Anomaly Detection in Biosensor Waveforms

SIEMENS HEALTHINEERS

Justine Filion | Neethu Gopalakrishna | Saisree GR | Sara Hall
UNIVESITY OF BRITISH COUMBIA | MASTER OF DATA SCIENCE CAPSTONE PROJECT FINAL REPORT

Contents

Executive Summary 3
1. Introduction 4
2. Background 5
3. Data 9
4. Methods 12
4.1 Pre-processing 13
4.1.1 Zeroing with respect to Sample Detection 14
4.1.2 Removing the wet-up period 14
4.1.3 Normalizing 15
4.1.4 Noise Reduction 16
4.2 Describing the pipelines 16
4.2.1 Pipeline 1 16
4.2.2 Pipeline 2 17
4.2.3 Pipeline 3 18
4.2.4 Pipeline 4 20
5. Results and Discussion 20
6. Conclusions 27
APPENDIX A : SUMMARY TABLES OF PIPELINES 30
Table 1. 30
Table 2. 31
APPENDIX B : REASONS AS TO WHY ITERATION 1 WAS UNSUCCESSFUL 31
APPENDIX C : TOOLS USED IN THE UNSUCCESSFUL PIPELINES 33
1. PREPROCESSING 33
1.1 Removing the wet-up period 33
1.2 Standardizing 33
1.3 Windowing 33
1.4 Noise Reduction 34
2. FEATURE EXTRACTION 34
2.1 TSFresh 34
3. CLUSTERING 35
3.1 Hierarchical clustering 35
3.2 Self-Organizing Maps (SOM) 36

APPENDIX D: OTHER NOISE REDUCTION TECHNIQUES 38
Moving Average Smoothing 38
Savitzky-Golay Filtering 38
APPENDIX E: IN-DEPTH DESCRIPTION OF TECHNIQUES USED IN THE PROMISING PIPELINES 39
1. K-Means Clustering 39
2. Random Forest Classifier 39
3. K-Shape Clustering 40
3.1 Autocorrelation 43
4. Autoencoder 43
5. Principal Component Analysis (PCA) 45
6. Gaussian Mixture Modelling (GMM) 47
APPENDIX F: DIAGNOSTIC PLOTS 49
References 50

Executive Summary

Siemens Healthineers specializes in manufacturing medical diagnostic equipment, an example of which is the epoc® blood system, which is used to perform comprehensive blood analyses. One of the main components of this system is a test card which contains the biosensors necessary to perform the testing for a variety of analytes contained in blood samples. To measure these analytes, the test card is inserted into the reader portion of the system, the sample is injected into the test card, and the results are reported by the host.

Before being shipped to customers for use, these test cards must undergo comprehensive quality control testing. If unsuccessful results are yielded during this process, an operator may manually inspect the raw data, which includes time-series signals, and the test cards in order to determine the root cause of failure. However, this is time consuming, and misclassification may occur due to subjectivity in manual review. As such, this project aimed to create an unsupervised machine learning pipeline that characterized these unsuccessful readings into different groups so that their occurrences over time can be tracked and tied back to manufacturing. This tool will help identify possible root causes of failure which can then be addressed, improving the manufacturing process.

One of the root causes that has been investigated occurs when an electrical circuit disruption (ECD) occurs during the testing of the card. Having the labels for these anomalies allowed us to evaluate whether or not our unsupervised pipelines were able to form clusters that could potentially be tied back to root causes. Thus, we used these labels to diagnose whether or not our pipelines were successful.

Before we could apply any machine learning models to cluster the signals, we had to pre-process them. Pre-processing included aligning the signals with respect to a semantically consistent time point, making sure all the signals were of the same length, and performing noise reduction. Various machine learning models, including K-Means, K-Shape and Gaussian Mixture Modelling were then used to cluster the unsuccessful readings. We obtained four promising pipelines that were successful to varying degrees in clustering the ECD errors from other unsuccessful readings. Furthermore, we observed distinguishable shapes across the different clusters which could be used in the future to identify and track potential root causes of failure.

To conclude, we successfully developed four pipelines that led to clusters with a high concentration of ECD errors and other clusters with characteristic signal shapes. This will be useful to Siemens Healthineers for helping identify root causes of failure and improve test card manufacturing.

1. Introduction

As the healthcare sector evolves to incorporate more technology for faster and more accurate diagnostics, there is an increasing demand for commercially produced biomedical devices. For these instruments to help improve patient care, they must be accurate and reliable, and thus undergo many tests for quality control. With the development of machine learning techniques in recent years, we are uniquely positioned to automate certain aspects of the quality control process.

One area of development is for systems that perform blood analysis, as it is one of the most common tests run by health professionals and is involved in the diagnosis of many different conditions [1][2].

The epoc® system has three components: a test card which contains the components necessary to perform the testing for a variety of analytes, a reader with a card slot for the test card that records the test results, and a host that interfaces with the reader to provide information to the healthcare professional [3]. The epoc® system fulfills these needs as a handheld wireless system that enables comprehensive blood analysis testing at the patient's side within minutes on a single test card [2]. In order to measure these analytes, chemical reactions occur in the test card over time, producing a signal which is recorded and can then be analyzed. Henceforth, we will refer synonymously to the signals produced when measuring a given analyte over time as time series, readings, and waveforms. Depending on the analyte of interest, the signal may measure different things like voltage, amperage, or resistance.

Since the epoc® system is a device that is used in patient care, the test cards must undergo comprehensive quality control testing before being sent out for use. This testing involves taking a subset of cards from a lot and making sure a low number of cards yield unsuccessful results when measuring the different analytes. If too many cards yield unsuccessful results, then the rest of the lot cannot be shipped for customer use. Beyond the necessity of testing to make sure the number of unsuccessful readings in a lot is sufficiently low, it is also important to understand the different ways in which the system is failing in order to make appropriate changes to the manufacturing process. One significant challenge is that there are many different reasons the results can be unsuccessful, and we don't necessarily know all of their root causes. Furthermore, new issues may occur, and old ones may disappear over time as changes in manufacturing are made. As a result, it is useful to have a way of automatically characterizing the readings into different groups so that their occurrence over time can be tracked and tied back to manufacturing.

One cause of unsuccessful readings that has been investigated occurs when an electrical circuit disruption (ECD) occurs during the testing of the card. With most analytes, the ECD errors are easy to identify and track. However, for the analyte that we are working with, this is not the case as the readings contain a significant amount of noise. This means that while we have access to some readings that have been labelled

as ECD errors, there is no guarantee that more do not exist in the dataset which have not been labelled. Furthermore, due to time constraints, not all of the unsuccessful readings are manually assessed.

As a result, the goal of this project was to use machine learning techniques on the unsuccessful readings for a particular analyte to see if they could be automatically clustered into different groups. By finding these different groups among the unsuccessful readings, future work will be able to analyze them and try to uncover underlying root causes and make suitable alterations to manufacturing. Having access to the labels for the ECD errors, which are one of the root causes of failure, allowed us to evaluate whether or not our unsupervised pipelines were successful in forming clusters that could be tied back to potential anomalies. Furthermore, isolating the occurrence of ECD errors will allow them to be tracked over time.

By not using the ECD error labels when training our models, we avoided an over-reliance on them since we knew that there are other exemplars in the dataset that have not been labelled, and that there are other unknown categories of errors. The main research objectives explored in this paper are as follows:

1. Develop machine learning pipelines to perform unsupervised learning to find clusters among the unsuccessful readings for our analyte of interest.
2. Determine whether a particular pipeline leads to clusters with better separation between the labelled ECD errors and the other unsuccessful readings.
3. Describe and compare the clusters that are formed by different pipelines.

During this project, we took an iterative approach to develop machine learning pipelines to solve this problem, wherein we tried many combinations of steps to extract information from the time series and then cluster it. In the end, we found four pipelines for which there were clusters containing a large number of ECD errors and relatively few other unlabelled unsuccessful readings. Throughout the rest of this paper, we will discuss previous work that has attempted to cluster time-series data, then outline the techniques that yielded the most promising pipelines. Finally, we analyze the results from these pipelines, discuss implications, and make suggestions for future work.

2. Background

While not a lot of previous work has looked specifically at blood analysis systems, there is a large body of literature describing time-series analysis and anomaly detection in various domains including electrocardiography and electroencephalography, which are biosensors that measure heart activity and brain activity over time, respectively [4]–[6]. However, while our goal is to perform a form of anomaly detection, it does not fit the same framework as most biosensor anomaly detection problems. Traditionally, most anomaly detection problems are concerned with finding a few anomalies among many ‘normal’ data points

[4], [6]. For instance, this could mean detecting an irregular heartbeat among plenty of regular ones. In contrast, we are looking at data where we know that all the observations are anomalous, and we want to find categories of different anomalous behaviours in the readings from the sensor we are working with. Given the lack of specific literature approaching this problem in our domain, we will focus more generally in this section on describing approaches to time series clustering.

When algorithms are used to find groups in data such that observations within a group are more similar to each other than to observations in other groups, this is referred to as clustering. Because the observations aren't grouped based on any pre-defined labels but purely based on similarity, clustering is an unsupervised machine learning technique [7]. A more specific form of clustering is time-series clustering, where each observation consists of measurements that are taken sequentially over time. Since each time point can be considered a feature, time series are naturally a high-dimensional form of data [7], [8]. Given this high dimensionality and time-dependency, time-series clustering does differ a bit from general clustering. Within the literature, there are two broad categories of time-series clustering – whole time-series clustering and subsequence clustering [7], [9]. In subsequence clustering, a single time series is examined, and groups are formed consisting of subsequences within that time series. This is useful when the goal is to find abnormal behaviour or other characteristics within a series. In contrast, whole time-series clustering is when a dataset consisting of multiple time series is clustered so that each cluster is a grouping of multiple different time series together [7]. Since we are trying to find groups between time-series rather than within them, we will focus on whole time-series clustering.

Within whole time-series clustering, approaches can be further subdivided into two categories – shape-based and feature-based [7]–[9]. In shape-based approaches, the signals of the time-series themselves are used for computing distances and clustering, although these may be pre-processed to reduce noise. As a result, clusters tend to end up consisting of readings that look similar to each other. Conversely, in feature-based time series clustering, features are first extracted from the time-series and distances between observations are computed using those features. The process of extracting features is quite varied and can range from something as simple as taking the mean and the standard deviation to feeding the time-series through special neural networks that learn what elements are important in describing the time-series[7]–[9]. Because of the exploratory nature of this project, we experimented with both shape-based clustering, feature-based clustering, and a combination of the two.

Regardless of whether shape-based or feature-based clustering is being performed, the process of reducing noise in the original signal is a critical step [8]. In signal processing, improving the signal-to-noise ratio is a key concept. However, in practice, it can be challenging when you don't know what is signal and what is

noise. In general, there are two ways of reducing noise in a signal – smoothing and filtering. Filtering techniques are usually based on Fourier decompositions, in which the signal is transformed into the frequency domain. Then particular frequencies can be removed before the signal is transformed back into the time domain. This means the frequencies to remove have to be defined ahead of time [10]. Alternatively, smoothing techniques can be used. These generally work with a sliding window that moves across the time-series and adjusts the data to get rid of abrupt changes. These can range in complexity and include processes like adjusting points according to the moving average or fitting a polynomial in the window. Typically, smoothing techniques reduce high-frequency noise, but exactly which frequencies to reduce are not specified [11]. More details on these techniques are discussed in the methods section.

If clustering is being performed with a shape-based method, then one challenge is choosing an appropriate distance measure. For instance, when time-series are of different lengths or are out of phase, traditional distance measures are insufficient for describing the similarity of the observations. Since describing the similarity (or distance) between two observations is a critical part of most clustering algorithms, alternatives need to be considered.

Traditionally, Euclidean distance is one of the most common distance measures used for clustering. Geometrically, it finds the length of a straight line connecting two points in n-dimensional space [12]. For this to work for time-series, all the series must be of the same length, t, and then the distance between two time-series is the square root of the sum of the difference between the series at each time point squared. In other words,

$$d_{euclidean} = \sqrt{\sum_{i=1}^{t} (ts1_t - ts2_t)^2}$$

where ts1 and ts2 are both time-series with t time points. Evidently, this distance measure falls apart when working with time-series of different lengths. It will also fail when two time-series are out of phase. For example, in Figure 1(a), the two waveforms would be considered fairly different according to Euclidean distance even though they have the same shape, just starting at different points in time.

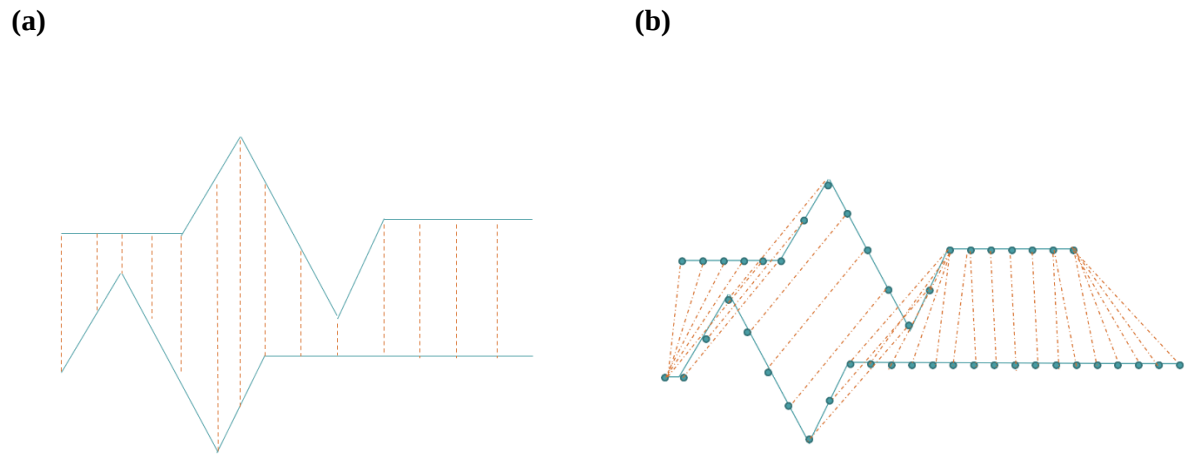


Figure 1. (a) An example of two very similar waveforms that would be considered far apart based on Euclidean distance but have the same shape, just out of phase with each other. (b) An example of how two waveforms might be aligned by DTW when computing the distance between them. Notice how peaks and troughs are lined up by allowing some points in one waveform to map to multiple points in the other.

An alternative to Euclidean distance used when working with time-series data is dynamic time warping (DTW). In this approach, the similarity of two waveforms can be determined even if they are of different lengths. The algorithm essentially finds the best way to match points up between waveforms to align the peaks and troughs (Figure 1(b)). By allowing one point to be matched with many points in the other time-series, the lengths of the two series are allowed to differ and they are also able to be out of phase with each other [13]. However, one of the major drawbacks of DTW is that it is computationally expensive to optimize the alignment of the two waveforms for the distance computation [14].

Another way that the distance between time-series can be computed is through cross-correlation. Similar to DTW, cross-correlation is a similarity measure that can be used with waveforms of different lengths [15].

Calculating the cross-correlation coefficients requires one of the sequences to be kept immobile while the other one is shifted past the first sequence, one time point at a time. At each shift, also referred to as lags, a different cross-correlation coefficient is calculated. A coefficient represents "the sum of the product of the newly lined up terms in the series" [16]. The goal is to find the lag where the cross-correlation coefficient is the highest, meaning where the two sequences are most similar [17]. In the example provided in Figure 2, we notice that after shifting by a lag of 2, the two waveforms are aligned in such a way that their similarity is maximized. This means that the cross-correlation coefficient will be highest at a lag of 2.

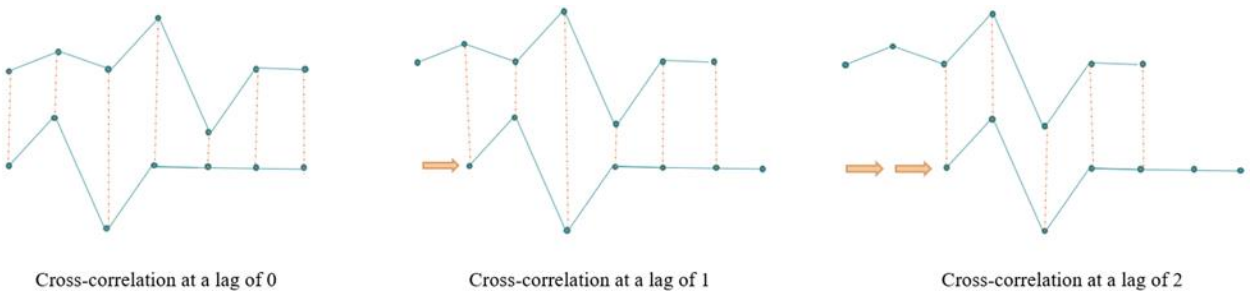


Figure 2. An example of how two waveforms might be aligned using cross-correlation. Notice that at a lag of 2, the two waveforms are aligned in such a way that their similarity is maximized. Thus, the cross-correlation coefficient will be highest at a lag of 2.

Once the largest cross-correlation coefficient has been identified, it is divided by a specific term for normalization purposes. This term varies depending on the domain or the application [17].

One of the main advantages of using cross-correlation is that, unlike Euclidean distance, it is shift-invariant [17]. This means that the measure remains adequate even when the waveforms are out of phase with each other. Furthermore, it can also be shown that it is less computationally expensive than dynamic time warping [17].

After performing noise reduction and/or feature extraction, the final stage in time-series clustering is to choose an appropriate clustering algorithm. There are many different options, and all have their advantages, disadvantages, and assumptions. Two of the most common ones that have been used for time-series data in the past include K-Means and Hierarchical Clustering [7]. Both methods will be discussed in detail in the methods section, along with Gaussian Mixture Modelling, which allows more flexible cluster shapes than K-means clustering [18]. Another algorithm developed specifically for use on time-series clustering called K-Shape which uses cross-correlation for distance computations will also be discussed [17].

As a result of all this previous work looking at time-series clustering, we decided to take an iterative approach to cluster the unsuccessful readings in hopes of isolating ECD errors and describing other modes of failure. Through these iterations, we tried different combinations and sequences of noise reduction, feature extraction, and clustering algorithms. These approaches will be further elucidated below.

3. Data

As part of this project, we were given two distinct datasets:

1. **Raw waveforms**: CSV files describing the raw time-series where each column represents a time point and each row represents a specific reading.

2. **Aggregate predictors**: CSV files containing a series of predictors which had been calculated from the time-series data. For example, metrics like slope, mean signal and noise for several different time intervals are included. For confidentiality, these are referred to through the report as AggPred1 – AggPred31.

These two datasets can be joined using their unique test identifiers. Figure 3 shows a sample of these two datasets for two different readings. In (a) we have the raw time-series data whereas in (b) a subset of the predictors is shown.

(a)

TestId	0.2	0.4	···	300
10695831	-1.85E-10	-3.07E-11	···	-8.65E-12
10695986	-1.56E-09	1.94E-10	···	8.21E-11

(b)

TestID	AggPred1	AggPred2	AggPred3	AggPred4
10695831	0.001747	1.21E-05	0.000806	-5E-05
10695986	0.00276	-2.42E-05	0.000854	-4.1E-05

Figure 3. Sample of the two distinct datasets we used. In (a), the value of the signal at each time point for two different readings is shown. In (b), a subset of the available predictors for the same two readings is shown.

Each of these datasets is comprised of three different types of readings :

1. **Successful readings**: Readings that pass the quality control testing. The raw time-series of these readings are never manually verified by an operator.
2. **Unsuccessful readings**: Readings that fail during the quality control testing. Only a subset of the raw time-series from these readings are manually verified by an operator and labelled as certain anomalies.
    a. **ECD errors**: These readings correspond to a subset of the unsuccessful readings. They have been labelled by the operator and represent the root cause of the error resulting from insufficient electrical contact between the card and the reader.

Since the focus of this project was to identify the ways in which the epoc® systems fails, we only took interest in the unsuccessful readings (which include the ECD errors). Figure 4 shows the raw time-series for the unsuccessful readings that were not labelled as ECD errors and for those that were.

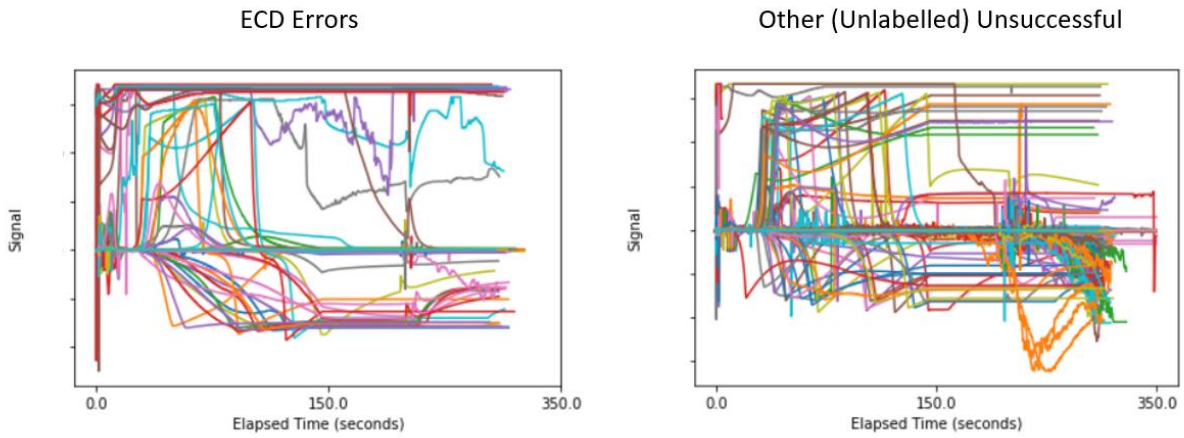


Figure 4. Raw waveforms for readings that were labelled ECD errors (left) and other unlabelled unsuccessful readings (right). Y-Axis scales removed for confidentiality.

In total, we had access to 10,090 unsuccessful readings of which 386 were labelled as ECD errors. The ECD readings were further subdivided into three categories :

1. **Original ECD errors**: Accounted for 246 of the 386 ECD errors. These were labelled during quality control testing by an operator.
2. **Synthetic ECD errors**: Accounted for 82 of the 386 ECD errors. These readings were intentionally created for this project to make it easier to train our machine learning models.
3. **Contaminated ECD errors**: Accounted for 58 of the 386 ECD errors. These readings were a by-product of the synthetic ECD errors from the residue left behind in the reader.

Figure 5 shows a diagram of the layout of our data. We know that the ECD errors are a subset of the unsuccessful readings, but we are unsure of how many other ways the system fails.

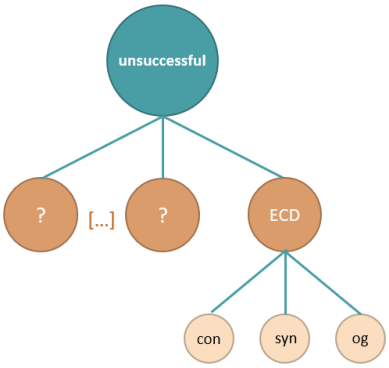


Figure 5. Diagram of the layout of the data that was used. We know that the ECD errors are a subset of the unsuccessful readings, but we are unsure of how many other ways the system fails.

Pre-processing this data represented a large portion of the work that had to be done in this project. Therefore, we dedicated section 4.1 to explain the various steps that were taken to format it effectively.

4. Methods

Due to the exploratory nature of this project, an iterative approach was taken to address the research questions. This approach consisted of experimenting with various pre-processing steps and clustering algorithms (Figure 6). As such, different pre-processing steps and algorithms were combined in numerous ways, leading to the formation of multiple pipelines.

Figure 6. Flow chart depicting the iterative approach taken to address the research questions.

All of the attempted pipelines in the first iteration led to unsuccessful results. A detailed summary of those pipelines can be found in Table 1 of Appendix A. We believe that the lack of results was mainly caused by insufficient pre-processing of the raw waveforms. Refer to Appendix B for further explanations of these possible causes.

Due to the lack of promising results in the first iteration, we drastically changed our pre-processing steps for the second iteration. All of the pipelines obtained during this second iteration are listed in Table 2 of Appendix A.

During iteration 2, we obtained four promising pipelines, meaning they were somewhat successful in clustering ECD errors from other unsuccessful readings. The pre-processing steps applied were common to all of these pipelines, but different feature extraction and clustering techniques were used (Table 3). A detailed description of the tools that were used in pipelines other than the ones listed in Table 3 can be found in Appendix C.

Table 3. Summary of the pipelines that yielded promising results. All of them were found using the same pre-processing steps during iteration 2.

Pipeline	Aggregate Predictors Used	Feature Extraction	Clustering
1	AggPred10, AggPred18, AggPred24	None	K-Means (EUC)
2	AggPred10, AggPred8, AggPred11, AggPred12, AggPred24, AggPred22, AggPred25, AggPred26,	None	K-Shape
3	List of 22 predictors	Autoencoder + PCA	GMM
4	List of 22 predictors	None	K-Shape
List of 22 predictors :	AggPred9, AggPred10, AggPred8, AggPred11, AggPred12, AggPred13, AggPred23, AggPred22, AggPred25, AggPred14, AggPred1, AggPred26, AggPred18, AggPred17, AggPred20, AggPred21, AggPred22, AggPred1, AggPred6, AggPred7, AggPred2, AggPred3, AggPred4		

In this section, we will focus on explaining the techniques that were used for the pipelines that yielded promising outcomes. To start, we will discuss the common pre-processing steps that were taken. Then, we will successively describe each pipeline from Table 3.

4.1 Pre-processing

As was previously mentioned, the four pipelines from Table 3 were obtained from the same pre-processed data. It is important to note that no pre-processing had to be made to the dataset containing the aggregate predictors. Regarding the raw waveforms, they contained information that could be made more meaningful by applying certain pre-processing steps to allow the algorithms to learn better.

Below is a list of the steps that were taken to pre-process the raw time-series.

1. Zeroing with respect to Sample Detection.
2. Removing the irrelevant period at the beginning of the waveform.

3. Normalizing.
4. Reducing the noise.

The code to run these pre-processing steps can be found in the *preprocessing.ipynb* notebook.

4.1.1 Zeroing with respect to Sample Detection

The first thing we did was set the time of sample detection to zero. The epoc® system initially pushes calibration fluid over the sensors with known concentrations of analytes, allows the waveforms to stabilize, and then pushes the sample fluid over the sensors for measurement. When this happens, there is usually a big peak in the signal resulting from high resistance caused by the air bubble that precedes the sample when it is pushed across the sensors. Hence, the sample detection time is easy to compute, and it gives a semantically consistent time point for alignment across all the readings. Some of the unsuccessful readings did not have a sample detection time, so these readings were excluded from the analysis. None of these were labelled as ECD errors.

4.1.2 Removing the Irrelevant Period of the Waveform

Once the readings were aligned by setting the sample detection time to zero, the next step was to get rid of the beginning of the waveform which has been confirmed by subject matter experts to be irrelevant. Throughout this time, the signals are very noisy and do not provide meaningful information for the analysis. As a result, we chose to remove this period before proceeding.

We chose to define the irrelevant period by looking at several example traces and removing the consistently noisy time frame. An example reading with the noisy beginning of the waveform and sample detection time labelled is shown in Figure 7.

We determined that sub-setting the readings from 30 seconds before sample detect time to 40 seconds after sample detect time effectively removed the noisy period. Furthermore, it ensured that all the readings had the same length. The few readings for which there was not a complete signal in this time range were excluded from further analysis. An example of this subset is also shown in Figure 7.

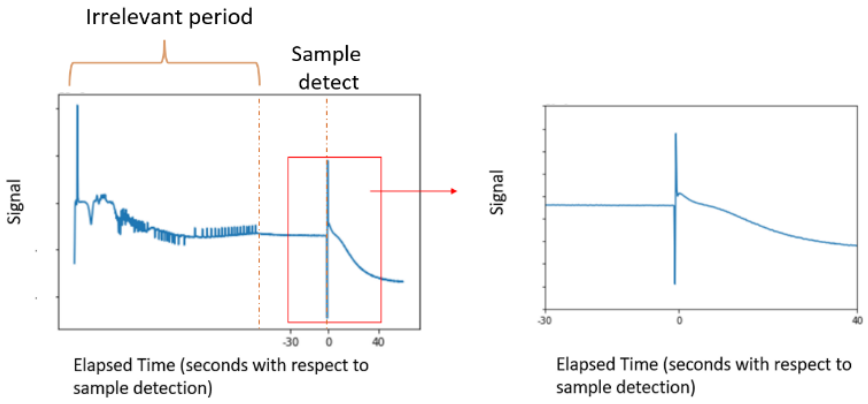


Figure 7. An example of the removal of the unnecessary time segment and ensuring all readings are of the same length by sub-setting from 30 seconds before sample detection to 40 seconds after sample detection.

4.1.3 Normalizing

Once the irrelevant time segment was removed, we normalized the waveforms.

Normalization. This is the process of taking a time-series and scaling it so that its values are all between 0 and 1. The formula to do this is :

$$ts_{standardized} = \left\{ \frac{ts_i - min(ts)}{max(ts) - \min(ts)} \; for\ i\ in\ the\ length\ of\ ts \right\}$$

After normalization, the overall changes and relative shape of the waveform are retained. Only the range of values is altered to lie within 0 and 1 (see Figure 8).

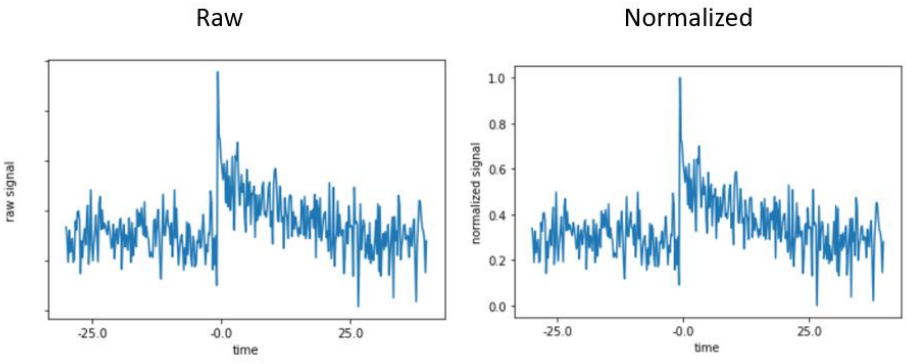


Figure 8. Demonstration that normalization does not change the shape of the waveforms, it simply scales the values so that they lie between 0 and 1.

Since this transforms the waveforms to fall within 0 and 1, it aids in visualization when we try to understand differences between the waveforms. Moreover, this helps prevent clustering algorithms from clustering based on the amplitude of the signals [8].

4.1.4 Noise Reduction

The analyte we were working with was very noisy. Thus, to extract the underlying signal, we tested various noise reduction techniques:

1. Moving Average Smoothing.
2. Savitzky-Golay Filtering.
3. Bartlett Window Convolution Smoothing.

We chose to go with the Bartlett Window Convolution Smoothing technique with a window of length 50 as it yielded what visually appeared to be the least noisy signal (see Figure 9). Refer to Appendix D for a description of the other two techniques.

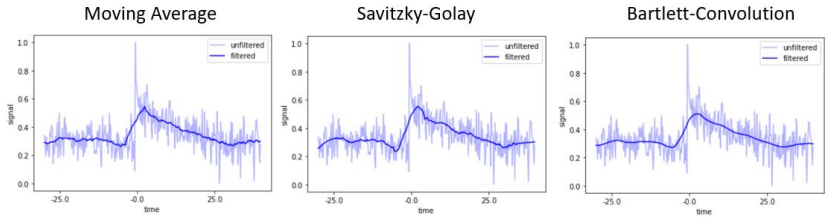


Figure 9. Illustration of the three different smoothing methods that were experimented with.

The *tsmoothie* Python package was used to perform the convolution [19]. While this smoothing technique is very similar to a moving average, using a Bartlett window is beneficial because the shape of the window means that less emphasis is placed on the edges, so there is a lower risk of distortion [20].

4.2 Describing the pipelines

In this section, we will individually describe the 4 most promising pipelines from Table 3.

4.2.1 Pipeline 1

The first pipeline that resulted in one of the clusters containing a large proportion of the total ECD errors was found by applying K-Means clustering on a dataset comprised of both the pre-processed waveforms and three of the aggregate predictors: AggPred10, AggPred19 and AggPred24. These features respectively represent the mean of the time-series for the calibration, post, and sample windows. Refer to section 1.3 of Appendix C for more details on these windows.

These three aggregate predictors were chosen because we knew that ECD errors should have a mean of approximately 0 in all three windows whereas this was not necessarily true for other unsuccessful readings. As such, it made sense to utilize this domain knowledge to try and distinguish the ECD errors from the rest of the unsuccessful readings.

We decided to cluster based on a combination of the aggregate predictors and the time-series data after many failed attempts during iteration 1 where only the waveforms were considered (Table 1 of Appendix A). Furthermore, we knew that using only the aggregate predictors had been tried in the past by Siemens Healthineers and that it was also not successful.

K-Means clustering requires the user to specify the number of clusters to find (K) and is suitable only for numerical data [21]. While we did not know exactly how many different modes of failure there were, we knew that there were around thirty different error codes that could be returned by the system. Hence, for this pipeline, we used K = 30 with Euclidean distance. Using Dynamic Time Warping as the distance measure was also tried but was computationally more expensive yet yielded similar results.

An in-depth description of K-Means clustering can be found in section 1 of Appendix E and the code to run this pipeline is included in the *KMeansClustering.ipynb* notebook.

4.2.2 Pipeline 2

After obtaining promising results with Pipeline 1, we decided to look into K-Shape; a similar clustering algorithm to K-Means, except it is tailored specifically for shape-based whole time-series clustering [17]. Furthermore, now that we knew that combining the information contained in the waveforms and the aggregate predictors could yield positive results, we decided to perform feature selection using a random forest classifier to identify which of the aggregate predictors were most relevant.

To do this, we used the estimated variable importance from the classifier. It allowed us to assess which aggregate predictors were most significant when trying to predict whether or not a reading was a ECD error [22]. For more details on how random forests can be used for feature selection refer to section 2 of Appendix E. Also, a complete implementation of this technique can be found in the *RandomForest.ipynb* notebook.

The 12 most important predictors according to this classifier were chosen to be:

AggPred10, AggPred8, AggPred11, AggPred12, AggPred24, AggPred22, AggPred25, AggPred26, AggPred19, AggPred17, AggPred20, AggPred21.

Once the pre-processed time-series and the above predictors were combined in a single dataset, we were able to apply the K-Shape algorithm to cluster the readings. K-Shape differs from K-Means in two regards [17] :

1. It uses normalized cross-correlation as a distance metric rather than Euclidean or DTW.
2. The centroids are calculated based on an optimization problem rather than computing the mean at each time point for all the readings in a cluster.

The steps taken by the algorithm are exactly the same as the ones taken by the K-Means algorithm (refer to section 1 of Appendix E for the K-Means algorithm). For the same reason mentioned in section 4.2.1 (the number of error codes that can be returned by the system), we selected 30 as the number of clusters to form. We also tried running the algorithm with various other numbers of clusters ranging from 20 to 35 but found the most interesting results when K = 30.

For more details on the distance metric, how the centroids are measured, and why it is important to both normalize and standardize the readings for this algorithm to be effective, refer to section 3 of Appendix E. This pipeline was implemented in the *KShapeClustering.ipynb* notebook.

4.2.3 Pipeline 3

For this third pipeline, we decided to take a slightly different approach to clustering. Because we were working with time-series data which is intrinsically high-dimensional, we thought using a feature extraction technique on the time-series prior to clustering might yield promising results.

Feature extraction consists of creating new features from the ones found in the original dataset. The hope is that these new features are more informative than the ones originally present and can thus lead to better modelling results [23]. Furthermore, feature extraction often leads to dimension reduction since the new features contain more information than the previous ones did [23].

For feature extraction, we first used a type of artificial neural network called an autoencoder. Briefly, an autoencoder is comprised of 2 distinct structures [24]:

1. An encoder that compresses the initial dataset into a smaller dimensional space.
2. A decoder that re-maps the output of the encoder to reconstruct the initial dataset using the features learned from the encoder.

Since our goal was to reduce the dimensional space of the time-series waveforms, the encoder portion of this architecture was used (see Figure 10). An in-depth description of how we built the autoencoder is

provided in section 4 of Appendix E. The complete implementation is available in the *autoencoder.ipynb* notebook.

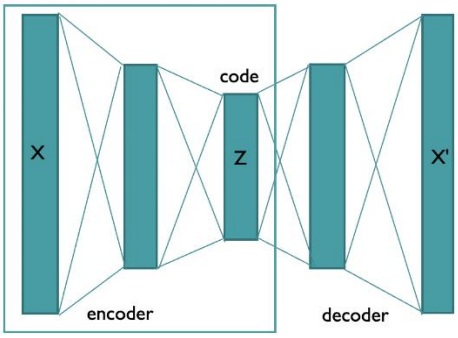


Figure 10. Illustrates the basic architecture of an autoencoder. There is a box around the encoder portion since that is the part that we used for feature extraction.

After obtaining the new set of features from the encoder, we applied another dimension reduction technique called principal component analysis (PCA). The resulting features from this algorithm are referred to as principal components. We kept the first three components to retain 95% of the variance contained in the output from the encoder. For more details on how this technique works and how it was used in the context of this project, refer to section 5 of Appendix E. A detailed walk-through of how to use PCA to obtain visualizations in a lower dimensional space is also provided in the *pca.ipynb* notebook.

Now that we had 3 new features that described the time-series data, we once again combined this information with that contained in the aggregate predictors. Initially, we only used the 12 most important predictors from the random forest classifier like in pipeline 2, but this did not yield good results. This may be explained by the fact that applying two methods of feature extraction resulted in a loss of valuable information and so to make up for that loss, more of the aggregate predictors had to be used.

As a result, we next used all of the aggregate predictors except the ones that had no tie to the actual waveforms such as the lot number and the date when the reading was produced. This corresponded to 22 predictors. The list of predictors can be found in table 3.

Having the new dataset comprised of 3 principal components for the time-series data and 22 aggregate predictors, we clustered the readings using Gaussian Mixture Modeling (GMM). GMM is a technique which assumes that each reading belongs to a cluster with a distinct probability associated with it [25]. The user must specify the number of clusters to form prior to running the algorithm. To assist us in making this decision, we used a plot of the Bayesian Information Criterion (BIC) to evaluate the performance of our model for a range of different numbers of clusters (Figure 11).

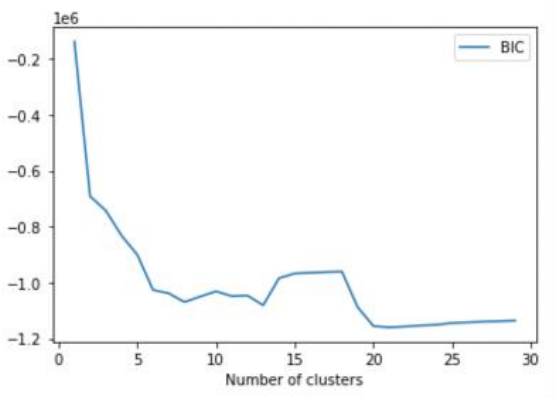


Figure 11. Shows the Bayesian Information Criterion for models with different numbers of clusters used to determine the optimal number of clusters.

In the graph, we saw that the BIC started to plateau with around 5 clusters, so we initially tried to perform GMM with 5 clusters, but we noticed that a lot of different waveform shapes were getting clustered together. As a result, we tried clustering with 20 clusters, since there was another dip in the BIC at that point, and we achieved some promising results.

More details regarding GMM and the BIC criterion are provided in section 6 of Appendix E. GMM was implemented in the *autoencoder.ipynb* notebook.

4.2.4 Pipeline 4

Seeing that using all of the 22 aggregate predictors yielded promising results in Pipeline 3, we decided to run the K-Shape algorithm again. This time, instead of using only the 12 most important predictors according to the random forest classifier as we did for Pipeline 2, we used these 22 aggregate predictors.

The only difference between this pipeline and Pipeline 2 is the aggregate predictors that were used. For the same reason explained in Pipeline 2, the number of clusters chosen was 30. This pipeline is also included in the *KShapeClustering.ipynb* notebook.

5. Results and Discussion

As we have mentioned, the main goals of this project were :

1. To develop machine learning pipelines to perform unsupervised learning to find clusters among the unsuccessful readings for our analyte of interest.
2. To find whether a particular pipeline(s) led to the formation of clusters with better separation between labelled ECD errors and other unlabelled unsuccessful readings.
3. To describe and compare the resulting clusters from our pipelines.

Throughout this project, we developed many different pipelines to achieve these goals but found only four satisfactory ones. In this section, we will discuss the results and implications of these pipelines and then spend some time more broadly discussing the implications of this project as a whole.

Pipeline 1. In this pipeline, we found 30 clusters, one of which contained 100 ECD errors and only 50 other unsuccessful readings (Figure 12). Of these, 35 were later manually identified as unlabelled ECD errors, while the remaining 15 were all caused by an issue with the sensor of another analyte. Although this was not the most promising of our developed pipelines, since most of the other ECD errors were in a cluster with around 2000 unlabelled unsuccessful readings, it was the first indication that we might be able to pull ECD errors out from the other unsuccessful readings.

Finding that most of the unsuccessful readings that were clustering in with these 100 ECD errors were actually unlabelled ECD errors themselves was also pivotal in the realization that these unsupervised pipelines could be useful for identifying more of the unlabelled ECD errors.

Looking at Figure 12, we can see that readings with a very characteristic shape are getting clustered together. Thus, we can say with some degree of certainty that this shape is characteristic of ECD errors. In the future, not only will these results help pull unlabelled ECD errors out from other unsuccessful readings, but it may help technicians to know that this is a shape to look for in normalized and smoothed waveforms when trying to diagnose ECD errors visually.

Overall, a lot of the other clusters contained readings with similar shapes. For instance, one cluster with 254 unlabelled unsuccessful readings and only 2 ECD errors seems to consist of waveforms that are flat before sample injection, and then follow a smooth decrease after (Figure 13). In contrast, another cluster with 127 unsuccessful readings and only 2 ECDs followed a similar pattern, but instead of a smooth decline, there was a noisy decline, which could be indicative of a difference in root cause of the error (Figure 14).

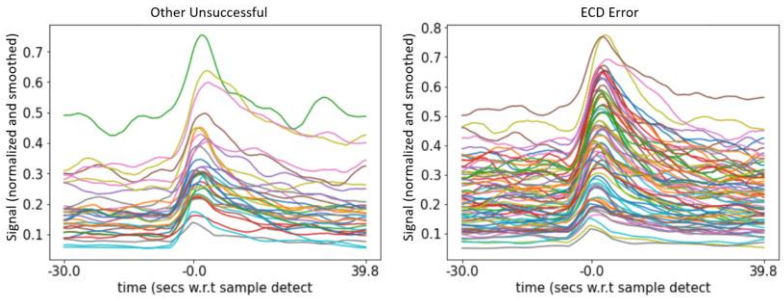


Figure 12. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 1 containing the highest ratio of ECD errors to other unlabelled unsuccessful readings.

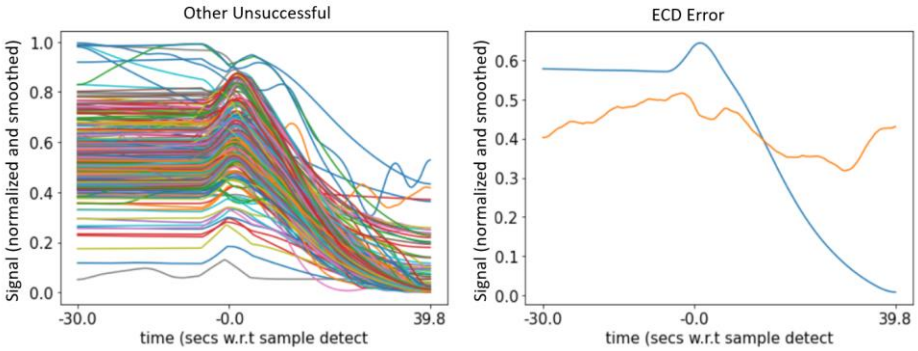


Figure 13. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 1 containing mainly unlabelled unsuccessful readings.

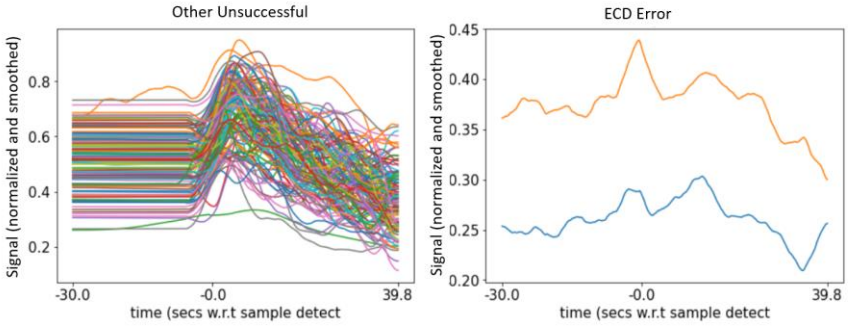


Figure 14. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 1 containing mainly unlabelled unsuccessful readings.

Pipeline 2. Knowing that there was a possibility that we would be able to get even better clusters, we moved on to developing the second pipeline, as described in section 4.2.2. Again, we got 30 clusters, two of which contained high ratios of ECD errors to other unsuccessful readings. One of these contained 177 ECD errors and 280 unlabelled unsuccessful readings. The pre-processed readings in this cluster were all relatively flat (Figure 15). The other cluster with a high concentration of ECD errors contained 160 ECD errors and only 22 unsuccessful readings. These 22 readings were reassessed manually and only 2 were found to be unidentified ECD errors while the remaining 20 were identified as being a result of an issue with the sensor of another analyte. The readings in this cluster followed the same shape as previously seen in pipeline 1, reaffirming that this is probably a characteristic shape for ECD errors (Figure 16).

Overall, these two clusters were able to account for around 80% of the ECD errors. Because we do not know how many of the unsuccessful readings in the cluster with 280 unsuccessful readings are unlabelled ECDs, it is challenging to assess the purity of this cluster. We again saw different shapes clustering together

in many of the other clusters containing mainly unsuccessful readings, indicative of different characteristic waveforms underlying unsuccessful readings (Figure 17).

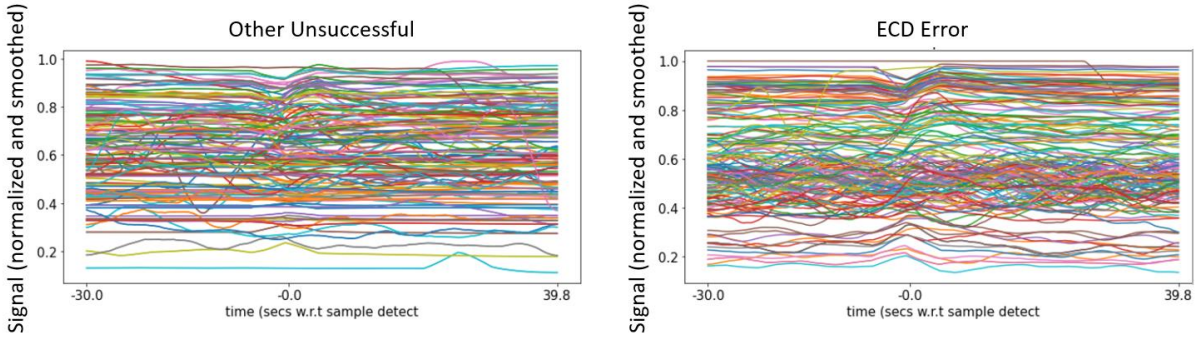


Figure 15. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 2 containing the highest number of ECD errors.

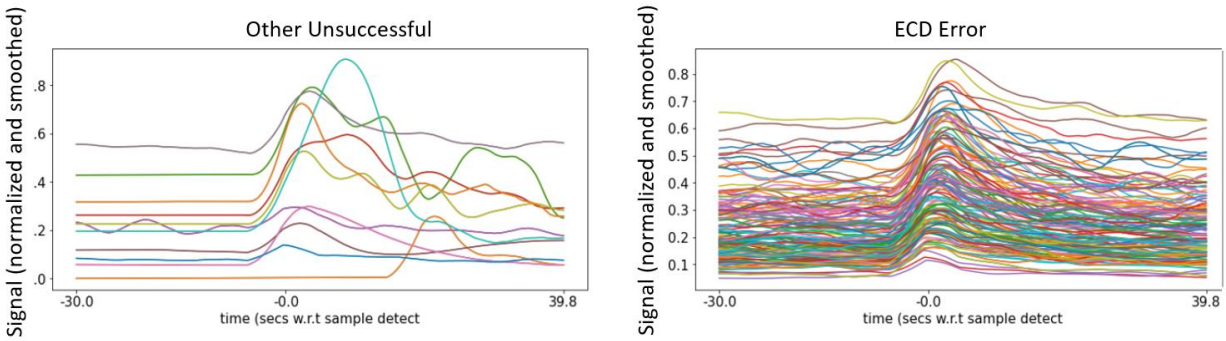


Figure 16. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 2 containing the second highest number of ECD errors.

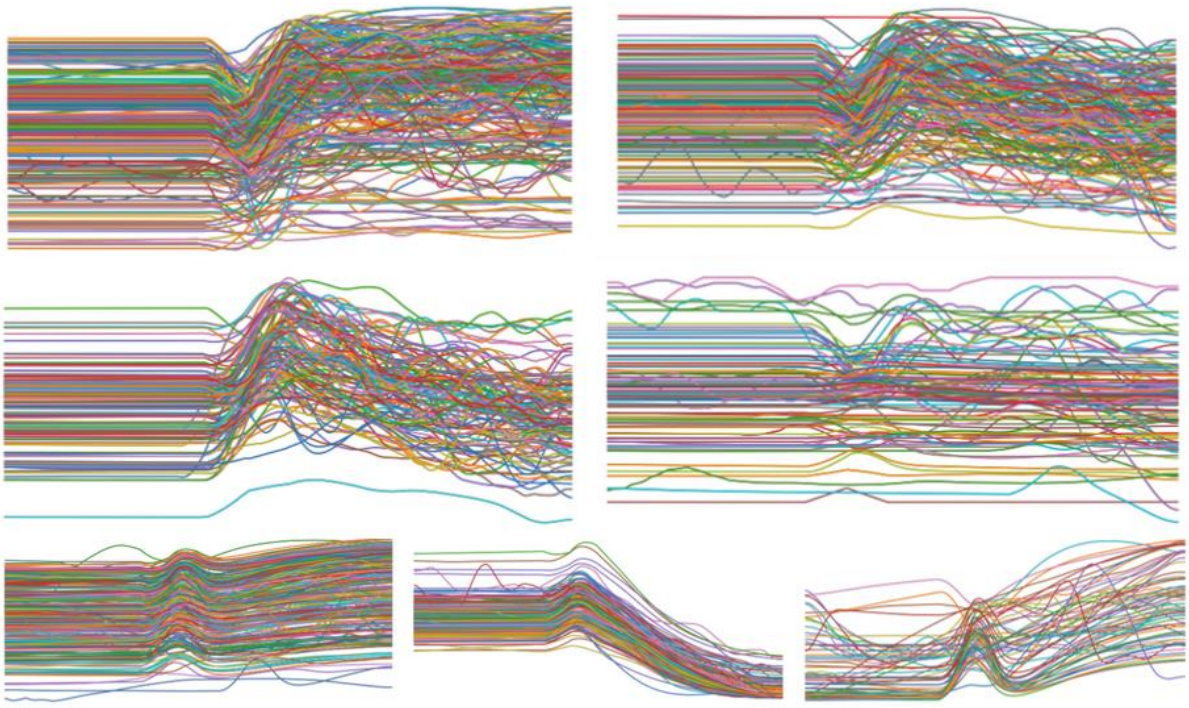


Figure 17. Examples of some of the other shapes that were clustered together in pipeline 2 where the pipelines mainly only contained unlabelled unsuccessful readings.

Pipeline 3. By using an autoencoder for feature extraction, our third pipeline resulted in 20 clusters, one of which contained 303 ECD errors and only 100 other unsuccessful readings. This means that at least 75% of the readings in this cluster are ECD errors since some of the unsuccessful readings may also be unlabelled ECD errors. Furthermore, this one cluster contains 72% of all the ECD errors. As a result, this pipeline is likely the most promising in terms of isolating the ECD errors. We can see that this cluster contains both of the shapes associated with ECD errors identified in pipeline 2, as expected (Figure 18). The unsuccessful readings in this cluster have yet to be manually reassessed, but it would be very interesting to see if they followed the same pattern as in Pipeline 1 and Pipeline 2, where all the unsuccessful readings in the cluster were either unlabelled ECD errors or an issue with the sensor of another analyte.

While this pipeline was very successful at isolating ECD errors, some of the other clusters were questionable and only contained a handful of readings. This makes it much more challenging to draw inferences from the other types of unsuccessful readings in the dataset as they did not cluster together into different groups very well. Future work would benefit from experimenting with using a different number of clusters to try and avoid having clusters with very few observations in them or try a clustering algorithm

that unlike GMM, does not assume that normal distributions underly the data since this assumption may not have been met.

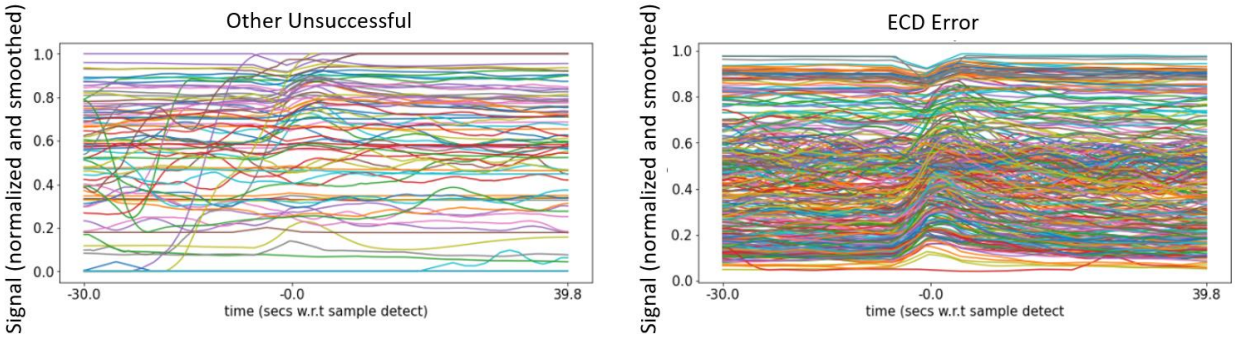


Figure 18. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 3 containing 72% of the total ECD errors.

Pipeline 4. In our final promising pipeline, we found a cluster with 343 ECD errors, accounting for 81% of the total ECD errors. However, there were also 570 additional unlabelled unsuccessful readings, so in total the cluster was comprised of only 38% ECD errors. Looking at the traces of the readings in this cluster, the ECD errors follow the same patterns identified in pipelines 1-3, however, there appear to be a lot of other shapes getting mixed in from the other unsuccessful readings (Figure 19). While there were a few homogeneous clusters with lots of unsuccessful readings as seen in Figure 20, there were also a few clusters with only a handful of readings that did not appear to belong together very well. Future work could experiment with reducing the number of clusters to try and prevent this problem.

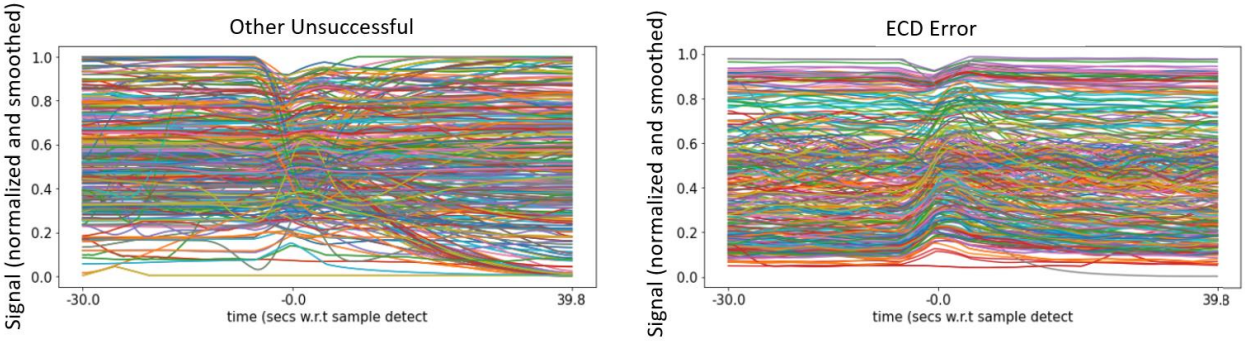


Figure 19. Overlayed traces for the ECD errors and other unsuccessful readings in the cluster from pipeline 3 containing 81% of the total ECD errors.

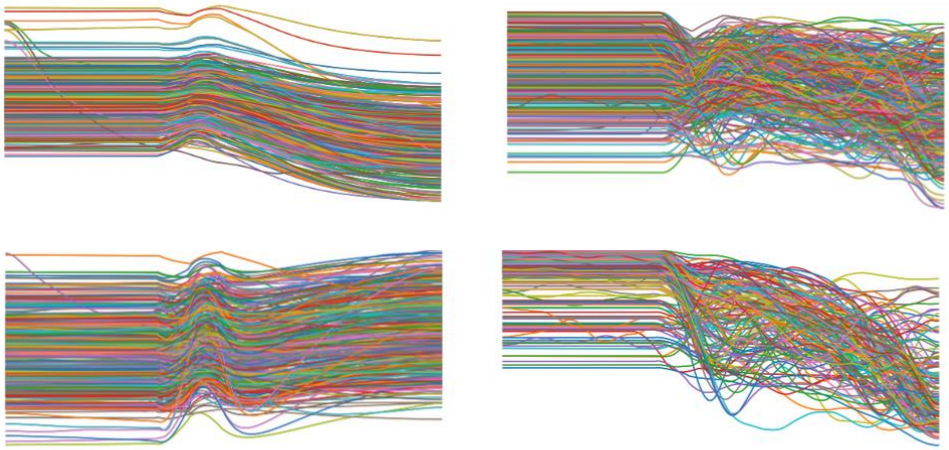


Figure 20. Examples of some of the most cohesive appearing clusters from Pipeline 4 containing almost no ECD errors.

Considering everything, which pipeline performs the best depends on interpretation. If the goal is to capture as many ECD errors as possible, then pipeline 3 may have been the most successful. However, if the goal is to characterize the different shapes of waveforms, then pipeline 2 may have been the most successful as it yielded the clearest definition between waveform shapes in different clusters. In either case, we did develop pipelines that can cluster unsuccessful readings into different groups and even found some which isolate ECD errors somewhat effectively. Furthermore, we were able to describe the readings that were put into different clusters by plotting the raw traces. We also developed a way of describing the different clusters by using histograms to describe the distributions of the aggregate predictors for the readings contained in a cluster. We found that these were hard for non-experts to interpret, so we have included further details on them in Appendix F. Appendix F also describes the cluster evaluation process in-depth.

Moving forward, these pipelines can be further developed and used to predict which unsuccessful readings are ECD errors to obtain a more accurate count of how many appear over time. In doing so, it will be possible to link any increases or decreases with relevant changes in manufacturing and make appropriate changes to decrease their frequency. Similarly, the different shapes seen in different clusters can be tracked over time to see how their frequencies of appearance change. While we do not currently know which kinds of failures in the system correspond to waveforms with these shapes, future work can correlate them with manufacturing changes in order to figure this out.

One additional finding from all of the pipelines that we tried is that the preprocessing steps are truly critical. Without performing noise reduction techniques and normalization we were completely unable to separate ECD errors from the other unsuccessful readings.

It is important to note that prior to clustering with pipelines 1 and 4, we selected aggregate predictors to use as features using either domain knowledge or a random forest to determine which were most useful for predicting ECD errors. We therefore expect these pipelines to perform better at clustering the ECD errors compared to other root causes of error. While it is still possible that these predictors are helpful for clustering other unknown causes of failure, it may also be the case that they would cluster more easily if an entirely different set of predictors were used.

Similarly, a potential limitation of how we iteratively designed our pipelines is how we were specifically looking for clusters with high proportions of ECD errors. Because we were not using the labels when grouping the data and were thus in the unsupervised domain [7], we decided to not use training and testing sets or worry about overfitting. However, by only selecting pipelines that led to clusters with a high proportion of ECD errors, we may have inadvertently selected models that leaned more heavily toward clustering ECD errors together and sacrificed the quality of other clusters. Overfitting is a big issue in machine learning, and generally means that results are less generalizable to new data [26]. As a result, while the pipelines that we developed are somewhat successful in isolating ECD errors, they may not do a good job at clustering other potential root causes of unsuccessful readings. Since one of our goals was to use clusters to try and identify root causes in the future, caution should be taken when interpreting results. Furthermore, future attempts could assess cluster formation using other metrics like silhouette scores, which can assess cluster quality without labels [27]. In doing so, the pipelines built may overfit less to ECD errors and generalize better to other root causes of unsuccessful readings.

Overall, we have built pipelines that help characterize all of these different waveforms seen in unsuccessful readings. In doing so, their analysis as part of the quality control process will be enriched moving forward, hopefully aiding in making intelligent design changes in the manufacturing of the epoc® blood system. While this work will aid in the automation of the quality control process, it will not eliminate the need for human experts moving forward.

6. Conclusions

Throughout this project, our main goal was to create unsupervised machine learning pipelines that characterized the unsuccessful readings for a particular analyte from the epoc® blood system into different groups. This will allow for their occurrences over time to be tracked and tied back to the root causes of failure in the manufacturing process. Having the labels for one of these causes of failure, ECD errors,

allowed us to evaluate whether or not our unsupervised pipelines were able to form clusters that could potentially be tied back to other root causes in the future.

We successfully developed four pipelines that led to clusters with a high concentration of ECD errors as well as clusters that contained waveforms with characteristic shapes. This gives us confidence that going forward, these pipelines can be useful for identifying other root causes that are still unknown as of now and thus lead to an improvement in the manufacturing process of the test cards.

These four pipelines all involved a common set of pre-processing steps including noise reduction using a convolutional smoother. Their success indicates that noise reduction is a key step in clustering noisy time-series data. We also found that combining the pre-processed time-series with other aggregate information about the waveforms like the mean of different time periods or amount of noise was helpful in finding better-defined clusters.

One of the most surprising and helpful results was discovering that many of the readings that cluster with known ECD errors are themselves unlabelled ECD errors or issues with a sensor for a related analyte. Moving forward, this means that identifying ECD errors may be easier after first diagnosing problems with the related analyte.

At the beginning of this project, we did not anticipate how challenging it would be to cluster the unsuccessful readings. It took us a very large number of attempts to find pipelines that worked, and in the end, we have discovered that using as much of the waveform as possible and reducing noise to get a clearer signal are both crucial steps. The biggest barriers we faced were staying creative, analyzing why things went wrong in previous pipelines and coming up with new things to try for future ones. Because of how long it took to find promising pipelines, we did not have much time to tune and improve upon them. As such, we have several suggestions for future work. These include:

1. Using smaller numbers of clusters for the four promising pipelines. We recognize that increasing the number of clusters leads to more homogeneous clusters as the number of readings per cluster decreases. Nevertheless, this does not necessarily mean that these clusters are of any more value in terms of inference compared to those that could be obtained from using a smaller number of clusters.
2. Smoothing the waveforms using a Savitzky-Golay filter and comparing the results to those obtained with the Bartlett Window Convolution filter. We decided to use a Bartlett Window Convolution filter rather than a Savitzky-Golay filter simply based on how smooth the waveforms looked visually. This being said, using a different type of filter might yield better results.

3. Run Pipeline 3 using only the aggregate predictors (no time-series data). In Pipeline 3, we used all the informative aggregate predictors and 3 components that contained information from the time-series data. It would be interesting to see if we obtained comparable results by using only the aggregate predictors. This would help measure how important the information contained in the time-series is.
4. Start by identifying the errors for the specific analyte that we know influences the readings of the analyte we were working with. Throughout this analysis, we found that some of the unsuccessful readings that were being clustered in the groups containing lots of ECD errors were actually ECD errors from this other analyte. This other analyte is less noisy than the one we were working with for this project. Thus, it could be useful to first filter out these anomalies from that analyte and then proceed to do the same for the noisier analyte.

APPENDIX A : SUMMARY TABLES OF PIPELINES

Table 1. Pipelines attempted during iteration 1. None yielded satisfactory results. All the pipelines were attempted with various numbers of clusters but still, none were effective at isolating ECD errors.

Time Series Pre-Processing	Windowing	Feature Extraction	Clustering	Results
1. Set sample detect time = 0	Window 1	TSFresh -> PCA	Gaussian	numerous different number of clusters ranging from 2 to 30.
1. Set sample detect time = 0 2. Remove first 150 secs 3. Standardize	Window 1	TSFresh -> PCA	Hierarchical	No separation between ECDs and unsuccessful in clusters. Tested with average, single, complete and ward linkage.
			Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.
		Autoencoder -> PCA	Hierarchical	No distinct separation between ECDs and unsuccessful readings in clusters, More unsuccessful got combined with the ECDs.
			Gaussian	Same results as above, tested it with various number of clusters ranging from 2-30 not good results.
		PCA	SOM	No separation between ECDcontact and unsuccessful after testing with various hyperparameters of SOM and different variation. However, clusters in this window were better than in windows 2 and 3.
		None	Kmeans with DTW	No separation between ECDs and unsuccessful in clusters.
		FFT	Hierarchical	No separation between ECDs and unsuccessful in clusters. Tested with average, single, complete and ward linkage.
		FFT -> TSFresh -> PCA	Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.
	Window 2	TSFresh -> PCA	Hierarchical	No separation between ECDs and unsuccessful in clusters. Tested with average, single, complete and ward linkage.
			Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.
		None	Kmeans with DTW	No separation between ECDs and unsuccessful in clusters.
			SOM	No separation between ECDcontact and unsuccessful after testing with various hyperparameters of SOM and different variation
		FFT	Hierarchical	No separation between ECDs and unsuccessful in clusters. Tested with average, single, complete and ward linkage.
		FFT -> TSFresh -> PCA	Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.
	Window 3	TSFresh -> PCA	Hierarchical	No separation between ECDs and unsuccessful in clusters.
			Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.
		None	Kmeans with DTW	No separation between ECDs and unsuccessful in clusters.
			SOM	No separation between ECDcontact and unsuccessful after testing with various hyperparameters of SOM and different variation
		FFT	Hierarchical	No separation between ECDs and unsuccessful in clusters. Tested with average, single, complete and ward linkage.
		FFT -> TSFresh -> PCA	Gaussian	No separation between ECDs and unsuccessful in clusters. Tested with numerous different number of clusters ranging from 2 to 30.

Table 2. Pipelines attempted during iteration 2. The four pipelines that yielded promising results are highlighted in orange.

Preprocessing	Aggregate Predictors Used	Feature Extraction	Clustering	Result
1. Set sample detect time = 0 2. Subset -30 secs : 40 secs 3. Normalize	None	None	SOM	The results were not better than what was obtained when normalizing and applying smoothing (convolution and savitzky-golay). However, it was suprisingly not terrible either.
1. Set sample detect time = 0 2. Subset -30 secs : 40 secs 3. Convolution smooth with bartlett window length 50	AggPred10, AggPred8, AggPred11, AggPred12, AggPred24, AggPred22, AggPred25, AggPred26, AggPred18, AggPred17, AggPred20, AggPred21	None	KShape	Not normalizing the timeseries before smoothing resulted in very bad clusters. The algorithm was not converging. No distinguishable shapes between clusters. Refer to KShapeClustering notebook for detailed explanation.
1. Set sample detect time = 0 2. Subset -30 secs : 40 secs 3. Normalize 4. Convolution smoothed with bartlett window length 50	None	None	Kmeans (EUC + DTW)	No separation between ECDs and unsuccessful in clusters.
			KShape	No separation between ECDs and unsuccessful in clusters.
			SOM	No separation between ECDcontacts and unsuccessful in clusters.
	AggPred10, AggPred18, AggPred24	None	Kmeans (EUC)	One cluster was promising - contained around 100 ECDs and 50 unsuccessful.
	AggPred10, AggPred8, AggPred11, AggPred12, AggPred24, AggPred22, AggPred25, AggPred26, AggPred18, AggPred17, AggPred20, AggPred21	None	KShape	Found one cluster with most of the ECDs (343) and approximately the same number of unsuccessful readings (376). Looking at the diagnostic plots, there seemed to be distinguishable shapes between the clusters.
		PCA on (time series + predictors)	KShape	No separation between ECDs and unsuccessful in clusters. Concerns with using PCA, more specifically standardizing each column, were identified. See KShapeClustering notebook for further details.
		(PCA on time series) + predictors	KShape	No separation between ECDs and unsuccessful in clusters. Concerns with using PCA, more specifically standardizing each column, were identified. See KShapeClustering notebook for further details.
	1. Set sample detect time = 0 2. Subset -30 secs : 40 secs 3. Normalize 4. Convolution smoothed with bartlett window length 50	None	Kshape	Found 2 clusters with high concentrations of ECDs. The first one had 177 ECDs vs 280 unsuccessful and the other had 160 ECDs and 22 unsuccessful. Looking at the diagnostic plots, there seemed to be distinguishable shapes between the clusters.
		Auto Encoder + PCA	Gaussian	Found 21 clusters. One of them contained 300 ECDs and only 100 unsuccessful readings. Diagnostic plots show clear distinguishable shape between the clusters.
	AggPred9, AggPred8, AggPred11, AggPred12, AggPred13, AggPred22, AggPred25, AggPred21, TranAggPred22,AggPred3	None	SOM	Results weren't great. This said, the sum of ECD readings from 3 different clusters was 170 versus 700 unsuccessful readings. Several clusters contained only unsuccessful readings.
		PCA	SOM	No seperation between ECDs and unsuccessful readings (2 PCs amounted to 95% variation)
1. Set sample detect time = 0 2. Subset -30 secs : 40 secs 3. Normalize 4. Smoothed with Moving Average with centred window of length = 30	| window 1 mean - window 3 mean | | window 1 mean - window 2 mean | | window 2 mean - window 3 mean |	Differences in means of the three widows	Hierarchical	No separation between ECDs and unsuccessful in clusters.
1. Set sample detect time = 0 2. Subset -40 secs : 30 secs 3. Normalize 4. Smoothed with Savitzky-Golay Filter	None	None	SOM	No seperation between ECD and unsuccessful readings. The main concern with savitzky-golay filter was that the edges of the waveforms seemed distorted.

APPENDIX B : REASONS AS TO WHY ITERATION 1 WAS UNSUCCESSFUL

There are numerous reasons for which all the pipelines from iteration 1 were unsuccessful. First, we did not perform any noise reduction even though the signals from the analyte we were working with are known to be very noisy. We justified this with the expectation that more noise might be an important characteristic to cluster on, but it is possible that it simply masked any differences between the ECD errors and the other unsuccessful readings.

Another reason that these initial pipelines may have failed is that we only ever used two clusters. While we do not know exactly how many different modes of failure there are, we do know that there are around thirty different error codes that can be returned by the system. While there may be overlap in the shapes of the waveforms between different error codes, or multiple waveform shapes that are associated with a particular error code, it does suggest that two clusters are likely insufficient to describe the wide variability across the readings.

Finally, we always clustered on windows 1, 2, and 3 separately. By not combining information from these windows, we may have lost important information about relative differences between them.

As a result of these issues, we drastically changed our preprocessing steps for the second iteration. Better results were obtained.

APPENDIX C : TOOLS USED IN THE UNSUCCESSFUL PIPELINES

1. PREPROCESSING

1.1 Removing the irrelevant period

For the first iteration, we chose to define the wet-up period by looking at several example traces and getting rid of the most consistently noisy time frame. Based on the readings that we examined, we decided that removing the first 150 seconds of each reading would be sufficient to remove the wet-up period. Because the readings were of different lengths prior to removing the wet-up, they were still different lengths after removing the wet-up. This worked well for the first iteration, where we then further subdivided into windows as described in section 1.3 below.

1.2 Standardizing

For the first iteration, once the wet-up period was removed, some of our pipelines also attempted standardizing the data.

Standardization. This is the process of taking a time-series and scaling it so that the standard deviation of the signal is 1 and the mean is 0. The formula to do this is

$$ts_{standardized} = \left\{ \frac{ts_i - mean(ts)}{standard\ deviation(ts)} \ for\ i\ in\ the\ length\ of\ ts \right\}$$

In other words, each time point in the sequence has the mean of the whole sequence subtracted from it before dividing by the standard deviation of the whole sequence.

The overall changes and relative shape of the waveform is retained, just the range of values is changed. Since this transforms the waveforms to fall within a range of values, it aids in visualization when we try to understand differences between the waveforms. Moreover, this helps prevent clustering algorithms from clustering based on the amplitude of the signals [8].

1.3 Windowing

In the first iteration, we divided the waveforms into three different windows and clustered them separately. We defined three windows: window 1, window 2, and window 3, as per subject matter experts (Figure 21).

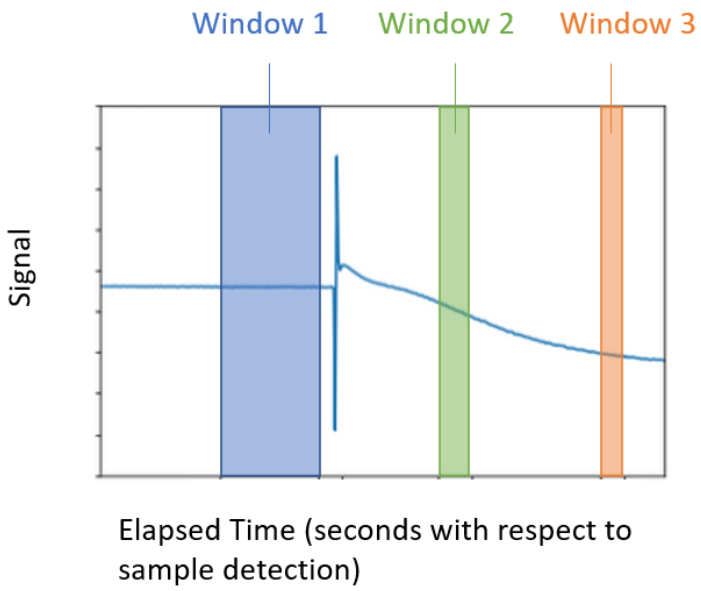


Figure 21. An illustration of the three windows used for clustering during the first iteration.

1.4 Noise Reduction

In the first iteration, we did not perform any noise reduction because we thought that different noise levels might convey important information. However, after seeing a lack of results, we focused on different ways of reducing noise. See Appendix D for filters that were tested but not used for the most promising pipelines.

2. FEATURE EXTRACTION

2.1 TSFresh

As was previously mentioned in the background section of this report, there exist two main sub-categories to whole time series clustering, one of which is feature-based clustering. Extracting features from raw timeseries is made easy with the TSFresh [28] python package. In fact, this package automatically extracts over 500 distinctive features for each waveform. These features are specifically designed to characterize time-series data. For example, some of these predictors include the waveform's autocorrelation at different lags, its kurtosis, its skewness as well as various others [29].

This package also includes a function to reduce the number of features extracted based on how useful each of them is in identifying the different classes contained in our data [30]. Since we had the labels for the ECD readings, we decided to use them to help filter out which features were most significant in helping us identify these anomalies. We recognize that this could have introduced some bias in our features as we wanted to cluster all the readings in an unknown number of clusters and not simply distinguish the ECDs

from the other unsuccessful readings. Nevertheless, as our approach was very exploratory, we decided to use this as a starting point.

After applying the filter function, we still had over 300 features and so to further reduce this number we applied principal component analysis (refer to section 5 of Appendix E).

Clustering was then performed on the principal components responsible for explaining a cumulative variance of at least 95%.

3. CLUSTERING

3.1 Hierarchical clustering

Hierarchical clustering is a popular clustering method as it does not require the user to specify the number of clusters to form. Furthermore, the clusters obtained can be visualized using a dendrogram as shown in Figure 22. The "faster" the observations join together, the more similar they are [31]. In Figure 22, observations A and B join together at the smallest distance and so are more similar to each other than the other observations are.

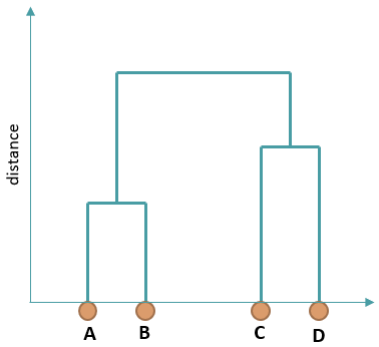


Figure 22. Dendrogram illustrating when observations are joined together in a cluster. Observations A and B are joined first, then observations C and D form a cluster. Finally, all the observations are grouped together.

There are two types of hierarchical clustering: agglomerative clustering and divisive hierarchical clustering [31]. For this project, we used agglomerative clustering. Below is a short overview of the algorithm [31].

1. Initially, each data point is in a separate cluster.
2. At each iteration, the distance between the observations and the clusters is calculated.
3. Observations that are closer to each other, meaning they are more similar, are grouped together forming new clusters.

4. Steps 2 and 3 are repeated until all the observations are in one large cluster.

Various measures exist to calculate the distances in Step 2. For this project, Euclidean distance with various linkage options (“single”,” complete”,” average” and” ward") was used. Linkage options determine how the distance between an observation and a cluster is calculated [31].

Since agglomerative clustering is based on a distance metric, it is important to standardize the dataset prior to using the algorithm. This ensures that all the variables contribute equally to how the clusters are defined [31].

We have used agglomerative clustering in various pipelines, one of which can be found in the *AutoencoderIteration1.ipynb* notebook.

3.2 Self-Organizing Maps (SOM)

Self-organizing maps (SOM) are artificial neural networks that are inspired by the human brain [32]. The human brain takes input from the environment and maps it to the appropriate sensory area to process it. Similarly, SOM maps each input vector to different regions called clusters.

The self-organizing map consists of input and output layers. Each neuron in the output layer is associated with a weight vector that is similar in shape to the input vector. These weights can be predefined or initialized randomly[32].

A distance metric is then used by the algorithm to calculate the distance between the input vector and the weight vector. For example, the Euclidean distance measure could be used to find the similarity between the input vector and the weight vector of neurons. The winning neuron is determined based on the weight vector that is closest in distance to the input vector [32]. The winning weight vector is then updated to become more like the input vector under consideration. Moreover, the neighbours of that weight are also updated to be similar to the input vector. This leads to clustering as similar inputs get joined together based on distance.

A simple working of SOM can be seen in Figure 23 below. This pictorially shows how the similar inputs get clustered together and how similar neurons come together to form these clusters.

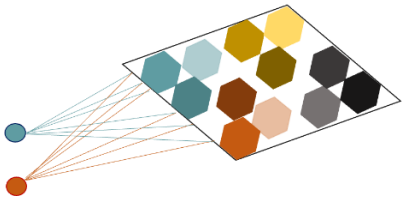


Figure 23: A simple diagram to depict the working of the self-organizing map algorithm.

For this project, we used a python package called *mini-som* which is a minimalistic python implementation of self-organizing maps to cluster our waveforms [33]. We developed several pipelines using self-organizing maps, each time with different input data. The input data to this algorithm were mainly either raw waveforms, waveforms processed differently using different pre-processing steps, or different windows and waveforms combined with predictor variables of choice. The code for this can be referenced in *SOM.ipynb*.

For each set of input data, the *mini-som* package was run to obtain maps which were processed to obtain the cluster format and then analyzed to characterize the waveforms. The algorithm can be customized using various hyper-parameters. Some of the main hyperparameters include *som_x* and *som_y* (the product of which gives the number of clusters of interest), the neighbourhood function (algorithm to find neighbour neurons), the activation distance (distance metric to calculate the distance between weight vector and input data), the learning rate, and the number of epochs.

Some of our best results were with the set of input data which was normalized, convolutionally smoothed, and combined with predictor variables which had the best feature importance scores as modelled using Random Forest Classifier. The best hyperparameters that worked well include som_x = 15, som_y = 2, Euclidean activation distance, Gaussian neighbourhood function, learning rate = 0.01, and 5000 epochs. Overall, the resulting clusters found with SOM were not the best. However, three clusters were obtained which in total consisted of about 177 ECDs vs 700 unsuccessful readings.

APPENDIX D: OTHER NOISE REDUCTION TECHNIQUES

Moving Average Smoothing

One noise reduction technique we tried was to adjust the signal at each point based on the average of a centred window thirty samples long. In other words, the signal at each time point was adjusted to have the value of the average of the fifteen time points before and after it. This was performed using the convolution smoother with a uniform window from the *tsmoothie* Python package, as this is mathematically equivalent to a moving average [19].

Savitzky-Golay Filtering

We also tried smoothing with a Savitzky-Golay filter, which functions similarly to the moving average approach, but instead of taking the mean, it fits a polynomial in each sliding window and adjusts the signal values according to that equation [11]. Following some trial and error, we chose a window fifty-one samples long. This filter was implemented using the *savgol_filter* provided by the Python package *Scipy* [34].

APPENDIX E: IN-DEPTH DESCRIPTION OF TECHNIQUES USED IN THE PROMISING PIPELINES

1. K-Means Clustering

K-Means clustering is one of the most powerful and widely used clustering algorithms among researchers [35]. It is up to the user to specify the number of clusters to find (K) and is suitable only for numerical data. Below is a short overview of the algorithm [21], [36]:

1. As an initial step, K readings from the dataset are randomly chosen as the initial centroids. A centroid represents the average reading within a cluster – its center.
2. The distance between each reading and the centroids is calculated.
3. Each reading is assigned to its nearest cluster.
4. New centroids are calculated for each cluster.
5. Steps 2 to 4 are repeated until every reading remains in the same cluster.

The most common distance measure used with the K-Means algorithm is the Euclidean distance [36]. This being said, we have already mentioned in the background section why Euclidean distance is not always suitable for time-series data. Hence, dynamic time warping is often a more appropriate distance measure.

In the context of this project, K-Means clustering was performed with both distance measures (Euclidean and DTW) and similar results were obtained. A possible reason for this is that the waveforms had already been pre-processed so that they were all aligned with respect to sample detect time and had the same length. We would expect to see the algorithm perform worse under Euclidean distance if this had not been the case.

Since K-Means clustering is based on a distance metric, it is important to standardize the dataset prior to using the algorithm. This ensures that all the variables contribute equally to how the clusters are defined [12], [37].

The code for running these algorithms is detailed in the *KMeansClustering.ipynb* notebook.

2. Random Forest Classifier

Random forest classification is a great supervised machine learning method built on top of decision trees [22]. The idea behind decision trees is to come up with binary splits based on the predictor variables which minimize the classification error when predicting the response variable. To avoid overfitting, a large tree is grown with lots of these splits and then pruned back using a penalty based on the number of terminal nodes in the tree combined with a tuning parameter, α. Decision trees are great because they have variable selection built in. If a predictor is not useful, then none of the splits in the tree will be based on it. By their

hierarchical nature in making decisions, they also inherently model any interactions between the predictors. However, they tend to be an oversimplification, so they have high bias and don't fit the training data very well and have high variance because they also tend to not perform well on new data [22].

Random forests address these issues with decision trees by creating lots of trees from bootstrap samples, randomly removing a certain number of predictors from consideration at each split, and then averaging together the results from all of the trees. By averaging the results from trees made from bootstrap samples, the results generalize much better to new data. Additionally, only considering a certain number of parameters at each split means that the generated trees are less correlated with each other, further improving predictive results on new data [22]. This averaging greatly improves the predictive capability of the model but does mean that it's much harder to explain how the predictor variables influence the response.

When we train a model using random forest it has the capability of determining which features have the most impact on the response variable. For us, this means whether an observation is a ECD error or one of the other unsuccessful readings. Since we wanted to know which of the predictors from the predictor dataset were most useful for predicting ECD errors, we used the *scikit learn* package function *feature_importances_* variable after training a random forest. This variable essentially measures how much each variable improves the performance when it is included in a decision tree [22], [38]. This function provided us with a score for all the predictors present in the aggregate predictor file. We then selected the most useful ones for finding ECD errors and used them in our clustering attempts along with other features. A complete implementation can be found in the *RandomForest.ipynb* notebook.

3. K-Shape Clustering

K-Shape clustering is very similar to K-Means except it is tailored specifically for shape-based whole time-series clustering. It differs from K-Means in two regards [17].

1. It uses normalized cross-correlation as a distance metric rather than Euclidean or DTW.
2. The centroids are calculated based on an optimization problem rather than computing the mean at each timepoint for all the readings in a cluster.

The steps taken by the algorithm are exactly the same as the ones taken by the K-Means algorithm (refer to section 1 of this Appendix).

First, let us discuss the distance measure used by the K-Shape algorithm. The distance measure is called Shape-Based Distance (SBD) and it is calculated as so:

$$SBD(ts1, ts2) = 1 - \max_{w}\left(\frac{CC_w(ts1, ts2)}{\sqrt{R_0(ts1, ts1) \cdot R_0(ts2, ts2)}}\right)$$

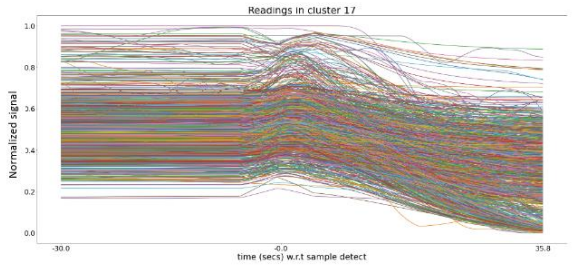
Where ts1 and ts2 are two time-series, $CC_w(ts1, ts2))$ corresponds to the cross-correlation coefficient between these two time-series at a lag of w and $R_0(ts1, ts1)$ and $R_0(ts2, ts2)$ are respectively the autocorrelation of ts1 and ts2 at a lag of 0. The geometric mean between these two autocorrelations found in the denominator is used to normalize the cross-correlation coefficients so that they remain between -1 and 1 [17]. For a brief introduction to autocorrelation see section 3.1 of this Appendix.

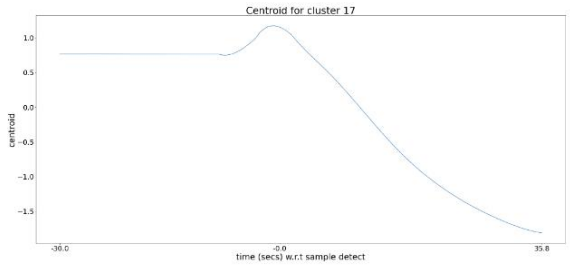
Once the cross-correlation coefficients have been calculated at each lag, we select the biggest cross-correlation coefficient, meaning the two sequences are optimally aligned. To go from a similarity measure to a distance measure, we subtract this term from 1, effectively obtaining the Shape-Based Distance between the two timeseries.

Since the normalized cross-correlation coefficient ranges between -1 and 1, the SBD measure will range from 0 to 2, where 0 indicates sequences that have exactly the same shape and 2 represents sequences that are completely opposite [17].

As was previously mentioned in the background section, cross-correlation is a similarity measure that is shift-invariant which means that the same can also be said about SBD.

Secondly, calculating the centroids of each cluster is a crucial step of the algorithm (refer to section 1 of this Appendix for the algorithm). To find these average sequences, K-Shape finds the sequence that maximizes "the sum of squared [similarities] to all other time-series sequences"[17]. Normalized cross-correlation is once again used as the similarity measure. Figure 24 illustrates the centroid obtained for 2 different clusters. We can see that the centroid reflects the general shape of the readings that are found in the corresponding cluster.





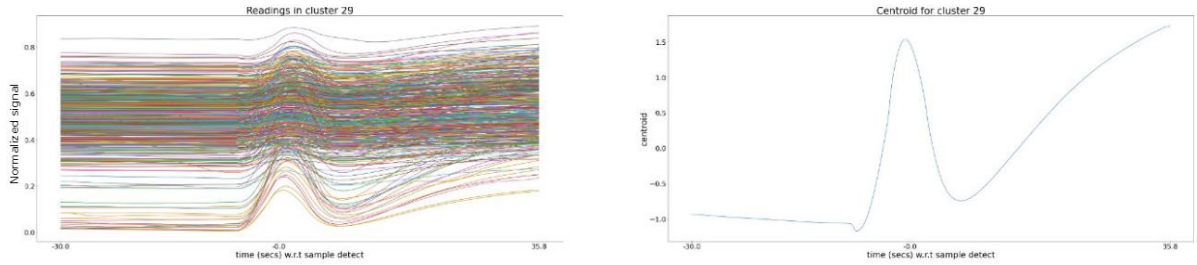


Figure 24. The plots on the right show the readings contained in two different clusters and their corresponding average sequence on the right.

Lastly, it is important to stress the importance of normalizing and standardizing each time-series prior to running the K-Shape algorithm. To get a clear understanding of why that is, let's look at the example in Figure 25.

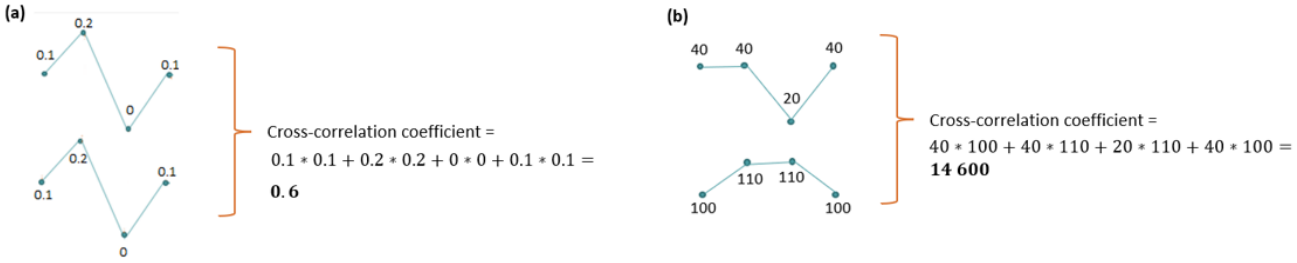


Figure 25. The sequences in (a) are more similar than those in (b) yet the cross-correlation coefficient in (a) is smaller. This illustrates the importance of normalizing so that all the timeseries are within a specified range and can meaningfully be compared.

From comparing the sequences in Figure 25 (a) and (b), we see that the sequences in (a) are more similar than those in (b) yet the cross-correlation coefficient in (a) is smaller because of the differences in scale. Based on these cross-correlations coefficients, since higher cross-correlations indicate higher similarity, we would assume that the sequences in (b) are more alike than those in (a). To prevent this from happening, it is important to normalize the timeseries to ensure that they are all expressed within the same specified range. Refer to section 4.1.3 for details on how the time-series were normalized.

Standardizing the waveforms so that they are centred at 0 and have a standard deviation of 1 is also crucial [17]. In fact, if we do not center the waveforms at 0, then the algorithm will cluster based on where the waveforms sit rather than based on their overall shape. Figure 26 shows how normalization and

standardization impact the waveforms. After both normalization and standardization are applied, cross-correlation will produce values that can be meaningfully compared.

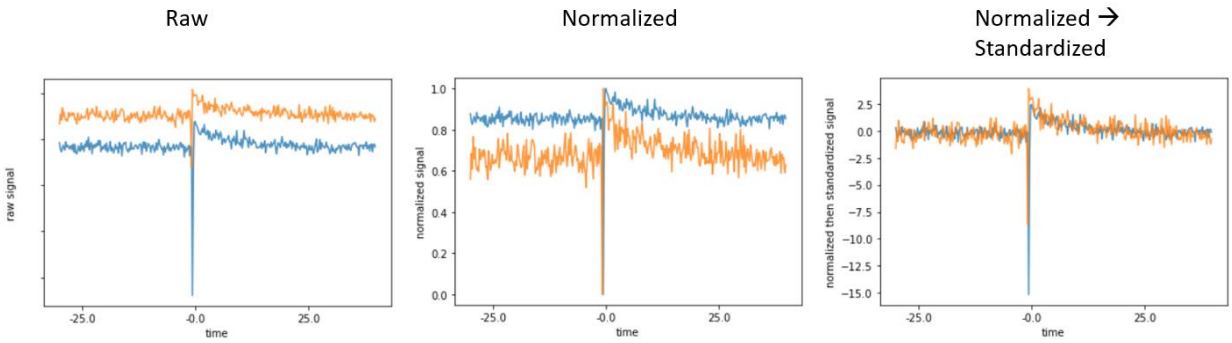


Figure 26. The plot to the left shows two waveforms on their original scale. In the middle, we have the normalized waveforms ranging from 0 to 1. Finally, on the right we have the waveforms normalized and scaled. We can see how after normalization and standardization have been applied, using cross-correlation will yield values that can be meaningfully compared. Raw signal scale removed for confidentiality.

3.1 Autocorrelation

Briefly, autocorrelation represents the correlation between a timeseries and itself. The autocorrelation is maximized at lag 0 as this is when all the peaks and the valleys are lined up with one another yielding the highest similarity between the 2 sequences [39]. This is illustrated in Figure 27.

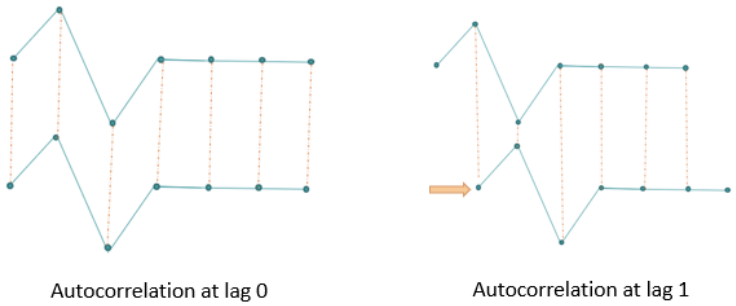


Figure 27. We can see that a sequence is most similar to itself at a lag of 0 as all the peaks and valleys are aligned with one another.

4. Autoencoder

An autoencoder is an artificial neural network used to learn more meaningful hidden features from unlabelled data. It is trained iteratively to propose hidden features and then reconstruct the original data from these features using the following architecture [24]:

1. An encoder that maps the input vector into a smaller vector.
2. A decoder that re-maps the smaller vector to reconstruct the input vector using the features learned from the encoder.

Both the encoder and decoder portions consist of a pre-set number of hidden layers, where each layer takes the input from the previous layer and after a series of convolutions feeds it to the next layer. During the training phase of building the model, the optimal way to perform these transformations is learned by the autoencoder [24].

Since our goal was to reduce the dimensional space of the time-series waveforms, the encoder portion of this architecture was used (see Figure 28).

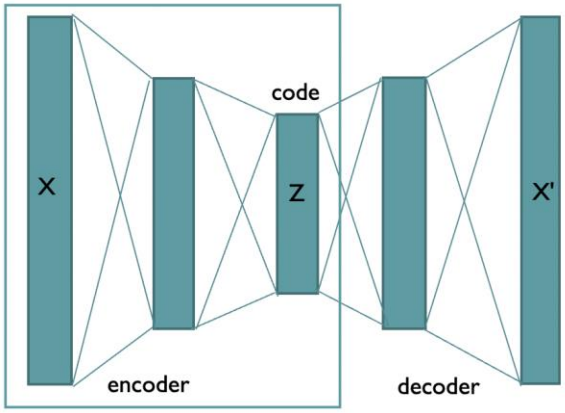


Figure 28. Illustrates the basic architecture of an autoencoder. There is a box around the encoder portion since that is the part that we are using for feature extraction.

By learning the optimal way to compress the input so that it can be rebuilt, the autoencoder determines which parts of the input are the most important in explaining it. This means that the smaller vector contains a smaller amount of rearranged information, in other words, hidden features, which describe the original [24].

Feature extraction can thus be performed by training the autoencoder on how to optimally perform these mappings, and then using the output from the encoder as a new set of features. As a result, we chose to use an autoencoder to extract features from the time-series data. It is important to note that the autoencoder needs to be trained using both the encoder and decoder before we can use the encoder bit to extract features.

To build a simple autoencoder, the *TensorFlow Keras* library in Python was used [40]. Our autoencoder had 13 hidden layers in both the encoder and decoder portions. When training the model on 80% of the pre-

processed time-series data, we used the Adadelta method for optimization along with binary cross entropy as a loss function [41]. We trained over 10 epochs with a batch size of 32 and then validated using the remaining 20% of the time-series data to see if our model was able to successfully reconstruct time-series from the extracted features. In doing so, we extracted 278 features from the pre-processed time series. The complete implementation is available in the *autoencoder.ipynb* notebook.

5. Principal Component Analysis (PCA)

Principal component analysis (PCA) is a dimensionality reduction technique used in unsupervised frameworks [42]. The term 'dimension' in the context of machine learning often refers to the number of variables/features that are fed to a modelling algorithm. Thus, dimension reduction techniques can be used when the set of predictors is too large making it challenging for the modelling algorithm to converge to sensible results. Furthermore, working in a lower-dimensional space also makes it easier for the user to explore and visualize the dataset [42]. However, one of the trade-offs with dimension reduction is the possible loss of pertinent information [43].

Principal Component Analysis aims to reduce the number of input variables while still retaining as much of the information from the original dataset as possible [44]. The algorithm uses algebra. More specifically, it decomposes the covariance matrix of the dataset[1] to create a new set of predictors. These new predictors are referred to as principal components. Each principal component is independent of each other and is a linear combination of all the variables from the original dataset [44]. The process of creating new features from the original data, here the principal components, is referred to as feature extraction.

The first principal component accounts for most of the information contained in the original dataset. Synonymously, it is the direction along which the original data has the highest variance (see PC 1 in Figure 29). The second principal component explains less variation than the first principal component did. Each component successively explains less variation than its preceding component [44]. Figure 29 illustrates the first (PC 1) and second (PC 2) principal component for a dataset with 2 variables.

[1] It is important that the dataset be standardized to have a mean of 0 and a variance of 1 before the covariance matrix is calculated.

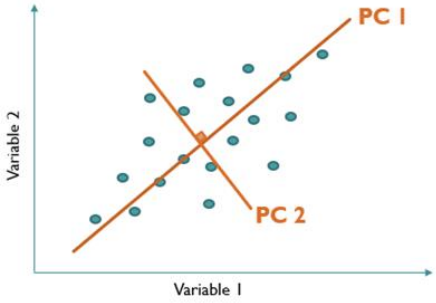


Figure 29. Illustration of the first two principal components for an original dataset comprised of 2 variables.

Applying principal component analysis will result in the same number of components as there are features in the original dataset. Dimensionality reduction only comes into play once some of these principal components are discarded [43], [45]. Since the components are ordered in decreasing order of explained variance, the last components can be omitted without this resulting in a big loss of information. In this project, we often kept enough principal components to retain 95% of the variance contained in the original data.

PCA was used in multiple pipelines during this project (as seen in Tables 1 and 2 of Appendix A). It served two main purposes:

1. Dimension reduction/Feature extraction
2. Visualization of the data in 2 or 3 dimensions

As a result of us working with time-series, which are intrinsically high dimensional, the number of features was often very large, sometimes reaching more than 500 features. Using PCA allowed us to consider our dataset in a lower dimensional space.

We also used PCA for visualization purposes. In fact, by plotting two or three of the first few principal components against each other, we were able to determine whether there seemed to be certain patterns in the data. For example, Figure 30 shows that most of the ECDs load around 0 on the second principal component.

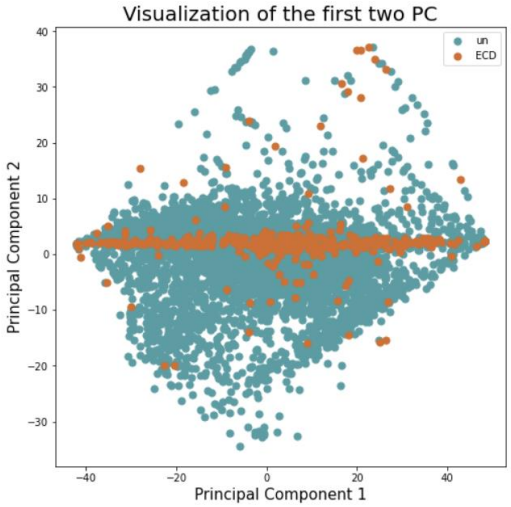


Figure 30. Example of how to use principal components for visualization.

A detailed walk-through of how to use PCA to obtain visualizations in a lower dimensional space is given in the *pca.ipynb* notebook.

6. Gaussian Mixture Modelling (GMM)

Another popular clustering method is Gaussian mixture modelling (GMM). Mixture modelling is a technique which assumes that each data point belongs to a subpopulation with a distinct probability distribution associated with it, and GMM falls under this umbrella, assuming that each subpopulation is generated from a gaussian distribution [25]. A large advantage of GMM is that the parameters for the normal distribution (mean and standard deviation) of each subpopulation do not need to be known in advance, the model learns what they are according to the data. Similarly, which subpopulation each observation belongs to does not need to be known ahead of time. These characteristics mean that GMM is extremely flexible when learning the underlying grouped structure of the data. Furthermore, by assigning each a probability of belonging to a group, the model can appropriately cluster the data even in the presence of outliers. One drawback is that the number of clusters must be pre-defined, and the assumption must be made that the subpopulations are in fact normally distributed [25].

The model essentially works by finding values for the cluster subpopulation parameters which maximize the likelihood of the data and then assigning the data probabilities of belonging to each subpopulation. This is done using an expectation-maximization algorithm, an explanation of which is beyond the scope of this report [25].

In the context of this project, GMM was performed on the PCA components accounting for 95% of the variance of the features extracted from the autoencoder, along with the most important predictors from the

aggregate predictor file, selected by random forest as previously described. To determine the optimal number of clusters, we used a plot of the Bayesian Information Criterion (BIC) for a range of models with different numbers of clusters (Figure 31). This gives us an estimate of how much the model improves when increasing the number of clusters by estimating how likely the model is given the data [25]. In the graph, we saw that the BIC started to plateau with around 5 clusters, so we tried to perform GMM with 5 clusters initially, but we noticed that a lot of different waveform shapes were getting clustered together. As a result, we tried clustering with 21 clusters next, since there was another dip in the BIC at that point, and we achieved some promising results. The code for running this clustering method and to see the clusters formed is an *autoencoder.ipynb*

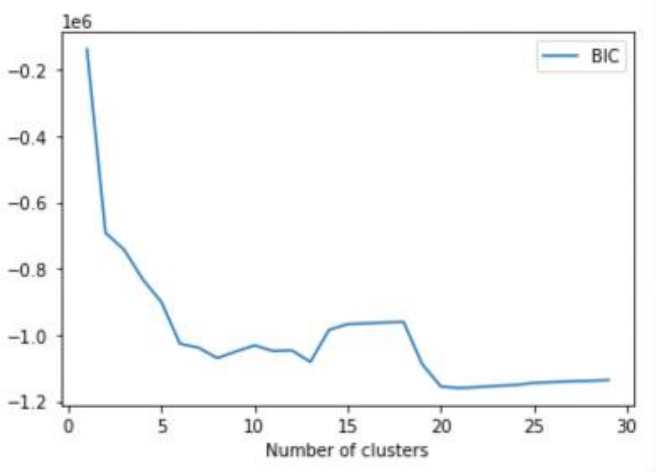


Figure 31. Shows the Bayesian Information Criterion for models with different numbers of clusters used to determine the optimal number of clusters.

APPENDIX F: DIAGNOSTIC PLOTS

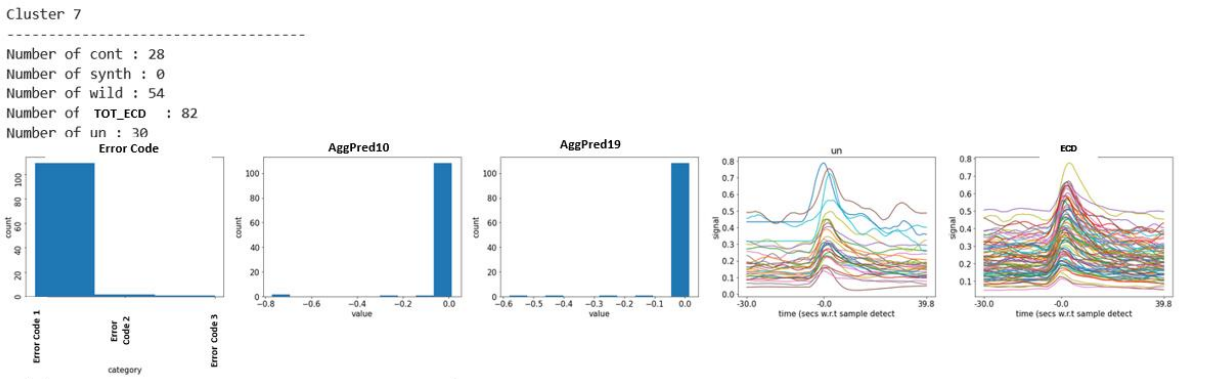
Another challenge with this project was describing the clusters that were output by the different pipelines and evaluating which pipelines led to successful results. While we had labels for some of the ECD errors, there was also an unknown number of additional ECD errors in the other unlabelled unsuccessful readings, and an unknown number of other categories of errors. This made using traditional measures of cluster evaluation challenging.

Since our main goals were to find clusters with as many ECD errors as possible and to describe the readings that were clustered together, we took a more subjective approach to evaluate the clusters. Hence, because people who work with readings from this sensor are already accustomed to using the aggregate predictors, we decided to use these same metrics to characterize the readings contained in each cluster. This involved a lot of visual diagnostic tools as shown in Figure 32.

The code to generate these diagnostics can be found in the *diagostics.py* file. The main ways we chose to describe the clusters are as follows:

1. Histograms of aggregate features describing the readings in each cluster (e.g., Mean window A or mean of window B) (Figure 32). These are useful for describing the characteristics of readings that fall into each cluster.
2. Traces showing the overlayed normalized and zeroed waveforms for all clusters formed; this is done in two separate plots, one with the ECD error readings in the cluster, and one with all the other unlabelled unsuccessful readings (Figure 32). These plots give us a key understanding of how the shapes are clustered and information about the similarities of ECD errors and unsuccessful readings in these clusters.
3. A table with information regarding the number of ECD errors (separated by the different categories: original, synthetic, and contaminated), and the number of unsuccessful readings. The rows in the table are sorted so that the clusters with the highest number of ECD errors are shown at the top (Figure 32). This table is useful for determining which clusters have the highest number of ECD errors and whether these clusters are concentrated with ECD errors or if they are diluted with other unsuccessful readings.
4. Density plots of selected aggregate features describing the readings from two different clusters of interest (Figure 32). These are useful when the goal is to compare the characteristics of two clusters.

a)



b)

Cluster	Label	cont	synth	wild	TOT_ECD	un
2		9	31	89	129	2547
7		28	0	54	82	30
3		4	8	17	29	230
0		0	16	9	25	465
10		5	1	14	20	36
4		0	0	17	17	80
6		1	4	11	16	697

c)

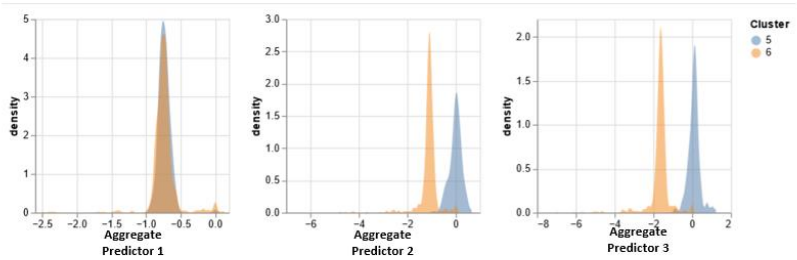


Figure 32. Shows some of the ways we described the readings in different clusters. a) shows the output of histograms for the aggregate predictors error code, *AggPred10*, and *AggPred19* for cluster number 7 from one of the pipelines. It also shows the overlayed normalized traces. These plots can be generated with *describe_clusters()* in the *diagnostics.py* file. b) Shows a table describing the counts of readings in each cluster. It can be generated using *get_label_counts()* in the *diagnositcs.py* file. c) Shows the overlayed density estimations for clusters 5 and 6 from a particular pipeline for three aggregate predictors. These plots can be generated using *compare_cluster_densities()* in *diagnostics.py*.

References

[1] "Blood Tests - Blood Tests | NHLBI, NIH." https://www.nhlbi.nih.gov/health/blood-tests (accessed May 04, 2022).

[2] "epoc® Blood Analysis System." https://www.siemens-healthineers.com/en-ca/blood-gas/blood-gas-systems/epoc-blood-analysis-system (accessed May 04, 2022).

[3] "epoc® Blood Analysis System Resource Guide".

[4] D. Azariadi, V. Tsoutsouras, S. Xydis, and D. Soudris, "ECG signal analysis and arrhythmia detection on IoT wearable medical devices," *2016 5th International Conference on Modern Circuits and Systems Technologies, MOCAST 2016*, Jun. 2016, doi: 10.1109/MOCAST.2016.7495143.

[5] M. Nawaz, J. Ahmed, G. Abbas, and M. Ur Rehman, "Signal Analysis and Anomaly Detection of IoT-Based Healthcare Framework," *2020 Global Conference on Wireless and Optical Technologies, GCWOT 2020*, Oct. 2020, doi: 10.1109/GCWOT49901.2020.9391621.

[6] D. Wulsin, J. Blanco, R. Mani, and B. Litt, "Semi-supervised anomaly detection for EEG waveforms using deep belief nets," *Proceedings - 9th International Conference on Machine Learning and Applications, ICMLA 2010*, pp. 436–441, 2010, doi: 10.1109/ICMLA.2010.71.

[7] S. Aghabozorgi, A. Seyed Shirkhorshidi, and T. Ying Wah, "Time-series clustering – A decade review," *Information Systems*, vol. 53, pp. 16–38, Oct. 2015, doi: 10.1016/J.IS.2015.04.007.

[8] S. Yazdi, "Clustering of Large Time-Series Datasets Using a Multi-Step Approach," University of Malaya, Kuala Lumpur, 2013.

[9] E. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: implications for previous and future research," *Knowledge and Information Systems 2004 8:2*, vol. 8, no. 2, pp. 154–177, Aug. 2005, doi: 10.1007/S10115-004-0172-7.

[10] G. Guo, H. Wang, and D. Bell, "Data reduction and noise filtering for predicting times series," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2419, pp. 421–429, 2002, doi: 10.1007/3-540-45703-8_39.

[11] A. Savitzky and M. J. E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures," *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, Jul. 1964, doi: 10.1021/AC60214A047/ASSET/AC60214A047.FP.PNG_V03.

[12] J. Andrews, "Distance Measures - UBC Okanagan Data 573 Lecture Notes," 2022. https://github.com/ubco-mds-2021/data573/blob/main/lectures/lecture1.pdf (accessed Jun. 15, 2022).

[13] D. F. Silva and G. E. A. P. A. Batista, “Speeding up all-pairwise dynamic time warping matrix calculation,” *16th SIAM International Conference on Data Mining 2016, SDM 2016*, pp. 837–845, 2016, doi: 10.1137/1.9781611974348.94.

[14] M. K. Brown and L. R. Rabiner, “Dynamic time warping for isolated word recognition based on ordered graph searching techniques,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 1982-May, pp. 1255–1258, 1982, doi: 10.1109/ICASSP.1982.1171695.

[15] J. Semmlow, “Basic Concepts in Signal Processing,” *Signals and Systems for Bioengineers*, pp. 35–80, Jan. 2012, doi: 10.1016/B978-0-12-384982-3.00002-X.

[16] P. Bourke, “Cross Correlation,” 1996. http://paulbourke.net/miscellaneous/correlate/ (accessed Jun. 12, 2022).

[17] J. Paparrizos and L. Gravano, “k-Shape,” *ACM SIGMOD Record*, vol. 45, no. 1, pp. 69–76, Jun. 2016, doi: 10.1145/2949741.2949758.

[18] R. Chellappa, A. Veeraraghavan, and N. Ramanathan, “Gaussian Mixture Models,” *Encyclopedia of Biometrics*, pp. 659–663, 2009, doi: 10.1007/978-0-387-73003-5_196.

[19] “cerlymarco/tsmoothie: A python library for time-series smoothing and outlier detection in a vectorized way.” https://github.com/cerlymarco/tsmoothie (accessed Jun. 13, 2022).

[20] “scipy.signal.windows.bartlett — SciPy v1.8.1 Manual.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.windows.bartlett.html (accessed Jun. 13, 2022).

[21] J. Andrews, “Clustering - Data 573 UBC Okanagan Lecture Notes,” 2022. https://github.com/ubco-mds-2021/data573/blob/main/lectures/lecture2.pdf (accessed Jun. 15, 2022).

[22] L. Breiman, “Random Forests,” *Machine Learning 2001 45:1*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[23] M. ‘Arif, H. Hassan, D. Nasien, and H. Haron, “A Review on Feature Extraction and Feature Selection for Handwritten Character Recognition,” *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 2, 2015, doi: 10.14569/IJACSA.2015.060230.

[24] N. Tavakoli, S. Siami-Namini, M. Adl Khanghah, F. Mirza Soltani, and A. Siami Namin, “An autoencoder-based deep learning approach for clustering time series data,” *SN Applied Sciences*, vol. 2, no. 5, pp. 1–25, May 2020, doi: 10.1007/S42452-020-2584-8/FIGURES/19.

[25] D. Reynolds, “Gaussian Mixture Models,” *Encyclopedia of Biometrics*, pp. 659–663, 2009, doi: 10.1007/978-0-387-73003-5_196.

[26] M. Decuyper *et al.*, “An Overview of Overfitting and its Solutions,” *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022022, Feb. 2019, doi: 10.1088/1742-6596/1168/2/022022.

[27] K. R. Shahapure and C. Nicholas, “Cluster quality analysis using silhouette score,” *Proceedings - 2020 IEEE 7th International Conference on Data Science and Advanced Analytics, DSAA 2020*, pp. 747–748, Oct. 2020, doi: 10.1109/DSAA49011.2020.00096.

[28] “tsfresh — tsfresh 0.18.1.” https://tsfresh.readthedocs.io/en/latest/ (accessed Jun. 15, 2022).

[29] “Overview on extracted features — tsfresh .” https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html (accessed Jun. 15, 2022).

[30] “Feature filtering — tsfresh.” https://tsfresh.readthedocs.io/en/latest/text/feature_filtering.html (accessed Jun. 15, 2022).

[31] T. Tullis and B. Albert, “Special Topics,” *Measuring the User Experience*, pp. 209–236, Jan. 2013, doi: 10.1016/B978-0-12-415781-1.00009-1.

[32] M. M. van Hulle, “Self-organizing maps,” *Handbook of Natural Computing*, vol. 1–4, pp. 585–622, Jan. 2012, doi: 10.1007/978-3-540-92910-9_19/FIGURES/001922.

[33] “JustGlowing/minisom: MiniSom is a minimalistic implementation of the Self Organizing Maps.” https://github.com/JustGlowing/minisom (accessed Jun. 15, 2022).

[34] “scipy.signal.savgol_filter — SciPy v1.8.1 Manual.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.savgol_filter.html (accessed Jun. 13, 2022).

[35] M. Ahmed, R. Seraj, and S. M. S. Islam, “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation,” *Electronics 2020, Vol. 9, Page 1295*, vol. 9, no. 8, p. 1295, Aug. 2020, doi: 10.3390/ELECTRONICS9081295.

[36] N. Shi, X. Liu, and Y. Guan, “Research on k-means clustering algorithm: An improved k-means clustering algorithm,” *3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010*, pp. 63–67, 2010, doi: 10.1109/IITSI.2010.74.

[37] G. W. Miuigan, M. C. Cooper, G. W. Milligan, G. W. Milligan, and M. C. Cooper, “A study of standardization of variables in cluster analysis,” *Journal of Classification 1988 5:2*, vol. 5, no. 2, pp. 181–204, Sep. 1988, doi: 10.1007/BF01897163.

[38] “sklearn.ensemble.RandomForestClassifier — scikit-learn 1.0.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (accessed Apr. 21, 2022).

[39] D. Dondurur, “Fundamentals of Data Processing,” *Acquisition and Processing of Marine Seismic Data*, pp. 211–239, Jan. 2018, doi: 10.1016/B978-0-12-811490-2.00004-9.

[40] “Intro to Autoencoders | TensorFlow Core.” https://www.tensorflow.org/tutorials/generative/autoencoder (accessed Jun. 14, 2022).

[41] “Adadelta.” https://keras.io/api/optimizers/adadelta/ (accessed Jun. 14, 2022).

[42] I. T. Jollife and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, Apr. 2016, doi: 10.1098/RSTA.2015.0202.

[43] R. M. Terol, A. R. Reina, S. Ziaei, and D. Gil, “A Machine Learning Approach to Reduce Dimensional Space in Large Datasets,” *IEEE Access*, vol. 8, pp. 148181–148192, 2020, doi: 10.1109/ACCESS.2020.3012836.

[44] C. Syms, “Principal Components Analysis,” *Encyclopedia of Ecology, Five-Volume Set*, pp. 2940–2949, Jan. 2008, doi: 10.1016/B978-008045405-4.00538-3.

[45] N. Salem and S. Hussein, “Data dimensional reduction and principal components analysis,” *Procedia Computer Science*, vol. 163, pp. 292–299, Jan. 2019, doi: 10.1016/J.PROCS.2019.12.111.