Unsupervised Machine Learning for Spacecraft Anomaly Detection in WebTCAD

Shawn Polson

Laboratory for Atmospheric and Space Physics

June 7, 2019

Abstract

This paper explores unsupervised machine learning techniques for anomaly detection in spacecraft telemetry with the aim of improving WebTCAD's automated detection abilities. WebTCAD is a tool for ad-hoc visualization and analysis of telemetry data that is built and maintained at the Laboratory for Atmospheric and Space Physics. This paper attempts to answer the question: "How good could machine learning for anomaly detection in WebTCAD be?" The techniques are applied to five representative time series datasets. Four algorithms are examined in depth: rolling means, ARIMA, autoencoders, and robust random cut forests. Then, three unsupervised anomaly definitions are examined: thresholding outlier scores with standard deviations from the data's mean, thresholding outlier scores with standard deviations from the scores' mean, and nonparametric dynamic thresholding. Observations from this exploration and suggestions for incorporating these ideas into WebTCAD and future work are included in the final two sections.

Contents

1 Introduction

Telemetry data is the cornerstone for analyzing the health, safety, and performance of spacecraft and spacecraft instrumentation. The Laboratory for Atmospheric and Space Physics (LASP) maintains thousands of telemetry datasets from the missions they support. And in an environment where a failure to detect and respond to potential hazards could result in the full or partial loss of spacecraft, anomaly detection is a critical tool to alert operations engineers of unexpected behavior. Ad-hoc analysis tools like WebTCAD [25] allow scientists to explore plots of these telemetries, and can flag when values stray outside of predefined limits, but they suffer from well-documented limitations [9]. Often at LASP, scientists must pore over these datasets by hand to uncover anomalies. This becomes less feasible over time as improving computing and storage capabilities lead to increasing volumes of telemetry data [6]. The need for improved anomaly detection systems will only grow. So, the goal of this paper is to explore and compare solutions for anomaly detection in spacecraft telemetry, with the aim of improving WebTCAD's automated detection abilities. The solutions are pulled from the field of machine learning, which can be defined as the science of getting computers to learn and then act without being explicitly programmed. They must be unsupervised because WebTCAD lacks any labeled anomalies that are necessary for supervised solutions.

An anomaly is anything that deviates from what is standard, normal, or expected. Anomalies are also referred to as outliers, abnormalities, discordants, or deviants in the data mining and statistics literature [3]. The output of an anomaly detection algorithm can be one of two types:

- **Outlier scores:** Most outlier detection algorithms output a score quantifying the level of "outlierness" of each data point. This score can also be used to rank the data points in order of their outlier tendency. This is a very general form of output, which retains all the information provided by a particular algorithm, but it does not provide a concise summary of the small number of data points that should be considered outliers [3].

- **Binary labels:** A second type of output is a binary label indicating whether a data point is an outlier or not. Although some algorithms might directly return binary labels, outlier scores can also be converted into binary labels. This is typically achieved by imposing thresholds on outlier scores, and the threshold is chosen based on the statistical distribution of the scores. A binary labeling contains less information than a scoring mechanism, but it is the final result that is often needed for decision making in practical applications [3].

Virtually all anomaly detection algorithms create a model of the "normal" patterns in the data, and then compute an outlier score of a given data point on the basis of the deviations from these patterns. In some applications, anomalies correspond to *sequences* of multiple data points rather than individual data points. In either case, it is necessary to consider the context of the data, because what can be called an outlier in one context may be normal in another. Weather is a classic example of this. A temperature reading of 38 °C during winter is likely anomalous, while the same temperature during summer is likely not. Moreover, it is often a subjective judgement as to what constitutes a "sufficient" deviation for a point to be considered anomalous. Some approaches to defining "sufficient" in an objective way, like nonparametric dynamic thresholding [17], will be covered later. However, there will always be trade-offs between missing anomalies and generating false positives (calling normal data anomalous).

The datasets studied in this paper, introduced in Section 2, are all time series datasets. Time series are series of values at successive times that are typically generated by continuous measurement over time. In our case, they are continuous readings from different hardware components on a particular spacecraft in orbit around Earth. The problem of outlier detection in time series data is highly related to the problem of *change analysis* because data values in consecutive time stamps are highly influenced by one another (i.e., the temporal context matters). Broadly speaking, many differentiate between the two problems as follows:

- **Change analysis** is generally focused on finding longer-term changes. For example, changes in a pattern or trend that evolve slowly over time—a phenomenon referred to as *concept drift*—or abrupt changes that become the new normal—referred to as *change points*.

- **Outlier detection** is focused on finding short-lived changes. For example, sudden spikes or dips in values or a one-time break from a pattern or trend.

Furthermore, these changes are detected either *offline* or *online*. In an offline setting, anomalies are detected in preexisting datasets. In an online setting, anomalies are detected in real time, as new data values arrive. Online anomaly detection is less straightforward than offline detection because a new data point that looks anomalous could either be a true outlier or a change point, and there is no way of knowing which it is except to wait for more data points. Of course, if the data does not follow any meaningful pattern (e.g., if the values are basically random or if they are generated by manual human input), then it is not helpful to think about these concepts and we really cannot do much better than detecting outliers by simple statistical measures like standard deviations from a mean or limit thresholds. But these concepts apply when the data does follow a meaningful pattern. A common pattern in spacecraft telemetry is that the data will have a seasonal component to it—that is, a pattern that will repeat in fixed-length cycles. In this case, it is often helpful to think about the data in terms of its *seasonal decomposition*—that is, in terms of its seasonality, trend, and residuals. This is demonstrated in Figure 1. This paper will explore solutions for anomaly detection in our time series telemetry, but it bears mentioning that there exist infinitely many time series, so there is no one solution that will work for all datasets. Part of the exploration will be to consider the kinds of data on which the solutions do not work.

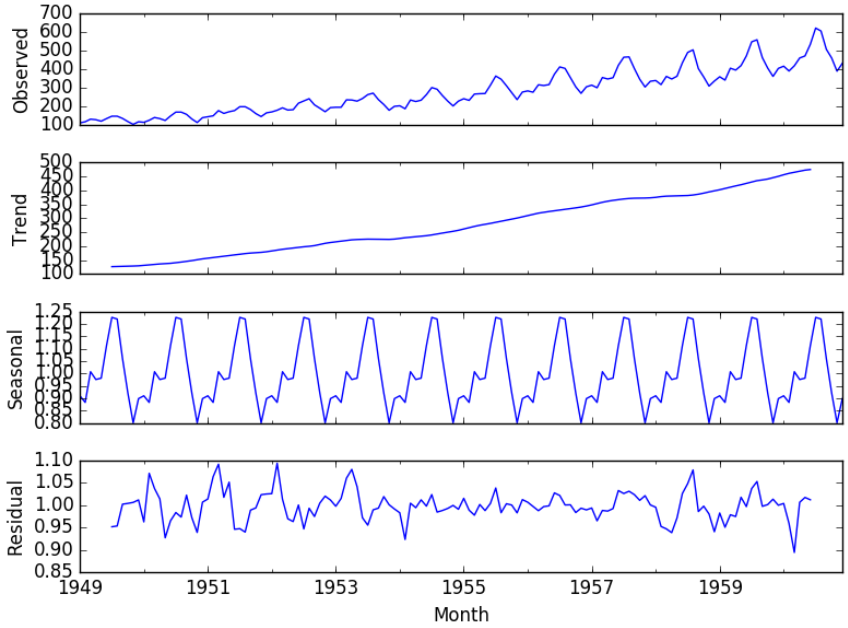


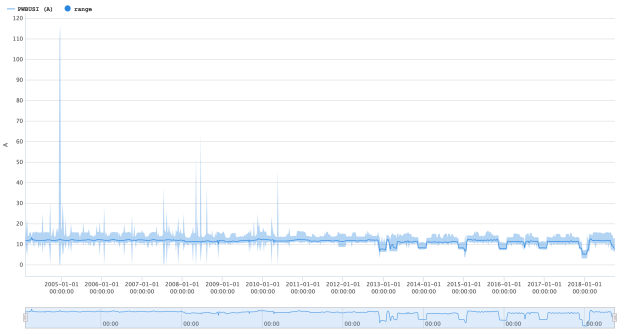
Figure 1: The seasonal decomposition of a time series dataset. The original data is shown in the top "Observed" plot. Its trend, seasonal, and residual components are plotted afterward, respectively. The original data is "decomposed" such that recombining the trend, seasonality, and residuals would reconstruct the original curve.

2 Dataset Descriptions

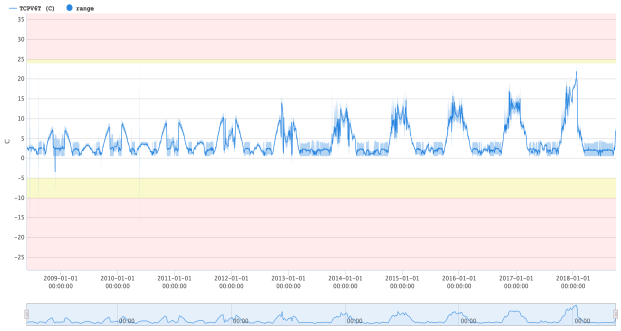
Our datasets come from the WebTCAD application which is built and maintained by the LASP Web Team [25]. They are five time series datasets, shown below in Figures 2a–2e, and they are representative of the types of data with which we are concerned. Copies of the datasets and associated metadata are publicly available in a GitHub repository[1]. The algorithms covered in Section 3 are applied to these datasets in Section 4.

[1]https://github.com/sapols/Satellite-Telemetry-Anomaly-Detection

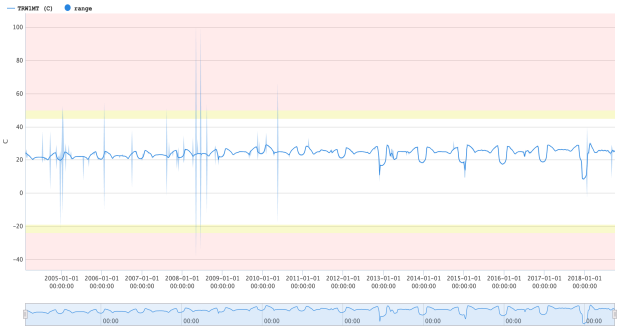
(a) The Bus Voltage dataset. The data shows a predictable yearly seasonal pattern with a trend that spikes upward after 2008.



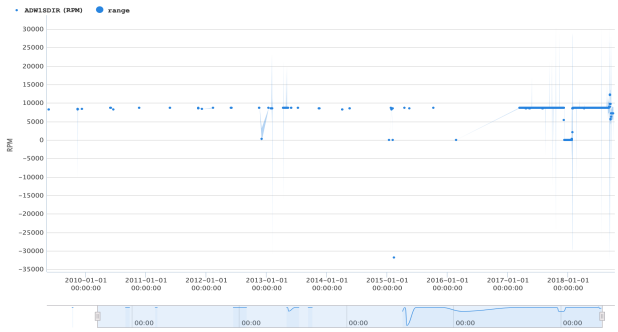
(b) The Total Spacecraft Bus Current dataset. The data does not appear to follow a predictable pattern, but the range of values is consistently bounded.



(c) The Battery Temperature dataset. The data shows somewhat predictable seasonal variations with increasing magnitude.



(d) The Reaction Wheel Temperature dataset. The data's seasonal variation and trend appear to change after 2012.



(e) The Reaction Wheel RPM dataset. The data is riddled with holes because the reaction wheel is not always running.

Figure 2: Bin-averaged plots of this paper's datasets viewed in WebTCAD. Minimum and maximum values for each bin are highlighted behind the averaged curves. The yellow and red shaded regions are limit violation thresholds.

Bus Voltage is a *univariate* time series with 51,704 readings for bus voltage recorded over a period of roughly 18 years. "Univariate" means that it tracks only one variable's values over time. The cadence of the data (the regular frequency at which measurements are taken) is one measurement every 3 hours. This cadence is the result of a mean resampling, as the original data had a cadence of 5 minutes. Missing values were filled using linear interpolation. The voltage dips predictably once a year, and the data's trend remains constant until a change point occurs at approximately the start of 2008. The trend spikes at the change point and slopes downward thereafter.

Total Spacecraft Bus Current is a univariate time series with 5,345 readings for total bus current recorded over roughly 15 years. The cadence is one measurement per day, which is the result of a mean resampling from its original 5 minute cadence. Missing values were filled using linear interpolation. The current does not follow a predictable pattern, but its values are consistently bounded between approximately 10 and 15 amps before the end of 2012 and 5 and 15 amps afterward.

Battery Temperature is a univariate time series with 128,288 readings for battery temperature recorded over roughly 10 years. The cadence is one measurement per hour, which is the result of a mean resampling from its original 16 second cadence. Missing values were filled using linear interpolation. The temperature shows regular yearly variations with increasing magnitude.

Reaction Wheel Temperature is a univariate time series with 769,745 readings for wheel temperature recorded over roughly 15 years. The cadence is one measurement every 10 minutes, which is the result of a mean resampling from its original 5 minute cadence. Missing values were filled using linear interpolation. The temperature follows a predictable yearly pattern until a change point occurs after 2011.

Reaction Wheel RPM is a univariate time series with 984,911 readings for RPM recorded over roughly 10 years. The cadence is one measurement every 5 minutes, which is the result of a mean resampling from its original 1 second cadence. Missing values were filled with zeros. The data is sparse because the reaction wheels are not always in motion. The RPM values are unpredictable because they are determined by manual human input, but the wheels tend to be spun up to similar RPMs.

3 Background and Review of the Literature

Anomaly detection is a broad field that has been studied in the context of a large number of application domains. M. Gupta et al. outlined various approaches for detecting anomalies in temporal data in their 2013 survey paper [15], and those approaches are summarized in figure 3. We are concerned with finding outliers in time series data, i.e., the left branch of figure 3.

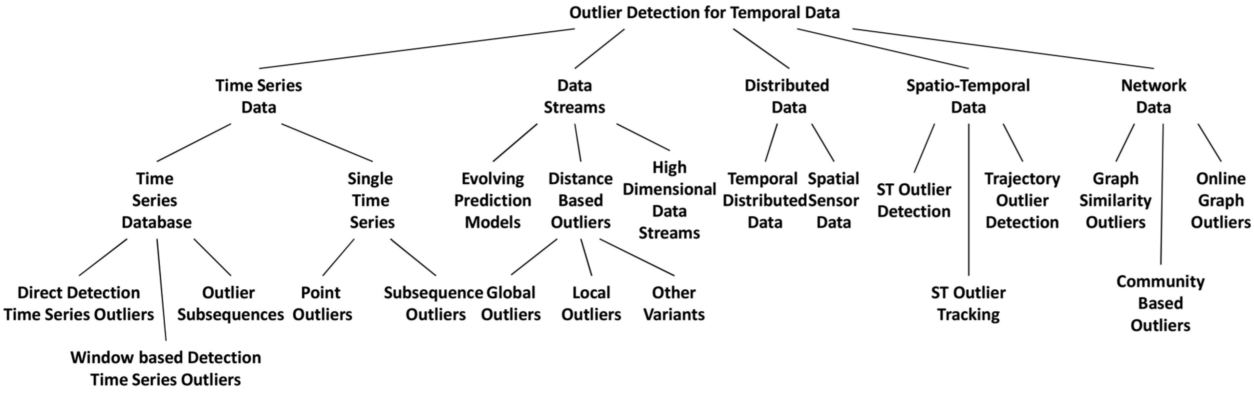


Figure 3: Organization of the survey by M. Gupta et al.

This paper will start with single time series, because that is what all five of the WebTCAD datasets are. Time series databases are covered briefly in Section 3.3. In singe time series datasets, we can detect either point outliers (outliers that are individual data points) or subsequence outliers (outliers that are multiple contiguous data points). Point outliers are covered first. Subsequence outliers are saved for Section 3.2.

It was stated in Section 1 that these outliers will be detected using unsupervised solutions because WebTCAD lacks any labeled anomalies. In the field of machine learning, there are two main types of tasks: *supervised* and *unsupervised*. The main difference between the two is that supervised learning is done using a ground truth, where there exists prior knowledge of what the output values for samples should be. Therefore, the goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between inputs and outputs. Unsupervised learning, on the other hand, does not have labeled outputs, so its goal is to infer the natural structure present within a set of data points. In the context of anomaly detection, a supervised solution would be trained on a collection of example anomalies such that it could recognize when data looks like one of the examples. An unsupervised solution detects anomalies based on a provided definition of what constitutes "anomalous" data.

3.1 Single Time Series Point Outliers

Numerous methodologies have been proposed to find point outliers in a time series. In addition to classic statistical measures like standard deviations from a mean, a large number of prediction models, profile-based models, and density-based models have been proposed [15]. An information-theoretic compression based technique has also been proposed to find "deviants" [18]. But for this paper, we will focus on prediction models, where the outlier score for a point in a time series is computed as its deviation from the value predicted by a summary prediction model. In other words, if actual values in the time series differ greatly from the model's predicted values, we call that anomalous. In practice, it is hard to define what "differs greatly" means because thresholding is, in a sense, a simplified version of the initial anomaly detection problem. But we will return to that in Section 3.4. First, we will explore the prediction models. Those will include a couple simple statistical models, *Auto Regressive Integrated Moving Average (ARIMA)* models, and *autoencoders*.

The latter two are both trained on *training sets* and the standard practice is to look for anomalies in the remaining *test sets*. The distinction between training and test sets is an important one in machine learning. Simply put, the training set is the set of data points on which a model is trained, and the test set is the set of points on which a model is evaluated [4]. As a rule, the better the training data, the better the model performs. Sometimes the training set is split into training and *validation* sets, where the validation set is used to find and optimize the best model. An analogy from real life is learning in a classroom. The example problems that the instructor solves in class while teaching a subject form the training set, exam questions are the validation set, and real-world problems solved outside the classroom are the test set. In time series data, the training set almost always comes chronologically before the validation set, and the test set is chronologically last.

3.1.1 Simple Statistical Approaches

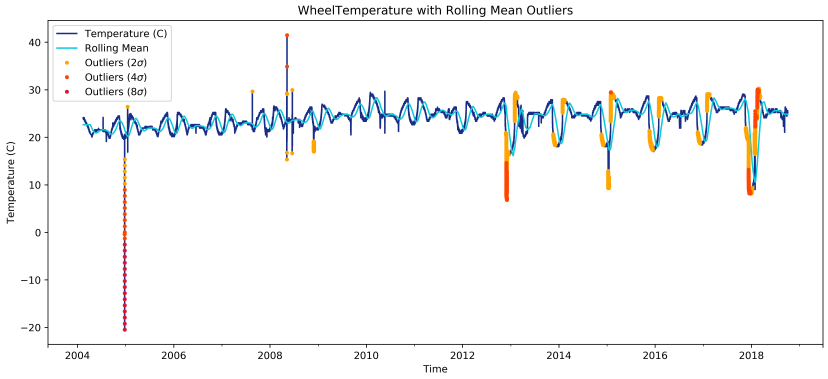


Figure 4: The Reaction Wheel Temperature data (dark blue) plotted with a rolling mean (light blue) using a window size equal to the length of the dataset $L/100$ and outliers at three different standard deviations.

Anomaly detection systems can get complicated. When first starting to build one and while considering the panoply of options from the fields of statistics and machine learning, it is important to start simple. For example, label as outliers all points that are more than two standard deviations from the dataset's mean and see the results. If that does not yield the desired results, diagnose what is wrong or missing and go from there. Statistics is a rich field for anomaly detection in spacecraft telemetries [5]; there are options that may work for the given use case without the need to employ advanced machine learning approaches, like neural networks, that are often harder to configure and require greater computing resources. Standard deviations from the dataset's mean is a common place to start, but that can be improved by comparing against a *rolling mean* with a fixed-sized window.

Given a series of numbers and a window of fixed size N, a rolling mean is obtained by first taking the mean of the initial N numbers in the series. Then the window is "shifted forward" by one, i.e., the first number of the series is excluded and the next number after the initial window is included, and the mean is recalculated. This repeats until the final window includes the last number. Pseudocode for this procedure is given in Algorithm 1. The rolling mean, while simplistic, can be used as a prediction model in historical data just like ARIMA or autoencoder predictions. See figure 4 for an example. Plus, it has the benefit of being computationally inexpensive.

Algorithm 1 Rolling Mean

Input: TS, a time series, and N, a window size
Output: $rollingMean$, a list of numbers

```
function RollingMean(TS, N)
    window ← the first N data points in TS
    x ← the last number in TS
    while x ∉ window do                          ▷ "Roll" the window across the whole time series
        mean ← Mean(window)
        rollingMean.Append(mean)
        window ← shift window forward by 1       ▷ Exclude the first number in window and
                                                   include the next number after window
    return rollingMean
```

A rolling mean is sensitive to outliers, which can be good or bad. If a measure that is not sensitive to outliers is preferred, consider a *rolling median* instead, which is just like a rolling mean except the median of each window is computed. The window size used for either can be tuned such that the resulting curve follows the data as closely as is needed. A window size of one will simply reproduce the original curve, whereas a window size equal to the length of the dataset will reproduce the flat mean/median. The window size should be somewhere in-between those extremes, such that the rolling statistic follows the original curve somewhat but does not overfit it. As a rule, overfitting should always be avoided.

3.1.2 ARIMA

The Autoregressive Integrated Moving Average (ARIMA) algorithm is a popular algorithm to try when first beginning to analyze a time series or use machine learning for time series forecasting. It constructs a mathematical model of a dataset from the training set and can then forecast values in, and even beyond, the test set [7]. An example ARIMA forecast is shown in Figure 5. For anomaly detection, the basic idea is that the constructed model represents the normal patterns in the data, so when data differs from the model, that can be called anomalous. The patterns represented by ARIMA models are effectively the trend, seasonal, and residual components of a seasonal decomposition like shown in Figure 1, so note that ARIMA should not be used on datasets that clearly lack those components (e.g., if the values are generated randomly or by unpredictable human input).

3.1.2.1 ARIMA Models

An ARIMA model is an equation consisting of *autoregressive* terms, *moving average* terms, and *differencing* operations [10]. The autoregressive terms express the dependency of the current value to its previous ones.

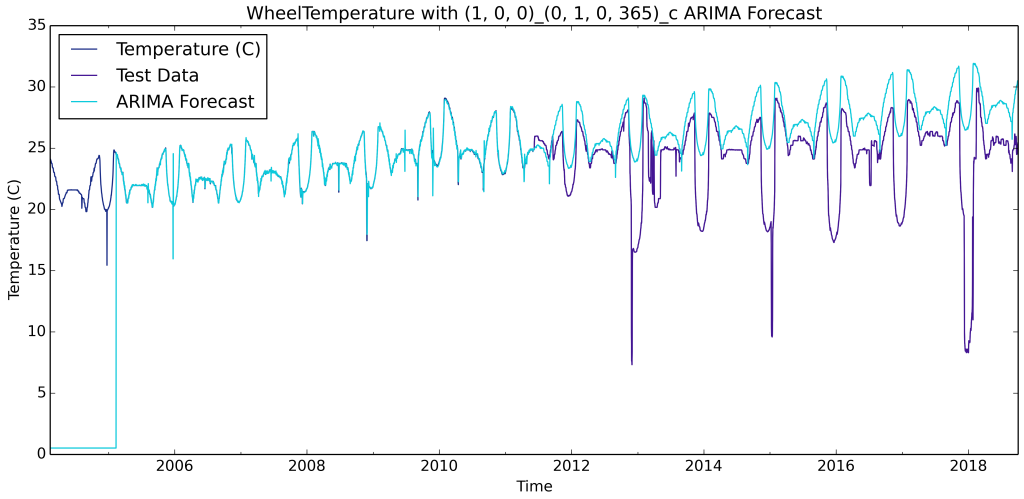


Figure 5: Forecast from an *ARIMA(1,0,0)(0,1,0)365* model trained on the first 50% of the Reaction Wheel Temperature dataset. The forecast begins after the first seasonal cycle because that data has been "differenced away."

The moving average terms model the effect of previous forecast errors on the current value. If the time series is not *stationary*, differencing operations are used to make it stationary.

A stationary process is a stochastic process with the property that its probability distribution—i.e., its mean, variance, and autocorrelation structure—does not change over time [10]. Stationarity can be defined in precise mathematical terms, but in simple terms it means a flat-looking series without trend nor periodic fluctuations (seasonality). See Figure 6 for an example. Recall that spacecraft telemetry often exhibit seasonality, so they are often not stationary.

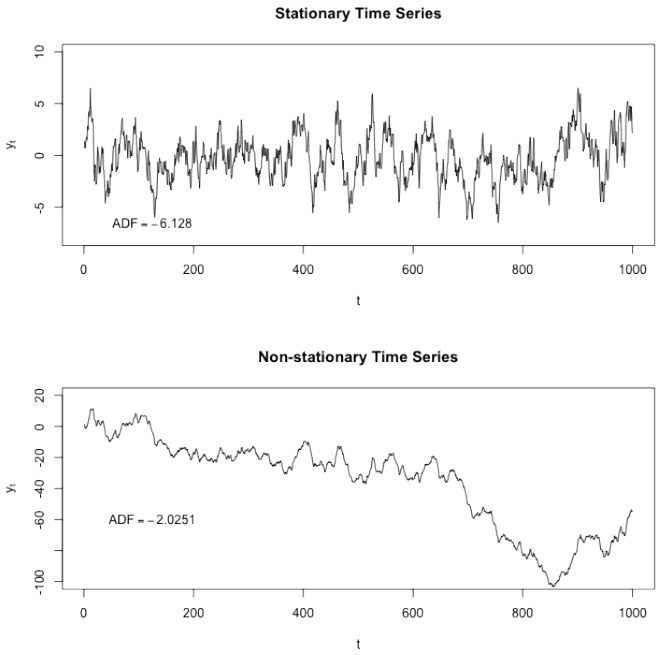


Figure 6: A stationary time series (top) and a non-stationary time series (bottom).

The terms of an ARIMA model are defined by seven hyperparameters that must be set precisely for a given time series. By convention, they are written as *ARIMA(p,d,q)(P,D,Q)m*. The hyperparameter definitions are as follows:

- ***p*:** The number of autoregressive terms
- ***d*:** The number of nonseasonal differences needed for stationarity (trend differencing)
- ***q*:** The number of lagged forecast errors in the prediction equation
- ***P*:** The number of seasonal autoregressive terms
- ***D*:** The number of seasonal differences needed for stationarity (seasonal differencing)
- ***Q*:** The number of seasonal moving average terms
- ***m*:** The number of points in one seasonal cycle

3.1.2.2 Grid Searching Hyperparameters

Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots can help inform how to set hyperparameter values for a time series [10]. Tests like the Augmented Dickey-Fuller test can also help [7]. However, a great deal of expertise is required to use those effectively. *Grid searching* offers a way to automate the tuning of hyperparameters without such expertise, but there are caveats [29]. The discovered hyperparameters may be suboptimal compared to the values an expert would set, and the procedure is computationally costly.

Algorithm 2 Grid search ARIMA hyperparameters

Input: TS, a time series, and *holdout*, a percentage
Output: *params*, the best-scoring hyperparameters

```
function GridSearch(TS, holdout)
    train ← the training portion of TS          ▷ Data points in TS minus the last holdout points
    pValues ← possible p values
    dValues ← possible d values
    qValues ← possible q values
    PValues ← possible P values
    DValues ← possible D values
    QValues ← possible Q values
    m ← length of a seasonal cycle in TS
    best ← ∞
    for p in pValues do
        for d in dValues do
            for q in qValues do
                for P in PValues do
                    for D in DValues do
                        for Q in QValues do
                            order ← (p,d,q)(P,D,Q)m      ▷ Try all possible hyperparameter combinations
                            RMSE ← ScoreModel(train, order)   ▷ Train and score a model with this
                                                                order using nested cross-validation
                            if RMSE < best then
                                best ← RMSE
                                params ← order
    return params
```

The idea behind grid searching ARIMA hyperparameters is to brute force search all possible combinations of values and then use the set of values that yielded the best score. Pseudocode for this procedure is given in

Algorithm 2. Every set of possible values is scored using *nested cross-validation*. In nested cross-validation, a portion of the dataset is held out as the test set, then the remaining data is iteratively split into training and validation sets. At each iteration, an ARIMA model with the current hyperparameters is fit using the training set and is scored using a metric like Root Mean Square Error (RMSE)—the standard deviation of the forecast errors. Figure 7 is a diagram of the nested cross-validation algorithm.

While grid searching, if the data is seasonal, the models should be trained on as many seasonal cycles as possible. More cycles help the models nail down the seasonality, and a low number of cycles risks overfitting. In fact, a grid search fundamentally will not yield certain combinations of hyperparameters if enough training data is not available. For example, an *ARIMA(0,0,0)(1,0,0)52* model needs at least 53 data points. An *ARIMA(0,0,0)(2,1,0)52* model needs at least 157 points.

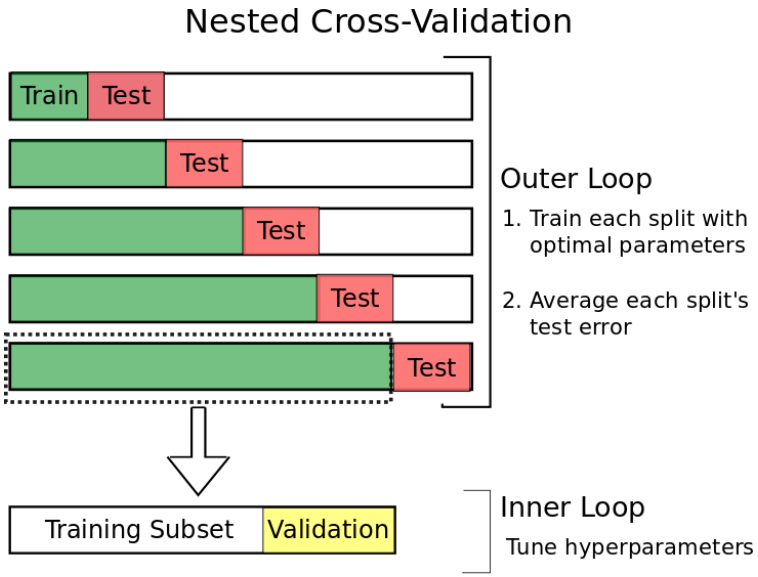


Figure 7: Diagram of the nested cross-validation algorithm.

3.1.3 Autoencoders

Autoencoders are a special type of *deep neural network* that are often used to compress or de-noise data [16, 27]. The general idea is that, given some input data, the first half of the network reduces the input into a lower-dimensional space to obtain a compressed representation of the data. This step is called *encoding* and is where the name "autoencoder" comes from. Once the data is encoded, the second half of the network performs the opposite operation, *decoding*, where the compressed data is enlarged back to its original size in an attempt to reconstruct the original input. This is shown in Figure 11. Because the output is reconstructed from the simplified representation, the autoencoder will learn only what is most important to represent the core attributes of the data and automatically ignore such things as anomalies and noise. The basics of neural networks should be covered before diving deeper into how autoencoders can be used for anomaly detection.

3.1.3.1 Neural Networks

Neural networks are machine learning algorithms that have recently seen a surge in popularity due to the boom of big data and the increasing relevance of artificial intelligence [24, 21, 26]. Neural networks are connected layers of *nodes* that each have several input values and a single duplicated output that depends on the node's inputs. The general structure of a node is shown in Figure 8. Sequences of nodes combine to create multiple layers in which each subsequent node depends on the output of its previous nodes, like in Figure 9. The reason for doing this is that it crudely approximates the network of connected neurons that comprise the human brain.

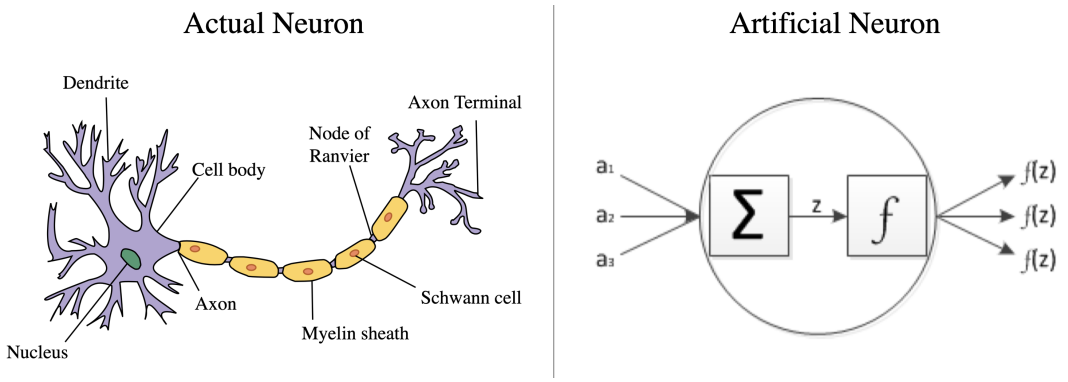


Figure 8: The general structure of a neuron cell (left) and a simplistic diagram of an artificial neuron (right) which is one example of a neural network node. There exist more complicated nodes such as Long-Short-Term-Memory nodes (LSTMs) [17].

In the node diagram in Figure 8, let a_1, a_2, a_3 be input values to the node and let w_1, w_2, w_3 be some initial weighting applied to each value. The first thing that is computed inside the node is a weighted sum of the inputs plus some additional bias value b. Define this summation as $z = a_1w_1 + a_2w_2 + a_3w_3 + b$. The node applies an activation function f to z in order to determine its output value. The activation function is usually one of several standard ones such as a linear, hyperbolic tangent, or sigmoid function [24]. The main idea behind a neural network is to create some architecture by chaining together layers of nodes and then provide that architecture a set of training data with the intent of generating predetermined output data. The network "learns" by cycling through all the input data and comparing its output to the predetermined output. After each cycle, the weights and biases of every node are updated so that the network slowly becomes better at producing the predetermined output. The result is a network of nodes that have been optimized such that the output is as close as possible to the desired output. Once the network has been trained, it can be saved as a model and then run against new test data that it has never seen before, thereby producing genuinely novel output [24]. Recall that autoencoders are a type of *deep* neural network. *Deep learning* is an umbrella term used to describe the subfield of machine learning that uses neural networks with several sequential hidden, a.k.a. middle, layers. Figure 9 shows such a network.

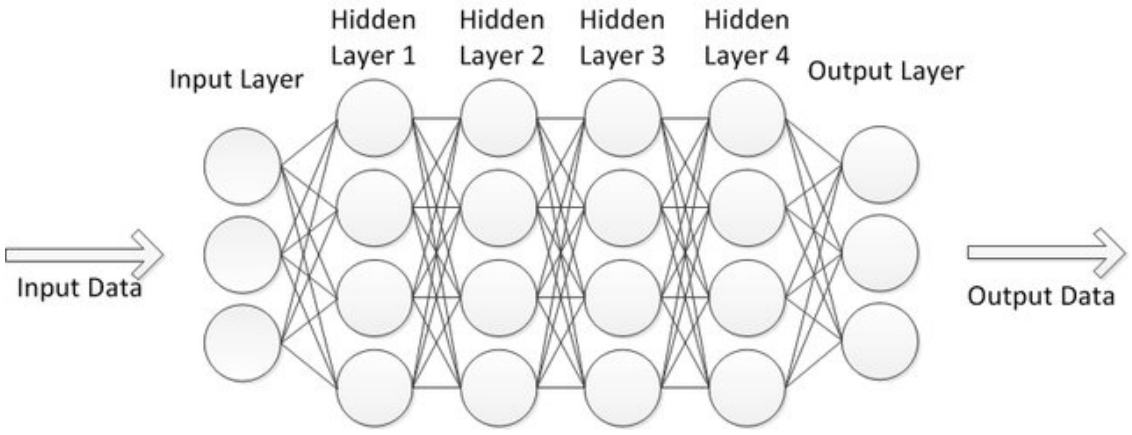
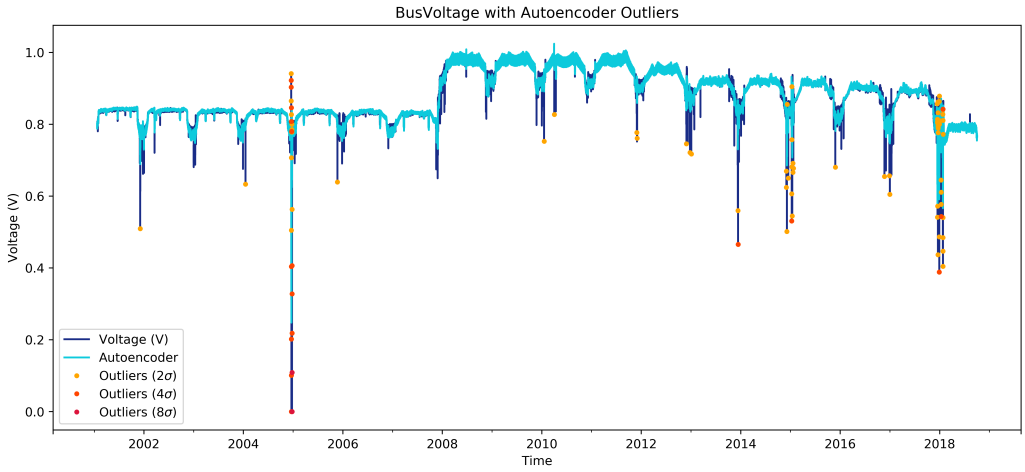


Figure 9: A multi-layer neural network with three input nodes, three output nodes, and four hidden layers consisting of four nodes each.

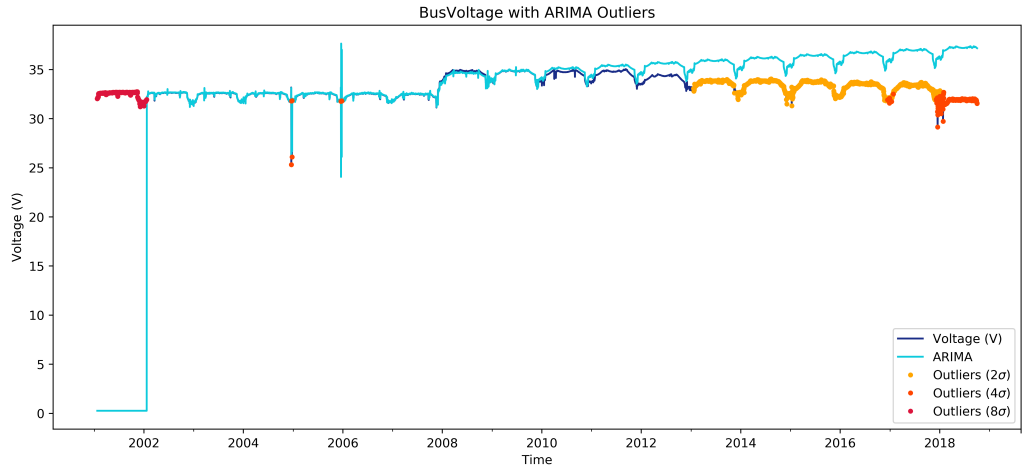
3.1.3.2 Autoencoders for Anomaly Detection

Autoencoders only learn what is most important to represent the core attributes of a dataset in their compressed middle layers, which excludes anomalies and noise. Because their outputs are reconstructed from these compressed forms, an immediate side effect of an autoencoder's behavior is that when input data has poor reconstruction accuracy, this implies that the input data was anomalous. Autoencoders can therefore be used to detect anomalies in time series datasets by examining the data's reconstruction accuracies and applying thresholds as to what is considered too poor of an accuracy.

The network architecture of an autoencoder typically contains far fewer input nodes than the number of data points in a given time series, so the series is fed through the network in small chunks. C. O'Meara et al. of the German Space Operations Center experimentally found that a network architecture consisting of 18 input and output nodes with hidden layers consisting of (150,75,10,75,150) nodes worked well to detect anomalies in their datasets [24]. In that case, a time series would be split into successive chunks of 18 points to be fed through their network.



(a) Anomalies detected from an Autoencoder prediction



(b) Anomalies detected from an ARIMA forecast

Figure 10: Anomalies detected using an autoencoder (top) and an ARIMA model (bottom). Both were trained on the first 50% of the Bus Voltage dataset and they use the same thresholding scheme.

Autoencoders have the benefit of being able to model non-stationary data which can give it an edge over ARIMA models, depending on the use case. Figure 10 shows the two different algorithms applied to the same dataset. One is not inherently better than the other because we are strictly dealing with unsupervised anomaly detection. The ARIMA model expects a certain pattern to continue indefinitely and labels an abundance of outliers when it does not. The autoencoder is more forgiving. It is crucial to note, however, that in contrast to ARIMA models, autoencoders are more of a magic "black box" solution. An ARIMA model is a clear-cut equation that details what the algorithm considers normal for the data. An autoencoder's predictions are by definition what the network has learned to consider normal, and if you want to know why it considers something normal versus anomalous, you have to shrug your shoulders because there is no clear-cut way to inspect the inner thoughts of the network.

3.1.3.3 Autoencoders for Feature Vector Extraction

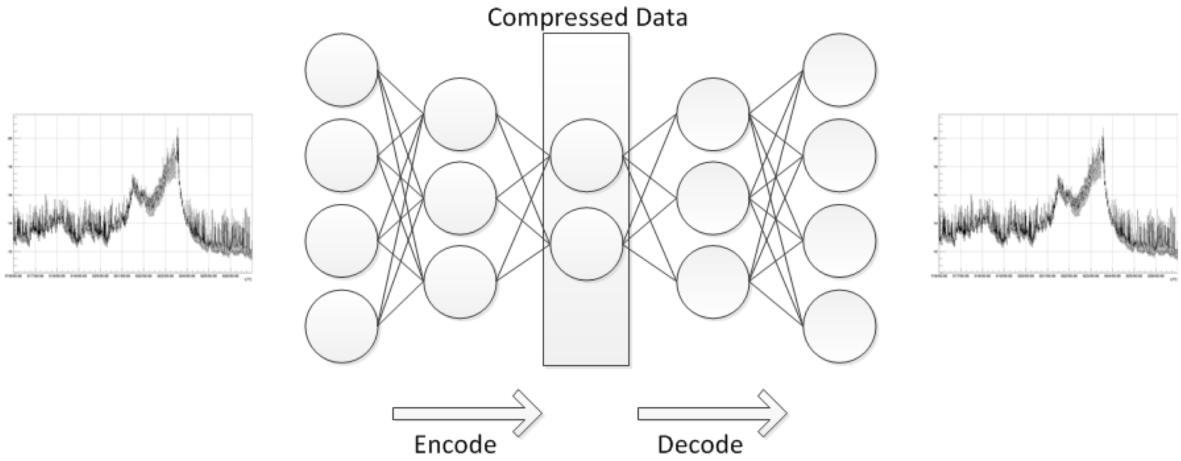


Figure 11: An autoencoder neural network architecture showing input and output layers with four nodes each and three hidden layers with three, two, and three nodes, respectively. The first half of the network encodes the data into a lower-dimensional space (a 2D middle layer here), then the second half decodes the data by enlarging the compressed representation in an attempt to reconstruct the original input data [24].

The previous sections described the basics of autoencoders and how they can be used for anomaly detection. Another fact about autoencoders is that the compressed representations of data on its way from input to output can be useful themselves. Formally, they are called *compressed feature vectors*. Compressed feature vectors represent time intervals of data and can be used to augment the existing human-engineered or human-understandable features currently in use [24, 16, 27]. In some cases, the compressed feature vectors can entirely replace the original time series data as input to an algorithm to achieve better results. We will see an example of this in Section 3.2.1 with *robust random cut forests*.

The key to obtaining the feature vectors is to first train an autoencoder so that it can correctly recreate the original input training data (up to some acceptable level of accuracy, usually between 80–90%). Then for each individual chunk of data fed into the network, we extract the output at the smallest hidden layer (the middle layer in Figure 11). Using Figure 11 as an example, each compressed vector of two numbers could stand in for each vector of four numbers that created it. If we do this for every data point in a time series, we obtain an entirely new representation of that time series. It may be incomprehensible to humans, but it can be useful in other algorithms.

3.2 Single Time Series Subsequence Outliers

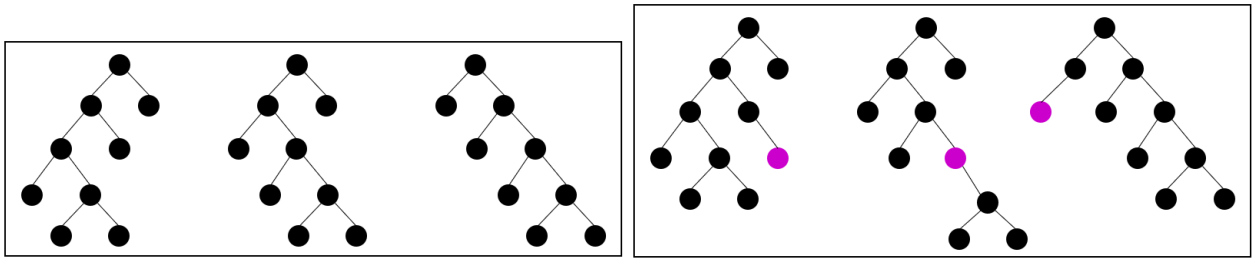
Section 3.1 focused solely on detecting point outliers in univariate time series datasets, but outliers can also be subsequences. Recall that subsequence outliers are multiple contiguous data points that collectively

count as one anomaly. This kind of anomaly is usually detected by algorithms that take as input sequential chunks of a dataset rather than single points from it or the whole thing at once. We saw in Section 3.1.3 that autoencoders take chunks of data as inputs; the reconstruction errors for each chunk could be averaged to yield subsequence outlier scores instead of individual scores for each output node. In general, however, according to the survey in [15], given a time series TS, the subsequence D of length N in TS is said to be an outlier when D has the largest distance to its nearest non-overlapping match. Most methods use Euclidean distance to compute the distance between subsequences, but Compression-based Dissimilarity Measure (CDM) is used as a distance measure in [20]. The brute force solution is to consider all possible subsequences of length N in TS and compute the distance of each with every other non-overlapping subsequence. Top-K pruning can be used to make this computation efficient. Furthermore, subsequence comparisons can be smartly ordered for effective pruning using various methods like heuristic reordering of candidate subsequences [19], locality-sensitive hashing [28], Haar wavelet and augmented tries [8, 11], or SAX with augmented trie [22]. None of these approaches will be covered in depth here, however. We will instead place the emphasis on robust random cut forests, and it will be the fourth algorithm used for the demonstrations in Section 4.

3.2.1 Robust Random Cut Forests

Robust random cut forests are collections of *robust random cut trees*, a data structure akin to a binary search tree that randomly "cuts up" sequences of data and places the points at leaf nodes [14]. Robust random cut forests provide the ability to label sequences of data with anomaly scores that are based on how "surprising" the points are, i.e., which points cause the most displacement in the depth of other points in the trees. This displacement is formally called a point's *collusive displacement*. Robust random cut forests offer a number of features that many competing anomaly detection algorithms lack. Specifically, they are designed to handle streaming data, they perform well on high-dimensional data, they reduce the influence of irrelevant dimensions, and their anomaly scores have clear underlying statistical meaning backed by mathematical proofs.

Let us build some intuition for what it means to be an anomaly in this context. Imagine someone observed a crowd of cats and dogs. They were a variety of colors, weights, and heights. Since they only observed cats and dogs of those colors, their model of an animal would consist of three traits: color, weight, and height. Now, they are shown a pink flamingo. It may weigh about the same as a cat but it has a very different color and height. To incorporate it into their model, they must amend their understanding of the world; it forces them to pay attention to a different set of features. Collusive displacement in a robust random cut forest conceptualizes the degree of anomaly as how much it forces this person to change their model.



(a) Before insert

(b) After insert

Figure 12: A diagram of a robust random cut forest before and after inserting a point.

A robust random cut forest can vary in four ways:

- **Number of trees:** The number of random cut trees in the forest
- **Subsample size:** The size of the random sample that the algorithm uses when constructing each tree
- **Time decay:** How much of the recent past to consider when computing an anomaly score
- **Shingle size:** The number of data points in a shingle (a sliding window)

This algorithm differs from the others described in the previous sections in that it does not attempt to recreate or predict data values. The calculated anomaly scores (collusive displacements) are not deviations from a model but instead act somewhat like probability densities except they are mathematically distinct from that. Moreover, while time series datasets can be input directly into the forests, in practice, some level of feature engineering is normally done on the data before it is input. Feature engineering is a broad enough process that detailing the possibilities is beyond the scope of this paper. But one technique that will be explored is to substitute shingles of data with the compressed features vectors that an autoencoder encoders from the shingles. This is done in the second demonstration of Section 4. The procedure is straightforward. A first shingle of data is fed into an autoencoder, its compressed feature vector is extracted, and that feature vector is input into a robust random cut forest. The resulting anomaly score is then assigned to the original shingle, and that process repeats for the rest of the dataset.

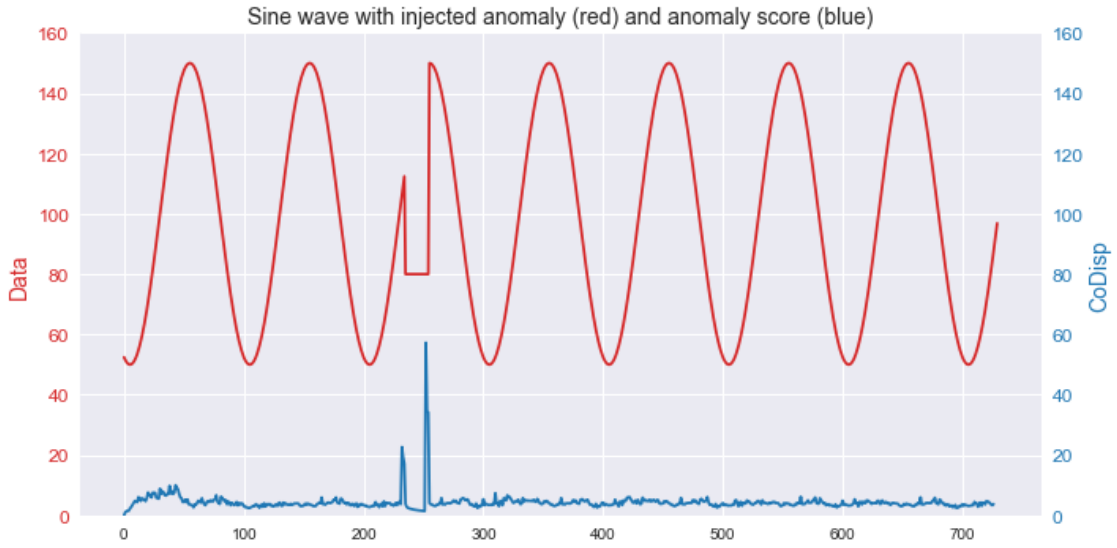


Figure 13: A sine wave with an injected anomaly plotted with the collusive displacement score from a robust random cut forest.

3.3 Time Series Databases

This paper focuses on single time series datasets, but approaches exist to detect anomalies within entire databases of telemetries. Some approaches go as far as to create a model of a whole spacecraft that captures the relationships between its various telemetry channels—e.g., if its thrusters fire, a nearby temperature reading is likely to go up [30, 13, 12]. In some cases, a priori knowledge of how the spacecraft's systems should interact is required. In others, the detection systems can learn those relationships with little a priori knowledge. This is the state of the art, but such approaches are beyond the scope of this paper. Lower-hanging fruit that is relevant to the topics of this paper are the use of autoencoders for *multivariate* time series. "Multivariate" means time series of multiple variables, like thruster output and temperature. We saw how to feed horizontal slices of univariate time series through an autoencoder network. With multivariate time series, vertical slices of the many variables at exactly one point in time are fed through during each input cycle. For a comprehensive overview of the possibilities for detecting anomalies in time series databases, refer to Section 2.1 of the survey in [15].

3.4 Unsupervised Anomaly Definitions

We have seen that a common pattern for unsupervised anomaly detection is to create a model of what is "normal" for a dataset, then when data "differs greatly" from that model, we call that anomalous. But we skirted the issue of defining "differs greatly" until this section. Recall from Section 1 that the output of an

anomaly detection algorithm can be either outlier scores or binary labels. If the output is the latter, the labels state what is an anomaly and there is no work to do. Outlier scores can be converted into binary labels by imposing thresholds on the scores, but thresholding is, in a sense, a simplified version of the initial anomaly detection problem. There will always be trade-offs between missing anomalies and generating false positives depending on where the threshold is set.

In a supervised setting, prelabeled anomalies can inform threshold values in that a threshold can be chosen that maximizes the number of true positives and minimizes the number of false positives. However, in our unsupervised setting, domain experts must intelligently decide their own thresholds with no objective metrics. This decision is often based on the statistical distribution of the data or the outlier scores. The following subsections describe three unsupervised approaches to thresholding outlier scores. They are by no means exhaustive of the possibilities, but they are practical starting points. Again, domain knowledge is ultimately required to know what is a good threshold for a particular dataset.

3.4.1 Thresholding with Standard Deviations from the Data's Mean

Given a time series and associated prediction model errors, one way to threshold those errors is to compare them against N number of standard deviations from the mean of the time series. That is, take the standard deviation σ of the time series, then where an error is the difference between a point in the time series and the same point in the prediction model, if that difference is greater than $N \cdot \sigma$, label that point an outlier (see Algorithm 3). This is a data-centric argument; it cares more about how the data normally varies than it does about how wrong the model tends to be. It is basically saying that the data shows some normal amount of variation, and the threshold should be based on the model being off by more than some multiple of that variation. Note that this may not be a reasonable approach for thresholding any outlier scores. It makes most sense with a prediction model that attempts to recreate the original time series curve.

Algorithm 3 Label outliers by thresholding errors with standard deviations from the data's mean

Input: TS, a time series, Y, a prediction model, and N, a number of standard deviations
Output: TS', a time series with labeled outliers

```
1:  function LabelOutliers(TS, Y, N)
2:      σ ← StandardDeviation(TS)
3:      for t in timestamps of TS do
4:          obs ← TS[t]
5:          y ← Y[t]
6:          error ← |y − obs|
7:          if error > N · σ then
8:              TS'[t, Outlier?] ← True          ▷ The data point at this timestamp is an outlier
9:          else
10:             TS'[t, Outlier?] ← False         ▷ The data point at this timestamp is not an outlier
11:     return TS'
```

3.4.2 Thresholding with Standard Deviations from the Scores' Mean

Given a time series and associated outlier scores from an anomaly detection algorithm, another way to threshold those scores is to compare them against N number of standard deviations from the mean of the scores themselves. That is, take the standard deviation σ of the outlier scores, then if a score is greater than $N \cdot \sigma$ plus the mean of the scores μ, label that point an outlier (see Algorithm 4). This can be expressed as $\tau = \mu + N\sigma$. This was used in [23], to give one example, but it is commonly used all over the place. It is a model-centric argument; it cares more about how wrong the model tends to be than about how the data normally varies. Note that this is a more general algorithm than the previous one because it works with any outlier scores, not just prediction model errors.

Algorithm 4 Label outliers by thresholding outlier scores with standard deviations from the scores' mean

Input: TS, a time series, *scores*, outlier scores, and N, a number of standard deviations
Output: TS', a time series with labeled outliers

function LABELOUTLIERS($TS, scores, N$)
  $\sigma \leftarrow$ **StandardDeviation(***scores***)**
  $\mu \leftarrow$ **Mean(***scores***)**
  $\tau \leftarrow \mu + N \cdot \sigma$
  for t in `timestamps of` TS **do**
    $obs \leftarrow TS[t]$
    $score \leftarrow scores[t]$
    if $score > \tau$ **then**
      $TS'[t, \texttt{Outlier?}] \leftarrow$ True ▷ The data point at this timestamp is an outlier
    else
      $TS'[t, \texttt{Outlier?}] \leftarrow$ False ▷ The data point at this timestamp is not an outlier
  return TS'

3.4.3 Nonparametric Dynamic Thresholding

The previous two approaches obtain binary labels for data points by thresholding outlier scores. This third approach, proposed by NASA's Jet Propulsion Laboratory (JPL), directly outputs binary labels for subsequences given a time series and prediction model values. It is called *nonparametric dynamic thresholding* [17]. The procedure is mathematically complicated, to the point that explaining it in detail is beyond the scope of this paper. JPL's own explanation is in Section 3.2 of [17]. At a high level, the procedure first generates a set of smoothed errors by taking the distances between actual data points and their predicted values and smoothing them using an exponentially-weighted average. An initial threshold is then set using a function of the mean and standard deviation of those smoothed errors. In simple terms, a threshold is found that, if all values above are removed, would cause the greatest percent decrease in the mean and standard deviation of the smoothed errors. The function also penalizes for having larger numbers of anomalous values and sequences to prevent overly greedy behavior. Once an initial set of anomalies is discovered, they go through a pruning procedure that can reclassify some anomalies as normal. The inner workings of nonparametric dynamic thresholding are complicated, but using it is exceptionally easy, and it can actually be a reasonable thing to try first if little domain knowledge about a problem exists. Pseudocode for using it is given in Algorithm 5. Although "nonparametric" is in its name, the algorithm uses six configuration variables that can optionally be tweaked to better suit a particular use case.

Algorithm 5 Label outliers by nonparametric dynamic thresholding

Input: TS, a time series, and Y, a prediction model
Output: TS', a time series with labeled outliers

function LABELOUTLIERS(TS, Y)
  $errors \leftarrow$ **GetErrors(**TS, Y**)**
  E_s, $scores \leftarrow$ **ProcessErrors(**$TS, scores$**)** ▷ E_s contains sets of outlier start and end indices (which, optionally, could be returned)
  for *anomaly* in E_s **do** ▷ Convert sets of outlier start and end indices into outlier points
    $start \leftarrow anomaly[0]$
    $end \leftarrow anomaly[1]$
    for t in *start–end* **do**
      $TS'[t, \texttt{Outlier?}] \leftarrow$ True ▷ The data point at this timestamp is an outlier
  return TS'

4 Demonstrations

Two demonstrations were carried out to showcase the four main algorithms from Sections 3.1 and 3.2 along with the three unsupervised anomaly definitions from Section 3.4. The first demonstration, diagrammed in the left half of Figure 14, compared how well the first three algorithms for modeling "normality"—rolling means, ARIMA, and autoencoders—could fit the WebTCAD datasets from Section 2. The second demonstration, diagrammed in the right half of Figure 14, compared how many anomalies were detected by the different anomaly definitions using the outputs of all four algorithms (including robust random cut forests). All anomalies were detected offline as opposed to online, and the results are in the tables following the two subsections. The data and plots were generated inside of Jupyter notebooks [1] that are publicly available in a GitHub repository[2].

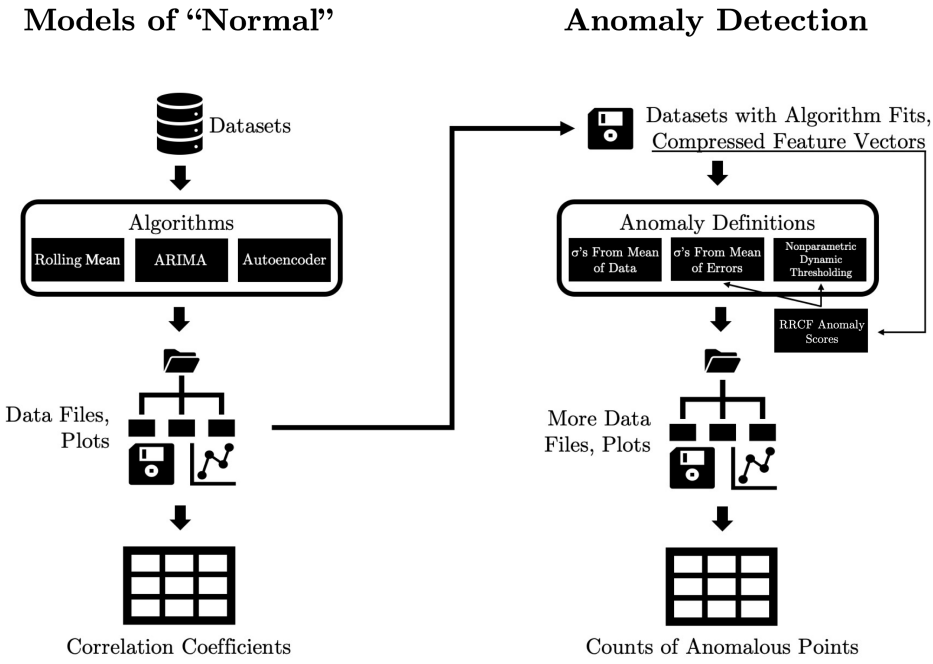


Figure 14: The flow of the two demonstrations in this paper.

4.1 Demonstration 1: Modeling "Normal"

Recall that the common procedure for unsupervised anomaly detection is to create a model of what is "normal" for a dataset, then when data "differs greatly" from that model, we call that anomalous. This first demonstration explored the first half of that procedure by comparing how rolling means, ARIMA, and autoencoders create models of our five datasets from Section 2, using correlation coefficients as an apples-to-apples comparison metric. The point was to explore how the algorithms work and to uncover any quirks in their behavior, not to exhaustively collect data on their nuances.

To use a rolling mean, a fixed window size must be set. To make an ARIMA model, all hyperparameters must be set, including the extra "trend" parameter added in the StatsModels *SARIMAX* implementation used here [2]. For autoencoders, a neural network architecture must be decided, and the data should be normalized between 0 and 1 for best performance. The rolling mean window sizes were set to be the length of each dataset $L/100$. ARIMA hyperparameters were set by grid searching with nested cross-validation, except for the the seasonal frequency m which should not be grid searched. A brute force search could, in theory, determine m but it is not worth the computational cost (plus, in production, the seasonal frequency

[2]https://github.com/sapols/Satellite-Telemetry-Anomaly-Detection/tree/master/demo

should be available in metadata). The daily averages of each dataset had to be used for ARIMA because the computational load of full-resolution datasets proved to be too much for the computer running the demonstrations. That is an unfortunate drawback of this demonstration. A good network architecture for the autoencoders was experimentally found to be $(18, 75, 10, 75, 18)$ nodes in each layer, but that certainly could have been more finely tuned if objective criteria for the model existed. Autoencoder training was done with batch sizes of 50 and 1000 epochs.

The ARIMA and autoencoder algorithms were run twice on each dataset: once where the first 50% of points were set aside as training data, and once where convention was purposefully ignored and the models were trained on the full datasets to demonstrate the problem of overfitting. The results are in Table 1. A selection of informative plots from this demonstration (Figures 15–17) are in Section 5.1.

	Bus Voltage	**Total Bus Current**	**Battery Temperature**	**Wheel Temperature**	**Wheel RPM**
Rolling Mean	0.926000	0.658593	0.712022	0.522871	0.909958
ARIMA (50% train)	0.941020	0.212979	0.945242	0.990172	0.619487
ARIMA (50% test)	-0.409896	-0.339883	0.409670	0.505791	0.133136
ARIMA (100%)	0.964081	0.873680	0.953052	0.986575	0.939315
Autoencoder (50% train)	0.978790	0.425565	0.942024	0.995748	0.997059
Autoencoder (50% test)	0.966947	0.827401	0.960382	0.997030	0.998642
Autoencoder (100%)	0.979787	0.871307	0.970714	0.998000	0.999488

Table 1: Correlation coefficients between the original datasets and the "normal" models generated by the three algorithms. ARIMA and autoencoder are separated into three rows each: *50% train*, where the model was fit to the first 50% of each dataset and the correlation was calculated on the training set, *50% test*, where the model was fit to the first 50% of each dataset and the correlation was calculated on the test set, and *100%*, where the model was fit to 100% of each dataset (no test set).

4.2 Demonstration 2: Detecting Anomalies

Using the models from the first demonstration, this second demonstration explored detecting point anomalies using the three unsupervised anomaly definitions from Section 3.4. For clarity, they are:

1. thresholding with standard deviations from the data's mean
2. thresholding with standard deviations from the scores' mean
3. nonparametric dynamic thresholding

The numbers of anomalies labeled from each definition were counted by dataset. Again, like the first demonstration, the point was not to be exhaustive but rather to explore how these different definitions affected the number of anomalies detected in each dataset.

The ARIMA and autoencoder algorithms were run twice on each dataset exactly like in the first demonstration: once with a proper 50% training set and once overfit on whole datasets. The robust random cut forest algorithm was also run twice on each dataset: once where the datasets themselves were given as input to the forests and once where the compressed feature vectors from autoencoders were used instead. The forests contained 100 trees, each of size 256, using a shingle size of 18. Robust random cut forest anomaly scores were input to definition 2 but not definition 1 because the forests are not prediction models. Definition 3 was experimentally tried on the forests' scores by substituting them for nonparametric dynamic thresholding's smoothed errors. The results are in Table 2. Figures 18–22 in Section 5.2 are a selection of informative plots from this demonstration, and they are visually more explanatory than the numbers in the table.

Rolling Mean

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from data's mean	96	172	1712	39720	19627
4σ from data's mean	27	36	13	5212	0
8σ from data's mean	6	0	0	15	0
2σ from scores' mean	1688	241	5656	41741	41375
4σ from scores' mean	167	73	830	9818	24941
8σ from scores' mean	58	7	4	415	2
Nonparametric dynamic thresholding	3828	700	5180	15770	15130

ARIMA (trained on 50% of data, test set anomalies)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from data's mean	2032	550	242	446	551
4σ from data's mean	303	79	11	60	0
8σ from data's mean	0	0	0	0	0
2σ from scores' mean	0	0	0	0	0
4σ from scores' mean	0	0	0	0	0
8σ from scores' mean	0	0	0	0	0
Nonparametric dynamic thresholding	280	750	910	1080	210

ARIMA (overfit on whole datasets, all anomalies)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from data's mean	9	30	27	10	57
4σ from data's mean	8	7	0	0	0
8σ from data's mean	0	0	0	0	0
2σ from scores' mean	0	0	0	0	0
4σ from scores' mean	0	0	0	0	0
8σ from scores' mean	0	0	0	0	0
Nonparametric dynamic thresholding	350	0	140	630	0

Autoencoder (trained on 50% of data, test set anomalies)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from data's mean	60	107	210	4	69
4σ from data's mean	5	13	9	1	0
8σ from data's mean	0	0	0	0	0
2σ from scores' mean	569	268	4089	10794	7018
4σ from scores' mean	230	37	1176	8389	3564
8σ from scores' mean	95	0	157	3715	1747
Nonparametric dynamic thresholding	420	673	1260	0	3306

Autoencoder (overfit on whole datasets, all anomalies)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from data's mean	84	56	132	36	17
4σ from data's mean	17	7	0	15	0
8σ from data's mean	1	0	0	1	0
2σ from scores' mean	839	207	4006	18445	6881
4σ from scores' mean	314	63	946	4218	2956
8σ from scores' mean	142	9	227	247	2071
Nonparametric dynamic thresholding	560	350	490	209	977

Robust Random Cut Forest (from data points)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from scores' mean	626	299	5280	26120	156837
4σ from scores' mean	323	68	1789	13676	0
8σ from scores' mean	70	0	172	1461	0
Nonparametric dynamic thresholding	3356	0	5188	71659	12531

Robust Random Cut Forest (from compressed feature vectors)

	Bus Voltage	Total Bus Current	Battery Temperature	Wheel Temperature	Wheel RPM
2σ from scores' mean	334	160	4718	13504	156358
4σ from scores' mean	216	90	1142	5803	0
8σ from scores' mean	60	18	287	1332	0
Nonparametric dynamic thresholding	2590	350	6752	40081	17893

Table 2: Counts of detected point anomalies for each algorithm, by dataset and anomaly definition.

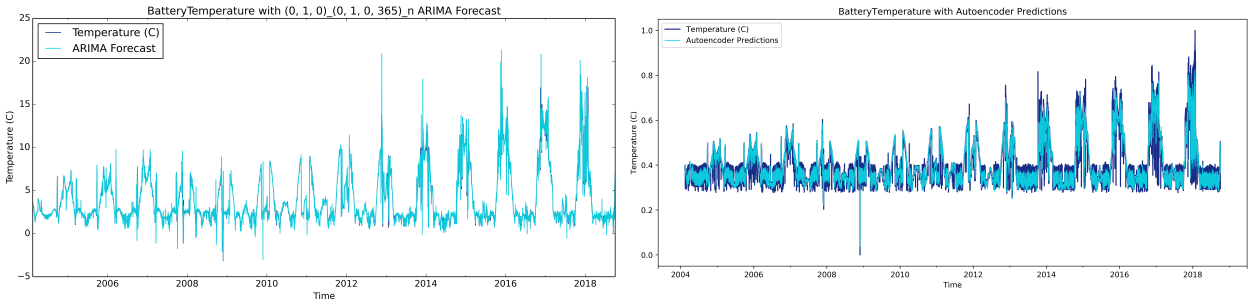
5 Observations

5.1 Demonstration 1 Observations

5.1.1 Overfitting

Demonstration 1 shed light on how rolling means, ARIMA, and autoencoders model the five WebTCAD datasets. To begin with the most important observation, and possibly the most obvious, overfitting should always be avoided. Prediction models work best for offline unsupervised anomaly detection when they are about 85–90% accurate, but all three algorithms are capable of recreating most of the datasets with nearly 100% accuracy. There is little room to call anything anomalous when so few deviations between data and predictions exist.

A rolling mean can perfectly recreate any dataset by setting its window size to one, so care should always be taken to set a reasonably large window. ARIMA models are not guaranteed to have such perfect replication accuracy, but they did turn out to be expressive enough to recreate the datasets with an average correlation coefficient of 0.94 when trained on whole datasets. The average rises to 0.97 if the Total Spacecraft Bus Current and Wheel RPM datasets are excluded; those two had relatively lower correlations because they are fundamentally not the kind of time series that can be predicted by ARIMA (see Figure 17b). Although the average correlation coefficient of the overfit autoencoders was 0.02 higher than that of ARIMA, autoencoders were less prone to overfitting (keep in mind that correlation coefficients measure how well two curves correlate, not necessarily how similar they are). That resistance to overfitting is evidence that their encoding and decoding behavior really does weed out anomalies and noise—compare Figures 15a and 15b. Note, however, that autoencoders become better at representing complex data as their networks grow in size. If the network is too large, it becomes too easy for the network to encode a dataset and that leads to overfitting.



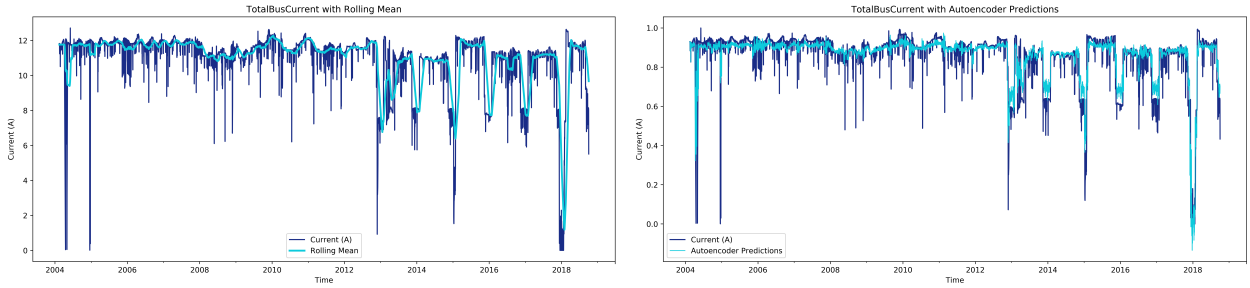
(a) ARIMA

(b) Autoencoder

Figure 15: ARIMA and autoencoder predictions, both trained on the entire Battery Temperature dataset. The autoencoder is less prone to overfitting.

5.1.2 Computational Complexity

Algorithm compute times were not given much attention in this paper, but they are worth discussing here. Both demonstrations were done on a MacBook Pro with eight CPU cores and sixteen gigabytes of RAM. That MacBook arguably did not have enough computing power, however, because it proved difficult to get certain algorithms, namely ARIMA, to finish on some of the datasets. ARIMA models would train for hours at a time—around five hours per dataset—and the program would often crash before the training finished. That is why daily averages of each dataset had to be used. The ARIMA training procedure crashed on every full-resolution dataset except for Total Spacecraft Bus Current which already had a daily cadence. Autoencoders usually took less than an hour to train and only crashed the MacBook once, while training on the Reaction Wheel Temperature dataset. Rolling mean was by far the fastest algorithm, taking only seconds or minutes to compute. And its output was not always substantially different from that of an autoencoder—compare Figures 16a and 16b. This demonstrates an important point from Section 3.1.1 that



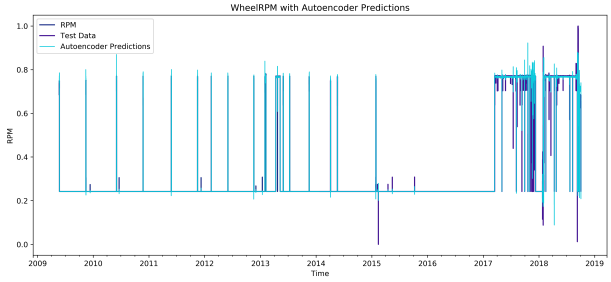
(a) Rolling mean

(b) Autoencoder

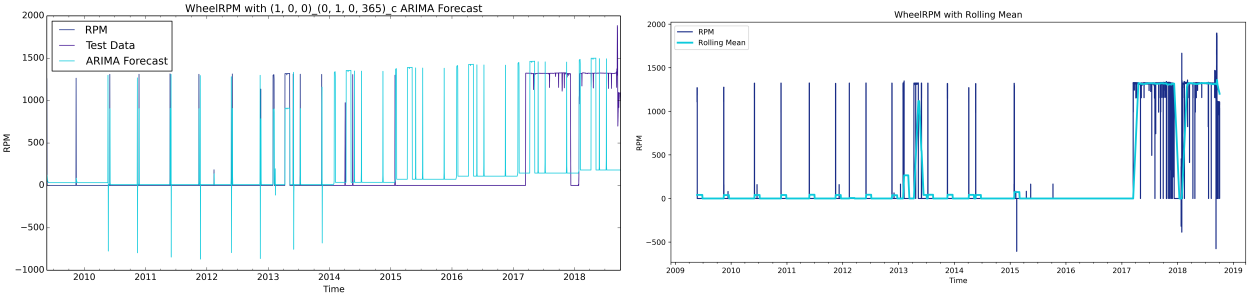
Figure 16: The Total Spacecraft Bus Current dataset with a rolling mean and an autoencoder prediction. The two results are not substantially different, but the rolling mean took orders of magnitude less time to compute.

simple statistical approaches can sometimes yield similar results to sophisticated machine learning approaches without the added computational complexity. In production, this could be the difference between being able to implement a solution or not. Rolling means should therefore be taken seriously despite their simplicity.

5.1.3 Advantages and Disadvantages



(a) Autoencoder



(b) ARIMA

(c) Rolling mean

Figure 17: The Wheel RPM dataset modeled with an autoencoder, an ARIMA model, and a rolling mean. The autoencoder arguably produced the best results. ARIMA was trained on the first half of the data and wrongly tried to deduce a repeating pattern from that training set. The rolling mean fails to capture the regular spikes in the data because of its window size.

The algorithms had clear advantages and disadvantages over each other. Their differences are well-

highlighted by the Wheel RPM dataset in Figure 17. Rolling mean's advantage is in its negligible computational cost and, when desirable, its ability to be tuned to fit unpredictable data by decreasing its window size (Total Spacecraft Bus Current and Wheel RPM being the most unpredictable of the WebTCAD datasets). The disadvantage of fitting rolling means to unpredictable data is that a small window may overfit and mask outliers. Plus, a window size that is good for one dataset is not necessarily good for another, so, in that sense, it does not generalize well. Autoencoders had the advantage of modeling the two unpredictable datasets reasonably well with static configurations, and deviations from an autoencoder's predictions are significant beyond simple statistics; anything an autoencoder does not predict is, by definition, not normal. ARIMA models were the least flexible in modeling unpredictable data but that can be an advantage depending on the use case. ARIMA is the only algorithm that can predict a pattern indefinitely into the future. If a dataset really can be described by a repeatable pattern and if that pattern should never change, ARIMA is the clear winner. An example of this ability is in Figure 5; if the pattern in the first half of the Reaction Wheel Temperature data was supposed to continue into the future, ARIMA is the only algorithm capable of marking the change point as an anomaly. However, ARIMA can produce strange predictions if a change point exists in the training set instead of the test set (this is shown in Figure 10b). Also, considering that correlations with an absolute value less than 80% are considered statistically insignificant, ARIMA was the only algorithm that could identify Total Spacecraft Bus Current and Wheel RPM as unpredictable datasets. The correlation coefficients for the training sets of those two datasets were both below 0.80 (0.212979 and 0.619487, respectively). By the same argument, autoencoders identified Total Spacecraft Bus Current as unpredictable with a correlation of 42.56%, but it considered Wheel RPM predictable.

5.1.4 Generalizations

Finally, a few generalization about how the algorithms work can be made. It is obvious what the output of a rolling mean will be: it will be the mean of each sliding window. The same cannot be said about the output of an autoencoder because autoencoders are essentially black box solutions. Autoencoders are also the most complicated to configure of all the algorithms due to the multitude of ways their neural networks can be configured, including the number of layers, the number of nodes in each layer, and the activation functions used by the nodes. The output of an ARIMA model is more predictable than an autoencoder's output. For seasonal data with trends, it seems that ARIMA tries to make the best representation of a seasonal cycle and the overall trend using the data in the training set. It then repeats that representation indefinitely during forecasting. This can be seen in Figures 5 and 17b. The ARIMA model will simply "learn" outliers if they exist in the training data. An approach has been suggested to allow ARIMA to detect anomalies in training data [31], but it is not covered in this paper. In simple terms, if a human can visually see a repeating shape in a time series, ARIMA can model it. Otherwise, it likely cannot.

5.2 Demonstration 2 Observations

Demonstration 2 showed how the unsupervised algorithms of this paper can detect anomalies. It also showed that the choice of how to define anomalies played just as important of a role as the choice of algorithm. Some combinations of algorithms and definitions did not yield any anomalies, while some yielded too many. This underscores the importance of domain knowledge in unsupervised settings. No one solution was objectively better than another because objective criteria cannot exist without prelabeled anomalies. The demonstration was therefore completely exploratory.

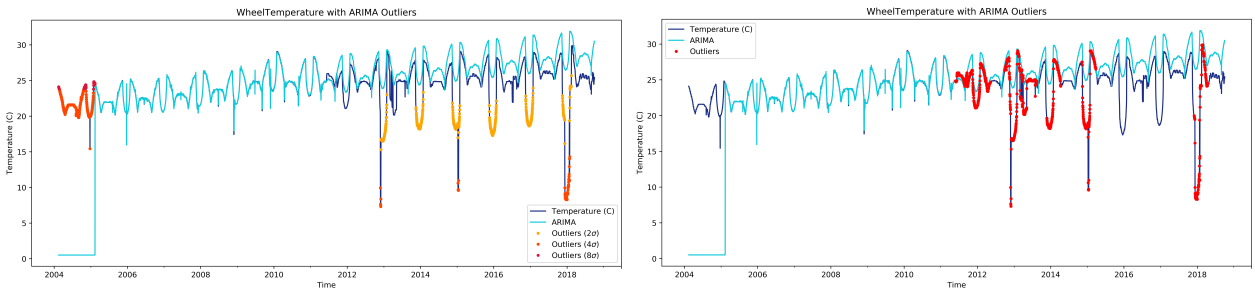
5.2.1 Rolling Mean Anomalies

Anomalies detected from a rolling mean are slightly more complicated forms of anomalies found in the classic sense of deviations from a normal distribution. An example of a normal distribution is height. Most adults are between five and six feet tall, making heights less than four feet or greater than seven feet outliers. In the context of time series telemetry, rolling mean outliers are abnormally high or low spikes data values. The best example of this is Figure 4. The benefit of using rolling means on time series instead of flat means is that they capture how the distributions of data change over time. The algorithm is incapable of detecting breaks from particular patterns because the rolling mean follows whatever the data does.

Labeling anomalies by thresholding outlier scores with standard deviations from the scores' mean yielded the most anomalies on average. Those anomalies were not categorically different than the ones labeled by thresholding with standard deviations from the data's mean, it was just more common for scores to be within two to eight standard deviations from the scores' mean. The two definitions yield extremely similar results when points are compared against a greater number of standard deviations from the data's mean. Nonparametric dynamic thresholding labeled subsequence anomalies rather than lone point anomalies, and its pruning procedure did exclude some of the least anomalous subsequences. That is evidence that nonparametric dynamic thresholding is likely best at minimizing false positives.

5.2.2 ARIMA Anomalies

The number of anomalies detected from ARIMA models depended largely on the anomaly definition used and whether the models had been overfit to whole datasets. If they were overfit, very few, if any, anomalies were labeled. Absolutely no anomalies were labeled by the definition of thresholding with the scores' mean because none fell within the minimum distance of two standard deviations. An abundance of anomalies were labeled by the other definitions when a change point occurred in a dataset (e.g., Figure 10b), which demonstrates ARIMA's effectiveness at detecting breaks from particular patterns. That is the algorithm's strength. It produced strange forecasts after being trained on the Total Spacecraft Bus Current and Wheel RPM datasets because, again, ARIMA is fundamentally incapable of predicting them. It is clear that ARIMA should never be used on any datasets that are similarly unpredictable because the labeled anomalies are meaningless.



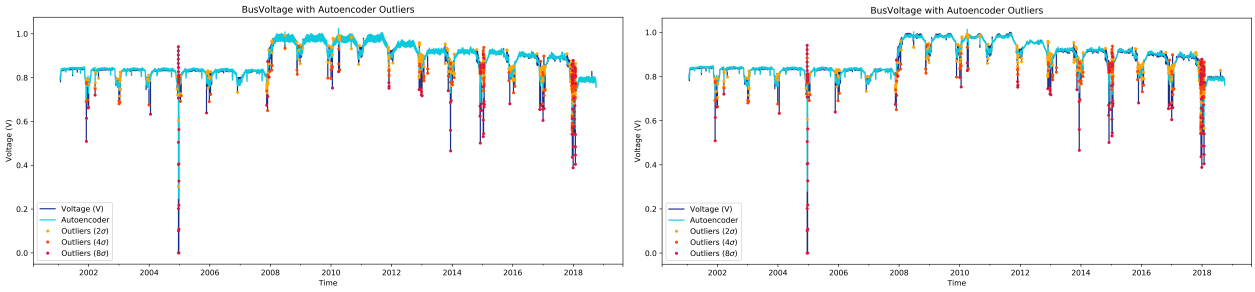
(a) Thresholding with standard deviations from the data's mean

(b) Nonparametric dynamic thresholding

Figure 18: Anomalies detected in the Reaction Wheel Temperature dataset by an ARIMA model trained on the first half of the data.

5.2.3 Autoencoder Anomalies

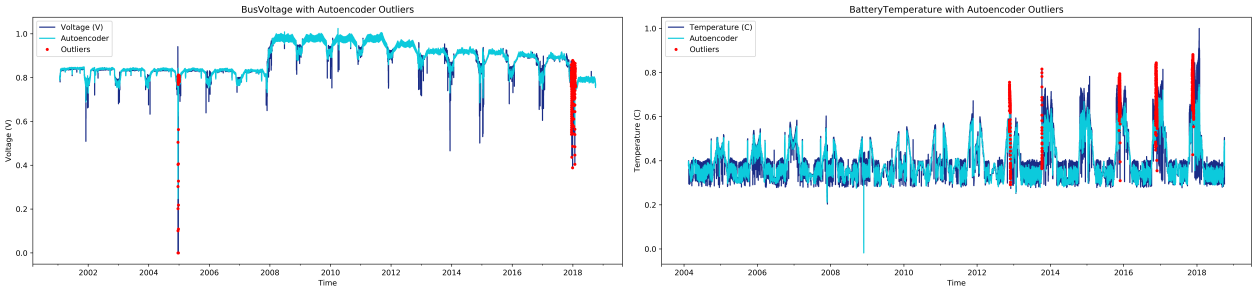
Anomalies detected from an autoencoder represent deviations from what it has learned to consider normal for a given dataset. Conclusions can be drawn from the demonstrations in this paper, but it must be noted that because such a wide variety of configuration options were not explored, these demonstrations are a drop in the bucket compared to what is possible with autoencoders. Much like ARIMA, the number of detected anomalies depended largely on whether the models had been overfit—minus a notable exception in Figure 19. Less were detected when they were overfit but not as few as with ARIMA because overfitting was not as much of a problem with autoencoders. In contrast to ARIMA, more anomalies were labeled while thresholding with the scores' mean than with the data's mean. The pruning procedure of nonparametric dynamic thresholding seemed to play a bigger role with autoencoders than with the other algorithms; only a couple subsequences were labeled per dataset on average (see Figure 20). The exception to this was the Wheel RPM dataset where about half of all instances of rotating wheels were labeled as anomalous by nonparametric dynamic thresholding, suggesting that definition is a bad pair with that dataset.



(a) Trained on the first half of the dataset

(b) Trained on the entire dataset

Figure 19: Anomalies detected in the Bus Voltage dataset by autoencoders, thresholding anomaly scores with standard deviations from the scores' mean. The results are largely the same because the two models produced very similar predictions. Eight standard deviations seems to be more reasonable than two in this case.



(a) Bus Voltage

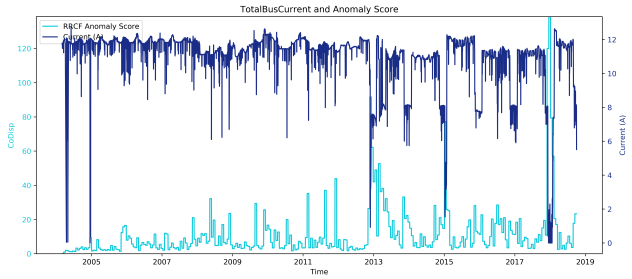
(b) Battery Temperature

Figure 20: Anomalies detected in the Bus Voltage and Battery Temperature datasets by autoencoders trained on the first halves of data, using nonparametric dynamic thresholding. Many anomalous subsequences were reclassified as normal by the pruning procedure.

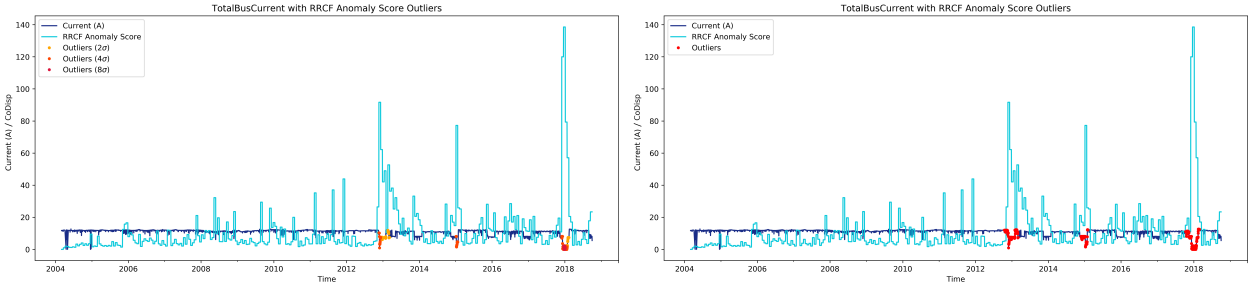
5.2.4 Robust Random Cut Forest Anomalies

The last algorithm in this demonstration was the robust random cut forest. The first observation is that it was the second slowest algorithm after ARIMA. It took on the order of one to three hours to assign anomaly scores to all the points in a dataset. In spite of this, had the anomalies been detected online instead of offline, this algorithm would have been the fastest by a wide margin. This is because after being slowly initialized with a set of data points, a new score can be assigned to an incoming point after one quick insertion into the forest. Similar to the autoencoders, configuration options were not systematically explored. Changes to any of the four robust random cut forest parameters could have affected run times and overall detection performance.

A compelling observation is that the algorithm performed better when the compressed feature vectors from autoencoders were given as input instead of the datasets themselves. Around half of the points in each dataset—except for Bus Voltage and Total Spacecraft Bus Current—were counted as anomalies when the original datasets were input. Figure 22 is a particularly striking example. The anomaly scores also showed less variability which makes it more difficult to distinguish anomalies from regular data. When the compressed feature vectors were input, the anomaly scores showed more variability and a more manageable number of anomalies were detected (see Figure 21). Robust random cut forests did produce the highest counts of anomalies on average, however, suggesting that a higher number of standard deviations in the anomaly definitions could have been used.



(a) Total Spacecraft Bus Current with robust random cut forest anomaly scores



(b) Thresholding anomaly scores with standard deviations from their mean

(c) Thresholding anomaly scores with nonparametric dynamic thresholding

Figure 21: The Total Spacecraft Bus Current dataset with robust random cut forest anomaly scores, where compressed feature vectors were input in lieu of the original data.

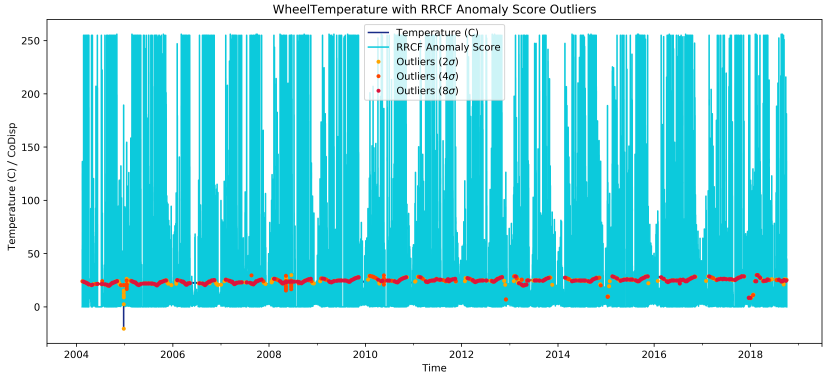


Figure 22: Anomalies detected in the Reaction Wheel Temperature dataset by a robust random cut forest that took the original dataset as input. The majority of data points were labeled as anomalies (thresholding the scores with standard deviations from their mean), suggesting this was not a good use of the algorithm. Greatly increasing the number of standard deviations may have helped.

6 Discussion

This has been, in part, an attempt to answer the question: "How good could machine learning for anomaly detection in WebTCAD be?" Four main algorithms were described and tested on the five time series datasets that represent the kinds of telemetry that exist in the WebTCAD database. Many other techniques were referenced but not described in detail. Only unsupervised techniques were explored because WebTCAD lacks any labeled anomalies that are necessary for supervised techniques. This was an informative view into how the techniques work, but it is hard to objectively evaluate them without being able to compare their results against prelabeled anomalies. If WebTCAD had prelabeled anomalies, statistical metrics could be used to minimize the number of false positives produced by an anomaly detection system. Even one prelabeled anomaly would be better than none.

This scratched the surface of what is possible in the field of anomaly detection, and none of these solutions are general enough to be applied to every dataset in the WebTCAD database. It is evident, however, that WebTCAD could be given the ability to detect anomalies through solutions that are far better than the limit violations in use today. Using robust random cut forests with the compressed feature vectors from autoencoders and then thresholding the resulting anomaly scores with nonparametric dynamic thresholding is state of the art for this field; the techniques were only proposed in the literature within the last few years. The other techniques have been around for a long time, but they are relatively low-hanging fruit. Any combination of the techniques in this paper could be the start of improving WebTCAD's automated detection abilities.

The biggest barriers here are the computational resources required by some of the algorithms and possibly the advanced machine learning expertise needed to optimize them. There is much room for future work. Model hyperparameters could be more finely tuned according to objective metrics. More complex models could be explored, specifically the ones that capture the relationships between various telemetry channels of a spacecraft. Anomalies could be detected online instead of offline. Finally, forecasting could be used to potentially detect anomalies before they happen, i.e, anomalies could be detected in the forecasts.

Acknowledgements

I gratefully acknowledge Dr. Claire Monteleoni and Marcus Hughes for valuable discussions.

References

[1] https://jupyter.org.

[2] https://www.statsmodels.org.

[3] Charu C Aggarwal. An introduction to outlier analysis. *Springer New York*, 2017.

[4] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[5] Laurent-Béatrice Loubes Jean-Michel Cabon Bertrand Barreyre, Clémentine and Loïc Boussouf. Statistical methods for outlier detection in space telemetries. 2018 SpaceOps Conference, 2018.

[6] Josh Blumenfeld. Getting ready for nisar - and for managing big data using the commercial cloud. *Earthdata*, 2019.

[7] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[8] Yingyi Bu, Tat-Wing Leung, Ada Wai-Chee Fu, Eamonn Keogh, Jian Pei, and Sam Meshkin. Wat: Finding top-k discords in time series database. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 449–454. SIAM, 2007.

[9] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[10] J. Contreras, R. Espinola, F. J. Nogales, and A. J. Conejo. Arima models to predict next-day electricity prices. *IEEE Transactions on Power Systems*, 18(3):1014–1020, Aug 2003.

[11] Ada Wai-Chee Fu, Oscar Tat-Wing Leung, Eamonn Keogh, and Jessica Lin. Finding time series discords based on haar transform. In *International Conference on Advanced Data Mining and Applications*, pages 31–41. Springer, 2006.

[12] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An anomaly detection method for spacecraft using relevance vector learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 785–790. Springer, 2005.

[13] Ryohei Fujimaki, Takehisa Yairi, and Kazuo Machida. An approach to spacecraft anomaly detection problem using kernel feature space. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 401–410. ACM, 2005.

[14] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. Robust random cut forest based anomaly detection on streams. In *International conference on machine learning*, pages 2712–2721, 2016.

[15] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, Sep. 2014.

[16] GE Hinton and RR Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313 (5786): 504–507, 2006.

[17] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 387–395. ACM, 2018.

[18] HV Jagadish, Nick Koudas, and S Muthukrishnan. Mining deviants in a time series database. In *VLDB*, volume 99, pages 7–10, 1999.

[19] Eamonn Keogh, Jessica Lin, and Ada Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. Ieee, 2005.

[20] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. Towards parameter-free data mining. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215. ACM, 2004.

[21] Y LeCun, Y Bengio, and G Hinton. Deep learning. nature 521 (7553): 436. *Google Scholar*, 2015.

[22] Jessica Lin, Eamonn Keogh, Ada Fu, and Helga Van Herle. Approximations to magic: Finding unusual medical time series. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pages 329–334. Citeseer, 2005.

[23] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

[24] Corey OMeara, Leonard Schlag, and Martin Wickler. *Applications of Deep Learning Neural Networks to Satellite Telemetry Monitoring.*

[25] Fernando Sanchez, Christopher Pankratz, Douglas M Lindholm, Ransom Christofferson, Darren Osborne, and Thomas Baltzer. Webtcad: A tool for ad-hoc visualization and analysis of telemetry data for multiple missions. In *2018 SpaceOps Conference*, page 2616, 2018.

[26] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[27] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[28] Li Wei, Eamonn Keogh, and Xiaopeng Xi. Saxually explicit images: Finding unusual shapes. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 711–720. IEEE, 2006.

[29] A. H. Yaacob, I. K. T. Tan, S. F. Chien, and H. K. Tan. Arima based network anomaly detection. In *2010 Second International Conference on Communication Software and Networks*, pages 205–209, Feb 2010.

[30] Takehisa Yairi, Yoshikiyo Kato, and Koichi Hori. Fault detection by mining association rules from housekeeping data. In *proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, volume 18, page 21. Citeseer, 2001.

[31] B. Zhu and S. Sastry. Revisit dynamic arima based anomaly detection. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*, pages 1263–1268, Oct 2011.