Anomaly Detection in Time Series Brain Data Collected Using Functional Near-Infrared Spectroscopy

Petra Kumi

May 2020

A Major Qualifying Project submitted to the Faculty of
Worcester Polytechnic Institute in partial fulfilment of the requirements for
the degree of Bachelor of Science
in
Mathematical Sciences and Computer Science

Report Submitted to:
Prof. Rodica Neamtu (CS), Prof. Erin Solovey (CS), and Prof. Suzanne Weekes (MA)
Worcester Polytechnic Institute

Abstract

This project explores some statistical and machine learning methods used to detect motion anomalies in data collected using Functional Near-Infrared Spectroscopy (fNIRS). fNIRS is a new type of noninvasive brain imaging technology that provides information about the dynamic cognitive state of individuals doing tasks in the real world. The portability and accuracy of fNIRS allow researchers to model brain activity of subjects in everyday situations. However, the data is collected using many highly sensitive sensors. This contributes to its high complexity and warrants analysis using a combination of statistical and machine learning methods. The study of anomaly detection of fNIRS is well sought after, as finding and post-processing anomalies leads to more accurate datasets and in turn helps scientists analyze their data more accurately. The results of this research effort include analyzing performance of a previously published anomaly detection method, creating and evaluating three new methods, and developing a platform that allows others to analyze their own data using these methods.

List of Figures

List of Tables

Contents

1 Introduction

Functional near-infrared spectroscopy, or fNIRS, is an emerging hemodynamic neuroimaging technology that can be used to map brain activity [5]. The technology gathers data about the amount of oxygen flowing in parts of a patient's brain, which helps scientists understand the level of activity in that part of the brain. fNIRS mapping is becoming increasingly popular because of its ease of use and practicality with more sensitive subjects such as kids and the elderly [2]. Among other applications, analyzing fNIRS data can potentially pave the way for more streamlined Human-Computer Interaction and more intuitive Brain-Computer Interfaces (BCI).

With any collection of data, there may be one or multiple data points that deviate significantly from the rest of the data, called anomalies. Anomalies are data points that lie abnormally far from other values and differ greatly from the trends of normal data [9]. Anomalies can provide a lot of critical information about the data: technical incidents such as a faulty sensor, new trends, etc. Therefore, an emerging type of data analysis is anomaly detection. The topic of anomaly detection sees many applications across disciplines such as cybersecurity, finance, marketing or neuroscience.

Scientists are generally interested in detecting anomalies in brain data to find unwanted movement of other external factors that lower the accuracy of the collected data, or to find problems with the device used to gather data. While more research and time has been devoted to analyzing data produced by fMRIs (functional magnetic resonance imaging) or EEGs (electroencephalogram) leading to certain standards being developed, research that analyzes fNIRS data is much newer and less standardized [7]. Our team aims to bridge this gap by applying well-known anomaly detection tools and models on fNIRS data and comparing the accuracy of the results.

The goal of this project is to determine methods for finding anomalies in fNIRS data and provide a tool to make these methods available to others. In order to achieve our goal, we break it down into smaller objectives, outlined below.

1. Understand the nuances of multivariate time-series data, fNIRS, and anomaly detection

2. Research appropriate algorithms that can be applied to multivariate time series data

3. Implement algorithms with fNIRS data
4. Create new anomaly detection algorithms
5. Evaluate and compare algorithms
6. Develop a platform to allow others to run these algorithms on their own data

2 Background and Literature Review

2.1 Time Series

Time series are a type of data where the observed points are indexed according to the order they are observed in time. We can consider a time series to be a sequence of random variables, $x_1 + x_2 + x_3 ...$, where the variable x_1 shows the value of the series at the first time point, x_2 at the second, and so on [14]. This series is append only, which means that if a value of x is updated, instead of overwriting the previous observation, we simply add another one at the new time at which it was recorded. This type of data is used in numerous industries, with more important and widespread use in meteorology, medicine, and finance [8].

2.1.1 Characteristics of Time Series

Time series possess three main characteristics: trend, seasonality/periodicity, and stationarity. Recognizing characteristics of time series data is very important as each one can unravel key features of the data such as the types of algorithms we can/cannot use, or the transformations we need to do in order to be able to use these types of algorithms.

Trend The trend of a time series describes the tendency of all values in the series to increase, decrease, or stay the same. We can determine trend by computing a linear fit of the series and finding the slope of the generated linear function. An example of this is shown on Figure 1, of the water level in Lake Huron, fitted using a least squares line and clearly displaying negative trend.

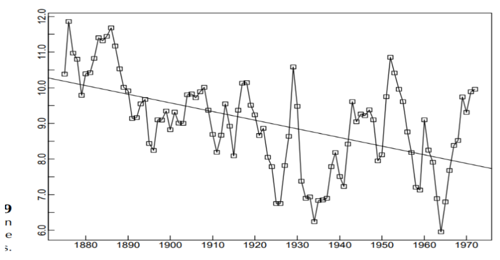
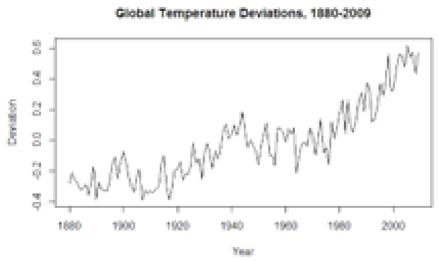
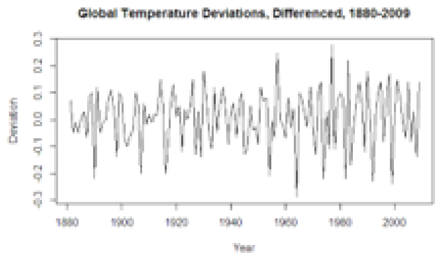


Figure 1: Level of Lake Huron 1875–1972 showing the line fitted by least squares. Source: Introduction to Time Series Forecasting, Brockwell et al. 2016

Seasonality/Periodicity While trend shows a series' overall tendency of values to increase or decrease, seasonality and periodicity describe cyclical variations in data which repeat at certain points in time. Figure 2 shows periodicity in fMRI data collected in various locations of the



(a) a



(b) b

Figure 3: Time series of global temperatures over time with [a] showing the raw data (non-stationary) and [b] showing the same data but is now stationary

brain of a person whose hand was periodically brushed.

Stationarity In its most widely used definition, stationary data is data whose mean and variance is constant [14]. Autoregression-based analysis methods require the data to be stationary to generate accurate output. There exist methods to test for stationarity and to turn non-stationary data into stationary, but we will not focus on them. Figure 3 shows raw data of global temperature deviations, followed by processed data to make the series stationary.

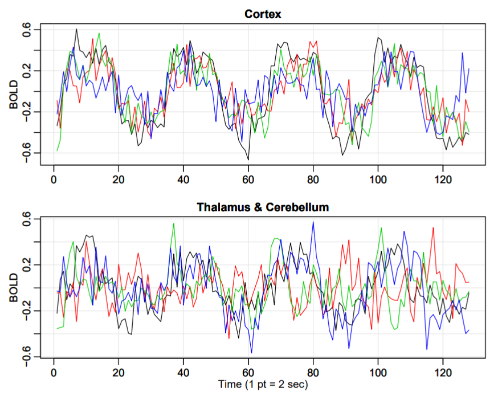


Figure 2: fMRI data collected in various locations of the brain of a person whose hand was periodically brushed. Source: Introduction to Time Series Forecasting, Brockwell et al., 2016

2.1.2 Uni-variate and Multi-variate Time Series

Another way to classify time series is into univariate and multivariate. Univariate time series are series where only one variable is dependent on time. We have been mostly focusing on univariate series in this paper up to this point, with examples in Figures

1 and 3. However, the fNIRS data is multivariate.

Multivariate series contain at least two series of variables which are dependent on time and each other [4]. Figure 2 is a perfect example of multivariate time series data: each plot shows multiple series dependent on time, and each other. Multivariate time series possess many of the same qualities that are present in univariate series because they are essentially composed of univariate series. The definitions for trend, periodicity, and stationarity are some of them. However, complications arise when taking into account the interdependence between individual components, as some parts of them could be strongly correlated with parts of other components [4]. Because of this not all algorithms that work on univariate time series also work for multivariate ones.

2.2 Functional Near-infrared Spectroscopy

fNIRS, or functional near-infrared spectroscopy, is a non-invasive form of neuroimaging that monitors the hemodynamics, or movement of blood, inside a person's head [15]. When compared to other methods of brain imaging, fNIRS is relatively new with its first clinical trial having been conducted in 1993 [5]. However, since then, the use of fNIRS technology has expanded dramatically with the number of publications related to fNIRS nearly doubling every 3.5 years [7]. The nearly exponential increase can be attributed to clinics and researchers seeing the method's benefits and new opportunities it can provide in fields such as Human-Computer Interaction (HCI) and Brain-Computer Interfaces (BCI) [5].

2.2.1 How fNIRS Technology Works

As the name suggests, fNIRS uses near-infrared light to monitor the hemodynamics of the brain [7]. The device itself, as seen in Figure 4, is made up of nodes that are placed around a black head cap. The nodes can be placed anywhere on the cap, meaning any area of the brain can be analyzed and measured. In our project, we are working with data obtained specifically from the anterior prefrontal cortex, a section of the brain that deals with decision-making, multi-tasking, memory, and attention [13]. This region of the brain is located behind the forehead. One particular benefit to placing nodes in that location is that there is very little hair in this area of the head, which leads to capturing cleaner data.

In our case, the cap used has eight light sources and eight detectors, the combination of which forms channels. In our current arrangement we obtain 20 channels of data in a session. When in use, each light source shines near-infrared rays 1-3 cm into the wearer's skull [13]. At this depth, the proteins oxygenated (Hb) and deoxygenated (HbO) hemoglobin which are contained in blood, are the primary absorbers of the light. The light that is not absorbed by the hemoglobin is then reflected and detected by the sensors. The change in levels of reflected light corresponds to the amount of oxygen in the blood [5]. This information can then infer which regions of the brain are using the most oxygen, telling scientists which parts of the brain are more activated than others. The data from the detectors is then fed into a computer where it can be analyzed and studied. When the data is visualized, each channel corresponds to one time series [5].

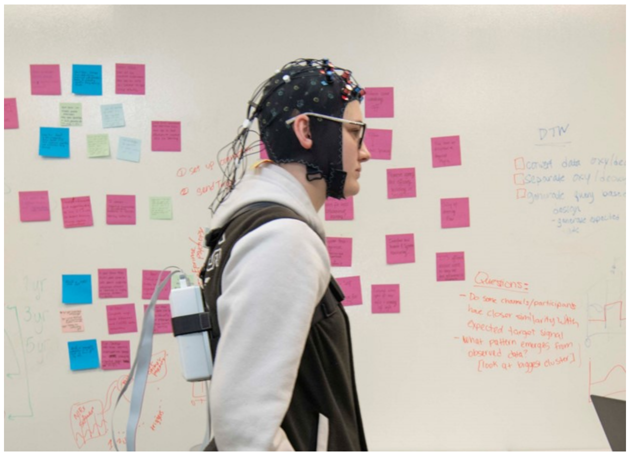


Figure 4: fNIRS device on a patient. The device includes the cap, sensors, and a small portable computer

2.2.2 The benefits of fNIRS for HCI/BCI

When compared to the other tools and devices that are used for mapping brain activity such as electroencephalograms (EEG) or functional magnetic resonance imaging (fMRI), fNIRS has many advantages, especially in the fields of Human-Computer Interaction (HCI) and Brain-Computer Interface (BCI). Some advantages to fNIRS are they are compact, less expensive to operate and use, easily portable, and relatively tolerant of body movements [5]. Another advantage is that unlike the other mapping tools, fNIRS allows the user freedom of movement. fMRIs and EEG force users to lay motionless to get accurate information as those methods require the use of large, cumbersome machines [2]. For fNIRS, the user has very little restrictions; they are allowed to move around, speak, and interact with other objects such as computers. This gives scientists and researchers more flexibility as now the user can interact with objects while their brain is being mapped (Wan, 2015). Previously in HCI, we could only detect certain signals from users doing work on a computer, such as facial cues, heartbeat and eye movement. With the introduction of fNIRS, we can now map how the user interacts with computers, opening possibilities for analyzing brain data in much more versatile environments.

2.3 Anomalies

Anomalies are data points that lie abnormally far from other values and differ greatly from the trends of normal data [9]. Another term that is frequently associated with anomalies is outliers. The word anomaly has many definitions, sometimes overlapping that of anomaly and sometimes describing a slightly different phenomenon. For our project we will refer to anomalies and outliers in the same context [1]. One common misunderstanding is anomalies equivalent to noisy data. Noise is the random error or variance in a measured variable and can be filtered out by applying appropriate preprocessing strategies [1]. On the other hand, anomalies are not only errors, but are also discordant data could reveal more information about the subject or data as a whole [11].

2.3.1 Types of Anomalies

From our research, we were able to find three different types of anomalies: global, contextual and collective. Global anomalies, or point anomalies, are individual data points that deviate significantly from the rest of the data set [6]. These anomalies are easier to find than the other types because global anomalies can be usually seen by graphing the data [1]. Figure 5 shows an example of a point anomaly.

Contextual anomalies are points that deviate significantly from other normal points in the same context (see Figure 6). With these types of anomalies, the data may be within the global range but is abnormal when compared to data in the same seasonality [1]. When determining possible contextual anomalies, it is very important to discuss the data with a domain expert as they can help determine if a data point is an anomaly or not.

Finally, collective anomalies are a certain subset of data points that deviate significantly from the entire set. With collective anomalies, the individual

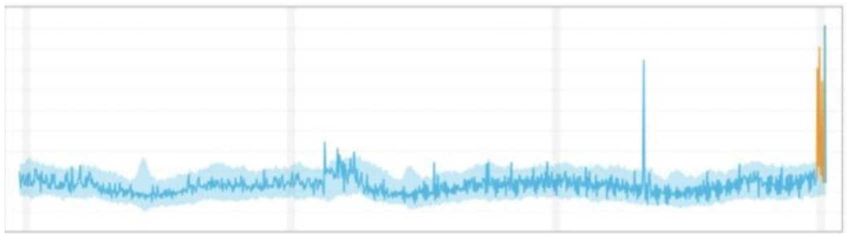


Figure 5: Visual example of global anomalies (colored orange). Source: anodot.com

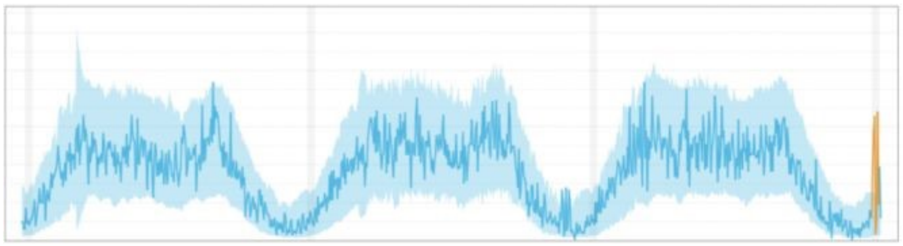


Figure 6: Visual example of contextual anomalies (colored orange). Source: anodot.com

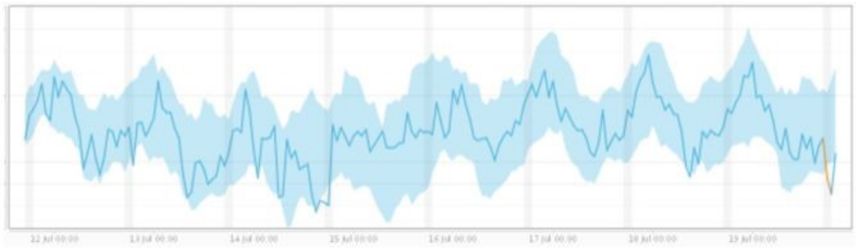


Figure 7: Visual example of collective anomalies (colored orange). Source: anodot.com

data points that are contained in the anomalous subset individually may or may not be anomalous themselves; it is when one looks at the data points as a whole that they are considered anomalous [1]. See Figure 7 for an example of collective anomalies.

2.3.2 Anomalies in fNIRS

In the domain of fNIRS, anomalies can come in a variety of ways. Some sources of anomalous data include external head/facial motion, “bad” channels, “bad” subjects, and other unknown states. Because of the location of the nodes and the way sensors receive brain signals, fNIRS data is very sensitive to extraneous movements of the face, head and body [12]. Any small movement of the light sources/detectors caused by shaking of the head or making facial expressions that involve the forehead, such as frowning and smiling, can distort the path of the near-infrared light which will misrepresent the data. Next, there is the detection of “bad” sensors. During an experiment, there may be times where one of the channels does not work properly. It could be that the light source/detector for that channel is malfunctioning or it could be that is not receiving a high-quality signal. This type of error can lead to faulty data and corrupt part or all of the data set [7]. The third type of anomaly is an anomaly occurring from respiration or heartbeat. Both these activities directly impact the amount of oxygen passing through a certain part of our body, causing sudden spikes in fNIRS data. Finally, we have the unknown states.

2.3.3 Difficulties in analyzing fNIRS anomalies

There are some difficulties to analyzing fNIRS data. First some types of anomalies could overlap with one another. In these cases it might become harder to find the anomaly, and when it is found it is harder for scientists to decide how to post-process it. Next, fNIRS data does not follow well-known patterns in time series like the ones discussed above. Hence, we cannot use on fNIRS data many of the common algorithms designed for time series analysis. Finally, at the beginning of this project we did not have any fNIRS data whose anomalies were previously labelled. This made it harder to understand in practice the shape and the behavior of anomalies in datasets and to evaluate the accuracy of anomaly detection algorithms.

2.4 Commonly used methods for detecting time series anomalies

Throughout our research, we compiled and examined a large number of algorithms used to analyze time series and categorized them based on various criteria. The criteria we used when examining the algorithms are:

- Its objective: whether it is noise reduction, classification, forecasting, anomaly detection, or dimension reduction
- Used for univariate time series, multi-variate time series, or both
- Whether or not the algorithm is used specifically for time series
- If it is a statistical or machine learning method, and if not statistical then whether it is supervised or unsupervised

We were most interested in algorithms that are either statistics-based or unsupervised, and those that are designed for use with time series, especially multivariate ones. Listed are some of the algorithms we researched in more depth: Auto-Regressive Integrated Moving Average, Isolation Forests, Bayesian Networks, One-Class Support Vector Machines, and Long-Short Term Memory Recurrent Neural Networks (LSTM-RNNs). A complete list of algorithms their categorization can be seen in Appendix A. LSTM-RNNs stood out among these algorithms and an anomaly detection method based on them seemed promising during our research. Below we describe LSTM-RNNs and the anomaly detection method in more detail.

2.4.1 Long-Short Term Recurrent Neural Networks

LSTMs made a continuous appearance during our research on tools to analyze time series data, with usages in financial predictions, meteorological forecasting, analysis of medical data, as well as in sequential non-time series data such as speech and text recognition [8]. LSTMs are a type of Recurrent Neural Network designed to better handle longer and more complex sequences. To understand how LSTMs work, we need to have a basic understanding of Neural Networks and Recurrent Neural Networks (RNNs).

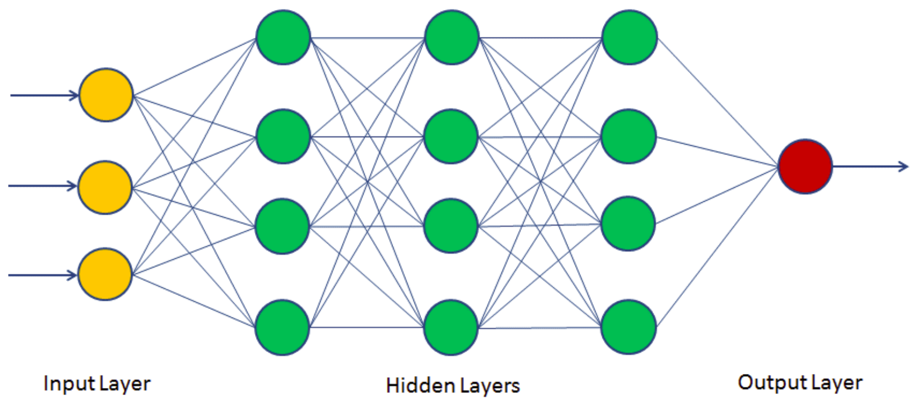


Figure 8: A simple neural network with one input layer, three hidden layers and one output layer. Source: datacamp.com

Neural Networks

Neural Networks are a system composed of nodes organized in layers, where every node of the previous layer is connected to every node of the next. Once a data point gets put through the network, it gets received by every node in the input layer. Then, these nodes send this information to every node in the second layer. Here, every node multiplies the data from every input it receives by a weight value (initially assigned at random). It then adds all the weighted values together to form one output. This output is then sent to every node of the next layer, where the cycle gets repeated again until the end of the network is reached. The goal of the nodes is to tweak the weights so that the final output produced from the network is as close as possible to a desired output decided on by the user. See Figure 8 for a simple architecture of neural networks with one input layer, three hidden layers (between the input and output), and one output layer [**aggarwal2018a**]. This is the simplest architecture of a neural network, also called a feed-forward network.

Recurrent Neural Networks

RNNs are a modification of feed-forward networks that learn by looking at past inputted values to find patterns in data. Because of this they are frequently used with sequential data. RNNs only contain one hidden layer, as seen in Figure 9. Once a data point gets inputted to an RNN, it gets multiplied by a given weight and inputted to an activation function, which

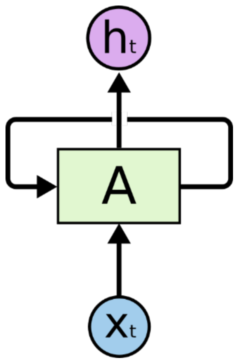


Figure 9: Simple diagram of a Recurrent Neural Network, showing the input xt which in this case is part of a sequence dependent on time, output ht, a hidden layer A, and the loop followed by the generated "memory" to be brought back as input. Source: Colah, 2015

is usually *tanh* of the form $f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$ (Aggrawal, 2018). Before the value gets outputted, the network creates a copy of the important information about the input. After the value gets outputted and compared to a given value in the training set provided, the weights get updated and the next inputted value gets processed. However, the next input is summed with the information that was remembered in the previous iteration, and then weighted and taken through the activation function. This allows the network to learn from the past values. The updated memory contains information about both the first and the second value. This process continues until the entire series has been processed.

Long-Short Term RNNs

RNNs work well for short and low complexity series. However, the system that is put in place to update the memory of an RNN tends to prioritize new information regardless of the importance of the old information. Because of this, RNNs are said to have a very "short-term" memory. LSTM-rNNs (or LSTMs for short) improve the RNN architecture to allow for processing much longer and more complex series, through mechanisms called gates. Gates allow them to decide what information from the data to "remember" and what to discard. The most basic LSTM architecture consists of three gates: the Forget Gate, Input Gate and Output Gate [10]. These gates are usually sigmoid functions of the form $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$, where the output is the decision of the gate and input is the Cell State (for Forget and Input Gates) or the input after it is processed as described in the next paragraph (Output Gate) [10].

When a new input is received, the Forget Gate looks at the current Cell State (the "memory" of the network, assuming this is not the first input of the network) to decide which parts of it to forget (remove) at the end of this iteration. Once that has been decided, the Input Gate decides which parts of the Cell State to update with the new values from the input. Afterwards, the new input is processed as it would in an RNN and the decided upon parts of the Cell State are updated: some information gets forgotten, some gets overwritten, and the rest stays the same. At this point, this cell state is what will be transferred to the next iteration of this network. Finally, the Output Gate decides which part of the processed input to output as the final result [10]. This process is essentially a way to ensure that information that is extremely important to the accuracy of the results does not get forgotten simply because it is old, which would have been the case in RNNs. Because of the internal structure described above, LSTMs are a powerful tool for analyzing highly interconnected and complex data, and the reason why we decided to focus on them for this project. However, LSTMs are designed to fit and predict data, not to detect anomalies, so they need to be used in conjunction with a type of error checking method to find points that differ the most from the LSTM-generated fit. We discuss such a method during the sample implementation of LSTMs in section 2.5.

2.4.2 Telemanom

Telemanom is an anomaly detection method built on top of LSTMs. It was initially proposed in a paper entitled "Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding" [3] as a way to find anomalies in spacecraft telemetry data. Telemetry data is information about the well-being of spacecraft systems, captured by various sensors which form different channels, similarly to fNIRS. The tool does not take advantage of any domain-specific information to find anomalies in data, which means that it can be used with any type of time series data that shows similar characteristics [3].

To find the anomalies in a dataset, Telemanom begins by training an LSTM model on part of the data that looks least anomalous. It then uses the trained model to predict all values of the dataset. Then, it uses mean square error (mse) to calculate the difference between each real and LSTM-predicted value, creating a sequence of errors. Mean squared error is calculated using the formula below and is a common way to calculate error between two points.

$$\text{mse} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2 \tag{1}$$

where n is the length of the error sequence, y_i is the i^th real signal value, and $\overline{y}_i$ is the i^th predicted signal value.

The error sequence is then split into smaller sequences of equal length. For each window, a threshold is calculated using the error values, of the form $mean + coefficient * standarddeviation$. All error values above this threshold are removed and the mean of the sequence is recalculated. This process is repeated until an optimal threshold is found such that removing all error values above it causes the biggest change in mean of the sequence. The original signal values corresponding to the errors that are above the optimal threshold are marked as anomalous.

In the Methodology section we implement this method on a labelled dataset and discuss its results.

3 Methodology

In order to reach our objectives, we followed a series of steps involving creating labelled datasets, finding a way to evaluate performance of an anomaly detection algorithm, implementing previously researched algorithms, then creating and implementing our own anomaly detection methods. Below we discuss each step in higher detail.

3.1 Creating a labelled set of anomalies using fNIRS

To determine the validity of any method, we have to test its performance. In the case of Telemanom we are interested in finding how good the method is at predicting anomalies. However, this is impossible to do without a set of labelled anomalies - a set where the known anomalies are marked. As none of the fNIRS datasets that were already available to us had labelled anomalies, we created our own. We recorded 5.5 minutes of brain activity of a subject sitting still. When directed to do so, the person would shake their head causing a motion artifact. This artifact was considered to be an anomaly. The fNIRS cap used on our subject was connected to 20 channels, four of which did not provide a stable signal. Hence, the final recorded dataset contained 16 reliable time series of 2821 points each, 1033 being anomalies. Each anomaly corresponded to 28 data points. With this dataset in hand, we are able to compare the predicted anomalies to the real ones to calculate how well the method predicts anomalies. We have made this dataset available for the purpose of reproducing the results of this study.

3.2 Performance measure of anomaly detectors

The next step in determining the performance of a method is to decide on a metric that is objective and does not require human input to determine the most accurate method. Because the anomalies detected by the methods we are developing will be re-evaluated by a specialist in the field, we want our method to find as many of the real anomalies as possible, even if it means falsely labeling some normal values as anomalous. To determine a way to express the statement above in mathematical terms, we started by looking at the most commonly used measurements of performance and their meaning.

Ideally, we would like for both precision and recall to be equal to one, because that would mean no LSTM values were incorrectly flagged as either

Table 1: Definitions and explanations for some common terms used to measure algorithm performance

True Positives (TP)	Values flagged as anomalous which are truly anomalous
False Positives (FP)	Values flagged as anomalous which are not anomalous
True Negatives (TN)	Values flagged as not anomalous which are truly not anomalous
False Negatives (FN)	Values flagged as not anomalous which are truly anomalous
Precision	$$P = \frac{TP}{(TP + FP)}$$ Fraction of values correctly flagged as anomalies out of all values flagged as anomalies. Ranges between 0 and 1. Shows us how good the method is at restricting its predicted anomalous points to the ones that are truly anomalous. If the model only identifies some true positives and marks everything else as normal then precision would be 1.
Recall	$$R = \frac{TP}{(TP + FN)}$$ Fraction of values correctly flagged as anomalies out of the total number of truly anomalous points in the dataset. Ranges between 0 and 1. Shows us the fraction of real anomalies that were correctly flagged as anomalous. If all points in the dataset were labeled as anomalous, recall would be = 1.
F_β Score	$$F_\beta = (1+\beta)^2 \frac{PR}{((\beta^2 P) + R)}$$ An example of a harmonic mean, a kind of averaging method. We are taking a type of average of precision and recall, to create a balance between the two. The β parameter is always > 0. If β is equal to one, then precision and recall are given the same importance (weight). If $\beta > 1$, then more weight is being put on recall, and if $\beta < 1$ precision is prioritized. If precision and recall are both high which makes us happy, the F_β score is high regardless of the β value.

normal or anomalous. However, that is highly unlikely. From the definitions in Table 1 we notice we would ideally have a balance between precision and recall. We previously mentioned we would rather pick a method that labels as anomalous all areas that are truly anomalous and some areas that are not, than a method that does not incorrectly label any anomalies but does not label all the areas that are in reality anomalous. This means we are slightly more interested in a high recall than high precision. The F-score, i.e. F_β score, provides a way to numerically express this balance. As stated in 1, F_β is a weighted harmonic mean that gives recall (R) β times more weight than precision (P). For $\beta > 0$, F_β ranges from 1 to 1. To understand how it works we have to look at the formula for F_1 i.e the balanced harmonic mean between P and R. The formula for a harmonic mean between P and R is as follows:

$$F_1 = (\frac{\frac{1}{P} + \frac{1}{R}}{2})^{-1} \tag{2}$$

which is equivalent to

$$F_1 = 2\frac{PR}{P+R} \tag{3}$$

To calculate this mean, we take the reciprocals of P and R, average them, and then take the reciprocal of the result. This formula is also equivalent to:

$$F_1 = (\frac{1}{2}\frac{1}{P} + \frac{1}{2}\frac{1}{R})^{-1} \tag{4}$$

To make this a weighted average, i.e to give more importance to one variable than the other, we have to multiply P and R by two values different than 0.5 whose sum = 1. We multiply by the larger of the weights the variable that we want to give more importance to. In our case we pick our coefficients as

$$\frac{\beta}{\beta+1} and \frac{1}{\beta+1} \tag{5}$$

When $\beta = 1$, the above coefficients are equal. When β is less than 1 the first coefficient is smaller than the second, while when β is greater than 1 the first coefficient is greater than the second. Since we want F to change more significantly with the change of R than P for $\beta > 1$, we assign the larger weight to R. Thus, the weighted harmonic mean becomes:

$$F_1 = (\frac{1}{\beta+1}\frac{1}{P} + \frac{1}{\beta+1}\frac{1}{R})^{-1} \tag{6}$$

which is equivalent to

$$F_1 = (1+\beta^2)(\frac{PR}{(\beta^2 P) + R} \tag{7}$$

From this we expect F_β for $\beta > 1$ to change more rapidly with changes in R than with changes in P, because the coefficient in front of R will be bigger. Hence for $\beta < 1$ F_β will change faster for changes in P than in R because the coefficient in front of P will be bigger.

Before deciding on the β value, we wanted to see if a high F_β score necessarily means we have high precision and recall regardless of the β value. To determine this, we initially examined the partial derivatives of the F_β score, shown below.

$$\frac{\partial(F_\beta)}{\partial(P)} = \frac{R^2 + \beta^2 R^2}{\beta^4 P^2 + 2\beta^2 PR + R^2} \tag{8}$$

$$\frac{\partial(F_\beta)}{\partial(R)} = \frac{\beta^2 P^2 + \beta^4 P^2}{\beta^4 P^2 + 2\beta^2 PR + R^2} \tag{9}$$

where P is precision and R is recall. From the equations above we can see that the partial derivative with respect to P is always > 0 for B, $R > 0$, while the partial derivative with respect to R is always > 0 for B, $P > 0$. This means that if R is increasing and P is constant or if P is increasing and R is constant then F_β is also increasing. Also if P and R are both increasing at the same time, then F_β again increases. However, if one of either P or R increases while the other decreases then we cannot say whether F_β will increase or decrease. This suggests that if P and R are high then F_β is high, but it does not show us whether P and R are necessarily high when F_β is high. Thus, we developed graphs for various β values that showcase what the F_β score is for each possible combination of precision and recall for β ranging between 0 and 2, seen in Figure 10.

From Figure 10 we noticed that in general for all β, F_β is highest when precision and recall are both high. For betas closer to zero, we saw that the F_β score remains high even when recall decreases and precision is high,

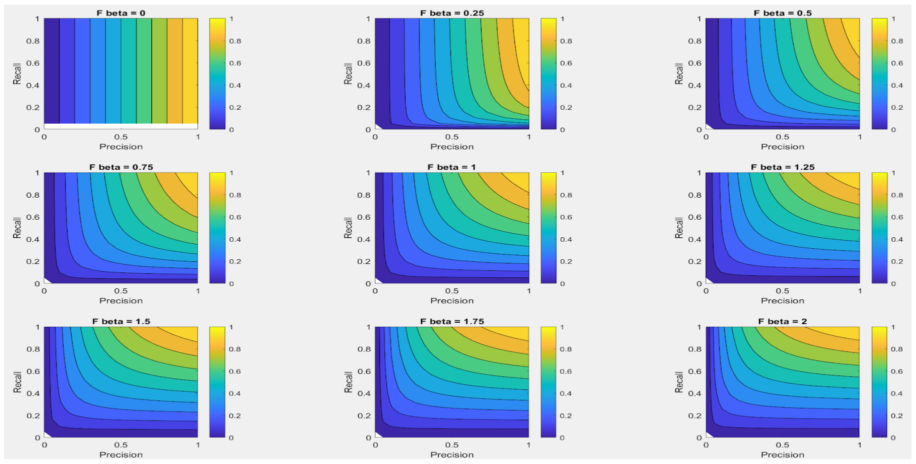


Figure 10: Contour plots of F_β scores for β between 0 and 2

while for β close to two, F_β remains high when precision decreases and recall remains high. We also noticed that when we fix a certain recall and F_β value while increasing the β value, higher precision is needed to reach the fixed F_β score. Similarly, when we fixed precision and F_β, we needed lower recall to reach the fixed F_β. From this we can say that for $\beta > 0.5$, a high F_β means we have high precision and high recall, which is what we want.

To decide on the value of β we first needed to decide on what F_β score we consider "good" i.e. we are content with. We decided on $F \geq 0.7$. Then, we needed to decide on how much we are willing to lower precision on high recall ($R = 1$) and still have a high F_β. Based on our needs we decided that to get a recall of 1 we are willing to sacrifice precision until $P = 0.5$. On the other hand if we had high precision and lower recall, how low can we allow recall to go and still get high F_β ($F_\beta \geq 0.7$)? We decided that for $precision = 1$, a recall of 0.6 is still high enough to ensure that enough values of the dataset are marked as anomalous. Thus, we can choose a β that meets the following criteria:

1. Given $R = 1$ then $F_\beta >= 0.7$ only for $P \geq 0.5$. That is, if $R = 1$ and $P < 0.5$ then $F < 0.7$.

2. Given $P = 1$ then $F \geq 0.7$ only for $R \geq 0.6$. That is if $P = 1$ and $R < 0.6$ then $F < 0.7$.

The β coefficient such that results in F_β best satisfy these two conditions is 1.5.

As stated before, we want our method to find as many of the real anomalies as possible, even if it means falsely labeling some normal values as anomalous. This means we value recall more than precision, so we are interested in choosing a β value higher than one. Since $F_{1.5}$ reflects the fact that a higher recall is slightly prioritized over high precision, we decided to use the $F_{1.5}$ value to measure the performance of our methods.

3.3 Widening anomalous intervals

During the process of generating sample $F_{1.5}$ scores for some Telemanom results, we noticed that the flagged anomalies corresponded with the general area where there was in reality an anomaly, but the labelled interval was somewhat shorter, longer, or shifted from the expected anomaly. Because of this we decided to post-process the real data by adding a buffer area, delta, on each side of a real anomalous interval, which results in a widening of the interval. This process extends the anomalous region from $[a_i, a_f]$ to $[a_i - \delta, a_f + \delta]$ where a_i is the initial point of an anomalous interval and a_f is the final point. Thus, if a predicted anomalous interval falls within $[a_i\delta, a_f + \delta]$, then it is considered a true positive. This also helps take into account the difference between when the subject of the experiment is given a prompt during an experiment and when they react to it. The subject could either react with a delay or expect to find an anomaly and react preemptively. The reaction could either result in the anomalous section being visible before or after the start or end of a section labelled as truly anomalous.

3.4 Implementing Telemanom

We used the Telemanom algorithm to predict anomalies in our labelled dataset with the purpose of evaluating its performance on fNIRS data. We trained an LSTM model on the first channel of the labelled dataset, then used this model to predict signal values of all channels. To measure method performance we averaged $F_{1.5}$ scores of predictions in all channels. Figure 11 graphs some average $F_{1.5}$ scores of predictions made using various Telemanom parameters. These values are calculated after adding the buffer mentioned in the previous section. The plot shows that $F_{1.5}$ scores do not go above 0.3, telling us that regardless of the LSTM or Telemanom parameters, the

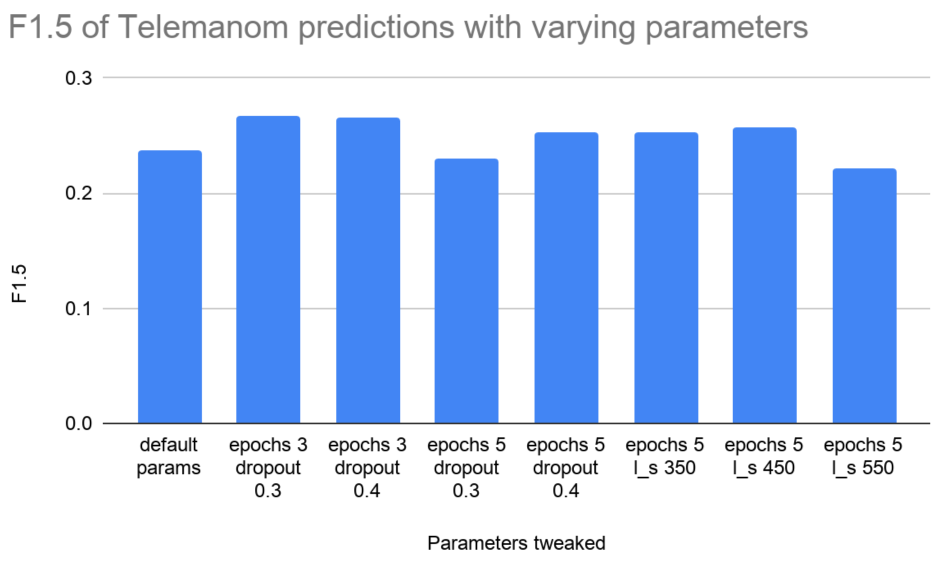


Figure 11: Average $F_1.5$ scores of all channels in the labelled dataset, analyzed using various Telemanom parameters.

algorithm does not find enough motion anomalies in the data, meaning that it performs poorly on fNIRS data.

Figures 12 and 13 show the real and predicted anomalies in Channel 0 of the labelled dataset. From these plots one can also visually tell that Telemanom does not flag many of the real anomalies in the dataset. However the anomalies that it does find correspond to regions that are in reality anomalous. Moreover, it seems that the LSTM does a good job at predicting the general shape of the real signal curve.

3.5 Creating new anomaly detection methods

In the previous section we showed that Telemanom does not detect motion anomalies in fNIRS well but the LSTM does a good job at predicting the shape of the original curve, acting as a smoother. Thus, we decided to build new anomaly detectors on top of LSTM predictions. Alongside LSTMs, we used the concept of variation as the base of these methods. Below we describe variation and its relationship with anomalies in a dataset.

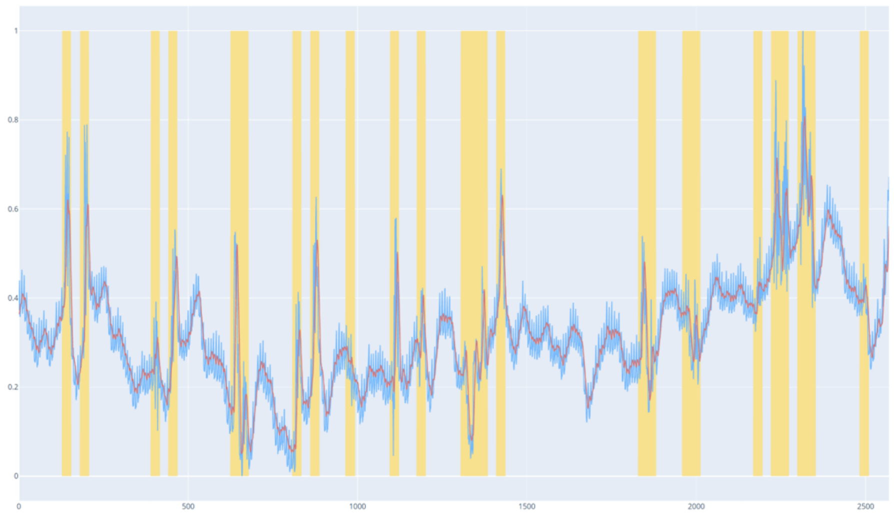


Figure 12: LSTM-smoothed signal values (red), real signal values (blue), and real anomalies (yellow) for channel 0 of the labelled dataset

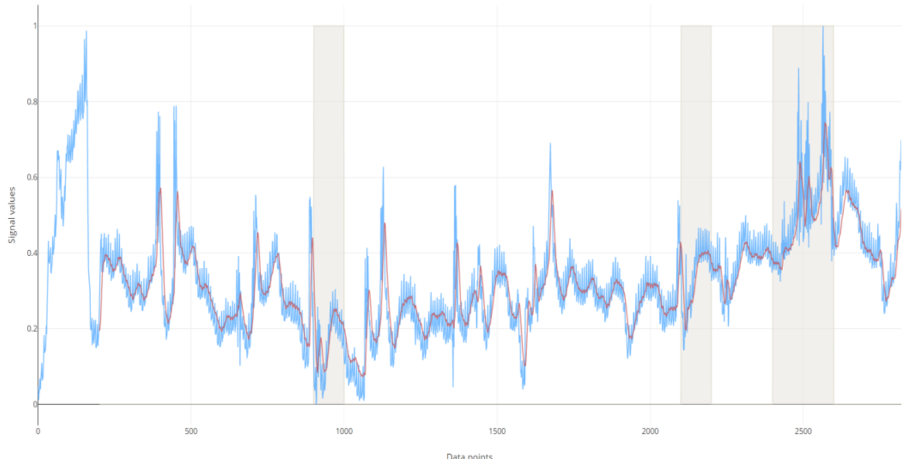


Figure 13: LSTM-smoothed signal values (red), real signal values (blue), and Telemanom-predicted anomalies (grey) for channel 0 of the labelled dataset

3.5.1 Variation as an indicator of anomalies in a dataset

Figure 14 shows the plot of the LSTM prediction and real anomalies of Channel 0 in the labelled dataset. The common characteristic of all anomalous sections is a spike in the signal values that after peaking return close to the ones before the anomalous section started. This spike lasts only for a short interval of time. In other words, there is a higher variability in the values of the LSTM curve during an anomalous interval than during a normal one. Mathematically, this variability is captured through variation. The variation of a curve over an interval of size B determines how much the curve changes from the first to the last point of the interval. A very noisy curve has high variation, while a smooth curve (low noise) has low variation. We compute the variation v_i of y from index i to $i+B$ for every index $i = 1, .., N-B$ of our time series. The formula for variation is

$$\{v_i\}_{i=1}^{N-B} : v_i = \sum_{j=i}^{i+B-1} |y_{j+1} - y_j| \tag{10}$$

For continuous functions $f(x)$, the analogous variation formula is

$$\{v_i\}_{i=1}^{N-B} : \int_{c}^{x+B} |f(x)| \tag{11}$$

To see if there is a significant difference in variation for anomalous versus non-anomalous sections, we plotted the variation curve of Channel 0 along with the LSTM-predicted signal, shown in Figure 14 where B was chosen to be 28, as we already know the anomalous intervals last for 28 data points. From Figure 14 we can see that there is in fact a significant difference in variation values between normal and anomalous sections. This shows us that high variation is an indicator of anomalies in a dataset. Because of this, we decided to use signal variation as the base of our anomaly detection methods.

3.5.2 Variation of Variation

As described in the sections above, variation measures the variability of the signal during a given interval. If this variability is consistent then the variation curve will be somewhat constant. If the variability changes then the variation values will vary as well. If we have an fNIRS curve that is constantly noisy the variation curve might not be able to help us find the anomalies in

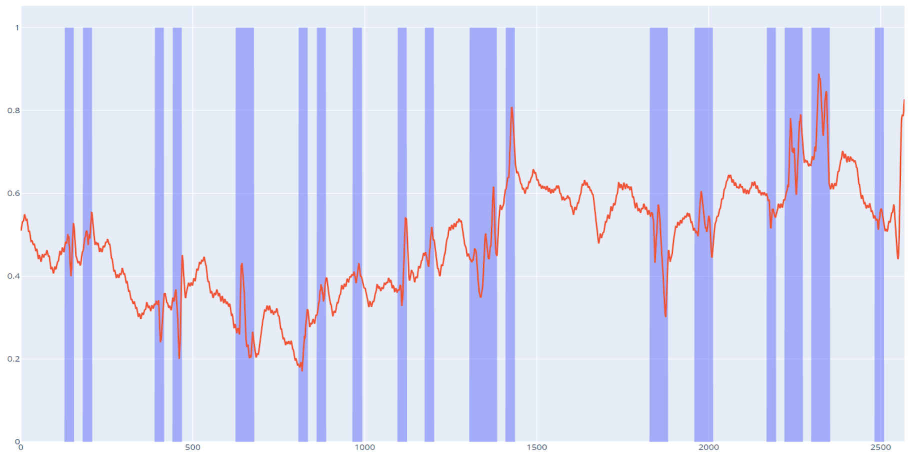


Figure 14: Smoothed LSTM curve (red) and real anomalies(purple) in Channel 0 of the labelled dataset.

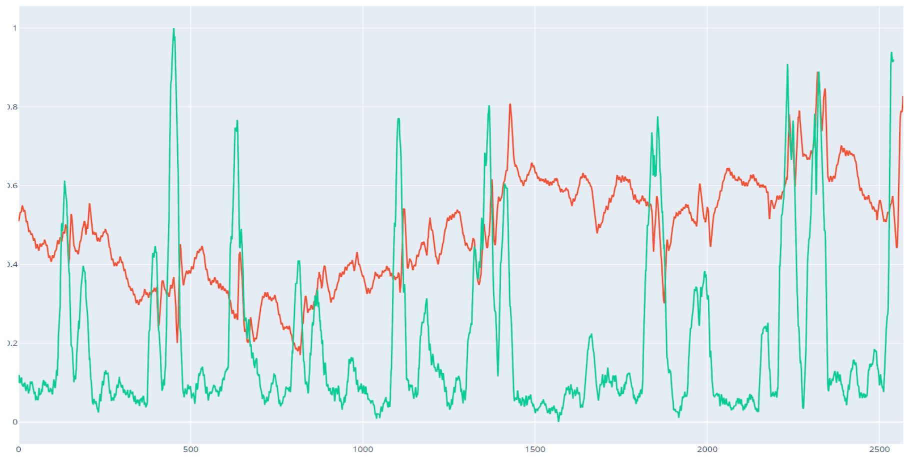


Figure 15: Smoothed LSTM curve (red) and the variation curve (green) in Channel 0 of the labelled dataset.

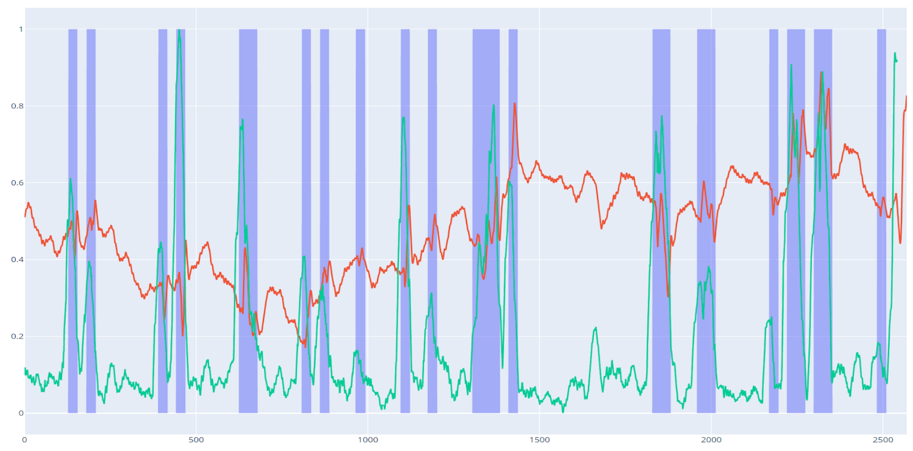


Figure 16: Smoothed LSTM curve (red), variation curve (green), and real anomalies (purple) in Channel 0 of the labelled dataset.

the curve. Hence, we became interested in the notion of variation of variation so that we can detect changes in the variability measure. If we define noise in a signal as what causes the signal values to change, then variation would show us the level of noise in each interval, while variation of variation shows the change in noise levels. Mathematically, it is defined as:

$$\{w_i\}_{i=1}^{N-2B} w_i = \sum_{j=i}^{i+B-1} |v_{j+1} - v_j| \tag{12}$$

where $\{w_i\}$ is the sequence of variation of variation values, B is the length of the interval, and v_j is the j^{th} element of the variation sequence.

When we look at LSTM-smoothed fNIRS signal, we see the curve is noisy throughout but the noise levels vary, with anomalous sections showing the highest change in noise. Figure 17 shows the LSTM-smoothed signal with variation of variation, and Figure 18 shows the variation of the LSTM-smoothed signal alongside variation of variation, both for channel 0 of the dataset. From Figure 17 we see that variation of variation also spikes when there is a spike in the LSTM-smoothed signal, while Figure 18 depicts how the variation of variation curve differs from that of variation. The biggest difference we find is that the top of every spike in the variation curve is represented by a flattening of the variation of variation curve. Moreover, if the variation curve spikes up, down, then back up again in an instant, variation

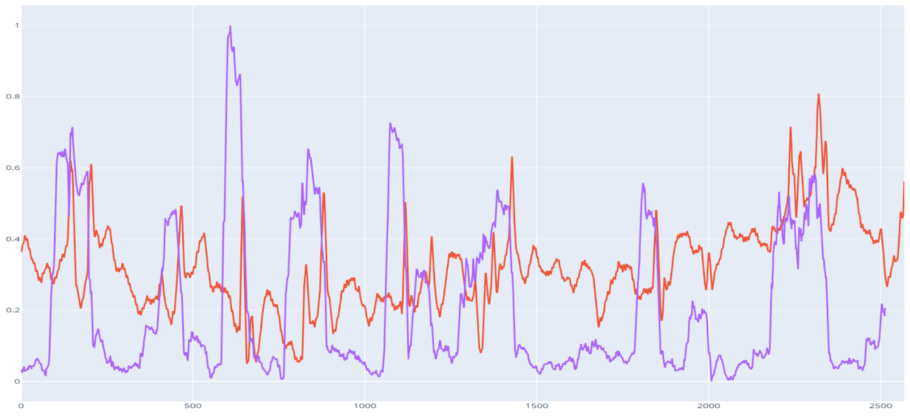


Figure 17: Smoothed LSTM curve (red) and the variation of variation curve (purple) in Channel 0 of the labelled dataset.

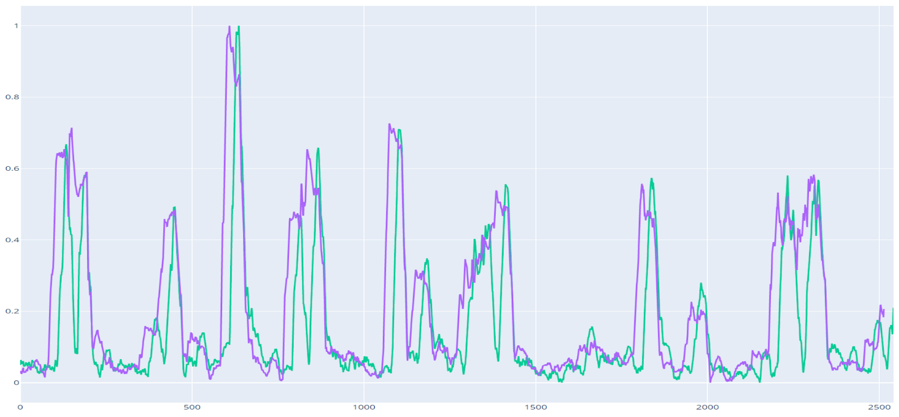


Figure 18: Variation curve (green) and the variation of variation curve (purple) in Channel 0 of the labelled dataset.

of variation is not affected by this change. This behavior gives us an indication that variation of variation can also be used in the process of anomaly detection using our methods.

4 Results and Conclusion

4.1 New Anomaly Detection Methods

As discussed in the methodology section, we decided to use LSTMs and variation to create our own anomaly detection methods. We also noticed that high variation or variation of variation is an indicator of anomalous sections. Thus, for the methods to work they simply need to define a threshold such that all variation or variation of variation values above it correspond to anomalous signal values. With this in mind, we created the following three methods:

- Variation of dataset with a percentile-based threshold
- Variation of dataset with a standard deviation-based threshold
- Variation of variation of the dataset with a standard deviation-based threshold

Using the $F_{1.5}$ score as an accuracy metric, we concluded that all three methods perform similarly, but the most accurate method for this dataset is Method 1: Variation of dataset with a percentile-based threshold. We also used these methods to analyze labelled data from a real life experiment to find that the percentile-based method again performed best. Below we describe in detail the methods stated above and make comparisons between them.

4.1.1 Method 1: Variation with Percentile-based threshold

We can see there is a difference in variation values between anomalous and non-anomalous sections. However, we want to find a threshold such that all variation values above it correspond to anomalous sections. One way to do so is by looking at percentiles of variation. A percentile P denotes all values of a curve that are larger than P percent of all values. We decided to leverage this by marking as anomalous all signal values whose variation is

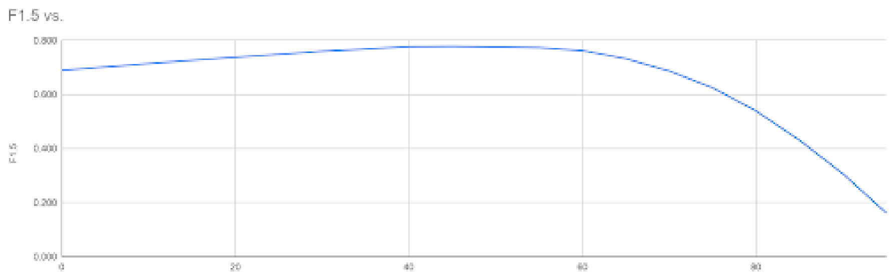


Figure 19: Average $F_{1.5}$ scores of the variation with percentile-based threshold for various thresholds

above a given percentile threshold. Mathematically, the steps in this method are as follows:

1. $\{y_i\}_{i=1}^{N}$: sequence of N signal values predicted by the LSTM
2. $\{v_i\}_{i=1}^{N-B}\ v_i = \sum_{j=i}^{i+B-1} |y_{j+1} - y_j|$: variation curve
3. $\{r_i\}_{i=1}^{N}\ r_i = \begin{cases} 0 & \text{if } v_i < P_{45}(v_i) \text{ or } i > N - B \\ 1 & \text{if } v_i \geq P_{45}(v_i) \end{cases}$

   where $\{r_i\}$ is the resulting array of predicted anomalies structured such that 1 is an anomalous value and 0 is a normal value. We will use the same notation for this array of results throughout the following sections.

The advantage of using percentiles is that sparse very big or small values in the variation curve do not have a big impact on the results of a percentile. To understand the behavior of the algorithm we applied this method to the labelled dataset. We found the q^{th} percentile of variation values and flagged all points with variation larger than that percentile. We then calculated the $F_{1.5}$ score of the results. We repeated the same process for all channels of the dataset and then averaged the results of all channels for each percentile, seen in Figure 19. We also looked at the resulting precision and recall values across various thresholds, shown in Figures 20 and 21.

Figure 19 shows us that even if we mark the entire dataset as anomalous (recall close to 1, happens at the 0^{th} percentile), we still get a $F_{1.5}$ score of 0.69, which is relatively high. Precision at the 0th percentile is 0.4. When we chose the F_β coefficient, we wanted precision to be at least 0.5 when recall =

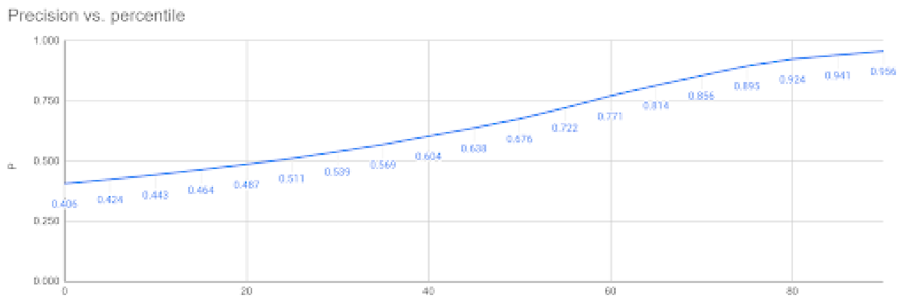


Figure 20: Average precision scores of the variation with percentile-based threshold for various thresholds

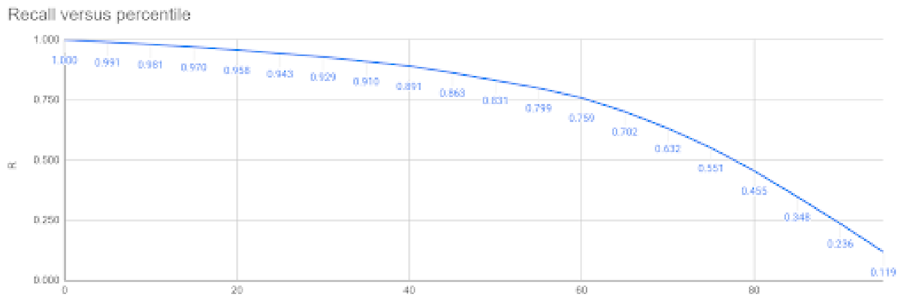


Figure 21: Average recall scores of the variation with percentile-based threshold for various thresholds

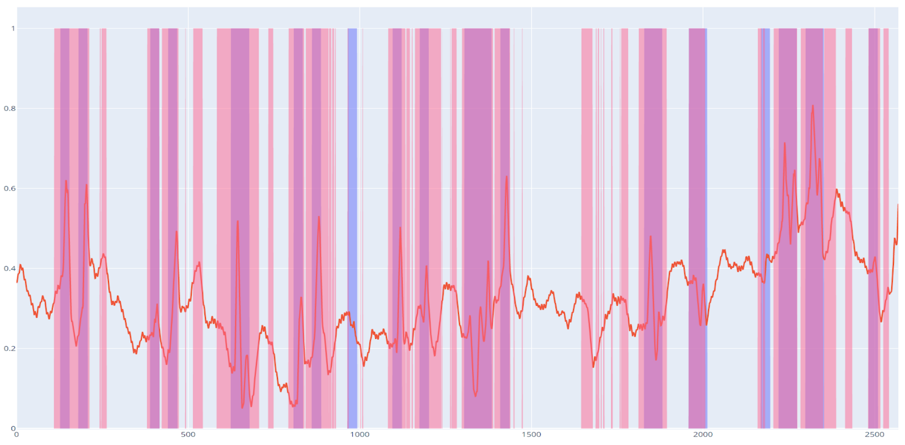


Figure 22: LSTM-smoothed signal values (red) of channel 0 vs real anomalies (purple) and predicted anomalies using 45th percentile (red highlights)

1 in order to get an F_β of 0.7. In this case precision is slightly lower at 0.4, and so is F_β. This means the results for a low percentile shown in this graph are not surprising. As the percentile threshold increases we notice an increase of the $F_{1.5}$ score followed by a steep decrease. The increase in $F_{1.5}$ happens as we reach percentile thresholds where the method flags as anomalous enough values to increase both precision and recall at the same time. $F_{1.5}$ starts to decline when the percentile becomes so high that a very small number of anomalies is being flagged. Even though the flagged values are indeed anomalous (average precision gets closer to 1 with higher percentiles), the recall is extremely low. As expected, this causes a steep decline in F scores. The results of these graphs are hence not surprising, and show us that the highest $F_{1.5}$ values come from percentiles 35 to 55, with the best-performing percentile being 45. A plot of Channel 1 of the labelled dataset with real (blue highlights) and predicted (red highlights) anomalies can be seen on Figure 22.

There is one disadvantage to this method, however. Marking all values above the 45^{th} percentile as anomalous means the method expects 55% of the variation values to be anomalous. Thus, this method would likely take into consideration the number of anomalies the user expects the dataset to contain. The user is most likely to not have this information in a real-life scenario. Moreover, the fact that the 45^{th} percentile was a good fit for this dataset does not mean it will be a good fit for others that might contain a bigger or smaller number of anomalies.

4.1.2 Method 2: Variation with standard deviation-based threshold

Since using percentiles as a way to determine the threshold of anomalous values is dependent on the user estimating the fraction of anomalies in a dataset, we turned to other ways to set a threshold. Another method of determining the values in a curve that deviate from the normal range of values is standard deviation. Standard deviation measures the amount of variation of a set of values, and when applied to the variation curve is expressed using the formula below:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(v_i - \mu)^2} \tag{13}$$

where σ is standard deviation, v_i is the i^{th} item of the variation array, and μ is the average of all variation values.

A lower standard deviation means the values of the set are generally all close to their mean, while a higher standard deviation means the values of the set are spread further away from the mean. We can use standard deviation to set a threshold above which all variation values are considered anomalous. The steps of this algorithm are shown below, where the first two steps are the same as in the previous method:

1. $\{y_i\}_{i=1}^{N}$: sequence of N signal values predicted by the LSTM
2. $\{v_i\}_{i=1}^{N-B}\ v_i = \sum_{j=i}^{i+B-1} |y_{j+1} - y_j|$: variation curve
3. $\{r_i\}_{i=1}^{N}\ r_i = \begin{cases} 0 & \text{if } v_i < \mu + T\sigma \text{ or } i > N-B \\ 1 & \text{otherwise} \end{cases}$

   where N is the number of data points in the dataset of signal values, v_i is the array of variation values, μ is the mean of v_i, σ is the standard deviation of v_i, and T is a threshold coefficient by which the standard deviation is multiplied.

To understand the behavior of this method, we calculated anomalies using various possible values of T ranging from -4 to 6, looking for the threshold

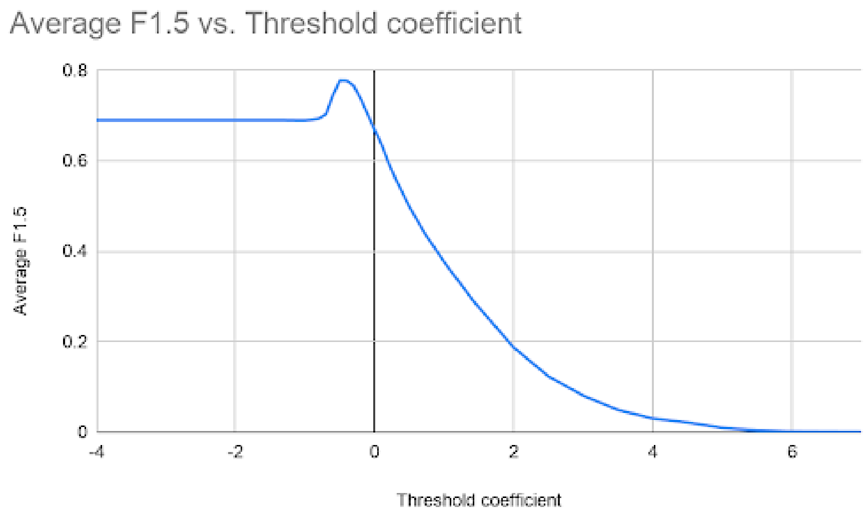


Figure 23: Average $F_{1.5}$ scores of the variation with standard deviation-based threshold for various threshold coefficients

that corresponded with the highest $F_{1.5}$ score. We used negative threshold values to label signal values that are greater than $T\sigma$. We averaged the $F_{1.5}$ scores of all 16 channels for each T value. Figure 23 shows the results of this operation, where it is clear that -0.5 is the best-performing T value.

The behavior observed in Figure 23 above is expected: when the threshold coefficient is low, more data points are flagged as anomalous, increasing the value of recall and decreasing precision. We notice $F_{1.5}$ stays constant until about $T = -0.8$. This is because the entire dataset has been labelled as anomalous, which has increased recall to 1 and decreased precision to about 0.4. Since the number of data points in the set is limited, the recall cannot go higher and precision cannot get lower than these values, keeping $F_{1.5}$ constant. As the threshold increases, less data points get flagged as anomalous because points are flagged that are increasingly farther from the interval where most data points are expected to be. This causes higher precision and lower recall. As the threshold increases, $F_1.5$ decreases to the point where it becomes 0. This happens when the threshold is so high that no anomalies get flagged, bringing the number of true positives, and hence precision and recall, to 0. From Figure 23 it seems that this happens when the threshold is 6, while the balance between flagging too many and too little anomalies is located at -0.5. It is also expected that we get a negative threshold. This dataset has many anomalous values which increase the average of the dataset. The rest of the data points are really small in value compared to the anomalous points, which means that when many anomalous sections begin the variation values have not yet reached the average of the variation curve. This method is less dependent on previous knowledge about the number of real anomalies in the dataset, making it more versatile.

4.1.3 Method 3: Variation of variation with standard deviation-based threshold

We mentioned in the methodology section that on data that exhibits fluctuations between noisy and smooth, variation of variation might perform better than variation alone. Since the standard deviation-based method requires less a priori knowledge than the percentile-based method, we decided to apply this method to detect anomalies with variation of variation. The steps for this method are outlined below:

1. $\{y_i\}_{i=1}^{N}$: sequence of N signal values predicted by the LSTM
2. $\{v_i\}_{i=1}^{N-B}\, v_i = \sum_{j=i}^{i+B-1} |y_{j+1} - y_j|$: variation curve
3. $\{w_i\}_{i=1}^{N-2B}$: $w_i = \sum_{j=i}^{i+B-1} |v_{j+1} - v_j|$: variation of variation curve
4. $\{r_i\}_{i=1}^{N}\, r_i = \begin{cases} 0 & \text{if } v_i < \mu + T\sigma \text{ or } i > N - B \\ 1 & \text{otherwise} \end{cases}$ where N is the number of data points in the dataset of signal values, v_i is the array of variation values, w_i is the variation of variation μ is the mean of w_i, σ is the standard deviation of w_i, and T is a threshold coefficient by which the standard deviation is multiplied.

We generated performance results of this method in the same way as for the previous two. Figure 24 shows the resulting $F_{1.5}$ score for various coefficients used to create the threshold, between -4 and 6. This graph shows the same pattern of $F_{1.5}$ scores across thresholds as graphs for Method 2 where a standard deviation-based threshold is applied to variation. This behavior is expected since the variation of the variation curve follows about the same pattern as variation. The best-performing coefficient is expected to be slightly different since the variation of variation curve does take different

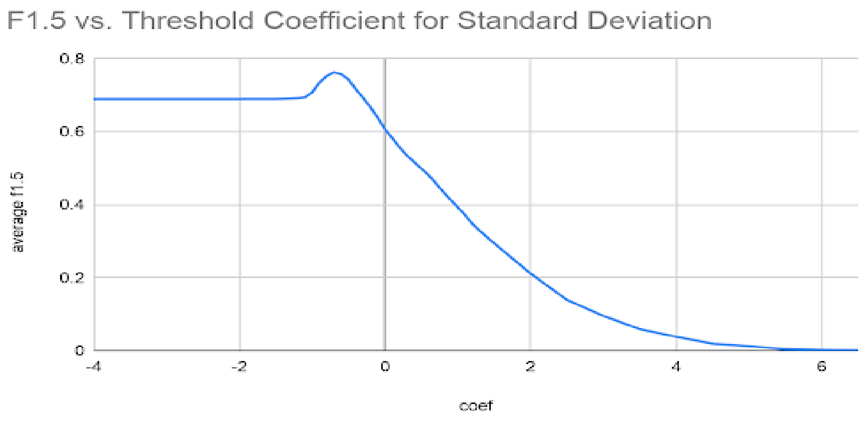


Figure 24: Average $F_{1.5}$ scores of the variation of variation with standard deviation-based threshold for various threshold coefficients

values than the variation curve. This slightly changes the mean and standard deviation, hence affecting the results for each threshold value.

From Figure 24 we can see the method performs quite well for coefficients T in the range of -1 to -0.5, with -0.7 being the best-performing coefficient. For the same reasons as explained in the results of Method 2, we are not surprised to see a negative value for threshold as the best performing coefficient. Figure 25 shows the results of this method applied to channel 0 of the artificial anomalies dataset, where the curve is variation of variation, the purple highlights are real anomalies, and the red highlights are predicted anomalies. We can see that the method flags as anomalous almost all regions that are truly anomalous, but goes beyond that to label the areas around these anomalies as well. Given that we created a buffer zone (see previous section) to widen anomalous sections, we expect and prefer this behavior to labeling only the truly anomalous regions.

4.2 Comparing performance across methods

After generating performance metrics for all methods, we were interested in finding which method performs best on the dataset of artificial anomalies. Although this does not guarantee that the same method will give the best performance on other datasets, it can still help us make comparisons between methods and generate a recommended one for similar datasets. Table 2 shows

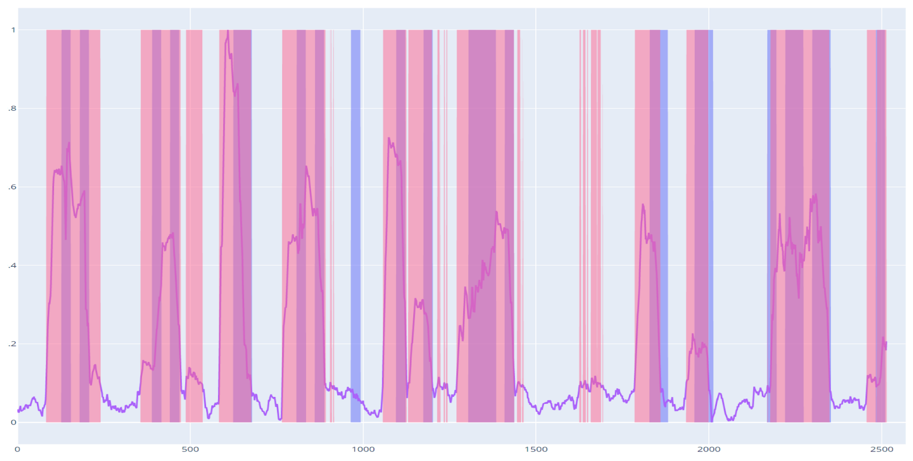


Figure 25: Variation of variation (purple) vs real anomalies (purple highlights) and predicted anomalies (red highlights) fr channel 0 analyzed using variation of variation with standard deviation-based threshold

Table 2: Best performing thresholds for analyzing LSTM-smoothed signal values in the labelled dataset for each method and the respective $F_{1.5}$ scores

Method No	Method type	$F_{1.5}$ score
1	Variation + Median	0.779
2	Variation + Mean + Stdev	0.778
3	Variation of Variation	0.763

the average $F_{1.5}$ scores of the three methods discussed above.

Table 2 shows that there is not a big difference in performance for all methods. As expected, the first method, which uses variation and a median-based threshold, performs best. However, we previously mentioned that this could also come from the fact that we have previous knowledge about this dataset's number of anomalous points and length of each anomalous interval. Without prior knowledge of the anomalies of a dataset, the variation method with standard deviation-based thresholding performed best.

Table 3: Best performing thresholds for analyzing real fNIRS signal values of the labelled dataset for each method and their respective $F_{1.5}$ scores

Method No	Method type	$F_{1.5}$ score
1	Variation + Median	0.722
2	Variation + Mean + Stdev	0.713
3	Variation of Variation	0.722

4.2.1 Performance of methods using raw fNIRS signal

We generated the results for the above methods using the LSTM-smoothed signal values as they were easier to analyze and describe. However, training an LSTM could potentially be a really time-consuming task. Hence we were interested to see if these methods would perform as well on the real signal values of the dataset of artificial anomalies. Table 3 shows the average $F_{1.5}$ score of each method computed on the real signal values of each channel in this dataset.

Although the performance metrics are not as good as when LSTMs are used, they are still quite acceptable suggesting that these methods can be used on raw signal as well.

4.2.2 Performance of methods using a real-life dataset

The results described in the sections above are generated using a very clean dataset that was created specifically to help us develop these methods. We were also interested in trying these methods on a dataset generated through a real experiment, namely the SART experiment conducted at Drexel university by Solovey et al. This experiment was conducted to understand when users get bored or distracted while doing a task. We used the datasets of 6 channels from one patient, each of which contains 13697 points. An expert in the field labelled anomalies in each of these channels, with each anomalous interval being 100 points. We then expanded the true anomalous sections by 10 data points on each side as explained in the Methodology section. Figure 26 shows LSTM-smoothed signals of each channel with their respective labelled anomalies.

To collect the LSTM-smoothed sets for each channel, we trained an LSTM

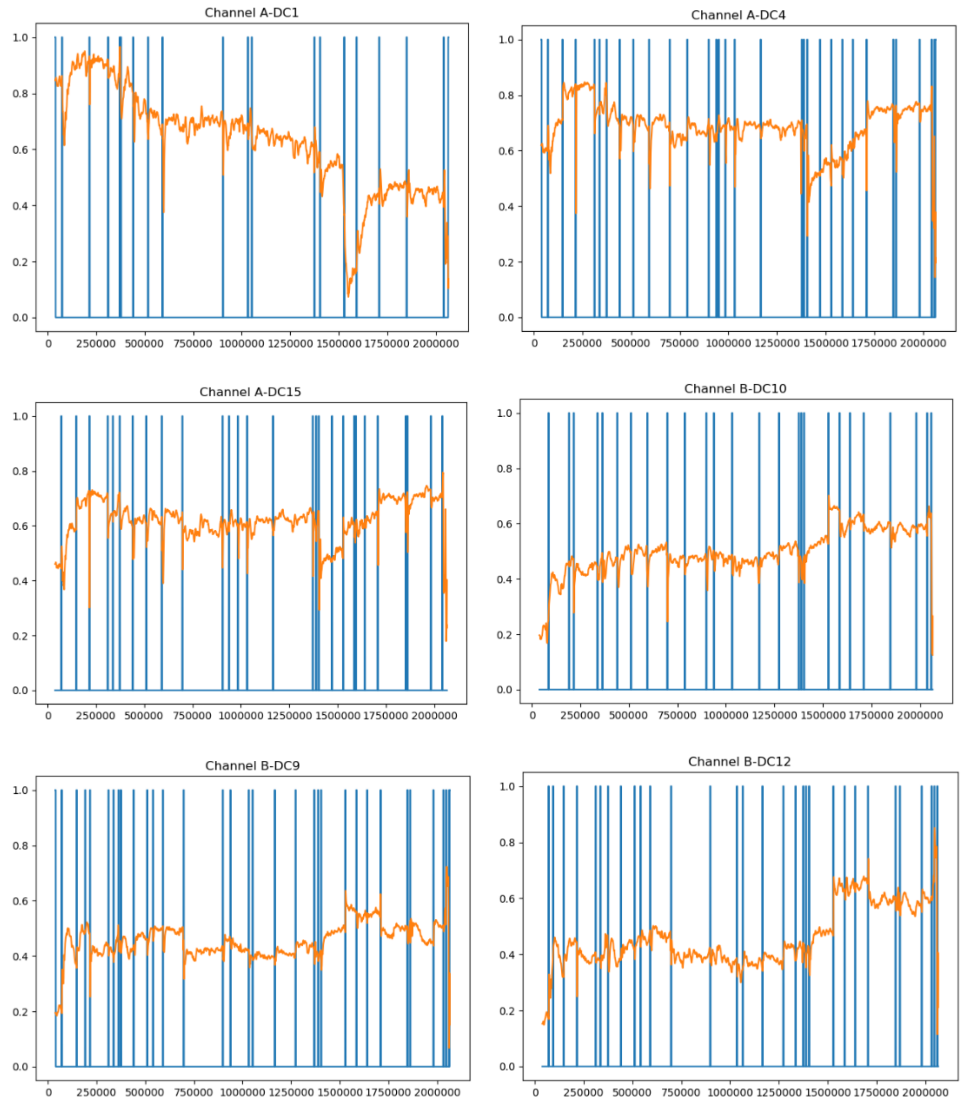


Figure 26: Real anomalies of the 6 labelled channels from the SART dataset

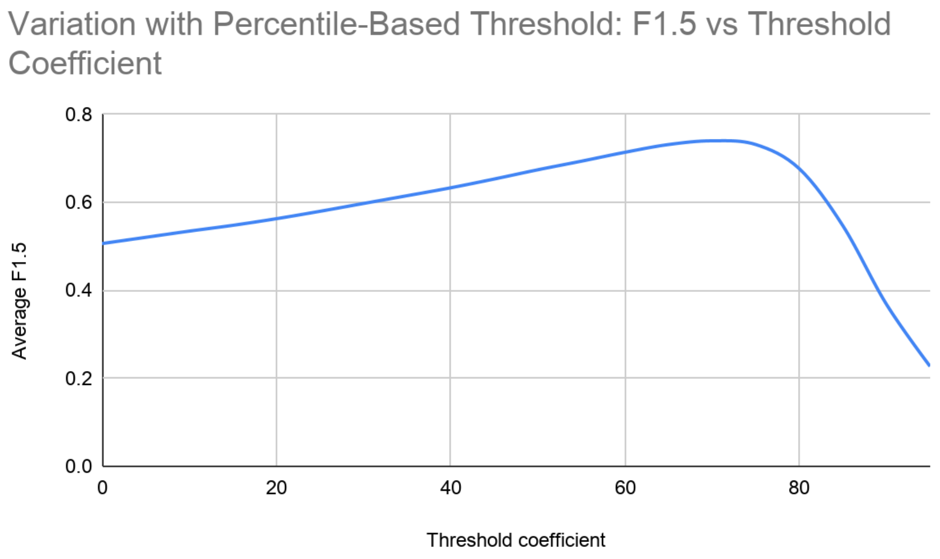


Figure 27: Average $F_{1.5}$ of SART channels calculated with the percentile-based method

model on the first channel of the dataset, and used it to predict the signal value of all other channels. We continued to use an interval of 28 data points to calculate variation and variation of variation. We then ran each method on all channels for various thresholds, collected the $F_{1.5}$ scores, and averaged the scores of all channels to generate one $F_{1.5}$ score per threshold value. The graphs of these results and their explanations can be seen below.

Performance of Method 1 on SART data: Variation with Percentile-based Threshold

The results from variation with a median-based threshold on SART data resemble those of the previous dataset. Although the best $F_{1.5}$ score is achieved at a higher percentile, the 70th, this is expected as the ratio of normal to anomalous points changes from the previous dataset.

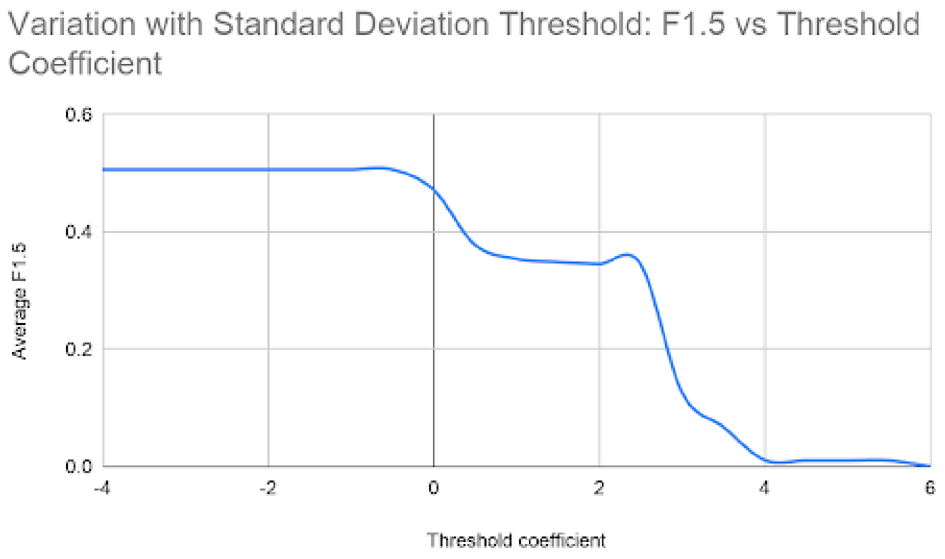


Figure 28: Average $F_{1.5}$ of SART channels calculated with the standard deviation-based method

Performance of Method 2 on SART data: Variation with Standard Deviation-based Threshold

The curve of Figure 28 1 shows the $F_{1.5}$ scores for method 2 calculated using various threshold coefficients with SART data. This curve is very different from that of the $F_{1.5}$ scores generated by running the same method on the first dataset. This might result from the fact that the anomalies seem to cause higher spikes in variation compared to normal values. Figure 29 side by side comparisons of variation curves of a channel from the first and second datasets.

From Figure 28 we see that most anomalies represented by variation values spike to about 0.5. However, there are some that go beyond 0.5. When the coefficient is low, the method labels more values as anomalous, flagging all visible spikes in variation as anomalies. Thus we see higher $F_{1.5}$ scores when the threshold coefficient is less than zero. As this coefficient increases, less values are marked as anomalous. The first drop in the $F_{1.5}$ curve values can be attributed to the threshold values increasing to above 0.5 and the method labelling as normal all but the highest spikes. The second drop

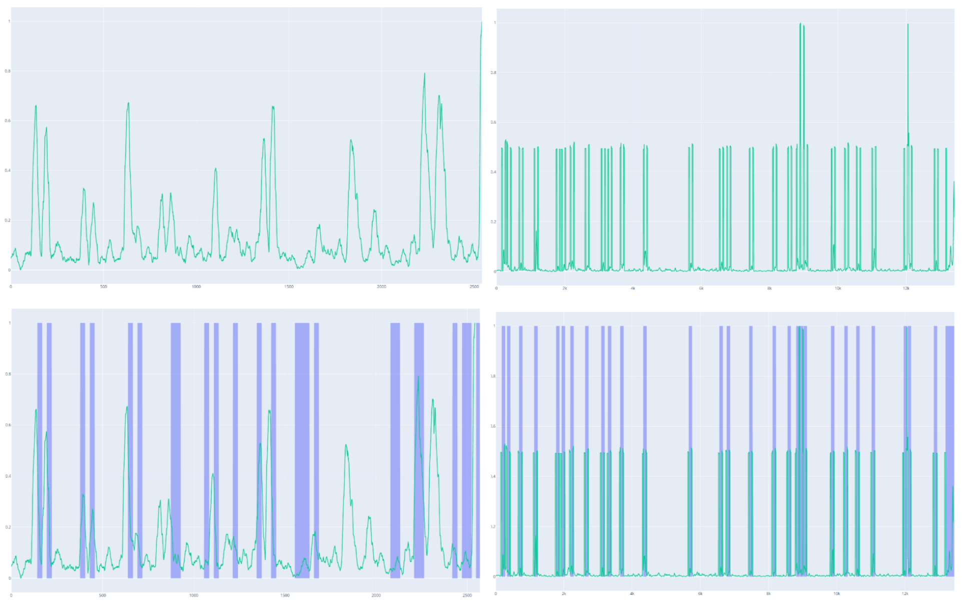


Figure 29: Comparison between variation (green) and real anomalies (purple) of one labelled dataset of artificial anomalies (left) and SART (right)

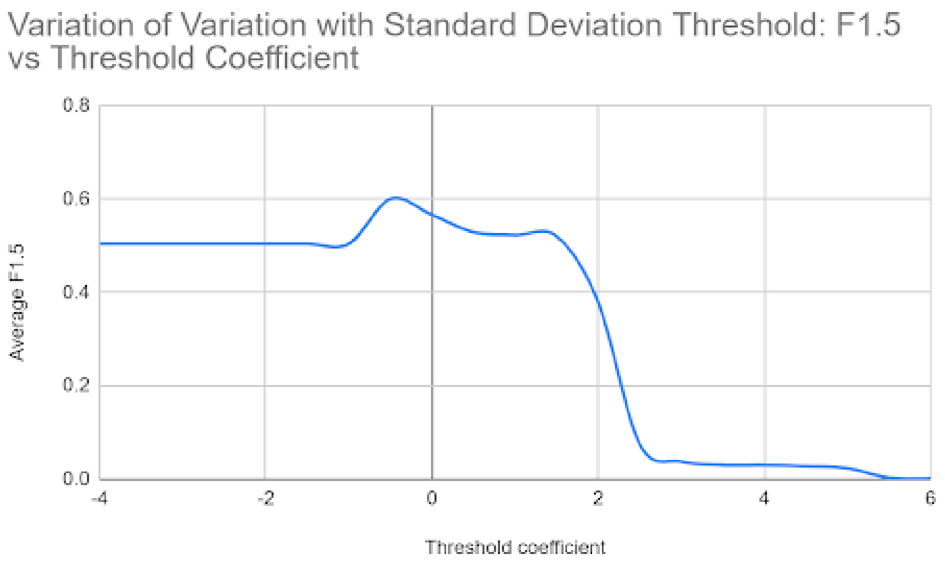


Figure 30: Average $F_{1.5}$ of SART channels calculated with the variation of variation with standard deviation-based threshold method

can be attributed to the fact that the threshold value approached 1, at which point no values were labelled as anomalous giving us an $F_{1.5}$ score of virtually zero.

Performance of Method 3 on SART data: Variation of Variation with Standard Deviation-based Threshold

Figure 30 represents $F_{1.5}$ scores generated while using method 3 with varying threshold coefficients on the SART dataset. The shape of this graph also does not resemble that of $F_{1.5}$ scores of the same method computed on the first dataset. We expect the reason for this behavior to be similar to that of Method 2. As the threshold increases, slightly less anomalies are labeled. Since the anomalies that were re-labeled as normal were not real anomalies, re-labeling them as normal increases the $F_{1.5}$ score. This is the case for threshold coefficients between -1 to 1.7. Threshold coefficients above 1.7 result in threshold values increasingly closer to 1, meaning only a very small number of anomalies to no anomalies are labelled. This behavior causes the drop of $F_{1.5}$ scores when the threshold coefficient goes beyond 2.5.

Overall, we see slightly worse $F_{1.5}$ scores when running all three methods on this dataset. However, the methods' performance is still acceptable. Further improvement of these methods could include shifting the anomalies detected using variation and variation of variation by B or $2B$ elements respectively, where B indicates the interval at which the variation values were computed.

4.3 Relevance of method to users of anomaly detection toolkit

From these results we suggest that users opt for Method 1 if they know the expected number of anomalous regions in their dataset. They can decide on a threshold based on the percentage of the entire dataset that the real anomalies take. We expect method 2 to work well for most datasets. If the user has some labeled data, we suggest they run a comparison of $F_{1.5}$ scores for various coefficients to find the one that will perform best on their dataset. If this option is not available then the user can use our suggested coefficient, -0.5. Method 3 is useful when part of the data contains normal sections whose values change very rapidly.

4.4 Anomaly detection platform

We aimed to make the anomaly detection algorithms described above available and easy to use by analysts as well as users with little technical knowledge. To do so we first needed to understand who would benefit most from using these algorithms. First and foremost, we have scientists who are interested in getting anomaly detection results on their experiments as easily and quickly as possible. Then there are developers who are working on larger anomaly detection tools, want to automate the process, or for other reasons want to programmatically interact with our tool. As these two categories of users have different requirements and will interact with our tool in very different ways, we decided to create two different products: an Access Point Interface (API) to cater to developers, and a website directed towards the rest of the public. Both these products have the same functionality, but differ in how the user interacts with them to get the desired result. We also needed to create user manuals to help users get started with and navigate our tool. This resulted in the following three main deliverables:

1. API
2. Website
3. User Guide

These deliverables can be found in this Github repository. In the following sections we discuss in higher detail the workflows of the API and website, tools used to make it possible, and finally we provide examples of our tools at work.

4.5 Workflow of API

The main purpose of the Anomaly Detector API is to detect anomalies in user-submitted datasets and return them formatted as binary arrays where 1 is an anomalous value, and 0 is a normal value. To do so, the API goes through the following steps. Figures of each of these steps can be seen in the demonstration that follows:

1. Enable users to input a JSON object to indicate submission of an anomaly detection job. The JSON object will contain the following information. Figure 31 shows a sample user input. There are no default values for any of them except the algorithm parameters:
   - **job_name**: The name of this anomaly detection job
   - **job_description**: The description of this anomaly detection job
   - **test**: An array containing paths to .csv files with signal values of all channels to be analyzed
   - **job_type**: The algorithm to use. The options are Telemanom, Variation with Percentile-based Threshold, Variation with Standard Deviation-based Threshold
   - **signal_type**: Whether or not LSTMs will be used to smooth the curve, and whether the LSTM model will be newly trained or is a previously trained model. The options are raw, LSTM-new, LSTM-prev
   - **alg_params**: Algorithm parameters. Default values for these parameters seen in Appendix B.

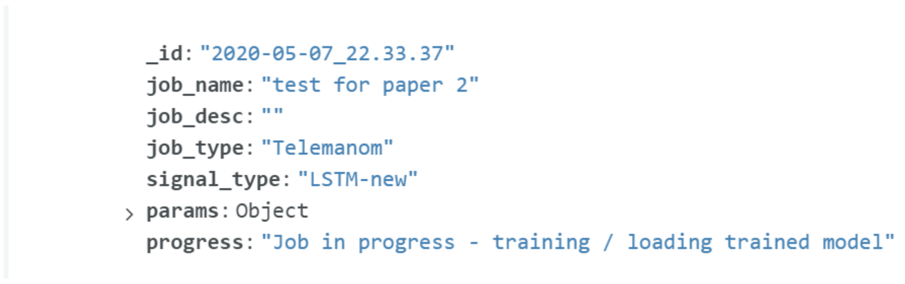
```
{
      "job_name": "testing for api",
      "job_desc": "",
      "job_type": "Variation with Percentile-based Threshold",
      "signal_type": "LSTM-new",
      "params": {
            "sets": {
                  "types": {
                        "train": "user_uploads",
                        "test": "user_uploads"
                  },
                  "train":["C:\\Users\\Petra Kumi\\Desktop\\0.csv",
                        "C:\\Users\\Petra Kumi\\Desktop\\1.csv"],
                  "test": ["C:\\Users\\Petra Kumi\\Desktop\\2.csv",
                        "C:\\Users\\Petra Kumi\\Desktop\\3.csv"]
            },
            "times": {
                  "type": "times",
                  "param_1": 0,
                  "param_2": 2821
            },
            "alg_params": {}
      }
}
```

Figure 31: Sample JSON for submission of a new anomaly detection job

- **train**: An array containing the training set data. If the user is not training a new LSTM model, they can leave the array as empty
- **times**: A dictionary with three keys: type, param_1, and param_2. The value for type has to be times, while param_1 and param_2 are indices of data points in the set. All points between param_1 and param_2 in the dataset will be used to train the model
- **prev_model_id**: The ID of a previously trained LSTM model. This parameter is only necessary if the user wants to use a previously trained LSTM model by choosing LSTM-prev for the signal_type parameter.

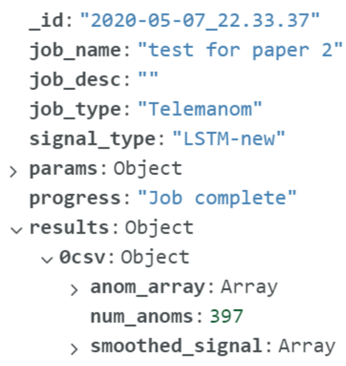
2. Parse input to obtain necessary information

3. Start anomaly detection job



```
  _id: "2020-05-07_22.33.37"
  job_name: "test for paper 2"
  job_desc: ""
  job_type: "Telemanom"
  signal_type: "LSTM-new"
> params: Object
  progress: "Job in progress - training / loading trained model"
```

Figure 32: Sample job in database with a Job in Progress status



```
  _id: "2020-05-07_22.33.37"
  job_name: "test for paper 2"
  job_desc: ""
  job_type: "Telemanom"
  signal_type: "LSTM-new"
> params: Object
  progress: "Job complete"
v results: Object
  v 0csv: Object
    > anom_array: Array
      num_anoms: 397
    > smoothed_signal: Array
```

Figure 33: Sample job in database with a Job Completed status

4. As job progresses, update its progress on a database, as seen in Figure 32

5. If job completes successfully, update the job with a results parameter as seen in Figure 33, containing the following:

   (a) **anom_array**: A binary array of results for each submitted channel, where 1 represents an anomalous point and 0 represents a normal point

   (b) **smoothed_signal**: The raw signal values or LSTM-smoothed values, depending on whether user chose to use LSTMs

   (c) **num_anoms**: The total number of anomalies found

6. If job fails, update the progress parameter with Job interrupted - error

Once the job is initialized, the API returns to the user the unique ID of that anomaly detection job. Alongside the basic job submission functionality, the API offers users the option to check the status and results of any previously submitted anomaly detection jobs. To do so, the user has to submit a separate request to the API and provide the name or ID of their request. Below we give brief descriptions of the main steps the API follows to complete the user's requests of submitting a new job or asking for the status of a previously submitted one.

4.5.1 Submitting a new anomaly detection job

Once the user submits the json object shown above in the form of a POST request to /submit_request, the API parses the JSON to see if it is a valid request to start an anomaly detection job. Then, the API retrieves the .csv files from the paths the user supplied, parses the file data, and saves the content as arrays of values in internal memory. The .csv file has to contain at least two columns, the first one containing signal values, and the others containing event values if there are any. If the user does not have or want to include events with their signal values, then the second column should be filled with zeros. If the arrays follow this pattern, the API assigns the job a unique ID, and places it in a queue. It then returns to the user a successful submission message and this job ID. If no other jobs are currently running, this job will be initialized. Otherwise, it will wait in the queue until all previously submitted jobs are finished. This is done to ensure that no code is being manipulated by two different processes at the same time. It also allows the user to keep interacting with the API without having to wait for any processes to finish. This means they can request to view previous submissions or submit a new job while the previous jobs are still waiting or being processed.

Parsing algorithm parameters from user input

Each anomaly detection method requires a set of parameters to properly be executed. Although the API offers the user the option to to input values for these parameters, they can also choose not to. In that case, the API will simply use a set of default values. These default values are saved in a .yaml file in the code. When a new job is about to start, the contents of this file are parsed, then updated with the values from the user's input. The default values for the algorithms used in our code can be found in Table 1 above.

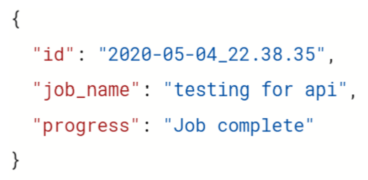
Preparing data for use with anomaly detection algorithms

Once a given anomaly detection job leaves the queue and starts running, the first step to execute is reformatting the signal values to fit the appropriate anomaly detection method. It has been shown in the previous section that the API initially saves the datasets inputted by the user as arrays. These arrays are now manipulated in various ways depending on whether the user wants to use LSTM smoothing on their data. If smoothing is performed, each element of each array gets normalized using minmax normalization. If a new LSTM model will be trained, all arrays selected for training will be merged. Then, the generated model will get used to predict values on each array of the provided testing set. If a new model will not be trained, then the model selected by the user will be used to make the predictions. If no LSTM smoothing is performed, the code simply normalizes the data arrays between 0 and 1.

Running anomaly detection algorithms and saving results to the database

After the successful manipulation of the data and algorithm parameters, the API calls the anomaly detection algorithm and submits the parameters of this job on the database, with the Job initialized status. As the algorithm runs, it updates this database entry with its progress. When the job is completed, the database entry is updated with the status Job complete, and the submission on the jobs collection of the database is updated. This object includes the unique id of the job, its name and description, the parameters used, as well as for each dataset its filename, predicted signal value, array of anomalies, and number of anomalous intervals.

If the job also required training a new LSTM model then the resulting .h5 file is saved in another cluster of the database alongside the name and ID of the current job. When a user wants to predict using a previously trained model, the API looks in this cluster for a model with a matching name or ID. The user can also see details associated with all the previously saved models, including their IDs, by submitting a request to the API.



```
{
  "id": "2020-05-04_22.38.35",
  "job_name": "testing for api",
  "progress": "Job complete"
}
```

Figure 34: Returned detail and status after requesting job progress

4.5.2 Requesting status of a given job

The user can also request the status of a submitted job. To do so, they have to submit a POST request to /job_progress, with the request body being a JSON with a single key called id, and the value being the ID of the job. The API looks for such a job in the database, and if one is found it returns to the user its details and status, as seen in Figure 34.

The following are the possible status messages:

1. Processing user input
2. Preparing datasets
3. Job queued
4. Job initialized
5. Job in progress - preparing sets for LSTM
6. Job in progress - training / loading trained model
7. Job in progress - predicting output
8. Job in progress - calculating errors
9. Job in progress - calculating anomalies
10. Job complete
11. Job interrupted - error

4.5.3 Retrieving all details of a given job

To get all information relating to a given job, the user has to submit a POST request to /job_details, with the request body being a JSON with a single key called id, and the value being the ID of the job. The API requests the job with the given ID from the database and returns it to the user in the same format as shown in Figure 33.

4.5.4 Retrieving all submitted jobs

By submitting a POST request to /get_jobs the user can receive all previously submitted jobs, regardless of whether they have been completed or not. The returned result is an array of JSON objects containing information about all jobs in the database, again in the same format as in Figure 33.

4.5.5 Retrieving all saved LSTM models

The user is also able to get information about all saved LSTM models by submitting a POST request with the same body structure as above to /get_saved_models.This returns all information stored in the database for all jobs whose LSTM models have been stored.

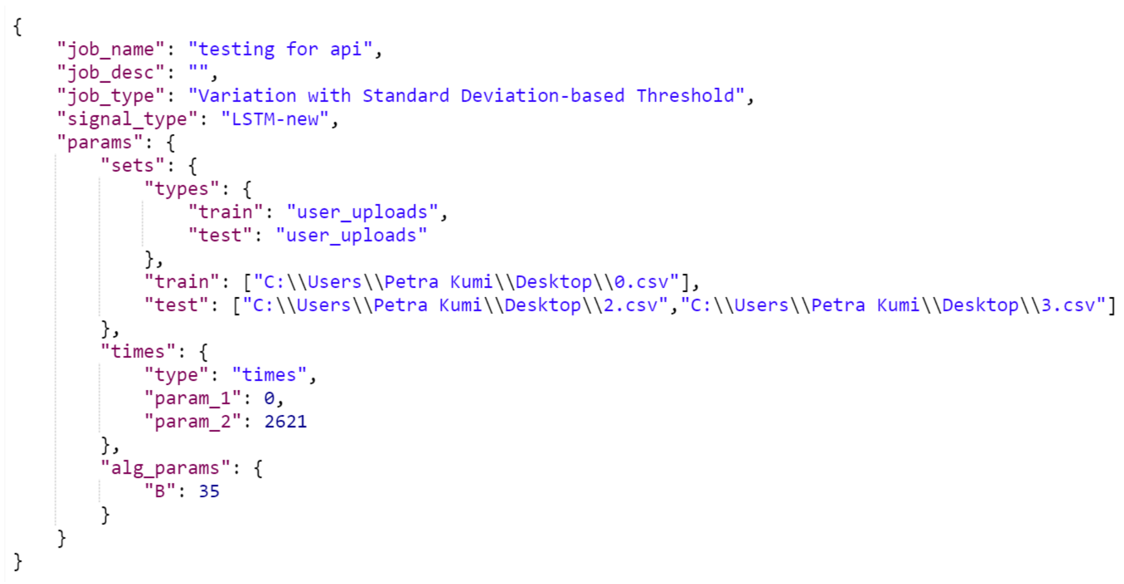
4.5.6 Demonstrating the job submission process

To demonstrate the API's functionality we will submit a sample job running Variation with Standard Deviation-based Threshold on an LSTM-smoothed signal from a newly trained model.

1. We submit a request containing the information seen in Figure 35 to the /submit_request url endpoint. Once the request is submitted, we receive a response like in Figure 36:

2. To understand how the job is being processed we can see the logs outputted in the backend. First, we see the parameters in Figure 37 will be used. Since we only added the B parameter, all others were set as default.

3. Then, we see a progress update followed by starting the LSTM



```
{
    "job_name": "testing for api",
    "job_desc": "",
    "job_type": "Variation with Standard Deviation-based Threshold",
    "signal_type": "LSTM-new",
    "params": {
        "sets": {
            "types": {
                "train": "user_uploads",
                "test": "user_uploads"
            },
            "train": ["C:\\Users\\Petra Kumi\\Desktop\\0.csv"],
            "test": ["C:\\Users\\Petra Kumi\\Desktop\\2.csv","C:\\Users\\Petra Kumi\\Desktop\\3.csv"]
        },
        "times": {
            "type": "times",
            "param_1": 0,
            "param_2": 2621
        },
        "alg_params": {
            "B": 35
        }
    }
}
```

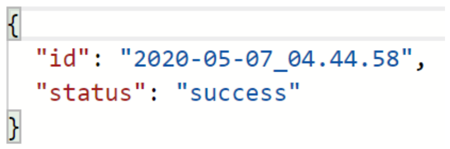
Figure 35: JSON of sample request submission

4. Afterwards, the model gets trained. After it is done training, it updates progress and predicts outputs for the user-defined channels. We can see this in the logs in Figure 39

5. Once predictions have been made, the smoothed arrays are sent to the Variation with standard deviation method and anomalies are calculated separately for each channel. The progress update seen in Figure 40

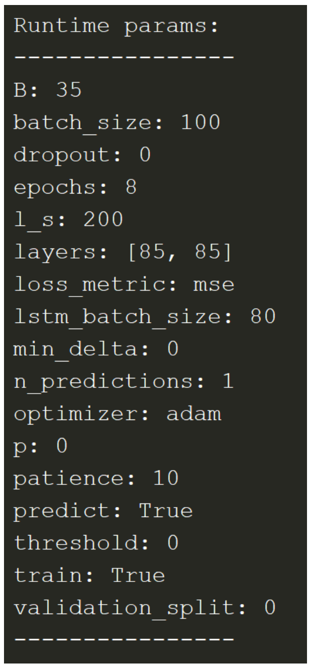
6. Once this step is also completed, our algorithm is finished. We can see this through the following output as well.

   In the meanwhile, the job has been saved to the database and we can retrieve its information through the API through a POST request to /job_details. Figure 42 shows the body of the request and Figure shows a snippet of its returned response.



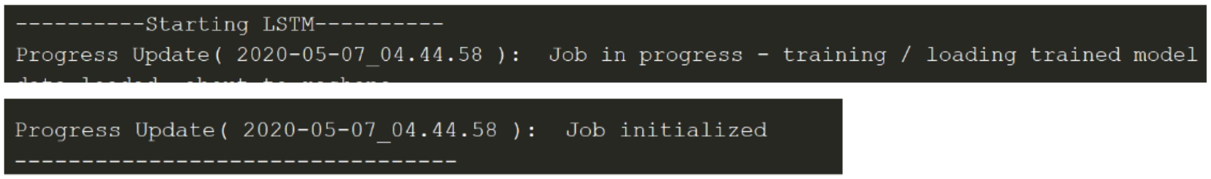
```
{
  "id": "2020-05-07_04.44.58",
  "status": "success"
}
```

Figure 36: Response to request submitted in Fig. 35



```
Runtime params:
----------------
B: 35
batch_size: 100
dropout: 0
epochs: 8
l_s: 200
layers: [85, 85]
loss_metric: mse
lstm_batch_size: 80
min_delta: 0
n_predictions: 1
optimizer: adam
p: 0
patience: 10
predict: True
threshold: 0
train: True
validation_split: 0
----------------
```

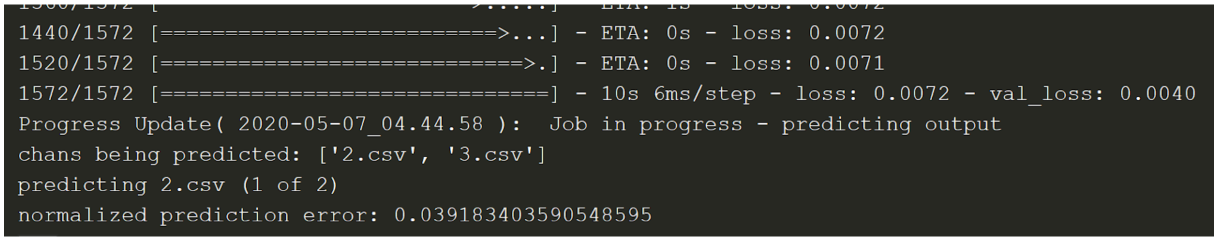
Figure 37: Runtime parameters when running an anomaly detection job



```
----------Starting LSTM----------
Progress Update( 2020-05-07_04.44.58 ):  Job in progress - training / loading trained model
```

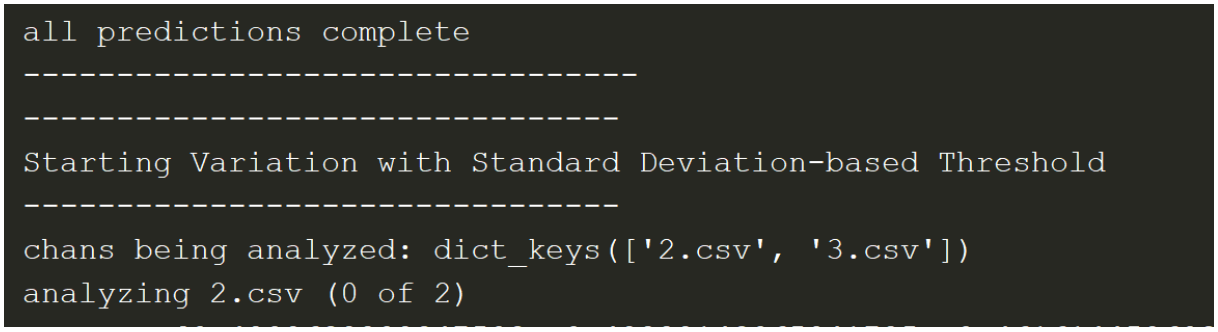
```
Progress Update( 2020-05-07_04.44.58 ):  Job initialized
---------------------------------
```

Figure 38: Progress update of job to ”Starting LSTM”



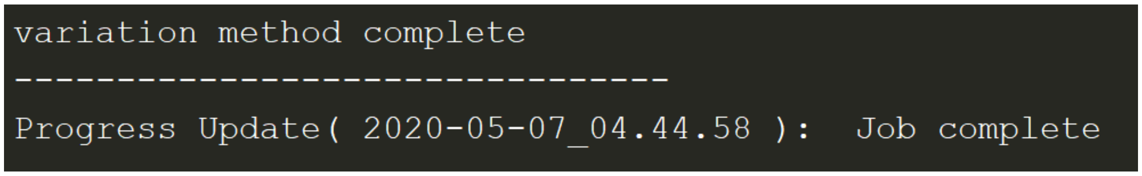
```
1440/1572 [========================>...] - ETA: 0s - loss: 0.0072
1520/1572 [==========================>.] - ETA: 0s - loss: 0.0071
1572/1572 [============================] - 10s 6ms/step - loss: 0.0072 - val_loss: 0.0040
Progress Update( 2020-05-07_04.44.58 ):  Job in progress - predicting output
chans being predicted: ['2.csv', '3.csv']
predicting 2.csv (1 of 2)
normalized prediction error: 0.039183403590548595
```

Figure 39: Job progress update to predicting output



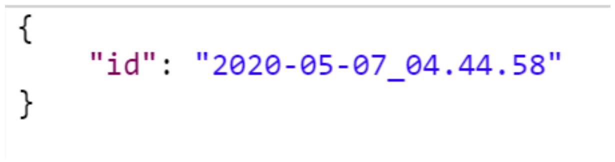
```
all predictions complete
---------------------------------
--------------------------------
Starting Variation with Standard Deviation-based Threshold
--------------------------------
chans being analyzed: dict_keys(['2.csv', '3.csv'])
analyzing 2.csv (0 of 2)
```

Figure 40: Job progress update to starting anomaly detection algorithm



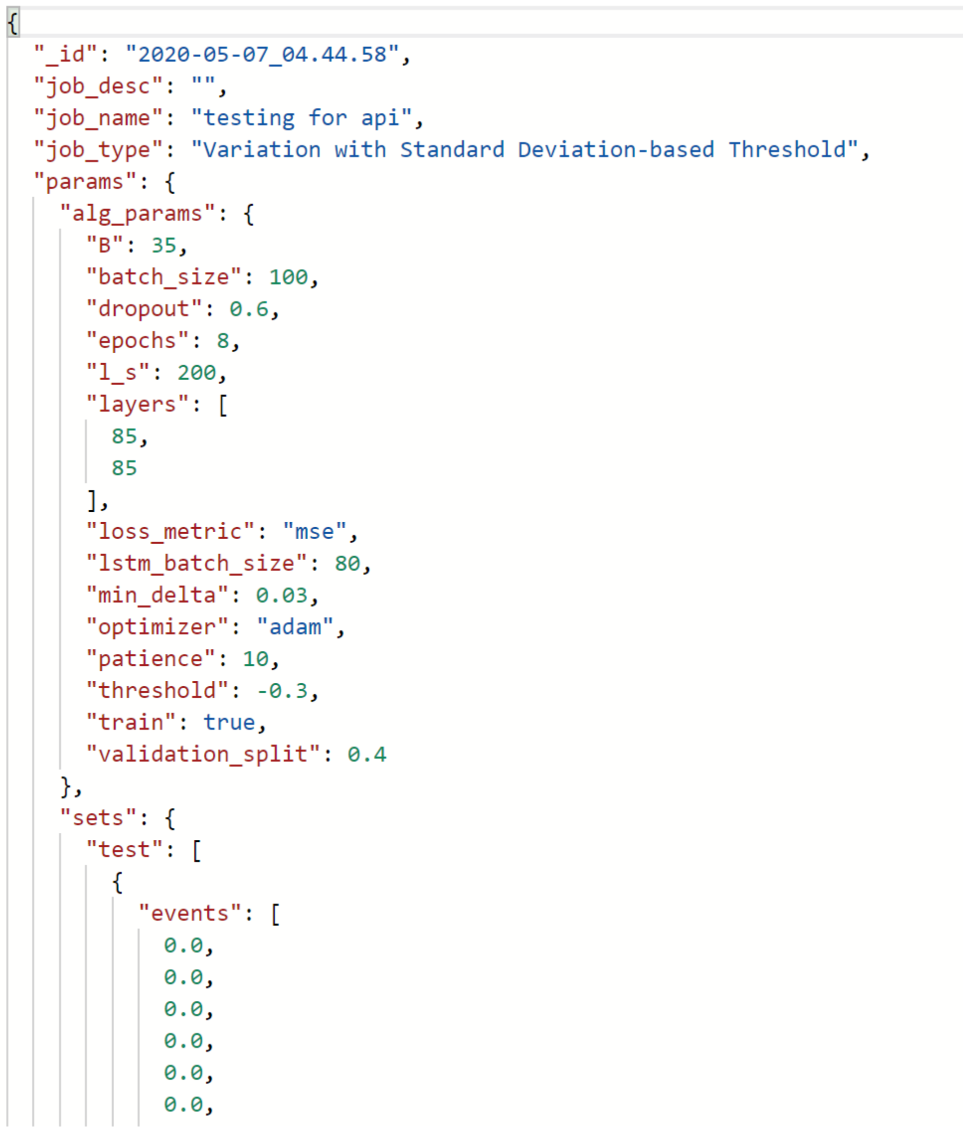
```
variation method complete
-----------------------------------
Progress Update( 2020-05-07_04.44.58 ):  Job complete
```

Figure 41: Job progress update to job complete



```
{
    "id": "2020-05-07_04.44.58"
}
```

Figure 42: Body of request submitted to /job_details url



```
{
  "_id": "2020-05-07_04.44.58",
  "job_desc": "",
  "job_name": "testing for api",
  "job_type": "Variation with Standard Deviation-based Threshold",
  "params": {
    "alg_params": {
      "B": 35,
      "batch_size": 100,
      "dropout": 0.6,
      "epochs": 8,
      "l_s": 200,
      "layers": [
        85,
        85
      ],
      "loss_metric": "mse",
      "lstm_batch_size": 80,
      "min_delta": 0.03,
      "optimizer": "adam",
      "patience": 10,
      "threshold": -0.3,
      "train": true,
      "validation_split": 0.4
    },
    "sets": {
      "test": [
        {
          "events": [
            0.0,
            0.0,
            0.0,
            0.0,
            0.0,
            0.0,
```

Figure 43: Snippet of response to request submitted to /job_details url

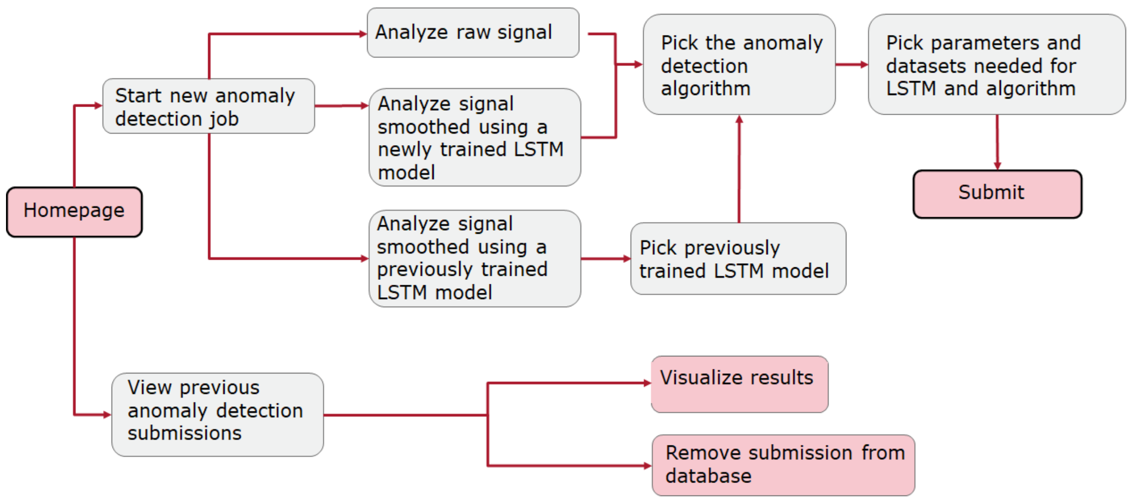

Homepage
Start new anomaly detection job
Analyze raw signal
Analyze signal smoothed using a newly trained LSTM model
Analyze signal smoothed using a previously trained LSTM model
Pick the anomaly detection algorithm
Pick parameters and datasets needed for LSTM and algorithm
Submit
Pick previously trained LSTM model
View previous anomaly detection submissions
Visualize results
Remove submission from database

Figure 44: Diagram of website workflow. Pink boxes indicate the first or last pages of the site.

4.6 Workflow of website

To build the website, we added some extra functionality to the API code and a user interface to make the site as easy as possible to use. The site was developed using only HTML, CSS, and plain Javascript, with the help of libraries like Bootstrap and Plotly. The site can complete the same functions as the API, with the addition of the possibility to visually see all details of all previously submitted jobs. The diagram in Figure 44 shows the workflow of the site.

Below we show the layout of main pages and show step-by-step instructions of what the users should do at every screen depending on the outcome they want to achieve.

4.6.1 Website demonstration: Starting a new anomaly detection job using LSTM signal values

This is a demo where we will submit a new anomaly detection job using signal values smoothed with LSTMs and view it visually once it has completed. There are also other scenarios but we will focus on this for this presentation.

1. From the homepage, click on Start New Anomaly Detection Job

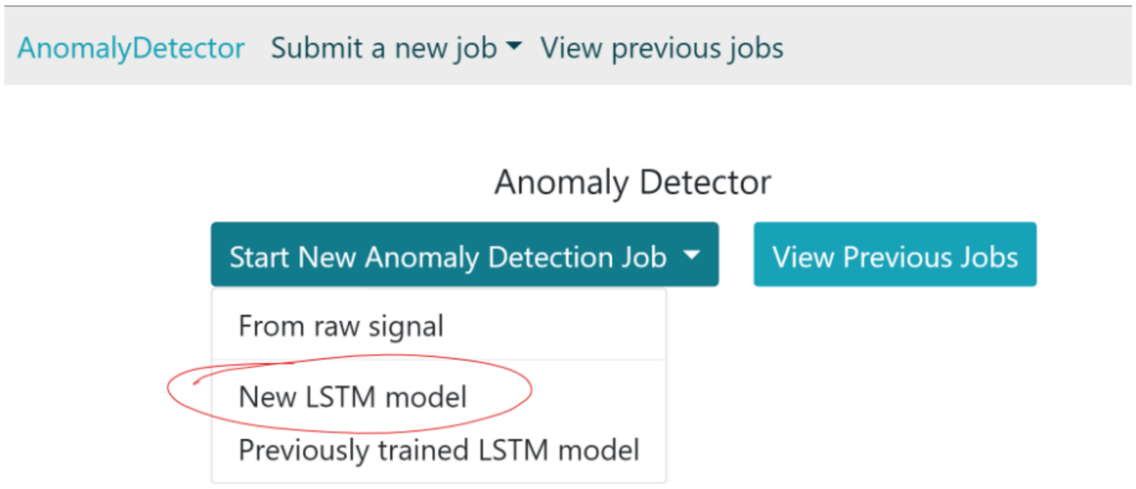
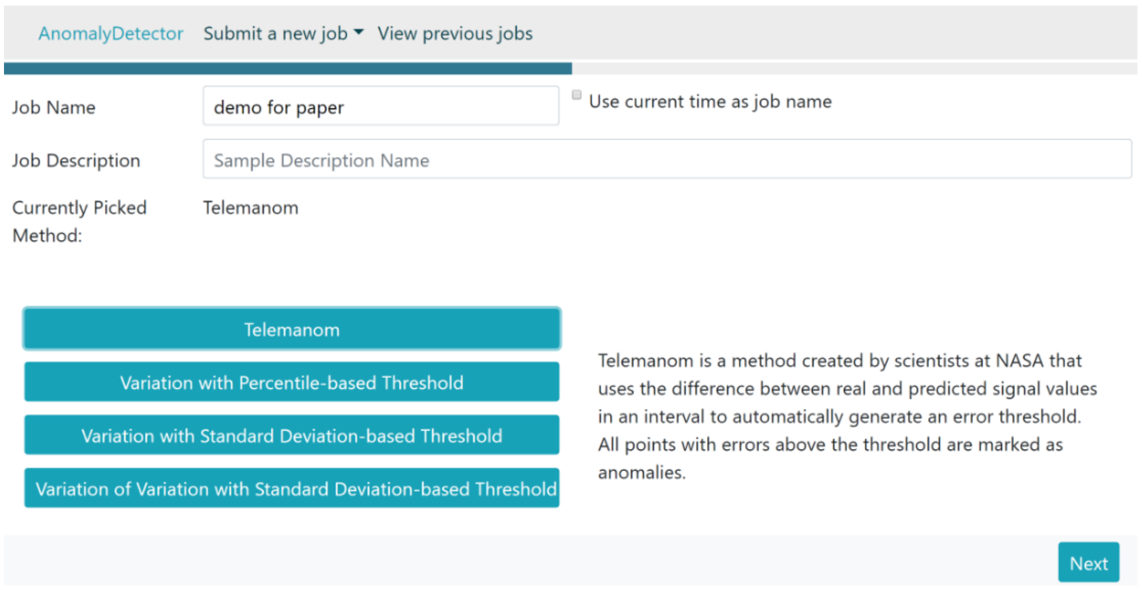


Figure 45: Homepage of Anomaly Detector site with clicked dropdown button indicating the button that will initiate an anomaly detection job on signal smoothed with a new LSTM model

2. On the dropdown menu, click New LSTM Model as shown in Figure 45. Doing so will take you to the next page.

3. The page in Figure 46 allows users to pick the anomaly detection algorithm. In this page, the next button is deactivated until the user selects at least a method and a job name. A description can also be added to allow the user more space to give additional details about the job, but it is not necessary. The job name can take any form the user likes. Clicking the checkbox to use current time as job name automatically names the job after the current date and time. For this demonstration we will name the job "demo for paper" and pick Telemanom as the anomaly detection algorithm because it is the only one whose functionality requires the use of LSTMs.

   By selecting this algorithm we notice the currently picked method changes from None to Variation with Standard Deviation-based Threshold, and a short paragraph containing some information about the method appears. This is done to make it easier for users to understand what algorithm they have chosen. If users switch between different


AnomalyDetector
Submit a new job
View previous jobs

Job Name: demo for paper

Use current time as job name

Job Description: Sample Description Name

Currently Picked Method: Telemanom

Telemanom
Variation with Percentile-based Threshold
Variation with Standard Deviation-based Threshold
Variation of Variation with Standard Deviation-based Threshold
Next

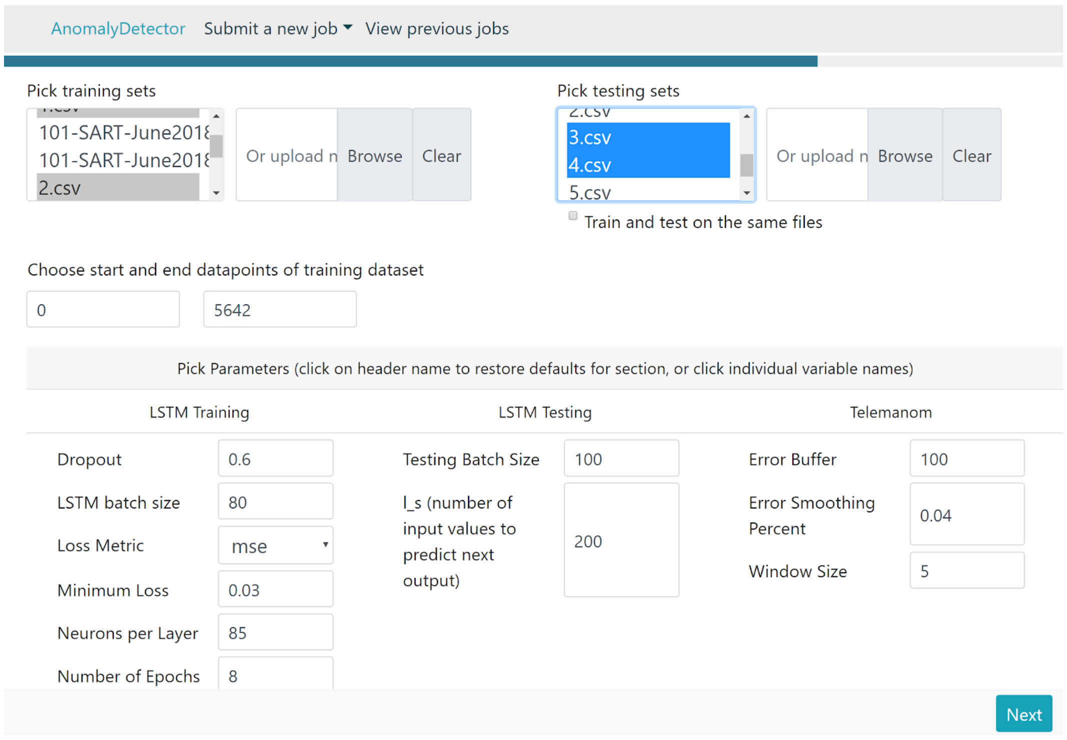
Telemanom is a method created by scientists at NASA that uses the difference between real and predicted signal values in an interval to automatically generate an error threshold. All points with errors above the threshold are marked as anomalies.

Figure 46: Page where users can input a job name and select an anomaly detection algorithm

algorithms, both these fields change accordingly.

4. Pressing the Next button on the page of Figure 46 page takes the user to a page devoted towards inputting datasets and algorithm parameters, shown in Figure 47. The content of this page changes based on the picked method and whether or not the signal will be LSTM-smoothed. We can start by inputting training and testing sets. The user can either upload new sets or pick previously uploaded ones from the selection box. There are no limits to the number of sets the user can choose, as long as all the arrays combined are less than 5MB in size. Picking one option disables the other. In this case we can pick two previously uploaded sets to train on, 1.csv and 2.csv, and two others to predict on, 3.csv and 4.csv. When the training sets are selected, the boxes asking for start and end data points get automatically filled with 0 and the summed length of the selected training arrays. The user can lengthen or shorten this input as desired within the given range of numbers.

   All the parameter fields get automatically populated by the default values stored in the config.yaml file mentioned previously. If the user



Pick training sets

1.csv
101-SART-June2018
101-SART-June2018
2.csv

Pick testing sets

2.csv
3.csv
4.csv
5.csv

☐ Train and test on the same files

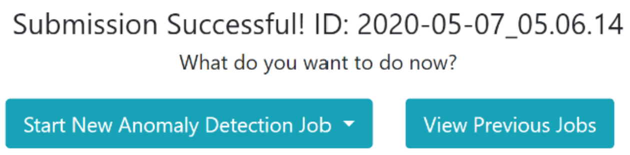
Choose start and end datapoints of training dataset

0 | 5642

Pick Parameters (click on header name to restore defaults for section, or click individual variable names)

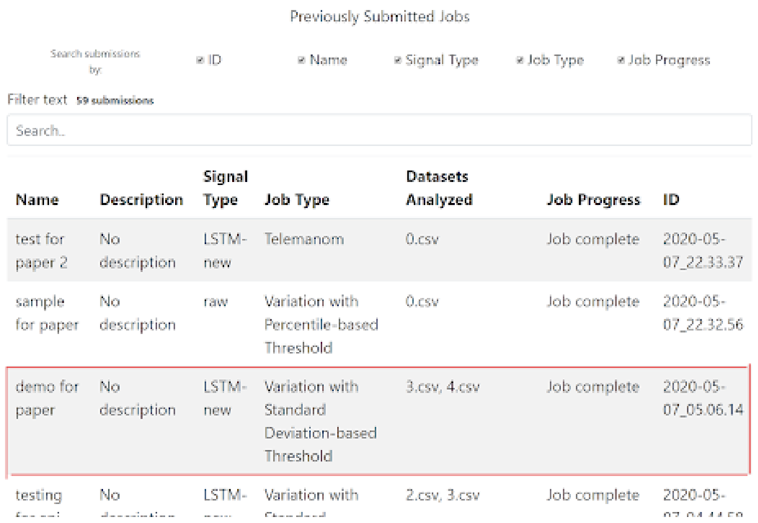
LSTM Training		LSTM Testing		Telemanom	
Dropout	0.6	Testing Batch Size	100	Error Buffer	100
LSTM batch size	80	l_s (number of input values to predict next output)	200	Error Smoothing Percent	0.04
Loss Metric	mse			Window Size	5
Minimum Loss	0.03				
Neurons per Layer	85				
Number of Epochs	8				

Figure 47: Page where users can select LSTM and anomaly detection parameters, as well as sets used to train the LSTM model and predict anomalies.


Submission Successful! ID: 2020-05-07_05.06.14

What do you want to do now?

Figure 48: Page showing successful submission of the anomaly detection job


Previously Submitted Jobs

Search submissions by: ☑ ID ☑ Name ☑ Signal Type ☑ Job Type ☑ Job Progress

Filter text 59 submissions

Search..

Name	Description	Signal Type	Job Type	Datasets Analyzed	Job Progress	ID
test for paper 2	No description	LSTM-new	Telemanom	0.csv	Job complete	2020-05-07_22.33.37
sample for paper	No description	raw	Variation with Percentile-based Threshold	0.csv	Job complete	2020-05-07_22.32.56
demo for paper	No description	LSTM-new	Variation with Standard Deviation-based Threshold	3.csv, 4.csv	Job complete	2020-05-07_05.06.14
testing	No	LSTM-	Variation with	2.csv, 3.csv	Job complete	2020-05-

Figure 49: Page showing all previously submitted jobs

wants to change any of them, they can do so, and they can restore the default value of the parameter by clicking on its name. For this demo, we will keep default parameters.

5. When the next button is pressed, the information is sent to the server and the job gets initialized. The user sees the success page shown in Figure 48. The backend process at this point is almost identical to that shown on the API demonstration above.

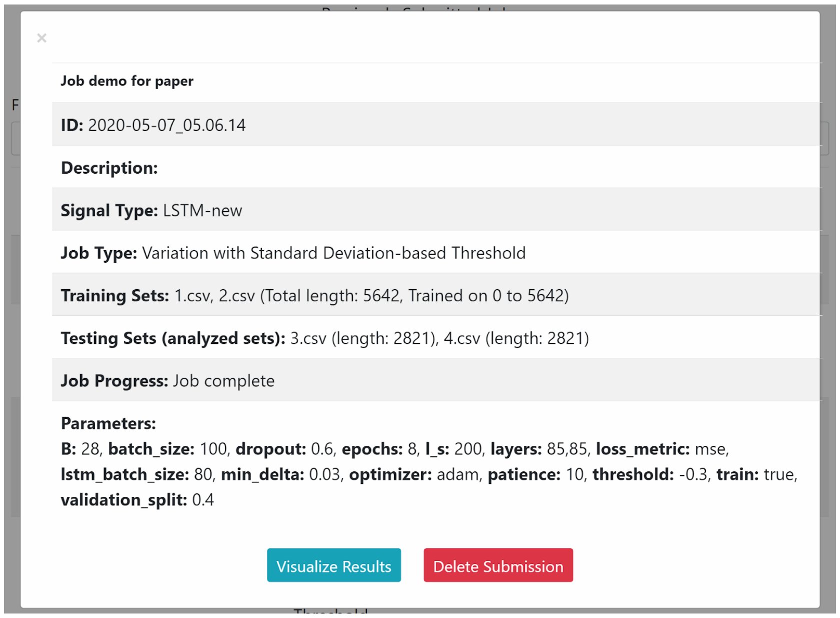
6. Clicking on View Previous Jobs sends the user to a page where they can see all previously submitted jobs and their details shown on Figure 49. In this page they can search and filter jobs, as well as click on them

to see more details.

Clicking on the job we just submitted shows us all details related to the job and gives us the option to delete the job or visualize its results, as shown on Figure 50.

Clicking on Visualize results takes us to a new page where all arrays or curves corresponding to this submission are visualized. The initial layout of the page looks like Figure 51:

Differently colored graphs represent different types of curves (blue - real signal curve, red - LSTM-predicted signal curve, orange - variation curve, grey highlights - anomalies). These colors remain the same regardless of how many channels are being displayed. If we want to only analyze a single channel we can use the toggle buttons on the top of the page to toggle on just those curves. We can also toggle individual curves by clicking on the legend entries. This is helpful if we want to analyze and compare how two specific curves affect each other. This graph is generated with Plotly and as such it offers by default options to drag and zoom, pan, show or compare data points on hover, and more. Finally, clicking on the button at the top of the page shows a dropdown of the job information similar to the one on the previous page.


×

Job demo for paper

ID: 2020-05-07_05.06.14

Description:

Signal Type: LSTM-new

Job Type: Variation with Standard Deviation-based Threshold

Training Sets: 1.csv, 2.csv (Total length: 5642, Trained on 0 to 5642)

Testing Sets (analyzed sets): 3.csv (length: 2821), 4.csv (length: 2821)

Job Progress: Job complete

Parameters:
B: 28, **batch_size:** 100, **dropout:** 0.6, **epochs:** 8, **l_s:** 200, **layers:** 85,85, **loss_metric:** mse, **lstm_batch_size:** 80, **min_delta:** 0.03, **optimizer:** adam, **patience:** 10, **threshold:** -0.3, **train:** true, **validation_split:** 0.4

Figure 50: Details of submitted job displayed once user clicks on job entry on the table

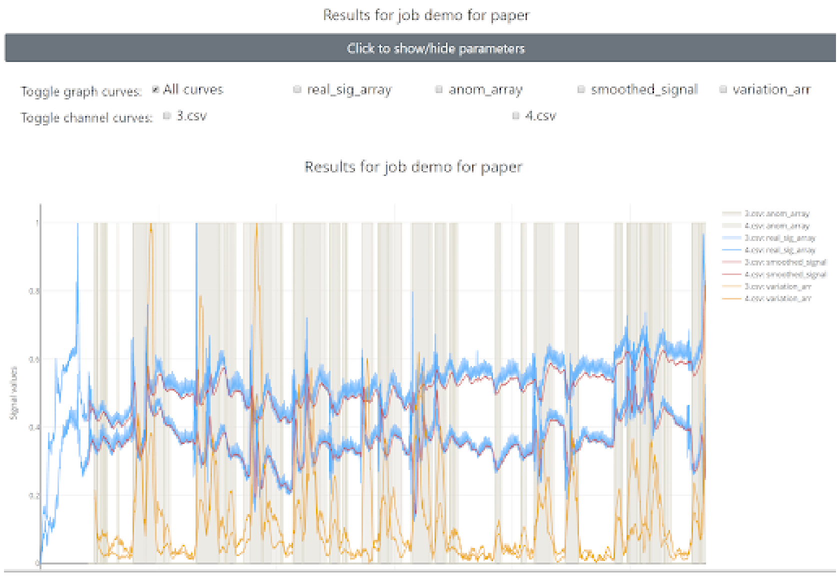


Figure 51: Visualization of anomaly detection results

4.7 User Guide

Both the website and API come with a user guide which aims to provide general guidelines for using these tools. The guide shows the user how to set up their digital environment to run the anomaly detection jobs, how to format the datasets, and how to navigate the API and website, and can be found here.

5 Future work

Future work on this project includes developing and expanding of the current anomaly detector methods and of the toolkit.

5.1 Future work on anomaly detector methods

There are two main parts where progress can be made on the anomaly detection algorithms. Once is improving the way variation is calculated, and the other is the development of a new anomaly detector method that finds optimal thresholds similarly to Telemanom.

5.1.1 Recalculating variation

During testing of the new anomaly detector methods we noticed that detected anomalies were shifted by at least B points from the real anomalies. This is because every i^{th} point of the variation array corresponds to points i through $i + B$ or the real signal array. To fix this issue we can recalculate variation so every i^{th} point of variation corresponds to points $i - \frac{B}{2}$ to $i + \frac{B}{2}$. The new formula for variation then becomes

$$\{v_i\}_{i=1}^{N-B} \quad v_i = \sum_{j=i-\frac{B}{2}}^{i+\frac{B}{2}} |y_{j+1} - y_j| \tag{14}$$

The variation array could also be appended by $\frac{B}{2}$ 0's in its beginning and end to make its length equal to that of the signal array. This would shift the location of every i^{th} point in the variation array to $i + \frac{B}{2}$ aligning it with its respective signal value. These changes could improve the accuracy of the anomaly detection algorithms.

5.1.2 New anomaly detection method

A popular way to find outliers in a curve is by calculating the mean of the curve and picking values in it such that their removal causes the greatest change in the curve. We saw this method used in Telemanom and in the Mathematical Optimization class at WPI, MA 4235. We can adapt this concept to variation: we can split the variation array into smaller sections, and calculating a threshold such that removing all variation values above it causes the greatest change in the mean of the section. This method might have potential given the fact that anomalies in the real signal cause very large and concentrated spikes in variation.

5.2 Future work on toolkit

The website and API could benefit from many additions, which we will describe below.

5.2.1 Adding real anomalies

So far once an anomaly detection job is complete the user is able to visualize its results, but they cannot visualize its real anomalies even if they have the data for it. It would be good to give users the option to upload a file with the real anomalies so they can visualize them alongside the predicted ones.

This option could be added in the page where users visualize their results. This page gets generated by visualize_results.html and visualize_results.js. We can add an upload box in the html file and on the javascript file we can add its functionality. Once the user uploads a file we can extract the data from it and add it to the plot. The plot is a global object in this .js file, and we can add the new data to it by following the plotly documentation.

5.2.2 Comparing between algorithms

Users of the website could greatly benefit from the ability to compare between the results of different job submissions as it would allow them to view the differences in results and help them understand which methods work best for a given dataset.

To allow for comparing between job results we must first allow the user to choose between which jobs to compare. This could happen on the View

Jobs page: when the table containing all the jobs is loaded it could contain a column named Compare with a checkbox on each row. Once the user clicks on the jobs they want to compare they can click a button that will take them to a new page containing the information for all selected jobs. The code for adding the checkboxes should be added to the results.html and results.js files. In results.html, another column should be added to the table, while in results.js code to update the table rows should be added to the updateProjectTable() function.

Once the Compare command is called we need a way to change to a new page and get the information about the selected jobs. In the results page we already have all the data for all the jobs in the database. We can use the sessionStorage variable to pass the job data from results.html to the new page where comparison results will be visualized. This gives us all the information we need to compare all results both visually and numerically.

5.2.3 User Interface for adding anomaly detection algorithms

To make the process of adding anomaly detection algorithms easier we can add another section in the website where users can upload their own anomaly detection algorithms. The users should be able to submit a .py file with the anomaly detector code, and a .yaml file with the display name for the algorithm, whether it works on real signal values, their LSTM predictions, or both, and the default parameters for it. For the algorithm to work with the current code a screen name, a variable name, and the actual default value is required. The variable name and its value should be added to the config.yaml file located in the _config_files folder. The .py file should be put in its own folder added under the algorithms folder. Then, the algorithm type and the display names should be added to the default_parameters variable in the AnomalyDetector object. Finally, the run_job() method in the Anomalydetector object needs to be updated with another if statement that initializes the object of the new algorithm which is defined in the uploaded .py file.

References

[1] C. Aggarwal. *Outlier Analysis*. da. New York City: Springer, 2014.

[2] M. Biondi and T. Wilcox. In: (2015). fNIRS in the developmental sciences.

[3] V. Constantinou, C. Laporte, and K. Hundman. “Spacecraft Anomalies Using LSTMs and Parametric Dynamic Thresholding.” en. In: (2018).

[4] R.A. Davis and P.J. Brockwell. *Introduction to Time Series and Forecasting*. en. Springer, 2016.

[5] M. Ferrari and V. Quaresima. *A brief review on the history of human functional near-infrared spectroscopy (fNIRS) development and fields of application*. en. NeuroImage, 2012.

[6] J. Han, M. Kamber, and J. Pei. *Data Mining Techniques and Concepts*. Elsevier, 2012.

[7] L.M. Hocke et al. “Automated Processing of fNIRS Data—A Visual Guide to the Pitfalls and Consequences.” In: (2018).

[8] O. Honchar and L. Di Persio. “Recurrent neural networks approach to the financial forecast of Google assets.” en. In: *International Journal of MAthematics and Computers in Simulation* (2017).

[9] V. Kumar, A. Banerjee, and V. Chandola. *Anomaly Detection: A Survey*. en. ACM Computing Surveys, 2009.

[10] C. Olah. *Understanding LSTM Networks*. en. Retrieved from. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[11] C.M. Salgad et al. *Secondary Analysis of Electronic Health Records*. en. 2016, pp. 163–183.

[12] E. Solovey. Ed. by P. Kumi, J. Pardue, and Interviewers. 2019.

[13] E.T. Solovey et al. 2009.

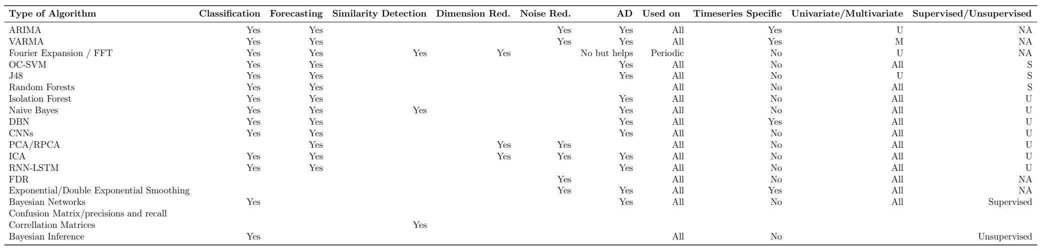
[14] D.S. Stoffer and R.H. Shumway. *Time Series Analysis and Its Applications: with R Examples*. en. Springer, 2017.

[15] N. Wan. *FNIRS: The In-Between for Brain Activity in Real-World Settings*. en. Cognitive Neuroscience Scoriety, 2015.

Appendices

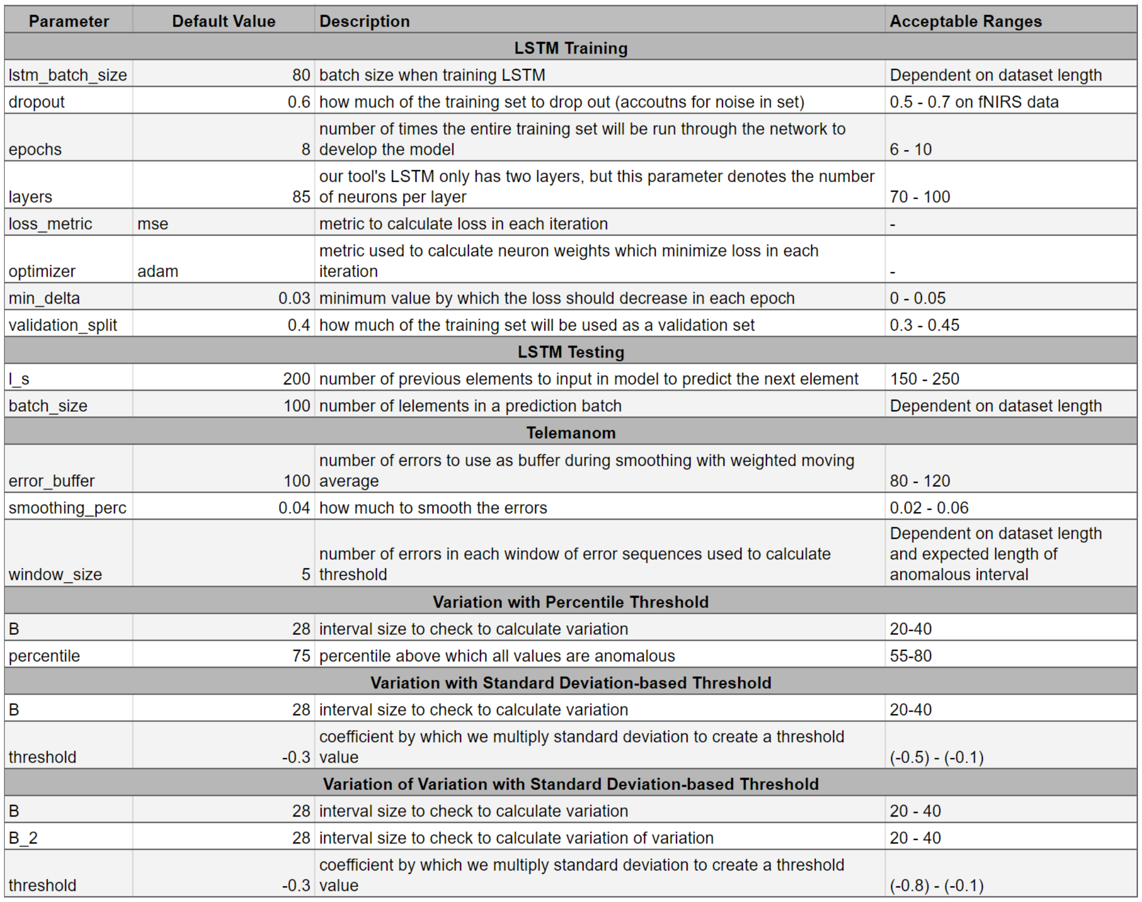
A Table of researched algorithms

The table in the image below shows algorithms we researched during the beginning of this project and their categorizations



Type of Algorithm	Classification	Forecasting	Similarity Detection	Dimension Red.	Noise Red.	AD	Used on	Timeseries Specific	Univariate/Multivariate	Supervised/Unsupervised
ARIMA	Yes	Yes			Yes	Yes	All	Yes	U	NA
VARMA	Yes	Yes			Yes	Yes	All	Yes	M	NA
Fourier Expansion / FFT	Yes	Yes	Yes	Yes		No but helps	Periodic	No	U	NA
OC-SVM	Yes	Yes				Yes	All	No	All	S
J48	Yes	Yes				Yes	All	No	U	S
Random Forests	Yes	Yes					All	No	All	S
Isolation Forest	Yes	Yes				Yes	All	No	All	U
Naive Bayes	Yes	Yes	Yes			Yes	All	No	All	U
DBN	Yes	Yes				Yes	All	Yes	All	U
CNNs	Yes	Yes				Yes	All	No	All	U
PCA/RPCA		Yes		Yes	Yes		All	No	All	U
ICA	Yes	Yes		Yes	Yes	Yes	All	No	All	U
RNN-LSTM	Yes	Yes				Yes	All	No	All	U
FDR					Yes		All	No	All	NA
Exponential/Double Exponential Smoothing					Yes	Yes	All	Yes	All	NA
Bayesian Networks	Yes					Yes	All	No	All	Supervised
Confusion Matrix/precisions and recall										
Correllation Matrices			Yes							
Bayesian Inference	Yes						All	No		Unsupervised

B Table of default parameters for each method



Parameter	Default Value	Description	Acceptable Ranges
LSTM Training			
lstm_batch_size	80	batch size when training LSTM	Dependent on dataset length
dropout	0.6	how much of the training set to drop out (accoutns for noise in set)	0.5 - 0.7 on fNIRS data
epochs	8	number of times the entire training set will be run through the network to develop the model	6 - 10
layers	85	our tool's LSTM only has two layers, but this parameter denotes the number of neurons per layer	70 - 100
loss_metric	mse	metric to calculate loss in each iteration	-
optimizer	adam	metric used to calculate neuron weights which minimize loss in each iteration	-
min_delta	0.03	minimum value by which the loss should decrease in each epoch	0 - 0.05
validation_split	0.4	how much of the training set will be used as a validation set	0.3 - 0.45
LSTM Testing			
l_s	200	number of previous elements to input in model to predict the next element	150 - 250
batch_size	100	number of lelements in a prediction batch	Dependent on dataset length
Telemanom			
error_buffer	100	number of errors to use as buffer during smoothing with weighted moving average	80 - 120
smoothing_perc	0.04	how much to smooth the errors	0.02 - 0.06
window_size	5	number of errors in each window of error sequences used to calculate threshold	Dependent on dataset length and expected length of anomalous interval
Variation with Percentile Threshold			
B	28	interval size to check to calculate variation	20-40
percentile	75	percentile above which all values are anomalous	55-80
Variation with Standard Deviation-based Threshold			
B	28	interval size to check to calculate variation	20-40
threshold	-0.3	coefficient by which we multiply standard deviation to create a threshold value	(-0.5) - (-0.1)
Variation of Variation with Standard Deviation-based Threshold			
B	28	interval size to check to calculate variation	20 - 40
B_2	28	interval size to check to calculate variation of variation	20 - 40
threshold	-0.3	coefficient by which we multiply standard deviation to create a threshold value	(-0.8) - (-0.1)