10769757. doi: 10.1613/jair.3198.

[13] Paolo Toccaceli. Tutorial on Conformal Predictors and Venn Predictors. 2018.

[14] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. Algorithmic Learning in a Random World, 2005.

[15] Alexander Gammerman and Vladimir Vovk. Hedging predictions in machine learning. *Computer Journal*, 50(2):151–163, 2007. ISSN 00104620. doi: 10.1093/comjnl/bxl065.

[16] Wikipedia. K-nearest neighbors algorithm. URL https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[17] Vladimir Vovk and Ivan Petej. Venn-Abers predictors. pages 1–18, 2018. URL http://arxiv.org/abs/1211.0025.