Journal of Artificial Intelligence Research 72 (2021) 849-899 Submitted 01/2021; published 11/2021

Experimental Comparison and Survey of Twelve Time Series Anomaly Detection Algorithms

Cynthia Freeman SHINGLIFREEMAN@GMAIL.COM
Jonathan Merriman JONATHAN.MERRIMAN@VERINT.COM
Ian Beaver IAN.BEAVER@VERINT.COM
Verint Intelligent Self-Service
12809 Mirabeau Pkwy, Spokane Valley, WA 99216

Abdullah Mueen MUEEN@CS.UNM.EDU
University of New Mexico Computer Science Department
1901 Redondo S Dr, Albuquerque, NM 87106

Abstract

The existence of an anomaly detection method that is optimal for all domains is a myth. Thus, there exists a plethora of anomaly detection methods which increases every year for a wide variety of domains. But a strength can also be a weakness; given this massive library of methods, how can one select the best method for their application? Current literature is focused on creating new anomaly detection methods or large frameworks for experimenting with multiple methods at the same time. However, and especially as the literature continues to expand, an extensive evaluation of every anomaly detection method is simply not feasible. To reduce this evaluation burden, we present guidelines to intelligently choose the optimal anomaly detection methods based on the characteristics the time series displays such as seasonality, trend, level change concept drift, and missing time steps. We provide a comprehensive experimental validation and survey of twelve anomaly detection methods over different time series characteristics to form guidelines based on several metrics: the AUC (Area Under the Curve), windowed F-score, and Numenta Anomaly Benchmark (NAB) scoring model. Applying our methodologies can save time and effort by surfacing the most promising anomaly detection methods instead of experimenting extensively with a rapidly expanding library of anomaly detection methods, especially in an online setting.

1. Introduction

Time series are used in almost every field: intrusion and fraud detection, tracking key performance indicators (KPIs), the stock market, and medical sensor technologies. One important use of time series is the detection of *anomalies* for ensuring undisrupted business, efficient troubleshooting, or, in the case of medical sensor technologies, lower the mortality rate. However, time series anomaly detection is a notoriously difficult problem for a multitude of reasons:

1. **What is anomalous?** An *anomaly* in a time series is a pattern that does not conform to past patterns of behavior in the series. What is defined as anomalous may differ based on application. The existence of a one-size-fits-all anomaly detection method that is optimal for all domains is a myth (Laptev et al., 2015). In addition, inclusion of contextual variables may change initial perceptions of what is anomalous. Suppose, on average, the number of daily bike rentals is 100, and one day, it was 10. This

initially may appear anomalous, but if this occurred on a cold, winter day, this is actually not so surprising. In fact, it might appear more anomalous if there were 100 rentals instead. There are also different *types* of anomalies (e.g. point, contextual, and collective anomalies (Banerjee et al., 2008), and some anomaly detection methods are better than others at detecting certain types.

2. **Online anomaly detection.** Anomaly detection often must be done on real-world streaming applications. Strictly speaking, an *online* anomaly detection method must determine anomalies and update all relevant models before occurrence of the next time step (Saurav et al., 2018). Depending on the needs of the user, it may be acceptable to detect anomalies periodically. Regardless, efficient anomaly detection is vital which presents a challenge.

3. **Lack of labeled data.** It is unrealistic to assume that anomaly detection systems will have access to thousands of tagged datasets. In addition, given the online requirement of many such systems, it can be easy to encounter anomalous (or not anomalous) behavior that was not present in the training set.

4. **Data imbalance.** As an anomaly is a pattern that does not conform to past patterns of behavior, non-anomalous data tends to occur in much larger quantities than anomalous data. This can present a problem for a machine learning classifier approach to anomaly detection as the classes are not represented equally. Thus, an accuracy measure might present excellent results, but the accuracy is only reflecting the unequal class distribution in the data (the *accuracy paradox*). For example, if there are 100 data points and only 2 anomalies, a classifier can deem every point as non-anomalous and achieve 98% accuracy.

5. **Minimize False Positives.** It is important to detect anomalies as accurately and efficiently as possible, but minimizing false positives is also of great necessity to avoid alarm fatigue. Alarm fatigue can lead to a serious alert being overlooked and wasted time in checking for problems when there are none.

6. **What should I use?** There is a massive wealth of anomaly detection methods to choose from (Campos et al., 2016; Emmott et al., 2015; Wu, 2016; Cook et al., 2019; Chandola et al., 2009a; Hodge & Austin, 2004).

Because of these difficulties inherent in time series anomaly detection, we believe there is a strong need for a comprehensive experimental comparison which can:

1. Survey the landscape *and* demonstrate which anomaly detection methods are more promising given different types of time series characteristics.

2. Highlight the differences between several scoring methodologies (windowed F-scoring, Area Under the receiver operating characteristic Curve, and Numenta Anomaly Benchmark scoring)

3. Provide a foundation for time benchmarks essential in an online setting.

4. Reveal important omissions in the anomaly detection methods themselves.

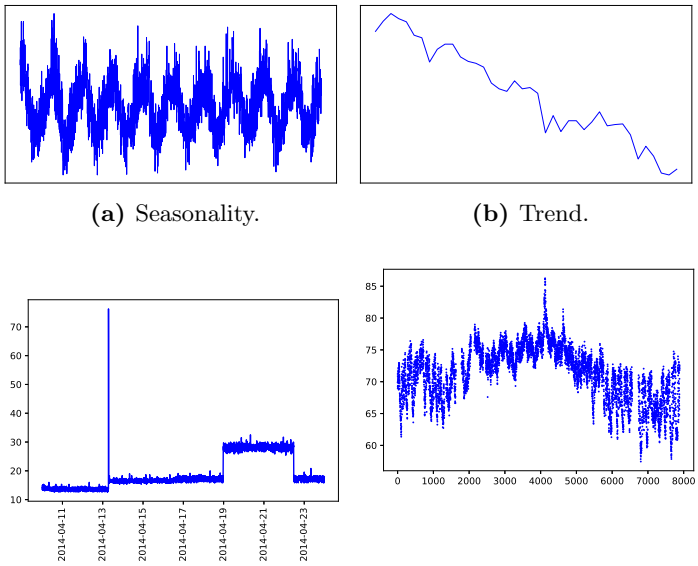
We present guidelines for automating the classification of univariate time series and choice of anomaly detection method based on the characteristics the time series possesses.[1] For example, if the time series in a user's application exhibits concept drift and no seasonality, which anomaly detection method would perform best? We make these analysis by conducting a thorough experimental comparison of a wide range of anomaly detection methods and evaluate them using both windowed F-scores, AUC (Area Under the receiver operating characteristic Curve), and NAB (Numenta Anomaly Benchmark)[2] scores. Anomaly detection often must be done on real-world streaming applications. Thus, we include the time it took to train the methods as well as detection. Finally, in experimentally testing anomaly detection methods on a wide variety of datasets, we reveal areas where many of these methods are lacking but are not brought to light. For example, Twitter AnomalyDetection VEC can only be used with seasonal datasets.

These are our main contributions:

- Either re-implemented or used existing libraries to test 12 different anomaly detection methods[3]
- Test these methods on different time series characteristics (seasonality, trend, concept drift, and missing time steps). For example, we could determine how well Facebook Prophet performs on concept drift by observing its results on 10 time series datasets all exhibiting concept drift.
- Created new benchmark datasets for anomaly detection.
- Compare and contrast multiple scoring methods: windowed F-scoring, AUC, and Numenta Anomaly Benchmark scoring. As mentioned previously, accuracy, alone, is a poor measure of an anomaly detection method's performance.
- Provided analysis and guidelines based on these results. Applying these guidelines can save time and effort by surfacing the most promising methods for anomaly detection instead of experimenting extensively with a rapidly expanding library of anomaly detection methods, especially in an online setting.

The rest of the paper is organized as follows. After a discussion of time series characteristics and datasets, we proceed with a description of all anomaly detection methods we experiment with, parameters used, and how we evaluate performance. After obtaining the results, we derive several guidelines on method selection by time series characteristics and outline areas for future work.

1. Similar analysis (Emmott et al., 2015) has been performed before to compute the influence of meta-data on anomaly detection but for feature-vector-based datasets instead of time series.
2. See Appendix for a table of all acronyms and their definitions.
3. See `https://github.com/dn3kmc/jair_anomaly_detection` for all source code implementations, Jupyter notebooks demonstrating how to determine characteristics, and datasets.



(a) Seasonality. **(b)** Trend.

(c) Level Change Concept Drift. **(d)** Missing points.

Figure 1: **a** An example of a non-stationary time series exhibiting seasonality. **b** An example of a non-stationary time series exhibiting a downward trend. **c** An example of a non-stationary time series displaying level change concept drift around 2014-04-19 and 2014-04-23. Another concept drift occurs around 2014-04-13 shortly after an anomalous spike. **d** An example of a time series with missing time steps. The time series is displayed as a scatter plot to showcase the missing points, especially around time step 6500.

2. Preliminaries

We follow the same definition established by Wang et al. (2013) where a *time series* is defined as a sequence of pairs

$$T = [(p_1, t_1)...(p_n, t_n)](t_1 < t_2 < ... < t_n)$$

where each p_i is a data point in a d-dimensional space and each t_i represents the time stamp at which the corresponding p_i occurs. Given the challenges present in anomaly detection, we aim to simplify the process in choosing the most promising anomaly detection methods for a given *univariate* time series. In the following subsections, we will:

1. Discuss several characteristics that a time series can possess
2. Consider ways to automatically detect these characteristics
3. Provide example datasets for said characteristics
4. Review a variety of anomaly detection methods

2.1 Time Series Characteristics

We begin with a description of time series characteristics (see Figures 1a to 1d).

Stationarity A time series is *stationary* if the mean, variance, and autocorrelation structure are constant for all time t (Hyndman & Athanasopoulos, 2018). A white noise process is stationary, for example. Most time series encountered in the real world are not stationary. Many anomaly detection methods require a time series to be made stationary first as a preprocessing step.

Non-stationarity Non-stationary time series may exhibit seasonal behavior (see Figure 1a). For example, it is likely that there are more bike rentals in the summer and fewer in the winter, and that this behavior repeats year after year. Other behaviors that may be exhibited by non-stationary time series include trend (see Figure 1b) and concept drift (see Figure 1c which shows an example of *level change* concept drift) where the definition of normal behavior changes over time (Saurav et al., 2018). Initially, a concept drift may trigger an anomaly, but an *adaptive* system should quickly determine that a new pattern of behavior has occurred and is the new "normal". Here we focus on level change concept drift and leave other types of concept drift for future work.

Sampling Rate An *online* anomaly detection method requires that anomalies be determined and models be updated before occurrence of the next time step or some potentially larger time frame as specified by the user. In cases where anomaly detection is done online, the sampling rate can restrict the kinds of anomaly detection methods under consideration.

Missing Time Steps Missing time steps occur in time series often due to the inherent nature of what is being measured. Suppose we create a time series of average load times of a webpage over one minute windows. We cannot expect that pages are being loaded every minute; this is dependent on how customers are using the webpage. Thus, some one minute periods will not have a measurement. Missing time steps are a frequent occurrence in real-world datasets (see Figure 1d). Missing time steps may make it difficult to apply anomaly detection methods without some form of interpolation. However, other methods can handle missing time steps innately.

In summary, the characteristics of time series we discuss are:

- Non-stationarity
  - Seasonality (10)
  - Level Change Concept Drift (10)
  - Trend (10)
- Sampling Rate
- Missing Time Steps (10)

The number of datasets we analyze exhibiting seasonality, concept drift, trend, and missing time steps are in parenthesis. We consider 5 minute, 30 minute, 1 hour, 1 day, and 1 month sampling rates.

2.2 Detection of Time Series Characteristics

Although some of the characteristics discussed are easy to determine visually, it can be more difficult to determine if a time series is stationary in an automated fashion. Naively, one could partition the dataset and compare summary statistics of every partition. But it is cumbersome to determine the location and sizes of these partitions; one partitioning scheme can reveal differences while another may suggest constant mean and variance. We make use of several statistical tests to detect time series characteristics.

2.2.1 Detecting Trend

We detect two types of trend: **stochastic** and **deterministic** trend. For stochastic trends, the mean trend is stochastic. Stochastic trend is removed using differencing, where we subtract the previous observation from the current observation. Because it is removed via differencing, stochastic trend is sometimes called **difference-stationary trend**. For deterministic trends, the mean trend is deterministic. Deterministic trends are removed by detrending the time series, where you find some line that best fits the data and then subtract it from the data. That is why they are sometimes called **trend-stationary**.

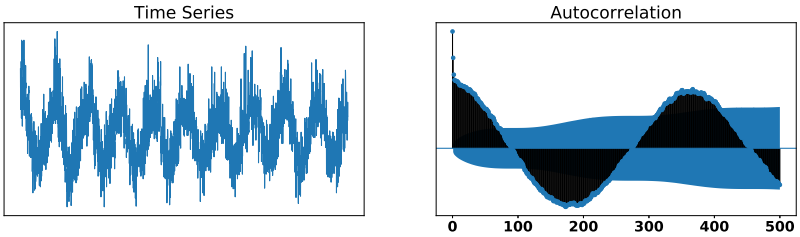
We use the formulation established by Barbosa (2011) to detect stochastic and deterministic trend. Barbosa (2011) uses two statistical tests: the KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test and the PP (Phillips-Perron) test. The null hypothesis of the KPSS test is that the time series is trend-stationary (deterministic trend exists) whereas the null hypothesis of the PP test is that the time series is difference-stationary (stochastic trend exists).

If the null hypothesis of the KPSS test is rejected but the null hypothesis for the PP test is not rejected, the time series exhibits difference-stationarity. If the null hypothesis of the KPSS test is not rejected but the null hypothesis for the PP test is rejected, there is trend-stationartiy. For all other cases, the tests and/or time series is not informative enough for discriminating behavior.

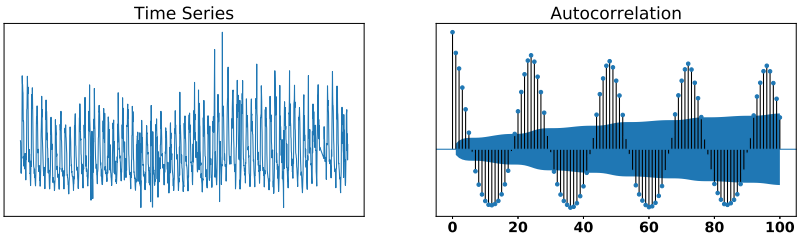
2.2.2 Detecting Seasonality

Although seasonality can sometimes be determined at a glance (see Figure 2a), other times it is not so obvious (see Figure 2b). Thus, it is often necessary to analyze the ACF (Auto-Correlation Function) plot which displays the correlation for time series observations with its lags (observations from previous time steps). A time series that exhibits seasonality will show a repetitive pattern in its autocorrelation plot like in Figure 3a. The x-axis shows the lag value. The y-axis shows the Pearson correlation. This correlation coefficient is between -1 and 1 and describes a negative or positive correlation, respectively. A value of zero indicates no correlation. The dashed line[4] represents the 95% confidence interval. Values outside of this dashed line are very likely a correlation and not a statistical fluke. For example, in Figure 3b, the time series does not contain seasonality.

4. Figures 2a and 2b are generated using *Pandas*, with the confidence interval represented as a blue cone. Figures 3a and 3b are generated using *stats* in R, with the confidence interval represented as the dashed, blue line. The reason for the difference in display of confidence intervals is that *Pandas* takes into account the fact that time series values that are close in time would be more highly correlated than ones that are more far apart.

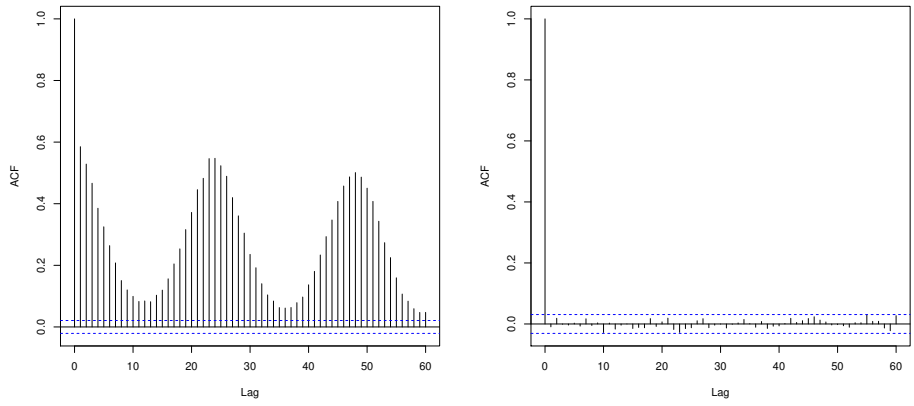


(a) On the left is a time series displaying seasonal behavior. The ACF of this time series (on the right) also provides evidence of this behavior. However, seasonality is obvious from visual inspection of the time series.



(b) Compared to the time series in **a**, seasonality for this time series (left) is not obvious at a glance. Instead, we must refer to the ACF (right) to reveal this behavior.

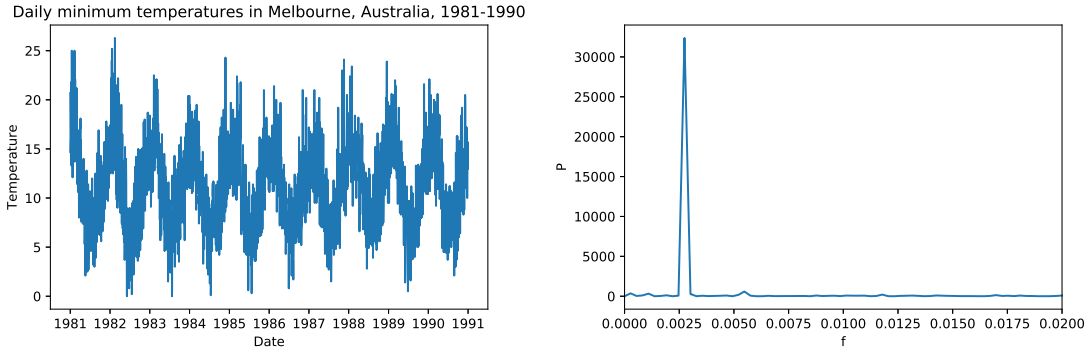
Figure 2: **a** Time Series with obvious seasonality. **b** Time Series with seasonality determined by analysis of the ACF.



(a) ACF with seasonality.

(b) ACF without seasonality.

Figure 3: **a** The ACF of a non-stationary time series displaying daily seasonality. Similar behavior occurs once every 24 time steps where each time step is 1 hour. **b** The ACF of a stationary time series without seasonality which shows exponential decay.



(a) Time Series with Seasonality

(b) Periodogram of the time series.

Figure 4: a A time series with seasonality **b** The periodogram of the time series with a massive spike in P at frequency 0.0025. Periodicity and frequency have an inverse relationship, so the periodicity here is 365 (yearly seasonality).

Another common method is to find the maximum in the Fourier transform of the time series (Oppenheim et al., 2001). Given a time series with n distinct values, we can represent it as a sum of sine and cosine waves:

$$x_t = \sum_{j=1}^{n/2} \left[\beta_1 \left(\frac{j}{n} \right) \cos(2\pi\omega_j t) + \beta_2 \left(\frac{j}{n} \right) \sin(2\pi\omega_j t) \right].$$

where $\omega_j = \frac{1}{n}, \frac{2}{n}, ... \frac{\frac{n}{2}}{n}$, are the harmonic frequencies (positive integer) and $\beta_1 \left(\frac{j}{n} \right)$ and $\beta_2 \left(\frac{j}{n} \right)$ are parameters that can be estimated using Fast Fourier Transform.[5]

After β_1 and β_2 are estimated using FFT, a periodogram can be created where the x axis is the frequency j/n and the y axis is P or the sum of the squared coefficients β_1 and β_2 at the frequency j/n. The periodogram graphs a measure of the relative importance of possible frequency values that might explain the oscillation pattern of the observed data. A relatively large value of P at frequency j/n indicates relatively more importance for the frequency j/n in explaining the oscillation in the observed series (see Figures 4a and 4b). We make use of the findfrequency function in the R forecast library (Hyndman & Khandakar, 2008) which returns the period with the maximum spectral amplitude of the signal.

2.2.3 Detecting Level Change Concept Drift

Concept drifts can be difficult to detect especially if one does not know beforehand how many exist. This number need not be known as established by Adams and MacKay (2007). An implementation of this methodology is available (Kulick, 2016) using t-distributions for every new concept, referred to as a *run*. The posterior probability ($P(r_t|x_{1:t})$) of the current run r_t's length at each time step (x_i for $i = 1...t$) can be displayed, using a logarithmic color

5. Note that this equation is working under the assumption we have an even number of distinct values n in the time series; there is a separate equation for the case where it is odd.

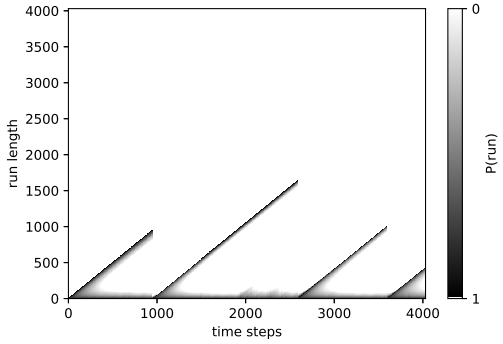


Figure 5: Using the same time series in Figure 1c, this plot shows the posterior probability of the run length at each time step using a logarithmic color scale (Kulick, 2016).

scale (see Figure 5). Although the number of concept drifts need not be known beforehand, Adam's method is memoryless; if a change point has occurred with high probability, the previous distribution's parameters are not used in determining the next distribution's parameters.

2.3 Example Datasets

For every characteristic in Section 2.1, we create a *corpus* of 10 datasets each containing this characteristic (see Table 4). Some datasets come from the Numenta Anomaly Benchmark repository (Numenta, 2018b) which consists of 58 pre-annotated datasets across a wide variety of domains and scripts for evaluating online anomaly detection algorithms. No multivariate datasets are provided in Numenta's repository. Meticulous annotation instructions for Numenta's datasets are available (Numenta, 2017). The Numenta Anomaly Benchmark repository also contains code for combining labels from multiple annotators to obtain ground truth. For every such annotated dataset, there is a *probationary period* (first 15% of the dataset) where models are allowed to learn normal patterns of behavior. For this reason, no anomalies are labeled in the probationary period. See Figures 6a, 6b, and 6c which include vertical dividing lines to display the location of the probationary period. In cases where we do not use Numenta datasets, we have computer science undergraduate students from Eastern Washington University tag the datasets for anomalies following the Numenta instructions. There are also several instances where we inject outliers (Liu et al., 2017; Choudhary et al., 2018; Vallis et al., 2014).

We also determine the *type* of anomalies present in each dataset: point or collective. A data point is considered a point anomaly if its value is far outside the entirety of the data set. A subset of data points within a data set is considered a collective anomaly if those values as a collection deviate significantly from the entire data set, but the values of the individual data points are not themselves anomalous. The first point in the subset is marked. Out of a total of 34 datasets, 28 contain anomalies that are point anomalies, and 24 contain collective anomalies. Some datasets exhibit both types of anomalies.

For datasets with missing time steps, we also consider another version of the dataset with missing time steps filled using linear interpolation.

We proceed with briefly describing the datasets making up our corpora. A summary of all datasets in available in Table 1 where the *Numenta* column indicates whether or not the dataset came from the Numenta repository, and P and C under the *Outlier Type* column indicate point and collective, respectively.

2.3.1 ex-2_cpm_results, ex-3_cpm_results

ex-2_cpm_results and ex-3_cpm_results consist of an online advertisement's cost per thousand impressions (CPM). CPM is a marketing term referring to the cost (as charged by the website publisher to the advertiser) of having 1000 customers view an advertisement. One duplicate record was found on August 24, 2011 for ex-2_cpm_results; we arbitrarily chose one of the two values.

2.3.2 ex-2_cpc_results, ex-3_cpc_results

These datasets are similar to those in Section 2.3.1 except they track the CPC, or cost-per-clicks (as charged by the website publisher to the advertiser), instead of the CPM.

2.3.3 Twitter_FB, Twitter_GOOG, Twitter_AMZN

These Numenta datasets record the number of Twitter mentions of large publicly-traded companies (Facebook, Google, and Amazon) every 5 mintes.

2.3.4 req_count_8c0756

req_count_8c0756 consists of request counts for Amazon Web Services' Elastic Load Balancer which distributes application traffic. There are approximately 2 weeks' worth of data in 5 minute intervals, 8 missing time steps, and 2 annotated outliers.

2.3.5 grok_asg_anomaly

grok_asg_anomaly consists of auto scaling group Grok metrics over the course of approximately half a month in 5 minute intervals. An auto scaling group is a similar collection of Amazon Elastic Compute Cloud instances, and Grok is used to monitor patterns in environments and identify unusual behaviors.

2.3.6 rds_cpu_util_cc0c53, e47b3b

rds_cpu_util_cc0c53 and rds_cpu_util_e47b3b consist of CPU utilization metrics for Amazon Web Services' Relational Database Service (RDS) over the course of 2 weeks in 5 minute time step intervals. cc0c53 has 1 missing time step. cc0c53 and e47b3b both have 2 annotated anomalies,

2.3.7 amb_temp_sys_fail

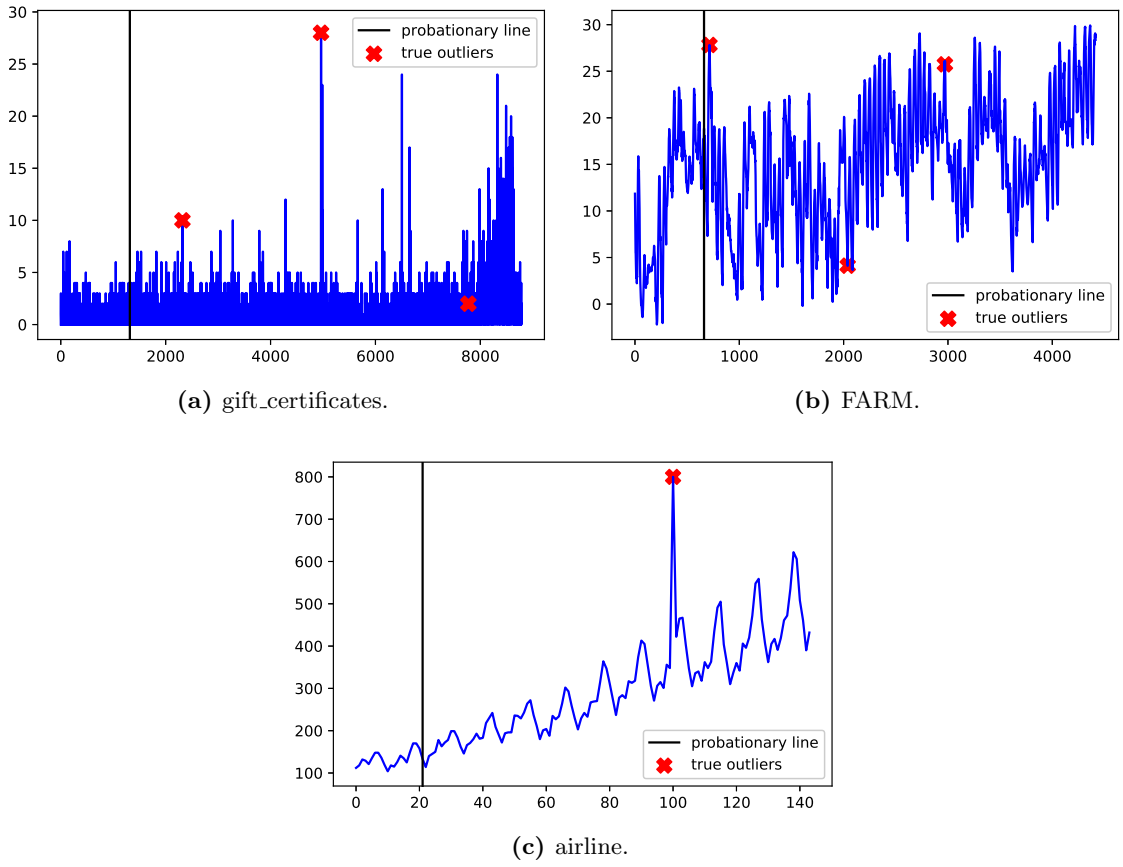
amb_temp_sys_fail consists of hourly ambient temperatures in an office setting over the course of approximately a year. There are 621 missing time steps and 2 anomalies out of 7267 total time steps.

2.3.8 art_daily_flat_middle, art_daily_no_jump

From Numenta (2018b), these are artificially-generated datasets with level change concept drift and collective anomalies (Chandola et al., 2009a), groups of data instances that are anomalous with respect to the entire dataset.

2.3.9 cpu_util_5f5533, ac20cd

These time series track CPU usage data from a server in Amazon's East Coast datacenter. They end with complete system failures resulting from a documented failure of AWS API servers.



(a) gift_certificates.

(b) FARM.

(c) airline.

Figure 6: **a** The hourly counts of a specific *intent* being hit in conversations with an Intelligent Virtual Assistant (IVA) for an airlines company. There are three annotated outliers (red x's). **b** Air temperature measurements in Bowling Green, Kentucky resampled to 30 minute time steps. **c** Monthly totals in thousands of international airline passengers.

2.3.10 gift_certificates

This dataset involves the hourly counts of a specific *intent* being hit in conversations with an Intelligent Virtual Assistant (IVA) for an airlines company. An intent is the interpretation

of the statement or question that allows one to formulate the best response (Dumoulin, 2014). Intents can be thought of as labels in a classification task. As an example, if a user asks if he can bring his dog with him on a plane flight, the user intent would be "travel policy with pets". The dataset in question counts the number of times the intent "Gift Certificates" was hit over the year of 2016. Anomalies were annotated using Numenta's instructions; there were 3 such anomalies. See Figure 6a.

2.3.11 FARM

This dataset consists of air temperature measurements in Bowling Green, Kentucky. The dataset was collected in 5 minute time intervals over a 3 month period and was then re-sampled into 30 minute time intervals grouping by the mean. 3 anomalies were annotated (after resampling) using Numenta's instructions. See Figure 6b.

Dataset	TS	Size	Min	Max	Med	Av	An	T	Mi	Nu	Se	Pe	Tr	TrT	CD
Twitter_FB_f	15833	5m	0	1,258	14	18	2	p	0	Y	T	17	T	d	F
req_count_8c0756_f	4040	5m	1	656	48	62	2	pc	0	Y	T	30	T	d	T
FARM_f	4416	30m	-2	30	15	15	3	pc	0	N	T	50	U	U	F
cpu_util_ac20cd_nf	4032	5m	2	100	35	41	1	pc	5	Y	T	3	T	s	T
amb_temp_sys_fail_f	7888	1H	57	86	72	71	2	p	0	Y	T	24	U	U	T
ibm-stock_nf	1008	1D	306	599	461	463	1	c	452	N	F	1	T	s	T
rds_cpu_util_cc0c53_f	4033	5m	5	25	6	8	2	pc	0	Y	F	1	U	U	T
ex-2_cpc_results_nf	1623	1H	.03	.2	.1	.1	1	c	25	Y	F	1	U	U	T
ex-2_cpm_results_nf	1623	1H	0	1	.3	.3	2	p	25	Y	F	1	U	U	T
cpu_util_ac20cd_f	4037	5m	2	100	35	41	1	pc	0	Y	T	3	T	s	T
rds_cpu_util_cc0c53_nf	4032	5m	5	25	6	8	2	pc	1	Y	F	1	U	U	T
rds_cpu_util_e47b3b_f	4032	5m	13	76	17	19	2	pc	0	Y	F	1	U	U	T
airline_f	144	1MS	104	800	266	283	1	p	0	N	T	12	T	s	T
art_daily_flatmiddle_f	4032	5m	-22	88	-18	19	1	c	0	Y	T	14	T	d	F
artificial_cd_3_f	1000	5m	1	50	25	25	4	pc	0	N	T	2	U	U	T
grok_asg_anomaly_f	4621	5m	0	46	33	28	3	pc	0	Y	T	23	T	s	T
cpu_util_5f5533_f	4032	5m	35	68	43	43	2	pc	0	Y	T	3	U	U	T
ex-3_cpc_results_f	1647	1H	.04	1	.1	0.1	3	p	0	Y	F	1	U	U	T
ex-2_cpm_results_f	1648	1H	0	1	.3	0.3	2	p	0	Y	F	1	U	U	T
ex-3_cpm_results_nf	1538	1H	.3	6	.7	.8	1	p	109	Y	F	1	U	U	F
artificial_cd_1_f	800	5m	1.	20	10	10	3	pc	0	N	T	12	U	U	T
ex-3_cpc_results_nf	1538	1H	.04	1	.1	0.1	3	p	109	Y	F	1	U	U	T
ex-3_cpm_results_f	1647	1H	.3	6	.7	.8	1	p	0	Y	F	1	U	U	F
artificial_cd_3_nf	997	5m	1	50	25	25	4	pc	3	N	T	2	U	U	T
artificial_cd_2_f	1800	5m	1	60	11	15	8	pc	0	N	T	8	U	U	T
req_count_8c0756_nf	4032	5m	1	656	48	62	2	pc	8	Y	T	30	T	d	T
gift_certificates_f	8784	1H	0	28	0	.9	3	pc	0	N	T	24	U	U	T
Twitter_AMZN_f	15831	5m	0	1,673	50	53	4	pc	0	Y	T	21	T	d	T
ex-2_cpc_results_f	1648	1H	.03	.2	.1	.1	1	c	0	Y	F	1	U	U	T
artificial_cd_1_nf	796	5m	1	20	10	10	3	pc	4	N	T	13	U	U	T
art_daily_nojump_f	4032	5m	18	88	21	41	1	c	0	Y	T	13	T	d	F
ibm-stock_f	1460	1D	306	599	461	463	1	c	0	N	F	1	T	s	T
amb_temp_sys_fail_nf	7267	1H	57	86	72	71	2	c	621	Y	T	24	U	U	T
Twitter_GOOG_f	15842	5m	0	465	16	21	4	pc	0	Y	T	11	T	d	F

Table 1: Summary of all datasets. TS is the number of time steps, $Size$ is the time step size, Min is the minimum, Max is the maximum, Med is the median, Av is the average, An is the number of anomalies in the dataset, T (type) is whether or not the outliers the dataset contains are point (p) and/or collective (c), and Mi (miss) is the number of missing time steps in the dataset. For datasets that have missing time steps (nf for no fill), we create another version of the dataset with time steps filled (f for filled) using linear interpolation (e.g. ibm-stock_f is ibm-stock_nf with missing time steps filled). A dataset will have a value of Y in the Nu column if it was collected from the Numenta repository. Otherwise, it is N. Se, Tr, and CD indicate whether or not seasonality, trend, and concept drift are present, respectively. Pe is the number of time steps in a period, and TrT (trend type) indicates whether or not the kind of trend present is stochastic (s) or deterministic (d). If U (Unknown), this means that the KPSS and PP test have not revealed any trend information about the time series.

2.3.12 ibm-stock

From Hyndman (2018a), *ibm-stock* contains the daily stock closing prices for IBM from 1962 to 1965. There is no weekend data, giving us approximately 450 missing data points. There is 1 annotated anomaly.

2.3.13 airline

From Hyndman (2018a), airline contains the monthly totals in thousands of international airline passengers from January 1949 to December 1960, and one outlier is artificially injected (see Figure 6c).

2.3.14 artificial_cd_dataset_1,2,3

As we want 10 datasets for every characteristic, three concept drift datasets are also artificially generated (artificial_cd_dataset_1,2,3) using the random uniform function at varying intervals[6]. Anomalies were chosen based on the starting location of the level change concept drifts.

2.3.15 Summary of All Datasets

For time series displaying seasonality, trend, and/or concept drift, any missing time steps are filled using linear interpolation. For the missing time step characteristic corpus, we either choose datasets that already contain missing time steps or we randomly remove data points from datasets with originally no missing points to generate the corpus. See Table 1 for a summary of all datasets.

2.4 Survey of Anomaly Detection Methods

We now proceed with a survey of all anomaly detection methods considered. We then discuss the computational complexity of these methods and categorize them to determine which methods are more promising given a characteristic.

2.4.1 Windowed Gaussian

Serving as a baseline, the Windowed Gaussian method is a sliding window anomaly detector that determines anomaly scores of data points in the time series by computing their probabilities from the Gaussian distribution over a window of previous data points (Numenta, 2018b). More specifically, the method uses the complement of the Q-function of the normal distribution to obtain an anomaly score between zero and one. The Q-function $Q(x)$ is the probability that a normal random variable will obtain a value larger than x standard deviations. To determine the Q-function, the mean and standard deviation are specified from a sliding window of the time series. The size of this window is *window_size*. This window moves forward every *step_size* many time steps. See Subsection 2.7 for information on parameter selection.

6. See `https://github.com/dn3kmc/jair_anomaly_detection` for details.

2.4.2 SARIMAX

ARMA models are tools for understanding and forecasting future values by regressing the variable on its own past values (**A**uto**R**egressive) and modeling the error term as a linear combination of current and past error terms (**M**oving **A**verage). By differencing between current and past values, the time series can hopefully be made stationary (**I**ntegrated). If seasonality is incorporated, we have a SARIMA (**S**easonal **A**uto**R**egressive **I**ntegrated **M**oving **A**verage) model. If we include e**X**ogeneous variables, we have SARIMAX.

The predictors depend on the parameters $(p, d, q) \times (P, D, Q, s)$ of the SARIMAX model. The number of AR terms are the lags (p). For example, if p is 5, the predictors for p_t are p_{t-1}, p_{t-2},...p_{t-5}. The number of MA terms are the lagged forecast errors (q). If q is 5, the predictors for p_t are e_{t-1}, e_{t-2},...e_{t-5} where every e_i is the difference between the actual value and the moving average at the *ith* step. d is the differencing parameter for removing trend in a time series. If $d = 1$, we take the difference of every observation with that of a direct, previous observation. P, D, Q are *seasonal* versions of the autoregressive component, moving average component, and difference (p, d, q) given the length of the season s (number of time steps in each season).

Much manual analysis can be necessary (see Box-Jenkins method (Hoff, 1983)) in choosing the parameters for a SARIMAX model, but we cannot expect this of a non-expert user. Thus, there are libraries that can automatically determine hyperparameters such as Pyramid (Smith, 2018) in Python and auto.arima in R according to a given information criterion such as the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC). The stepwise algorithm (Hyndman, 2008) is significantly faster than a traditional grid search and less likely to overfit. However, for long time series with seasonality, this can still be a very slow and memory intensive process.

To apply the SARIMAX model for anomaly detection, we use the following procedure:

1. The length of the seasonal cycle is chosen according to the findfrequency function in the R forecast library.
2. Pyramid is applied to the data to select the optimal model order.
3. A maximum likelihood fit is applied to obtain initial parameters.
4. The maximum likelihood parameters are used to initialize a Kalman Filter[7], and its residuals (Kalman Filter predictions) are fed to the Windowed Gaussian detector to obtain normalized anomaly scores.

Exogeneous variables are also included. We encode seasonal variation due to hour-of-day, day-of-week, day-of-month, and month-of-year using binary indicators. A subset of the aforementioned regressors are selected as appropriate for the scale of the time series

2.4.3 Generalized Linear Model (GLiM)

As an alternative to the SARIMAX model, we consider an online structured time series model encoding seasonal variation due to hour-of-day, day-of-week, day-of-month, and

7. Kalman filters (Kalman, 1960) produce estimates of unknown variables observed over time where these estimates are weighted by uncertainties around each point.

month-of-year using binary indicators. A subset of the aforementioned regressors are selected as appropriate for the scale of the time series.

The model is made adaptive using the Recursive Least Squares (RLS) algorithm (Haykin, 2002), and we modify the algorithm to enable the use of a Poisson distribution for the observations which may be more appropriate for some count-valued datasets that are not well-approximated with a Gaussian. The choice of a Gaussian or Poisson output distribution is determined for each dataset based on domain knowledge.

In the Gaussian case, this method differs from the SARIMAX model by the introduction of a tunable exponential weighting factor, λ, that controls how rapidly the influence of past observations decays. We further extend the RLS algorithm (Haykin, 2002) with an additional step-size parameter η, which aids with stability on some datasets.

The residuals from the one-step-ahead prediction are fed to the windowed Gaussian detector to obtain normalized anomaly scores.

2.4.4 Facebook Prophet

Facebook Prophet is an additive regression model that begins with a special time series decomposition method involving a piecewise linear or logistic growth curve trend $g(t)$, a yearly seasonal component modeled using Fourier series or a weekly seasonal component $s(t)$, and a user provided list of holidays $h(t)$ (Taylor & Letham, 2018):

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

ϵ_t is the error term and is assumed to be normally distributed. A key difference between what Prophet does compared to other techniques such as ARIMA is that it formulates this problem as a curve-fitting exercise. The above specification is similar to generalized additive modeling (Hastie & Tibshirani, 1987). Prophet uses Stan to determine parameters using maximum a posteriori (MAP) optimization and is available in R and Python. Prophet can automatically detect changes in trends by selecting change points between concept drifts although a sensitivity parameter (*prior_scale*) must be tuned. We use grid search to determine multiple sensitivity parameters (such parameters exist also for seasonality and trend). Prophet can also innately handle missing time steps.

2.4.5 Seasonal and Trend Decomposition Using LOESS (STL) Residual Thresholding

Seasonal and trend decomposition using LOESS (STL) (Cleveland et al., 1990) is a very commonly used preprocessing step in making a non-stationary time series stationary. This methodology decomposes a time series into three components: seasonality, trend, and remainder.

STL consists of an inner loop that determines these 3 components and an outer loop that increases robustness against anomalies (Hochenbaum et al., 2017). Trend is determined using a moving average filter. After removing trend from the data, the time series is partitioned into cycle subseries. For example, if there is yearly seasonality and the time series uses monthly time steps, there will be 12 cycle subseries where all January time steps form a time series, all February time steps form another time series, etc. These cycle subseries are then smoothed using LOESS (local regression), providing a smooth estimate

of the time series for all time steps; this means that STL can innately deal with missing time steps.

STL, like SARIMAX, is parameter-heavy, but the most important parameters are periodicity ($n_{(p)}$, which we determine using findfrequency) and the smoothing parameter for the seasonal filter ($n_{(s)}$). The cycle subseries becomes smoother the larger $n_{(s)}$ is. If the seasonal pattern is constant over time, $n_{(s)}$ should be set to a larger value. If the seasonal pattern, instead, changes quickly, $n_{(s)}$ should be reduced. We use grid search to determine this parameter as well as smoothing parameters for trend, degrees, and the number of inner and outer loops.

We follow the given anomaly detection methods in this section exactly; if STL is not mentioned, we do not apply it on the entire time series as a preprocessing step. The result of STL can be used for outlier detection; for example, the remainder component can be thresholded to determine outliers (Laptev et al., 2015).

We use the stlplus package in R (Hafen, 2016).[8] stlplus cannot be applied if the periodicity is less than 4.

2.4.6 Twitter Anomaly Detection

Twitter released the open-source R package, AnomalyDetection (Twitter, 2015; Hochenbaum et al., 2017) in 2015 that detects anomalies using modified versions of the extreme studentized deviate test (ESD). The entire time series is first made stationary by applying a modified STL decomposition where the median of the time series is used to represent the trend component. The residuals are then fed to median-based versions of the extreme studentized deviate test to detect anomalies. The traditional ESD test statistic is calculated like so:

$$C_k = \frac{max_k|x_k - \bar{x}|}{s}$$

where k is the upper bound on the number of anomalies, $\bar{x}$ is the mean, s is the standard deviation, and C_k is calculated for the k most extreme values in the dataset.

The mean and standard deviation are very sensitive to anomalous data and that the median is statistically more robust (Hochenbaum et al., 2017). Thus, for the ESD test statistic, instead of the mean, the authors use the median, and, instead of standard deviation, the authors use MAD (median of the absolute deviations from the sample median) to compute the test statistic.

There are two versions of Twitter AnomalyDetection: AnomalyDetection Vec (AD VEC) and AnomalyDetection Ts (AD TS). AD TS makes assumptions on the time step sizes, only allowing 1 minute, 1 hour, and 1 day time step sizes (Twitter, 2015). AD VEC does not make such restrictive assumptions but requires a periodicity parameter be given that is larger than 1, barring any non-seasonal dataset from being used. Although STL is used initially as a preprocessing step, if there are missing time steps, the Twitter AD library (Twitter, 2015) will error out and suggest using interpolation.[9] Because of the time step size restrictions, we use AD VEC.

8. stlplus allows for missing data as opposed to R's *stl* and gives access to more parameters compared to Python's *seasonal_decompose* function in *StatsModels* (Seabold & Perktold, 2010).
9. This is because Twitter AD uses R's *stl* instead of stlplus.

2.4.7 Half-Space Trees

Half-space trees (HS Trees) (Tan et al., 2011) are an online variant of isolation forests which are based on the decision tree algorithm. Outliers are isolated by randomly selecting a feature from the given set of features and then randomly selecting a split value between the max and min values of that feature. This random partitioning of features will produce shorter paths in trees for the anomalous data points; this distinguishes them from the rest of the data. Thus, anomalies can be detected via smaller path lengths in the tree, where path length is the number of edges traversed from the root node.

HS trees deal with streaming data by using two equally sized windows: the reference (r) and latest (l) windows. These windows capture *mass*, the number of data items within a subspace of the data stream. Data that falls in high mass subspaces are normal. Data that falls in low mass subspaces are anomalous.

Every node of the HS tree contains the feature that was used for the split, the split value, the mass profiles of the data stream in the reference and latest windows (r and l), the depth of the current node (k), and its left and right children nodes.

HS Trees are created in the following manner:

1. Create the workspace
    - For every dimension q, normalize its values between 0 and 1
    - Randomly and uniformly select a real number s_q between 0 and 1. Repetitively selecting s_q's creates an *ensemble* of trees.
    - Define q's work range as $(s_q - 2 * max(s_q, 1 - s_q),\ s_q + 2 * max(s_q, 1 - s_q))$
2. Initialize the tree
    - Randomly select a feature
    - Select the mid point of that feature's work range
    - Split the data space to form two half spaces
    - Initialize the mass of nodes to be 0 (for both r and l)
    - Repeat the above steps until the desired max depth is reached

For every data value x in the *reference* window of the time series, x traverses the tree based on the feature and split requirements of each node. For every node it visits, r increases by 1. For every data value x in the *latest* window, x traverses the tree based on the feature and split requirements of each node. For every node it visits, l increases by 1. The anomaly score for x given a single tree is its r value in its terminal node. Its final anomaly score is the sum of these scores obtained from each HS-Tree in the ensemble. A low r value is anomalous. After anomaly scores are obtained, all mass l values are transfered to mass r values, and all mass l values are set to 0.

2.4.8 Matrix Profile

The matrix profile (Yeh et al., 2018) of a time series can be used to identify *discords*, subsequences of the time series with the maximum distance from its nearest neighbors, and

has been reported to be very competitive anomaly detectors (Chandola et al., 2009b). A huge strength of the matrix profile is that it requires just one parameter: the subsequence length.

A *subsequence* of a time series is a contiguous subset of the time series, and an *all-subsequences set of length m* for the time series is an ordered set of all possible subsequences of the time series by sliding window of length m across the time series. The *distance profile* is a vector of Euclidean distances between a given query and each subsequence in an all-subsequences set. Mueen's Algorithm for Similarity Search (MASS) is used to determine this distance profile by exploiting overlap in distance calculations via convolution and provides the worst case time complexity of $O(N \log N)$.

The *profile index* is a vector of indices where element i is the index of the minimum distance in the distance profile of the subsequence starting at index i in the time series. The *matrix profile* is a vector of Euclidean distances where element i is the minimum distance in the distance profile of the subsequence starting at index i in the time series. Thus, the largest values in the matrix profile gives discords as these subsequences are unlike other subsequences in the time series.

There are several algorithms to compute the matrix profile such as STAMP (Scalable Time series Anytime Matrix Profile) and STOMP (Scalable Time series Ordered search Matrix Profile), and both utilize utilize MASS to obtain the distance profile. STOMP is faster than STAMP but suffers if motifs are near the end of the time series and random data is near the beginning (Yeh et al., 2018). We choose STAMP to determine the matrix profile (Cancino & Benschoten, 2020) and feed it to the windowed Gaussian detector to obtain normalized anomaly scores.

2.4.9 Variational Auto-Encoders (Donut)

Donut (Xu et al., 2018) is an unsupervised anomaly detection method based on variational auto-encoders (VAE). VAE is a deep Bayesian neural network and consists of an encoder, decoder, and a loss function. The encoder learns an efficient compression of the input data x in a lower dimensional space z. The decoder takes this lower dimension representation z and attempts to reconstruct the data. The loss function determines how much information is lost through this process. As the goal is to infer good values for z given x, the posterior $p(z|x)$ must be calculated. Calculating the posterior is intractable; thus, variational inference is used instead. $p(z|x)$ is approximated by using another distribution $q(z|x)$ which is tractable and similar enough to $p(z|x)$ where this similarity is determined via KL (Kullback-Leibler) divergence:

$$min(KL(q(z|x)||p(z|x)))$$

The above is equivalent to maximizing the ELBO (Evidence Lower BOund) (Bishop, 2006):

$$E_{q(z|x)} log\, p(x|z) - KL(q(z|x)||p(z))$$

where the first term is the reconstruction likelihood and the second term guarantees similarity between the two distributions. Typically, ELBO is optimized using SGVB (Stochastic Gradient Variational Bayes).

Donut is built for sequential data whereas VAE is not. Thus, the authors use a sliding window where each window is an input vector for the VAE. The reconstruction probability $E_{q(z|x)} log\ p(x|z)$ is used for the anomaly score. The last point of every window is used for sake of speed. If the reconstruction probability is high, the decoder has an easier time reconstructing the original signal from the compressed signal, and chances of an anomaly are low.

A point of interest is VAE's ability to deal with missing time steps. Missing data is initially filled as zeroes, and three techniques are used to impute the missing time steps:

1. A modified ELBO equation for training to allow Donut to corrrectly reconstruct the normal points within the input

2. Missing Data Injection for training by randomly setting a ratio of normal points to be 0 as if they were missing

3. MCMC (Markov Chain Monte Carlo) imputation for detection: Assuming input data is x and the lower dimensional space is z, x is divided into observed and missing parts (x_o, x_m) and a z sample is obtained from $q(z|x_o, x_m)$. A reconstruction sample (x'_o, x'_m) is obtained from $p(x_o, x_m|z)$, and (x_o, x_m) is replaced by (x_o, x'_m). Thus, the observed points are fixed and the missing points are replaced. (x_o, x'_m) is used to determine the reconstruction probability.

This modification helps Donut reconstruct missing points using a MCMC-based (Markov Chain Monte Carlo) missing data imputation technique where the missing values are treated as unknown parameters in a Bayesian paradigm. The missing value parameters can be sampled sequentially using MCMC simulation, and posterior distributions of the incomplete data given the observed data can be obtained.

We use the implementation of Donut (Xu et al., 2018) available in Python (Xu, 2018).

2.4.10 Anomalous

We also considered the R library, Anomalous (Hyndman, 2018b; Hyndman et al., 2015) which is a *feature-based* anomaly detection method. Anomalous determines anomalous time series relative to other time series in some given collection by computing a vector of features for every time series. The feature list is extensive and includes the mean, variance, spectral entropy, lag correlation, strength of seasonality and trend, etc. Principle component analysis is then performed to obtain the first two principal components. A two dimensional density based outlier detection algorithm is then used to determine the top k many outliers. Anomalous determines anomalous time series whereas we are looking for anomalies *within* time series data. However, we adapt Anomalous by dividing the entire time series into subseries and return the index of the first time step in an anomalous subseries.

2.4.11 Pattern-Based Anomaly Detection

Pattern-Based Anomaly Detection (PBAD) (Feremans et al., 2019) was designed with mixed type time series in mind. Mixed type time series are time series consisting of continuous

sensor values and discrete event logs. However, it can be used with univariate and multivariate time series without event logs. PBAD relies on mining *itemsets* and *sequential patterns*.

An *itemset* is a pattern in the form of a set where no temporal order between items is required. An itemset occurs in or is covered by a time series window if all items in the itemset occur in that window in any order.

A *sequential pattern* is a pattern consisting of an ordered list of one or more items. A sequential pattern occurs or is covered by the time series window if all items in the sequential pattern occur in that window in the same order (gaps and repeats allowed).

A sequential pattern or itemset is *frequent* if its occurrence is higher than a user defined threshold (called the minimal support)

PBAD computes anomaly scores for every time series window using the following procedure:

1. Preprocessing: Normalize time series between 0 and 1 and segment the time series using fixed-size sliding windows of length l.

2. Mine frequent itemsets and sequential patterns: Data is discretized using binning. Itemsets and sequential patterns are mined using existing libraries such as CHARM, CHARM-MF1, CM-CLASP, and MAXSP. Pattern sets are filtered either on closed or maximal patterns, and Jaccard similarity is used to remove itemsets and sequential patterns that co-occur in a large percentage of windows.

3. Construct a pattern-based embedding of the time series: Compute a distance-weighted similarity score between every time series window and every discovered pattern. Normal time series windows will be clustered together in the embedded space whereas the less frequent anomalous windows will have lower similarity scores and will be more scattered in the embedded space.

4. Construct an anomaly detection classifier to detect anomalous windows of the time series. The authors use isolation forests.

2.4.12 Hierarchical Temporal Memory Networks

Hierarchical Temporal Memory Networks (HTMs) are memory-based systems that rely on storing a large set of patterns and sequences (Hawkins et al., 2010). HTMs are heavily influenced by the properties of the neocortex, the part of the brain involved with sensory perception. They are similar to neural networks but store data in a more restrictive fashion; memory is time-based and has a hierarchical structure.

Input data is semantically encoded into a sparse distributed representation (SDR) that is very noise resistant. Like our brains, only a small number of neurons are active at a time (sparse), and many different neurons need to be activated at the same time to represent something (distributed). Similar inputs will have similar SDRs. Spatial pooling, a common trick in computer vision, is then done on the SDR to create sparse output vectors of a specified size and *fixed* sparsity. Spatial pooling retains *context* of the input data using an algorithm called *temporal memory*. The connections between active columns in the spatially

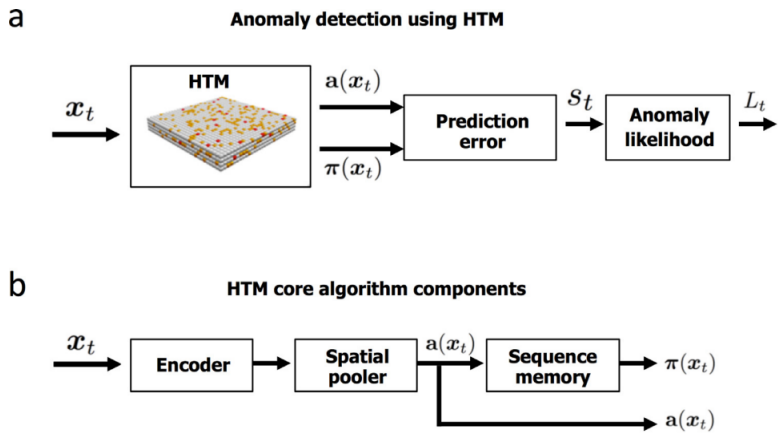


Figure 7: HTM Process (Hawkins et al., 2010).

pooled output array and active cells in the encoded sequence are strengthened whereas for inactive cells, they are weakened.

HTMs operate in real-time but do not output anomaly scores. In Figure 7a, x_t is the input, and the HTM outputs two things: $a(x_t)$ and $\pi(x_t)$. In Figure 7b, which is a zoomed in version of the first step in Figure 7a, the input x_t is fed to the encoder to create the SDR, and spatial pooling is done on it to create $a(x_t)$, the sparse encoding of x_t. The sequence memory component then models temporal patterns in $a(x_t)$ to create $\pi(x_t)$ which is the prediction of the next step. Going back to Figure 7a, we create the prediction error, s_t. The prediction error is not thresholded directly because doing so would introduce many false positives. Instead, the authors model the prediction error as a rolling normal distribution, and the anomaly likelihoods can be calculated for every time step (Ahmad et al., 2017).

We use HTM Studio (Numenta, 2018a) which aims to be user-friendly by setting parameters automatically but requires datasets have a minimum of 400 time steps.

2.5 Computational Complexity

In describing the complexity of the algorithms evaluated, the following variables are considered:

N	Length of time series
D	Size of state and/or external regressors
K	Maximum number of anomalies
W	Window size

Anomalous computes a fixed set of features from each time series, which, in our application, corresponds to a window of W time steps. Several of the features are based on the spectrum or autocorrelation; thus, the factor $O(W \log W)$. After computing the features, the method applies PCA and constructs the α-hull from the largest two components which can be done in $O(N \log N)$ time (Pateiro-Lopez, 2008).

Twitter's AnomalyDetection incorporates the complexity of STL followed by the application of the Hybrid-ESD test. Hybrid-ESD recomputes the median and median absolute

deviation for up to the maximum of K anomalies. We report $O(N \times K)$ complexity for this, assuming $O(N)$ for finding the median and median absolute deviation. The complexities of the methods that consume the entire time series at once are summarized below:

Prophet	$O(N \times W)$[10,11]
STL Residual Threshold	$O(N \times W)$[12]
Twitter AD	$O(N^2 \times K)$
Anomalous	$O(N \log N) + O(W \log W)$
Matrix Profile (STAMP)	$O(N^2 \times log\, N)$
PBAD	$O(\|P\| * \|S\| * o)$ where $o = O(W * m)$

The pattern based embedding for PBAD has time complexity $O(|P| * |S| * o)$ where $o = O(W * m)$, $|S|$ is the number of the windows, $|P|$ is the number of frequent patterns, and m is the length of the sequential pattern. As for the isolation forest step, it has low linear complexity (Chandola et al., 2009a).

We also evaluated four online algorithms, where the criteria for being online are (1) the entire time series does not need to be retained and (2) future values do not influence past predictions.

Windowed Gaussian	$O(N \times W)$
SARIMAX	$O(N \times D^2) + O(N \times W)$
Recursive GLiM	$O(N \times D^2)$
HS Tree	$O(t(h + W))$

Half-Space Trees have a worst time complexity of $O(t(h + W))$ where t is the ensemble size of the trees and h is the max depth level.

The state in a SARIMAX model must be sufficiently big enough to store up to the largest seasonal lag. This makes the quadratic term a significant constraint; hence, it is recommended to instead use indicators or Fourier bases to account for very long term seasonal variation.

We do not report complexity for Hierarchical Temporal Memory networks as we do not have information on the exact implementation employed by HTM Studio.

We also do not report complexity for VAE. Asymptotic running time analysis is not terribly useful for gradient descent for training machine learning models. In practical machine learning, gradient descent is run for some fixed number of epochs (e.g. for Donut, the default choice is 250 epochs which takes time proportional to 250 times the size of the training set multiplied to the time per evaluation of the neural network).

2.6 An Aside on Other Methods

Several methods were experimented with but ultimately not included in our final analysis. We describe them here as well as the reasons for why they were not included.

9. Includes cost of Windowed Gaussian detector.
10. Does not include MAP estimation which depends on solver.
11. Includes cost of Windowed Gaussian detector.

Recurrent Neural Networks We initially considered Recurrent Neural Networks (RNN), as an advantage to multi-layered recurrent neural networks is that they can model complex interactions across inputs without having to make them explicit in the model. However, as our datasets are all univariate, a RNN is overkill for our purposes and very time-consuming to train. For example, we experimented with multi-step prediction RNN (Saurav et al., 2018) which can adapt to concept drifts after an initial period of training and does not require non-anomalous training data. Training and inference time was even longer than SARIMAX (see Figure 21 and more in the Appendix). Due to this inefficiency, we ultimately did not investigate further.

If more complex applications were considered, there is a potential that the powerful nonlinear model could show an advantage over some of the simpler baselines.

Frameworks Other anomaly detection "methods" were considered that were not listed explicitly in this section because we do not wish to compare frameworks but anomaly detection methods themselves. For example, Yahoo's EGADS (Laptev et al., 2015) is a framework for anomaly detection. It includes 3 separate components: forecasting, detecting, and alerting. The user could choose ARIMA for the forecasting component, the prediction error for the detecting component, and k-sigma for the alerting component. Similarly, LinkedIn's Luminol is a Python library for analyzing time series that can detect anomalies using SAX, absolute thresholding, etc. Many of these components are already discussed such as ARIMA and STL. Our goal is to compare the methods and not frameworks. Other popular frameworks include: Etsy's Skyline (Etsy, 2015), Mentat Innovation's datastream.io (Ltd., 2018), Eleme's banshee (Eleme, 2018), Opprentice (Liu et al., 2015), and Lytics Anomalyzer (Lytics, 2015).

2.7 Experiment Setup

For anomaly detection methods that involve some form of forecasting, we perform grid search on the parameters to minimize the forecasting error. This includes methods such as Facebook Prophet and Generalized Linear Models. However, some methods do not perform such forecasting like Twitter AnomalyDetection and Anomalous. One might intuitively think to split the time series into a training and testing set and obtain parameters to minimize the number of erroneous detections and maximize the correct ones. However, as anomalies are by definition rare, it can be difficult to obtain a train and test set containing both anomalous and non-anomalous behavior. Looking back at Table 1, most datasets have very few anomalies (at most 1-3). Thus, we perform grid search on the forecast when possible instead of the actual detection. For anomaly detection methods that do not rely on forecasting, our goal is to choose models and parameters as intelligently as possible based on discovered time series characteristics. Once anomaly scores are obtained, we then determine the best thresholds using ROCs and the Youden threshold Index (see Subsections 3.4 and 3.3).

2.7.1 Windowed Gaussian

We mimic what was done in Numenta (2018b), using a sliding window with varying window sizes directly on the time series values. Step size is half the window size. Grid search was

Sampling Frequency	Indicator Features
Sub-daily	hour-of-day, day-of-week
Daily	day-of-month
Monthly	month-of-year

Table 2: Exogeneous Variables for SARIMAX.

performed on the parameters to minimize the mean squared differences between the time series values and the mean of the window.

2.7.2 SARIMAX

We use Pyramid (Smith, 2018) to determine the best set of parameters (order) according to a given information criterion (Akaike Information Criterion, or 'AIC') and the stepwise algorithm (Hyndman, 2008) which is less likely to overfit compared to traditional grid search. We maintain a cumulative/rolling estimate of the prediction error (Gaussian) distribution and use the Q-function to obtain an anomaly score between 0 and 1 (see Subsection 3.2). As for exogeneous variables (see Table 2), seasonal regressors are selected based on the sampling frequency of each time series.

2.7.3 Facebook Prophet

We fill in the appropriate period parameter (s) as seasonality is a time series characteristic determined beforehand using findfrequency. We use "linear" for the growth parameter as setting it to "logistic" requires knowing a maximum and minimum value the data will reach such as a max population size in an area. For the datasets we consider, a hard maximum is typically unknown. For the remaining parameters (changepoint and seasonality prior scales), we use grid search to minimize the mean squared error between the forecast (predictions) and the actual time series values.

2.7.4 STL

We fill in the appropriate period parameter ($n.p.$) as seasonality is a time series characteristic determined beforehand. The remaining parameters such as the number of outer and inner loops, the window width ($swindow$), the degree of the locally-fitted polynomial in seasonal extraction ($sdegree$), the span (in lags) of the LOESS window for trend extraction ($twindow$), and the degree of locally-fitted polynomial in trend extraction ($tdegree$) are determined by grid search, minimizing the sum of squares of the residuals.

2.7.5 Twitter AnomalyDetection

The direction of anomalies is set to 'pos', the confidence level $\alpha = .05$, and the upper bound percentage on the number of anomalies is set to .2% of the time series (Numenta, 2018b) as in a streaming setting, it is impossible to know the number of anomalies beforehand. The anomaly score *is* the label (0 or 1).

2.7.6 Matrix Profile

The length of the subsequence is set to the periodicity if seasonality is present and periodicity is greater than 5. Otherwise, it is set to 100.

2.7.7 VAE (Donut)

We generally use the same parameters as established by Xu et al. (2018). The number of latent dimensions is $K = 5$, the MCMC iteration count is 10, 1024 is the sampling number of Monte Carlo integration, 256 is the batch size, 250 epochs are used, and the optimizer is Adams. As for the structure of the neural network, there are 2 ReLU layers with 100 units, and .01 is the injection ratio. The learning rate is 10^{-3} and is discounted by .75 every 10 epochs. L2 regularization is used on the hidden layers with a coefficient of 10^{-3}. As for the window size, the authors use 120. However, if the window size is too small, patterns cannot be captured, and if too large, there is overfitting. We instead use $min(120, .25 * N, gws)$ where N is the length of the time series and gws is the gaussian window size used on the reconstruction probabilities to generate normalized anomaly scores.

2.7.8 HS Tree

We use the same parameters as established by Tan et al. (2011). The max depth is set to be 15, window size to 250, tree ensemble size to 25, and the size limit is 10% of the window size.

2.7.9 PBAD

We use the same parameters as established by Feremans et al. (2019). A window size of 12, a window increment of 6, an alphabet size of 100 (Feremans et al. (2019) used 30 to 100), a minimal support of .01, a Jaccard threshold of .9, and 500 isolation trees.

2.7.10 GLiM

The exponential forgetting factor, λ, and the step size parameter, η, are chosen via grid search by minimizing the mean squared error between the forecast (predictions) and the actual time series values. Additionally, the algorithm is modified to learn a Poisson output distribution for count-valued datasets. Seasonal regressors are selected based on the sampling frequency of each time series like with the exogeneous variables of SARIMAX. All models include an intercept term.

2.7.11 Anomalous

The number of anomalies k was set to be .2 %$\times$ number of time steps (Numenta, 2018b) as in a streaming setting, it is impossible to know the number of anomalies beforehand. The anomaly score *is* the label (0 or 1). If the time series has length greater than 1000, it is divided into multiple time series, each of length 100. Otherwise, it set to 25.

2.7.12 HTM

HTM parameters are determined automatically for HTM Studio. We do not allow for time step aggregation. As 400 time steps are needed for HTM Studio, international-airline-passengers could not be run through HTM Studio.

3. Comparison of Anomaly Detection Methods on Time Series with Different Characteristics

We experiment with several different anomaly detection evaluation methods: AUC (Area Under the receiver operating characteristic Curve), window-based F-scores, and Numenta Anomaly Benchmark (NAB) scoring. We discuss how anomaly scores are obtained, how thresholds are determined, define the evaluation methods, and provide results.

3.1 Anomaly Scores

Some anomaly detection methods return an anomaly score between 0 and 1 such as PBAD, HS trees, and HTM. However, others do not. Thus, we make use of the Windowed Gaussian as an output normalization technique. More specifically, the Windowed Gaussian is not just an anomaly detection method but also a technique we use to convert the outputs of many anomaly detectors to a score between 0 and 1. A sliding window computes the probabilities from a Gaussian distribution with a mean and standard deviation determined from this window. For SARIMAX, GLiM, and Prophet the prediction error is fed to the Windowed Gaussian. For the matrix profile, its vector elements are fed to the Windowed Gaussian. For STL, the residuals are used. For VAE, the reconstruction probability output from Donut (which is not normalized) is fed to the Windowed Gaussian. Feeding these outputs to the Windowed Gaussian creates an anomaly score between 0 and 1.

For Twitter and Anomalous which return 1 (if a point is an anomaly) or 0 (if not), we treat the output as the anomaly score.

Every anomaly detection method is run 10 times. If the output of the method needs to be converted between 0 and 1 (meaning we apply the Windowed Gaussian to the output), we also rerun the method's output on various Gaussian Window Sizes.

A threshold is set on this score to determine what is an anomaly. We use the *Youden Index* (Fluss et al., 2005) (see Subsection 3.2) as determined by the ROCs with the maximal AUCs for thresholding.

3.2 AUC and Thresholding

The ROC (receiver operating characteristic) curve is a plot of the false positive rate on the x-axis and true positive rate on the y-axis at different classification thresholds (Fawcett, 2006). The diagonal ($y = x$) divides the ROC space where a random classifier would have points close to the diagonal. Points above the diagonal represent good classification results.

The AUC is the area under the ROC curve and measures the entire two-dimensional area underneath the entire ROC curve from $(0, 0)$ to $(1, 1)$. The AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0 whereas a model whose predictions are 100% correct has an AUC of 1.0. An uninformative classifier yields

Dataset	Anom	Prop	GLiM	HS Tree	HTM	MP	PBAD	SARI	STL	Twitter	VAE	WG
ex-3_cpc_results_nf	1.0000	1.0000	0.9995	0.9999	0.0616	0.8121	N/A	1.0000	N/A	N/A	0.9995	N/A
ex-3_cpm_results_nf	1.0000	1.0000	1.0000	1.0000	0.2207	0.9873	N/A	1.0000	N/A	N/A	1.0000	N/A
FARM_f	1.0000	0.8550	0.7949	0.9998	0.0183	0.9617	0.2960	0.7673	0.8876	1.0000	0.9727	0.9297
artificial_cd_1_nf	1.0000	0.8980	0.9948	0.9997	0.0337	0.9147	N/A	0.9964	0.9950	N/A	0.9998	N/A
Twitter_GOOG_f	1.0000	0.8716	0.9961	0.9996	0.0183	1.0000	0.3727	0.9849	1.0000	1.0000	0.8683	0.9898
ex-2_cpc_results_nf	1.0000	0.9999	1.0000	0.9999	0.0301	0.9732	N/A	1.0000	N/A	N/A	1.0000	N/A
grok_asg_anomaly_f	1.0000	0.9993	0.9893	0.9996	0.0463	0.8718	0.2666	0.9744	0.9853	1.0000	0.7951	0.9997
cpu_util_5f5533_f	1.0000	0.9876	0.9985	0.9998	0.6303	0.9830	0.4415	0.9454	N/A	1	1.0000	0.9990
req_count_8c0756_f	1.0000	0.9258	0.9472	0.9999	0.1351	0.9808	0.2609	0.9715	0.9818	1.0000	0.9125	0.9802
cpu_util_ac20cd_nf	1.0000	1.0000	1.0000	1.0000	0.3000	0.9772	N/A	1.0000	N/A	N/A	1.0000	N/A
req_count_8c0756_nf	1.0000	0.8888	0.9466	0.9999	0.0800	0.7526	N/A	0.9727	0.9819	N/A	0.8344	N/A
gift_certificates_f	1.0000	1.0000	0.8379	0.9996	0.1542	0.7106	0.0060	0.9267	0.9203	1.0000	1.0000	0.8450
Twitter_FB_f	1.0000	1.0000	1.0000	1.0000	0.4499	0.8725	0.2870	1.0000	1.0000	1.0000	1.0000	1.0000
art_daily_nojump_f	1.0000	0.6246	1.0000	0.9980	0.0088	0.9775	0.8079	0.8368	0.5136	1	0.9573	0.7984
ex-2_cpc_results_f	1.0000	0.9998	1.0000	0.9999	0.0301	0.9828	0.0426	1.0000	N/A	N/A	1.0000	0.9987
rds_cpu_util_cc0c53_nf	1.0000	1.0000	1.0000	0.9999	0.1093	0.8095	N/A	1.0000	N/A	N/A	1.0000	N/A
ibm-stock_f	1.0000	0.9996	0.6517	0.9989	0.0088	0.9406	0.2031	0.9836	N/A	N/A	0.8588	0.9140
amb_temp_sys_fail_f	1.0000	0.9999	1.0000	0.9998	0.3000	0.9704	1	0.9927	0.7459	1.0000	1.0000	1.0000
cpu_util_ac20cd_f	1.0000	1.0000	1.0000	1.0000	0.3967	0.9810	0.0030	1.0000	N/A	1	1.0000	1.0000
amb_temp_sys_fail_nf	1.0000	0.9999	0.9998	0.9998	0.3000	1.0000	N/A	0.9974	0.7308	N/A	1.0000	N/A
ex-2_cpm_results_nf	1.0000	0.9828	0.9996	0.9999	0.0562	0.9697	N/A	0.6018	N/A	N/A	1.0000	N/A
artificial_cd_1_f	1.0000	0.8829	0.9948	0.9999	0.0418	0.9082	0.3748	0.8216	0.6162	1.0000	0.9946	0.9976
Twitter_AMZN_f	1.0000	1.0000	1.0000	0.9998	0.0053	0.8889	0.1711	0.9973	0.9999	1.0000	1.0000	1.0000
artificial_cd_2_f	1.0000	0.8626	0.9802	0.9992	0.0268	0.7865	0.2011	0.8568	0.8933	1	0.9970	0.9978
ex-2_cpm_results_f	1.0000	0.9998	0.9996	0.9999	0.0463	0.9509	0.3426	0.9944	N/A	N/A	1.0000	0.9902
artificial_cd_3_nf	1.0000	0.9705	0.9889	0.9999	0.0237	0.9650	N/A	0.7523	N/A	N/A	0.8856	N/A
ex-3_cpc_results_f	1.0000	0.9853	0.9875	0.9999	0.0674	0.7814	0.2402	0.9777	N/A	N/A	0.8426	1.0000
airline_f	1.0000	1.0000	1.0000	1.0000	N/A	0.9973	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
rds_cpu_util_cc0c53_f	1.0000	1.0000	1.0000	1.0000	0.1261	0.9734	0.4431	1.0000	N/A	N/A	1.0000	1.0000
ex-3_cpm_results_f	1.0000	1.0000	1.0000	1.0000	0.3000	0.9934	0.0978	1.0000	N/A	N/A	1.0000	1.0000
art_daily_flatmiddle_f	1.0000	0.9984	1.0000	1.0000	0.1857	1.0000	0.4749	0.5307	0.9997	1	1.0000	0.7347
artificial_cd_3_f	1.0000	0.9597	0.9891	0.9999	0.0463	0.8217	0.2794	0.8925	N/A	1.0000	0.9283	0.9543
rds_cpu_util_e47b3b_f	1.0000	1.0000	1.0000	0.9999	0.2207	0.9683	0.6100	1.0000	N/A	N/A	1.0000	1.0000
ibm-stock_nf	1.0000	0.9822	0.5692	0.9993	0.0103	0.9699	N/A	0.9239	N/A	N/A	0.8584	N/A

Table 3: Youden Index anomaly score thresholds for every dataset, method combination. *Anom* is for Anomalous, *Prop* for Prophet, *MP* for Matrix Profile, *SARI* for SARIMAX, and *WG* is for the Windowed Gaussian. (airline_f, HTM) is a N/A because that dataset does not have the 400 time step minimum required by HTM Studio. PBAD, Twitter and the Windowed Gaussian cannot take datasets with missing time steps. Twitter also requires seasonal time series. STL requires seasonal time series with periodicity at least four.

an AUC of 0.5. As the ROC is plotted at different classification thresholds, the AUC is classification-threshold-invariant.

The precision-recall curve (PR Curve) is an alternative plot of the precision (y-axis) and the recall (x-axis) for different thresholds. The PR curve tends to focus on the minority class in imbalanced problems, whereas the ROC curve covers both classes. As false positives often abound in anomaly detection algorithms, we display ROC curves in this paper and the appendix. However, AUCs of the PR Curve are also available on `https://github.com/dn3kmc/jair_anomaly_detection`.

We use the Youden Index (Fluss et al., 2005) to choose thresholds for anomaly scores generated from every method, dataset combination. The Youden Index threshold maximizes the difference between the true positive rate and false positive rate on the ROC line. The Youden index threshold for every method, dataset combination is shown in Table 3.

3.3 Window-Based F-Scores

To evaluate and compare the anomaly detection methods, in addition to the AUC, we use the standard metrics of precision and recall to compute the F-score (harmonic mean of the precision and recall, $2 \times \frac{precision \times recall}{precision + recall}$). An anomaly is considered to be the "True" class. Accuracy, alone, is not a good measure due to class imbalance (very few anomalies typically exist). We consider precision and recall on *anomaly windows*; points are too fine a granularity. An anomaly window is defined over a continuous range of points and its length can be user-specified. For the purposes of this paper, we use the same anomaly window size as established by Numenta (2018b): 10% of the length of a time series divided by the number of true anomalies.

3.4 NAB Scores

One might consider rewarding an anomaly detection method that detects outliers earlier rather than latter in a window. In addition, users may want to emphasize true positives, false positives, and false negatives differently. This gives rise to an *application profile*, $\{A_{FN}, A_{TP}, A_{FP}\}$, which are weights for false negatives, true positives, and false positives, respectively. Numenta (2018b) creates a methodology to determine NAB anomaly scores based on these application profiles. For every ground truth anomaly, an anomaly window is created with the ground truth anomaly at the center of the window. The length of the window can either be user-chosen or set as like in Subsection 3.3.

For every predicted anomaly, y, its score, $\sigma(y)$, is determined by its position, $pos(y)$, relative to a window, w, that y is in or a window preceding y if y is not in any window. More specifically, $\sigma(y)$ is:

$$(A_{TP} - A_{FP})(\frac{1}{1 + e^{5pos(y)}}) - 1$$

if y is in an anomaly window. If y is not in any window but there is a preceding anomaly window w, use the same equation as above, but determine the position of y using w. If there is no preceding window, $\sigma(y)$ is A_{FP}.

The score of an anomaly detection method given a single time series is:

$$A_{FN} f_{ts} + \sum_{y \in Y_{ts}} \sigma(y)$$

where f_{ts} represents the number of ground truth anomaly windows that were not detected (no predicted anomalies exist in the ground truth anomaly window), and Y_{ts} is the set of detected anomalies.

The score is then normalized by considering the score of a perfect detector (outputs all true positives and no false positives) and a null detector (outputs no anomaly detections). It is important to note that the normalized score can be negative when the false positive rate is high. A normalized score can also be larger than 100 if anomalies are detected very early in the window.

If multiple predictions exist in a window, the *earliest* detection is kept, and the rest are ignored. Multiple ground truth anomalies should not exist in a single window by virtue of the detailed anomaly tagging instructions (Numenta, 2017).

We use the code provided by Numenta (2018b) and the standard profile for NAB scoring, meaning that false positives and false negatives are given equal weights.

3.5 Friedman Average Rank Test and Post-Hoc Nemenyi Tests

To detect statistically significant differences between anomaly detection methods on a characteristic, we make use of the Friedman average rank test (Demšar, 2006), a non-parameteric test similar to ANOVA. The null hypothesis is that all anomaly detection methods have identical effects whereas the alternative is that there is a difference. A table is first created where the rows are the datasets making up a characteristic corpus, and the columns are the anomaly detection methods. The columns are ranked by the evaluation metric. As a small example, given dataset x and three anomaly methods A, B, C, suppose B has the highest NAB score[13], C has the second highest, and A does the worst. The corresponding row in the table would be $3, 1, 2$ assuming the columns are ordered as A, B, C. This creates a matrix where element r_{ij} is the rank of method j on dataset i.

Given N datasets and k anomaly detection methods, the test statistic is Q where

$$Q = \frac{12N}{k(k+1)} \sum_{j=1}^{k} \left(\bar{r}_{\cdot j} - \frac{k+1}{2} \right)^2$$

and

$$\bar{r}_{\cdot j} = \frac{1}{N} \sum_{i=1}^{N} r_{ij}$$

The null hypothesis is rejected if Q is larger than the Friedman critical value which can be checked up in a Friedman's ANOVA by Ranks Critical Value Table. If rejected, we proceed with a Nemenyi post-hoc test: For every two anomaly detection methods, the difference between their average ranks is ω, and if $\omega > q_\alpha * \sqrt{\frac{k(k+1)}{6N}}$, then the performance of the two algorithms is significantly different where q_α is the studentized range statistic divided by $\sqrt{2}$.

3.6 Corpora

To measure every anomaly detection method's performance on different time series characteristics, we create a corpus of 10 datasets for every characteristic (seasonality, trend, concept drift, and missing time steps) using the time series in Table 1. See Table 4 for the datasets comprising each corpus.

There are 34 time series in Table 1 but only 29 time series were used for characteristic corpora as some datasets were used for multiple characteristic corpora. We still report the ROCs and AUCs in the Appendix for the five that were not used in the corpora. Some datasets with missing time steps were filled and used in other corpora but not the non-filled (has missing time steps) version and vice-versa.

13. We repeat this test for windowed F-scores and AUCs.

Seasonality Corpus	Trend Corpus	Concept Drift Corpus	Missing Corpus
1. Twitter_FB_f	1. Twitter_FB_f	1. rds_cpu_util_cc0c53_f	1. cpu_util_ac20cd_nf
2. req_count_8c0756_f	2. req_count_8c0756_f	2. cpu_util_ac20cd_f	2. ibm-stock_nf
3. FARM_f	3. cpu_util_ac20cd_f	3. rds_cpu_util_e47b3b_f	3. ex-2_cpc_results_nf
4. amb_temp_sys_fail_f	4. airline_f	4. artificial_cd_3_f	4. ex-2_cpm_results_nf
5. airline_f	5. art_daily_flatmiddle_f	5. grok_asg_anomaly_f	5. ex-3_cpm_results_nf
6. cpu_util_5f5533_f	6. grok_asg_anomaly_f	6. cpu_util_5f5533_f	6. ex-3_cpc_results_nf
7. artificial_cd_1_f	7. Twitter_AMZN_f	7. artificial_cd_1_f	7. artificial_cd_3_nf
8. artificial_cd_2_f	8. art_daily_nojump_f	8. artificial_cd_2_f	8. req_count_8c0756_nf
9. gift_certificates_f	9. ibm-stock_f	9. gift_certificates_f	9. artificial_cd_1_nf
10. Twitter_GOOG_f	10. Twitter_GOOG_f	10. ibm-stock_f	10. amb_temp_sys_fail_nf

Table 4: Characteristic corpora.

Anomaly Detection Method	Best AUC	Mean AUC	95% CI
Anomalous	0.498760	0.498760	±0.00000
Prophet	0.996031	0.989618	±0.00195
GLiM	0.989581	0.952617	±0.00761
Matrix Profile	0.914971	0.629826	±0.05178
PBAD	0.621310	0.591987	±0.01489
SARIMAX	0.995287	0.992806	±0.00038
HS Tree	0.980278	0.979348	±0.00026
VAE	0.990821	0.904128	±0.02535
Windowed Gaussian	0.988837	0.948772	±0.00859
HTM	0.987100	0.987100	±0.02262

Table 5: AUC Table for rds_cpu_util_cc0c53_f.

3.7 Results

We show results using ROC and AUC scores, windowed F-scores, and NAB scores. Plots of all anomaly predictions for every data set, method combination are available on `https://github.com/dn3kmc/jair_anomaly_detection/blob/master/jair_work_step_four_evaluation/Plot%20Predictions.ipynb`. A deeper discussion of these results follows in Section 4.

3.7.1 AUC Results

Figure 8 displays ROCs of every method from the *best* AUC out of the 10 runs for every method + dataset combination. For example, the top right plot displays the best ROCs for all twelve anomaly detection methods on the dataset FARM_f. The ROC with the best AUC (.94) is VAE (light blue line). ROC plots for remaining datasets are available in the Appendix.

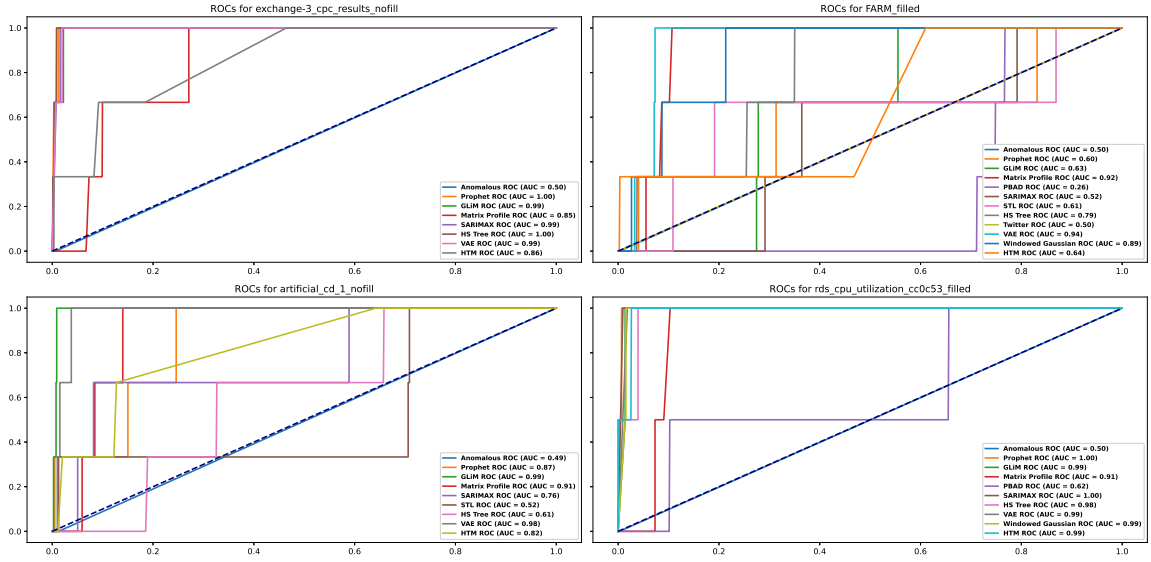


Figure 8: ROCs of anomaly detection methods for four datasets.

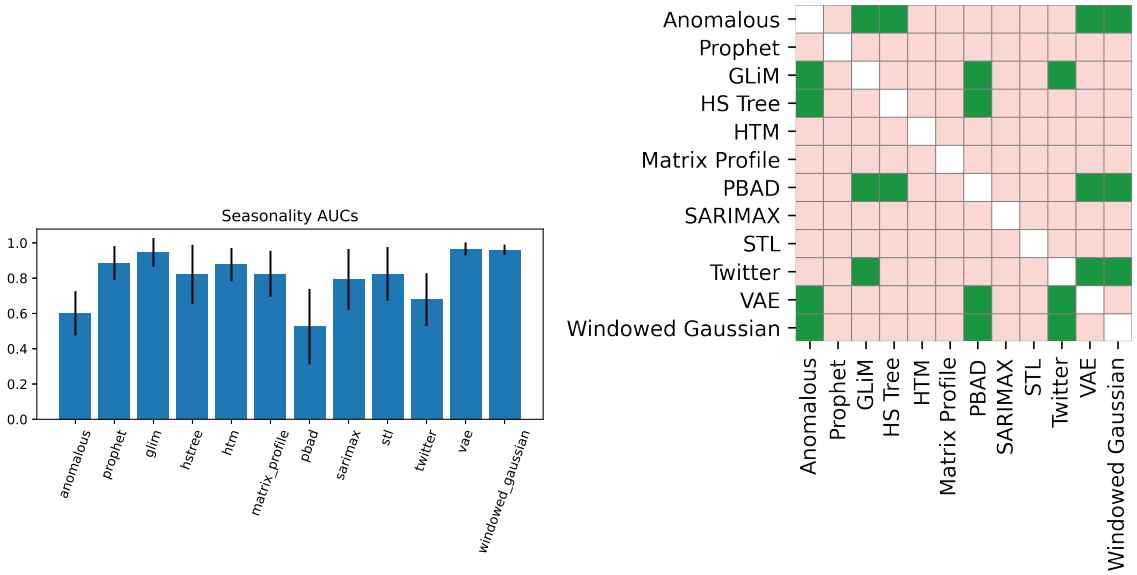
We additionally show the 95% confidence intervals of all AUCs in Table 5 (where in this case, the dataset is rds_cpu_util_cc0c53_f). *Best AUC* is the same AUC displayed in the legend of the ROC plots for that dataset. AUC tables for remaining datasets are available in the Appendix.

We also analyze the AUCs of methods on time series with seasonality, trend, concept drift, and missing time steps. For every method, we obtain its mean AUC from all the datasets comprising a characteristic corpus. For example, to determine Anomalous' abilities on seasonality, we obtain Anomalous' AUCs on every dataset in the seasonality corpus (see Table 4) and determine the mean of those selected AUCs.

We then conduct the Friedman average rank test (see Section 3.5) on AUCs and reject the null hypothesis at the 90% confidence level for all characteristics (seasonality, trend, concept drift, and missing time steps). Rejecting the null hypothesis means that there is a difference between the methods. If the null hypothesis is rejected we follow with the post-hoc Nemenyi test. These AUC results for every anomaly detection method, characteristic combination are shown in Figures 9a to 12b.

3.7.2 Windowed F-Score Results

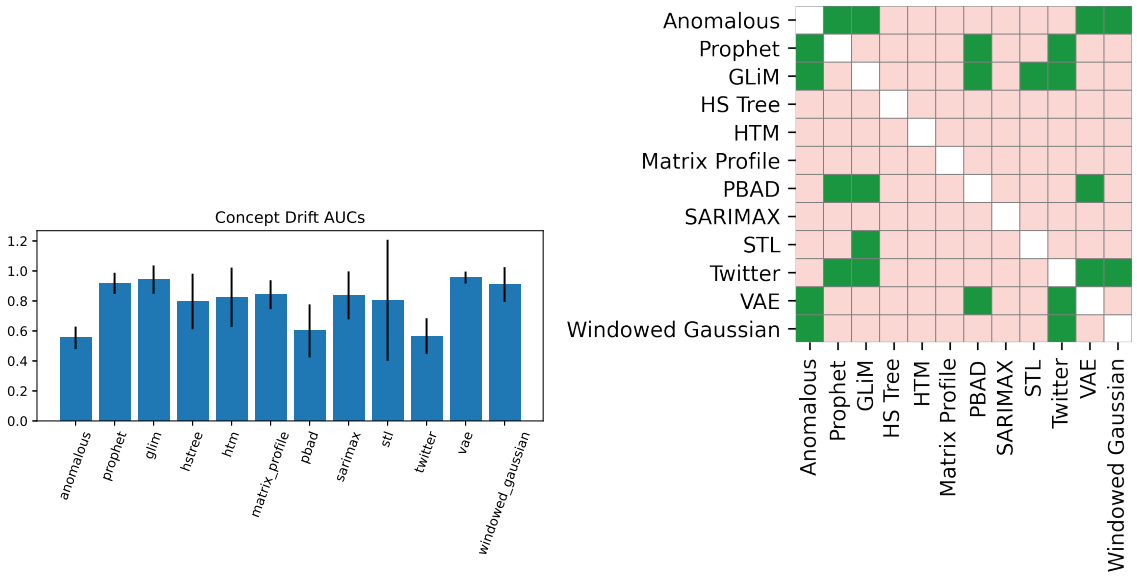
We repeat the above AUC process but use mean Windowed F-score bar charts with 95% confidence intervals for every characteristic corpus. We conduct the Friedman test on the Windowed F-Scores and reject the null hypothesis (i.e. there is a difference) for three characteristics: seasonality, trend, and concept drift. For these three characteristics we follow with the post-hoc Nemenyi test. The Windowed F-score results are shown in Figures 13a to 16.



(a) Mean AUCs for seasonality

(b) AUC Nemenyi results for seasonality

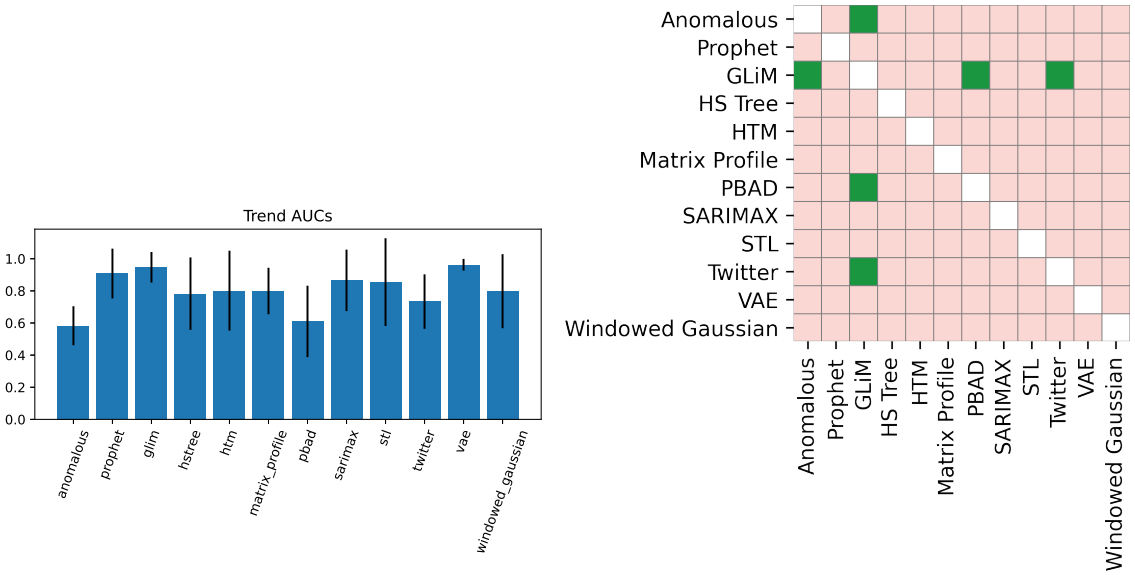
Figure 9: Mean AUCs and 95% confidence intervals of anomaly detection methods on the seasonality corpus (a). Nemenyi results for AUCs where dark green is statistically significant and light red is not significant (b).



(a) Mean AUCs for concept drift

(b) AUC Nemenyi results for concept drift

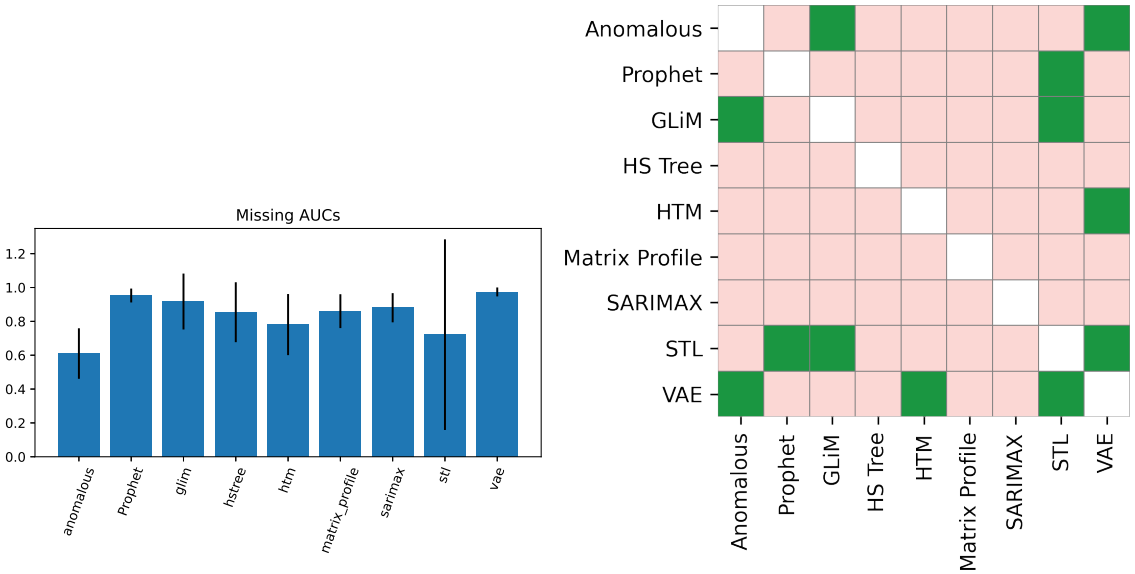
Figure 10: Mean AUCs and 95% confidence intervals of anomaly detection methods on the concept drift corpus (a). Nemenyi results for AUCs where dark green is statistically significant and light red is not significant (b).



(a) Mean AUCs for trend

(b) AUC Nemenyi results for trend

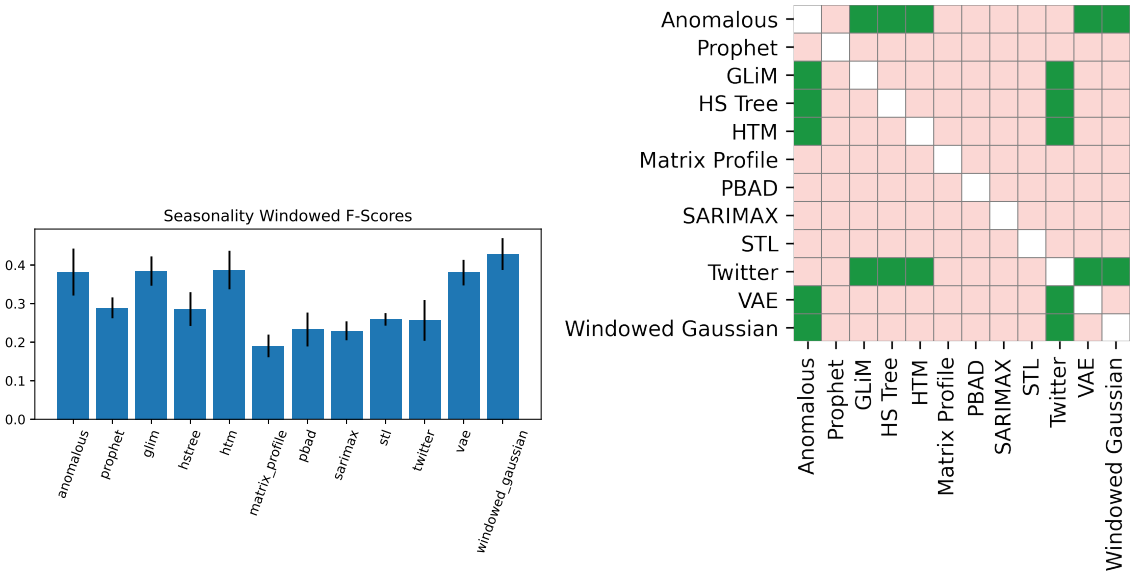
Figure 11: Mean AUCs and 95% confidence intervals of anomaly detection methods on the trend corpus (a). Nemenyi results for AUCs where dark green is statistically significant and light red is not significant (b).



(a) Mean AUCs for missing time steps

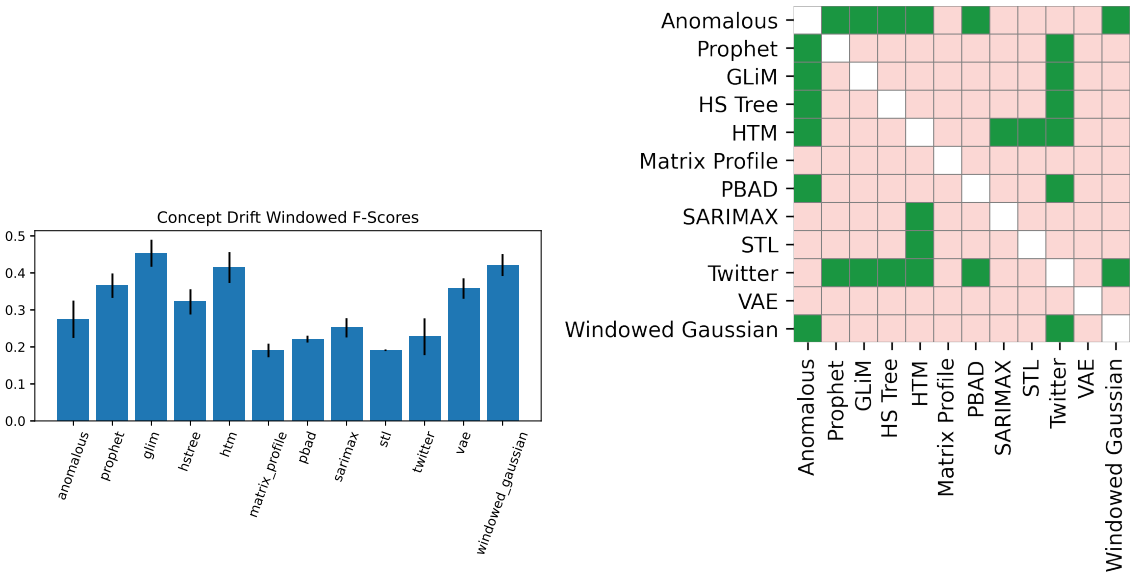
(b) AUC Nemenyi results for missing time steps

Figure 12: Mean AUCs and 95% confidence intervals of anomaly detection methods on the missing time steps corpus (PBAD, Twitter AD, and Windowed Gaussian cannot handle missing time steps) (a). Nemenyi results for AUCs where dark green is statistically significant and light red is not significant (b).



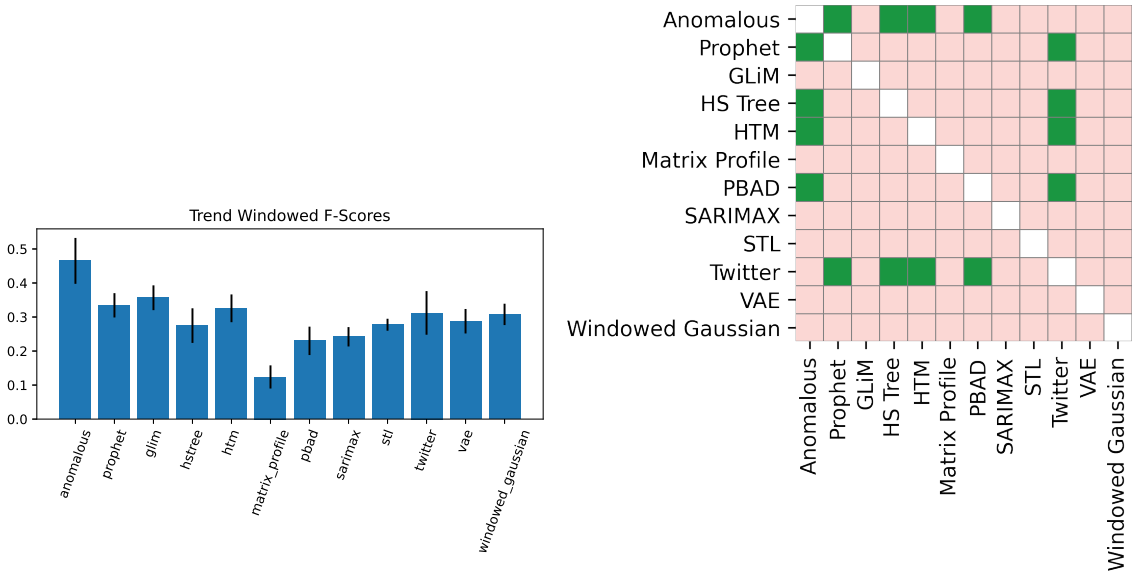
(a) Mean Windowed F-Scores for seasonality **(b)** Windowed F-Score Nemenyi for seasonality

Figure 13: Mean Windowed F-Scores and 95% confidence intervals of anomaly detection methods on the seasonality corpus (a). Nemenyi results for Windowed F-Scores where dark green is statistically significant and light red is not significant (b).



(a) Mean Windowed F-Scores for concept drift **(b)** Windowed F-Score Nemenyi for concept drift

Figure 14: Mean Windowed F-Scores and 95% confidence intervals of anomaly detection methods on the concept drift corpus (a). Nemenyi results for Windowed F-Scores where dark green is statistically significant and light red is not significant (b).



(a) Mean Windowed F-Scores for trend

(b) Windowed F-Score Nemenyi for trend

Figure 15: Mean Windowed F-Scores and 95% confidence intervals of anomaly detection methods on the trend corpus (a). Nemenyi results for Windowed F-Scores where dark green is statistically significant and light red is not significant (b).

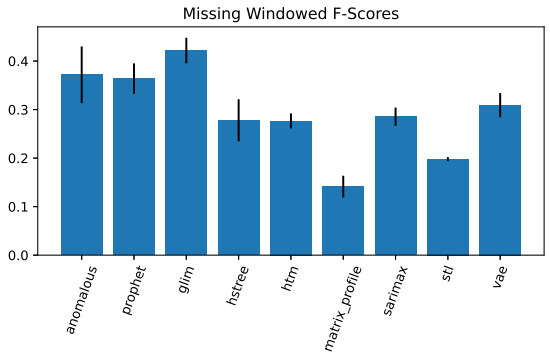
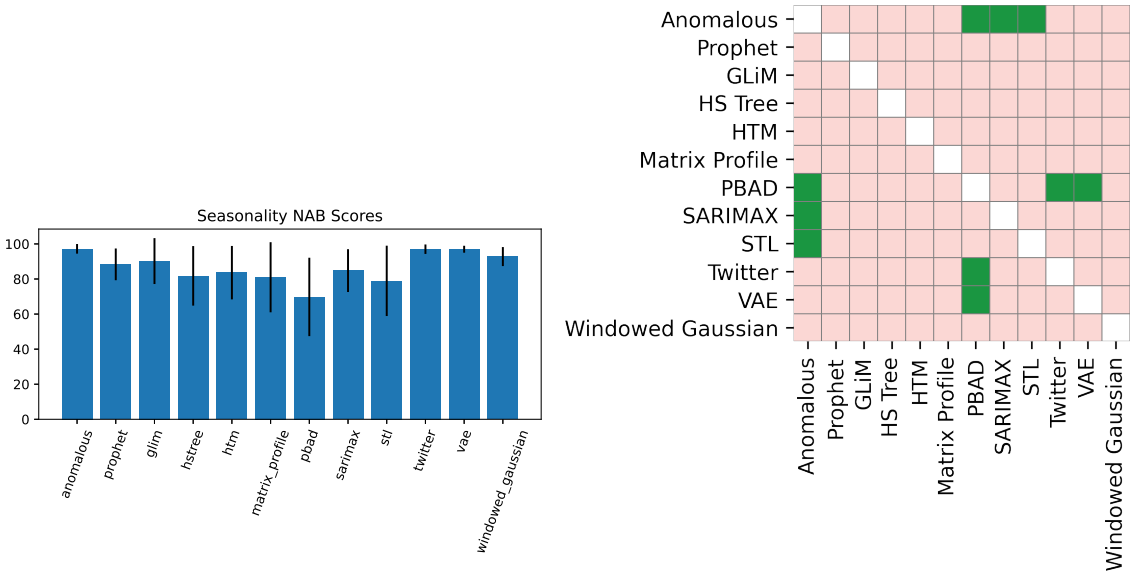
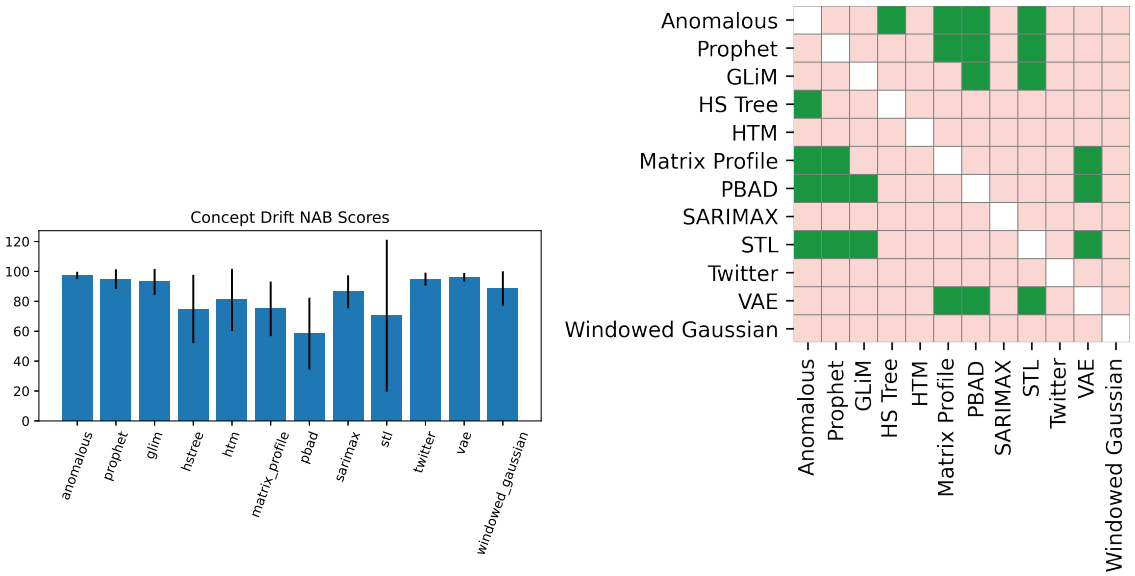


Figure 16: Mean Windowed F-Scores and 95% confidence intervals of anomaly detection methods on the missing time steps corpus.



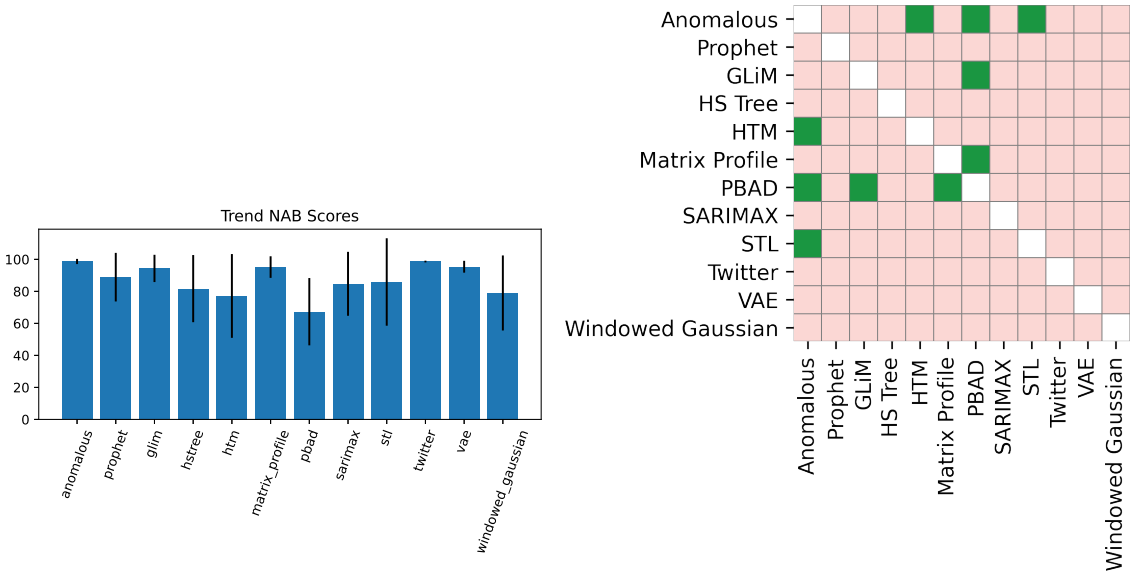
(a) Mean NAB scores for seasonality **(b)** NAB score Nemenyi results for seasonality

Figure 17: Mean NAB scores and 95% confidence intervals of anomaly detection methods on the seasonality corpus (a). Nemenyi results for NAB where dark green is statistically significant and light red is not significant (b).



(a) Mean NAB scores for concept drift **(b)** NAB score Nemenyi results for concept drift

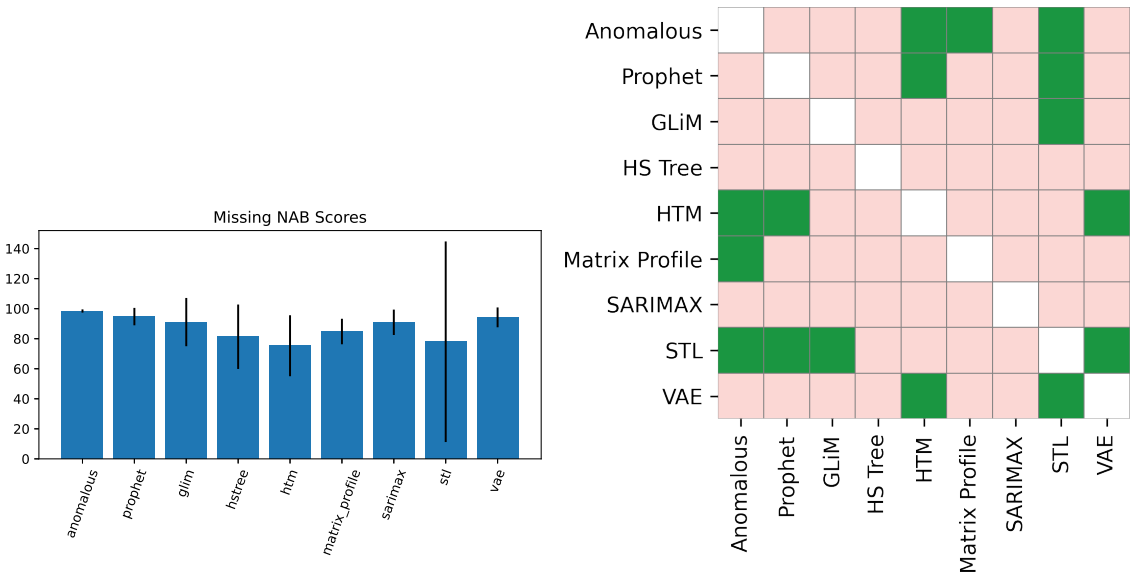
Figure 18: Mean NAB scores and 95% confidence intervals of anomaly detection methods on the concept drift corpus (a). Nemenyi results for NAB where dark green is statistically significant and light red is not significant (b).



(a) Mean NAB scores for trend

(b) NAB score Nemenyi results for trend

Figure 19: Mean NAB scores and 95% confidence intervals of anomaly detection methods on the trend corpus (a). Nemenyi results for NAB where dark green is statistically significant and light red is not significant (b).



(a) Mean NAB scores for missing time steps

(b) NAB score Nemenyi results for missing time steps

Figure 20: Mean NAB scores and 95% confidence intervals of anomaly detection methods on the missing time steps corpus (a). Nemenyi results for NAB where dark green is statistically significant and light red is not significant (b).

3.7.3 NAB Score Results

We repeat the above process but use mean NAB bar charts with 95% confidence intervals for every characteristic corpus. We conduct the Friedman test on the NAB Scores and reject the null hypothesis (i.e. there is a difference) for all characteristics. We follow with the post-hoc Nemenyi tests. Results for the NAB scores are shown in Figures 17a to 20b.

3.7.4 Times

As our time series have differing lengths, we record times needed to run anomaly detection methods on all datasets in characteristic corpora like in Figure 21. Time plots for all other datasets are available in the Appendix. See Subsection 2.5 for run time complexities of the methods discussed.

4. Discussion

Given these results, we proceed with answering the following questions:

1. How does the choice of window size affect the AUC, Windowed F-scores, and NAB scores?
2. What are the differences between the AUC, Windowed F-scores, and NAB scores? When should we use one over the other?
3. Given a time series characteristic, which anomaly detection methods are more promising?

4.1 How Do Gaussian Window Sizes Affect Scores?

This question is only applicable to anomaly detection methods that do not output an anomaly score between 0 and 1. Because no such score is immediately output by these methods, we use the Windowed Gaussian on the outputs of such methods to create a normalized score. Thus, the Windowed Gaussian is not just an anomaly detection method it is also a tool to create normalized anomaly scores between 0 and 1 for methods that do not produce one. This window slides across certain values (e.g. prediction errors, unnormalized anomaly scores, or the time series values themselves) and computes probabilities from a Gaussian distribution with the mean and standard deviation determined from this window. Our question in this subsection is whether or not the *size* of this window has a significant effect on anomaly detection.

It is difficult to establish patterns such as "anomaly detection method x improves in performance when the window size increases" as we have only looked at 5 window sizes: 128, 256, 512, 768, and 1024. However, what we can observe is that window size affects Prophet's outputs more than any other method.

Given an anomaly detection method and time series characteristic, we obtain 5 AUCs[14] (one for each window size: 128, 256, 512, 768, and 1024). We consider every 2-combination of these 5 AUCs and obtain the absolute difference of every combination. We then plot the variance of these differences in Figure 22.

14. We use AUCs as they are classification threshold invariant.

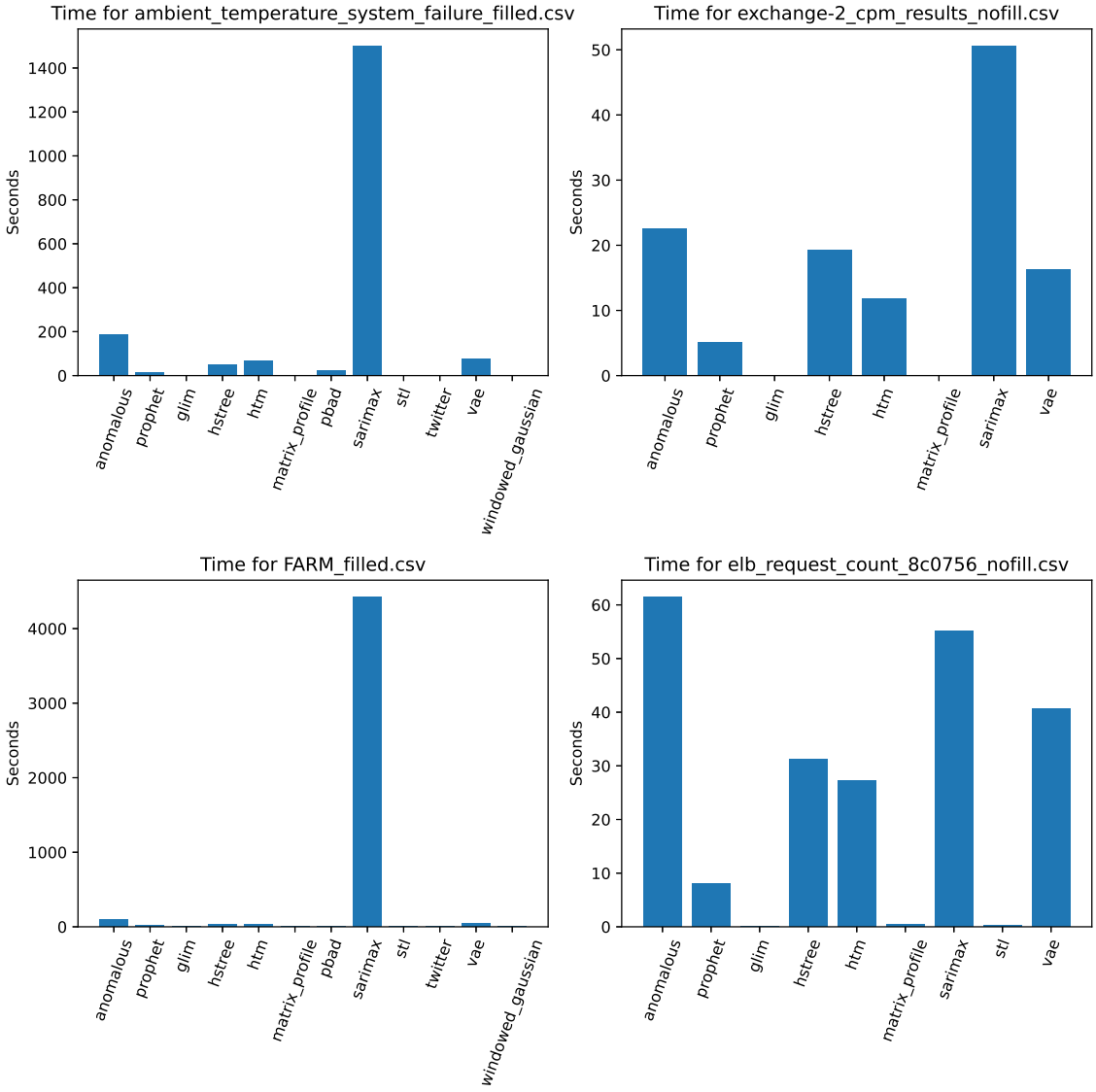


Figure 21: Times of anomaly detection methods for four datasets.

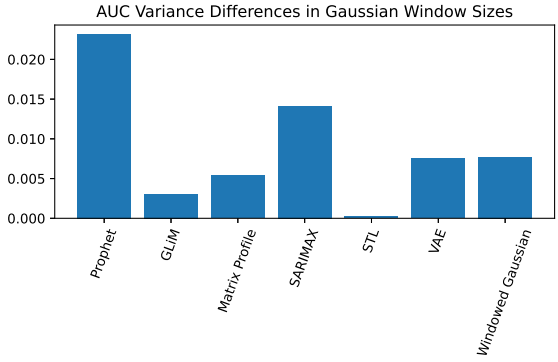
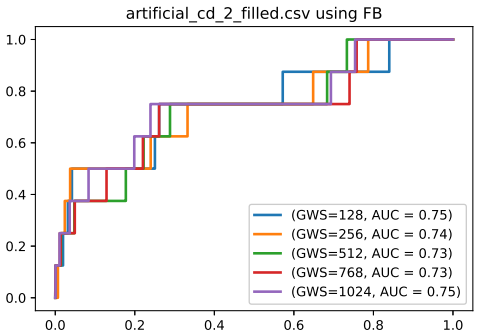


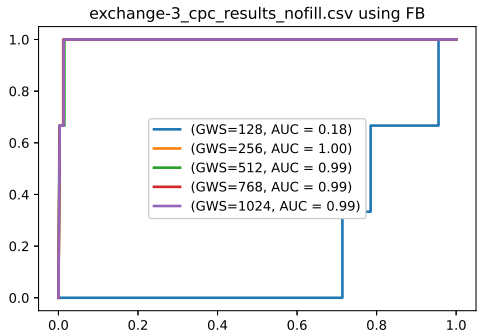
Figure 22: Variance of AUC differences based on Gaussian window size for every anomaly detection method on all datasets.

Note that we do not consider Anomalous, Twitter, PBAD, HS Trees, and HTM in this Figure because these methods already return a normalized anomaly score between 0 and 1 and do not need to be modified in some way to produce one.

According to Figure 22, Prophet is the most affected by the Gaussian window size, having the largest variance of AUC differences. We observe the ROCs of Prophet on multiple datasets and notice that although most AUCs are similar regardless of Gaussian window size (Figure 23a), in some cases the window size of 128 is too small for Prophet (Figure 23b).



(a) ROCs of Prophet on artificial_cd_2_f.



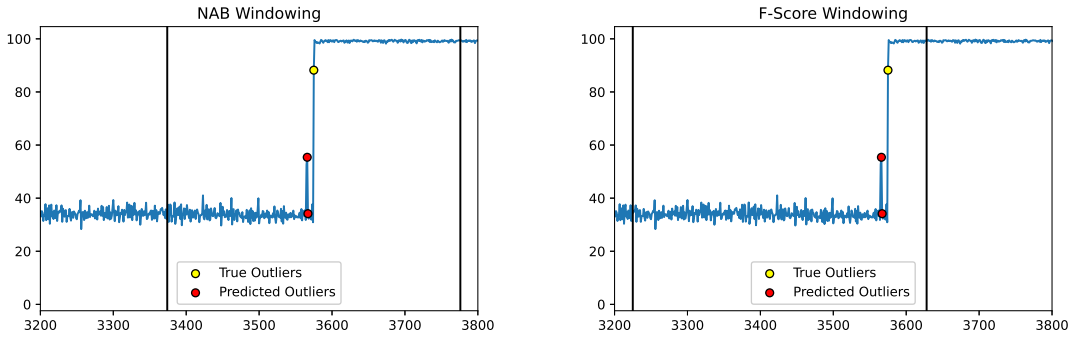
(b) ROCs of Prophet on ex-3_cpc_results.

Figure 23: A small Gaussian window size (GWS) can affect Prophet's AUC scores.

4.2 What Are the Differences Between the AUC, NAB Scores, and F-Scores?

Previous literature states that a direct comparison to NAB may be difficult due to several irregularities (such as determining anomaly window sizes a priori and non-normalized scoring due to no lower bound) (Tatbul et al., 2018). However, what we do know is that NAB scores reward early detection of anomalies, and that this can have a significant impact on how it behaves compared to other evaluation methods.

For NAB scoring, a window is created for every ground truth anomaly centered on the anomaly. Given a predicted anomaly, a position function determines the position of the prediction relative to the ground truth anomaly in its window. If the prediction is within the window but before the actual anomaly, the reward is even greater than having the prediction on the exact same time step as the actual anomaly.



(a) Using NAB score windows.

(b) Using F-Score windows.

Figure 24: HTM anomaly predictions (red) vs ground truth anomalies (yellow) for the time series cpu_util_ac20cd_nf. In a), we use NAB score windows, where windows are created around ground truth anomalies with the ground truth anomaly in the center of the window. In b), we use F-score windows, where the entire time series is divided into windows. Black vertical lines show these window divisions.

Windowed F-scores do not create windows around ground truth anomalies; the entire time series is divided into windows starting from the beginning of the time series. It then rewards a prediction anywhere in the same window as an actual anomaly. Thus, there is the possibility that a predicted anomaly may be rewarded under NAB as it is positioned in the same window as a ground truth anomaly but is punished under the windowed F-score method as the predicted anomaly may be in an entirely different window from the ground truth anomaly.

In Figure 24a, the ground truth anomaly is in the center of this Numenta window, and two outliers predicted by the HTM (red) occur within the Numenta window. Only the first detection point in the window matters; all others are ignored. The NAB score is 99.96 for the entire dataset. Similarly, in Figure 24b, where the entire time series is divided into windows, the predicted anomaly occurs in its F-Score window. In both cases, the prediction falls in the window; however, with NAB, the reward is even greater because the prediction occurs *before* the ground truth anomaly. Regardless of where the prediction falls in the window for the windowed F-score methodology, the reward is the same. The Windowed F-score is .5 for the entire dataset.

The AUC is classification-threshold-invariant which makes it very different from the NAB and Windowed F-score. The AUC provides an aggregate performance measure over all classification thresholds whereas the NAB and Windowed F-scores are calculated for the specified anomaly score threshold. The thresholds we use for NAB and the Windowed

F-score are based off of the ROCs producing the best AUCs for every dataset, anomaly detection method, and (if applicable) Gaussian Window size combination. We use the Youden Index (Fluss et al., 2005) to choose thresholds which maximizes the difference between the true positive rate and false positive rate on the ROC line.

An evaluation method that is classification-threshold-invariant is not always desirable. Depending on the task (e.g. email spam detection), it may be more desirable to minimize false positives at the cost of an increase in false negatives. The AUC is not a useful metric for optimizations such as these.

In addition, for imbalanced data, the ROC is not highly sensitive to false positives. The ROC curve maps the false positive rate on the x-axis where the false positive rate is the number of false positives divided by the total number of real negatives. Anomalies are, by definition, rare. Thus, the number of real negatives is typically very high, creating a very large denominator. Compare to precision: $\frac{tp}{(tp+fp)}$ where tp is the number of true positives, and fp is the number of false positives. Precision is more sensitive to false positives. Thus, the precision-recall curve (which is not as affected by data imbalance) is an alternative to the AUC and ROC.

In summary, what evaluation metric to use is entirely based on the needs of the user's application. It is worth reading Tatbul et al. (2018) and Singh and Olinsky (2017) for additional strengths and weaknesses of scoring methodologies. There is a definite need for more evaluation metrics, and research is progressing in this field (Tatbul et al., 2018). However, we found that the methods established by Tatbul et al. (2018) were not applicable as ground truth anomalies are assumed to be in the form of ranges. This is problematic as ground truths in anomaly datasets are typically given in the form of points.

4.3 Running Times

In addition to a discussion of anomaly detection method complexities in Subsection 2.5, we provide running times of anomaly detection methods on every time series in the characteristic corpora like in Figure 21 in the Appendix. Note that the time series are often of different length (see Table 1 for the number of time steps).

For most of these time series, SARIMAX and Anomalous have the longest running times, especially for longer time series. The running time for SARIMAX is heavily impacted on the periodicity. For example, the FARM dataset has a periodicity of 50, and takes over 4000 seconds ($>$ 1 hour) to run. Given that FARM has 30 minute time steps, only periodic updates of parameters would be possible if SARIMAX was to be applied online. As mentioned in Subsection 2.5, the quadratic complexity term for SARIMAX is a significant constraint for large seasonal lags. Alternative methods should be considered such as resampling the time series to reduce periodicity, Fourier bases, or even different anomaly detection methods altogether. Anomalous also has longer running times. Eighteen features are detected per subseries, and the index of the first time step in an anomalous subseries is returned. If the subseries are long, there are fewer subseries to determine features from; however, predictions may be far off from ground truth anomalies if ground truth anomalies occur near the end of subseries.

A difficulty encountered was the lack of publicly available, annotated, real-world benchmark datasets with high sampling rates. The highest sampling rate we have investigated

involves 5 minute time steps. However, many real-world applications involve even higher sampling rates (e.g. hundreds of measurements per second). Thus, methods such as Anomalous, Prophet, HS Tree, HTM, VAE, SARIMAX, and PBAD would either not be an option or be limited to only periodic updates.

4.4 Which Anomaly Detection Method Should I Use When?

We now discuss the results of our experiments and provide recommendations on which methods appear to perform better based on the characteristics of the time series to be analyzed. These recommendations can be used by practitioners to select the optimal anomaly detection method for their own applications.

To answer this question, we consult Figures 9a through 20b. An anomaly detection method X might perform better than Y according to some metric in the bar charts, but we must also check if this difference is significant by using the Nemenyi result matrix. For example, in the bar chart in Figure 9a, GLiM has a higher seasonality AUC than STL, but in Figure 9b, the (GLiM, STL) element is light red, indicating that this difference is not statistically significant. However, in Figure 9a, the method GLIM has a higher seasonality AUC than Twitter, and in Figure 9b, the element (GLiM, Twitter) is dark green, meaning that GLiM performs better than Twitter on seasonality, and this is a statistically significant difference.

Note that if method X outperforms method Y with statistical significance, and Y outperforms method Z with statistical significance, this does not guarantee that X outperforms method Z. In other words, the transitive property does not hold for Figures 9a through 20b. This is because each corpus is made up of 10 datasets and the methods can therefore outperform each other on different combinations of datasets. For example, X outperforms Y on datasets $\{a, b, c\}$ and Y outperforms Z on datasets $\{d, e, f\}$ but X did not outperform Z on datasets $\{d, e, f\}$ with statistical significance, so it is *not* the case that $X > Y \wedge Y > Z \implies X > Z$. A concrete example of this is in Figures 13a and 13b where Windowed Gaussian significantly outperforms Anomalous, which in turn significantly outperforms HS Tree, but Windowed Gaussian does not significantly outperform HS Tree.

In addition, not every method can be applied to all datasets within a characteristic corpora. For example, stlplus cannot be applied to time series with periodicity less than 4 and therefore STL performance is not always available for comparison.

4.4.1 Seasonality

The performance of the anomaly detection methods on the seasonality corpus as determined by the AUC is shown in Figure 9a. The Friedman test conducted on the AUCs confirmed a statistically significant difference between the methods on the seasonality corpus. A post-hoc Nenenyi test is conducted in Figure 9b showing that Anomalous is outperformed by the Windowed Gaussian, VAE, HS Tree, and GLiM. GLiM outperforms Twitter and PBAD. HS Tree, Windowed Gaussian, and VAE all outperform PBAD. The Windowed Gaussian and VAE outperform Twitter.

As for Windowed F-Scores on the seasonality corpus (see Figures 13a and 13b), there is a statistically significant difference between the methods also. A post-hoc Nemenyi test

shows that Anomalous is outperformed by the Windowed Guassian, HTM, and GLiM. Anomalous, however, outperforms VAE and HS Tree. Windowed Gaussian, VAE, GLiM, HS Tree, and HTM all outperform Twitter

As for NAB scores on the seasonality corpus (see Figures 17a and 17b), there is a statistically significant difference between the methods as well. A post-hoc Nemenyi test shows that Anomalous outperforms STL, SARIMAX, and PBAD. PBAD is outperformed by VAE and Twitter.

As can be seen by these bar charts and Nemenyi tests, the performance of methods can change drastically based on the evaluation method. For example, Anomalous is ranked in 11th place under the AUC but 1st place under NAB. With Anomalous, a subseries that is considered anomalous will predict the first point in the subseries as anomalous. So the anomaly typically occurs latter in the subseries. Thus, the prediction is made early, and NAB rewards early detection of anomalies.

Considering the AUC and windowed F-score performance, a simple methodology such as the Windowed Gaussian or GLiM can perform very well; they are both ranked in the top three methods for the AUC and windowed F-score evaluation methods (Figures 9a and 13a). Under the windowed F-score, both outperform VAE, but under the AUC, they are outperformed by VAE. The VAE is a complex model with many parameters, but the simple sliding Gaussian detector or generalized linear model can compete. This confirms recent analysis (Makridakis et al., 2018) where it is shown that machine learning and deep learning often struggle to outperform classical statistical time series forecasting approaches.

Of interest is which methods cannot handle *non*-seasonal time series or time series with small periodicities. Twitter AD VEC cannot handle time series with periodicity = 1 whereas STL Residual Thresholding requires periodicity to be 4 or higher due to usage of R stlplus.

4.4.2 Trend

For AUCs on the trend corpus (see Figures 11a and 11b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that GLiM outperforms Anomalous, Twitter, and PBAD.

As for Windowed F-Scores on the trend corpus (see Figures 15a and 15b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous outperforms PBAD, HTM, HS Tree, and Prophet. Prophet and HTM outperform Twitter. Twitter outperforms HS Tree and PBAD.

As for NAB scores on the trend corpus (see Figures 19a and 19b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous outperforms STL, PBAD, and the HTM. GLiM and the Matrix Profile both outperform PBAD.

Although the reader should verify with their own data, if AUC is the preferred evaluation metric we recommend starting with GLiM and VAE when processing time series containing trend. If using the evaluation metric of NAB scores or Windowed F-scores Anomalous is a good starting point.

Based on Figures 11a, 15a, and 19a, trend and seasonality have similarities; for NAB, the top three ranked methods are still Anomalous, Twitter, and VAE. For the AUCs,

VAE and GLiM are still in the top three; however Prophet jumps in the top three for trend. Prophet is a decomposition-based method with a component for trend which might explain its performance on this characteristic. Anomalous also becomes a much more appealing option as it is ranked not just first for NAB but also windowed F-scores. With NAB, it is expected given Anomalous makes "early" predictions which NAB rewards. Anomalous uses several trend or trend-like features (such as strength of trend, linearity, peaks, and troughs) that might explain its performance on this characteristic (Hyndman et al., 2015).

4.4.3 Concept Drift

For AUCs on the concept drift corpus (see Figures 10a and 10b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous is outperformed by the Windowed Gaussian, VAE, GLiM, and Prophet. Prophet outperforms Twitter and PBAD. GLiM outperforms Twitter, STL, and PBAD. VAE outperforms PBAD. Both the Windowed Gaussian and VAE outperform Twitter.

As for Windowed F-Scores on the concept drift corpus (see Figures 14a and 14b), there is also a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous outperforms PBAD but is outperformed by the Windowed Gaussian, HTM, HS Tree, GLiM, and Prophet. Twitter outperforms PBAD but Prophet, GLiM, HS Tree, Windowed Gaussian, and HTM all outperform Twitter. HTM also performs better than STL and SARIMAX.

As for NAB scores on the concept drift corpus (see Figures 18a and 18b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous outperforms STL, PBAD, matrix profile, and HS Tree. Prophet outperforms STL, PBAD, and matrix profile. GLiM outperforms STL and PBAD. VAE outperforms matrix profile, PBAD, and STL.

Promising initial methods for concept drift include Prophet, GLiM, and VAE. However, this greatly depends on the evaluation metric used. With NAB scores, Anomalous is a top choice outperforming STL, PBAD, matrix profile, and HS Tree. However, with AUCs, Anomalous is outperformed by the Windowed Gaussian, VAE, GLiM, and Prophet.

Compared to all other characteristics, concept drift displayed the most statistically significant pairing instances for post-hoc Nemenyi tests. We suspect that this is because many anomaly detection methods we experimented with explicitly took seasonality and trend into consideration (e.g. Prophet via a prior scale, Anomalous via the level shift feature, GLiM via λ, etc.) but not necessarily level change concept drift.

4.4.4 Missing Time Steps

For AUCs on the missing time step corpus (see Figures 12a and 12b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows that Anomalous is outperformed by VAE and GLiM, Prophet and GLiM outperform STL, and VAE outperforms HTM and STL.

As for Windowed F-Scores on the missing time step corpus, there is *no* statistically significant difference between the methods. Thus, no post-hoc Nemenyi test was conducted.

As for NAB scores on the missing time step corpus (see Figures 20a and 20b), there is a statistically significant difference between the methods. A post-hoc Nemenyi test shows

that Anomalous outperforms STL, matrix profile, and HTM. Prophet outperforms STL and HTM. GLiM outperforms STL. VAE outperforms HTM and STL.

VAE Donut was specifically constructed with missing time steps in mind, and is a promising initial choice for time series with this characteristic. However, the evaluation metric still plays a role, with VAE performing well under AUC and NAB scores but no statistically significant differences with Windowed F-Scores.

The missing time steps characteristic had the fewest statistically significant pairing instances for post-hoc Nemenyi tests. No comparisons could be done against the Windowed Gaussian, Twitter, and PBAD because these anomaly detection methods cannot be applied to time series with missing time steps. In addition, although there are instances where time series in the missing time step corpus have quite a number of missing time steps (e.g. ibm-stock has 452, ex-3_cpm_results has 109) most only have a few missing time steps (< 10). To see a more profound difference may require further experimentation on more datasets with significant irregular sampling.

5. Conclusions and Where Do We Go From Here?

We have analyzed the performance of 12 anomaly detection methods (the windowed Gaussian, SARIMAX, Facebook Prophet, Anomalous, Generalized Linear Models, STL residual thresholding, Twitter AD , Matrix Profile, VAE Donut, HS Trees, PBAD,and HTM) on several time series characteristics (seasonality, trend, level change concept drift, and missing time steps). We create corpora of behaviors with 10 datasets for every time series characteristic. We determine which methods tend to perform better or worse on these characteristics by analyzing three evaluation metrics: AUC, windowed F-scores, and NAB.

We observe that:

1. There are statistically significant differences as determined by the Friedman average rank test between anomaly detection methods for all characteristics under all scoring methodologies (AUC, windowed F-scores, and NAB) with exception of the missing time step characteristic with windowed F-scores. This may be due to fewer methods that can be applied to time series with missing time steps and because of the lack of benchmark datasets with a large variety of irregular sample sizes.

2. The differences between anomaly detection methods based on characteristics and scoring methodologies are summarized in:

   - Seasonality: Figures 9a, 9b, 13a, 13b, 17a, 17b
   - Trend: Figures 11a, 11b, 15a, 15b, 19a, 19b
   - Concept Drift: Figures 10a, 10b, 14a, 14b, 19a, 18b
   - Missing Time Steps: Figures 12a, 12b, 20a, 20b

3. NAB's reward for early detection may be so great that it overrides the presence of many false positives and can make it difficult to compare different methods. For example, with Anomalous, the predictions are made early, and NAB rewards early detection of anomalies. Anomalous often outperforms other anomaly detection methods under the NAB evaluation method but not with AUC and the windowed F-score method.

4. The AUC is a classification-threshold-invariant evaluation method which is not always desirable especially in applications that may want to minimize certain types of errors like false positives. ROCs are also not highly sensitive to false positives due to the large number of real negatives typically prevalent in anomaly detection tasks.

5. It is unfortuante that many benchmark datasets that are publicly available have few anomalies and that the anomalies present tend to be "obvious" outliers such as point outliers. We try to mitigate this by including different outliers such as collective outliers. However, this is a problem universal to most benchmark datasets. Possible future work that would improve research in the field would be the release of datasets with a larger variety of anomalies that are more "difficult" to detect where the measure of difficulty would have to be carefully defined.

6. There is a need for more anomaly detection methods for time series with missing time steps, especially if the time series is *irregularly sampled*. Many methods either completely ignore the missing time steps or simply replace them with 0s. Although there is work being done on this (Li & Marlin, 2016; Anava et al., 2015), these methods are not explicitly built for online anomaly detection. For future work, we plan on investigating methods established by Kowalska and Peel (2012) where a combination of extreme value theory and Gaussian Processes can be used to deal with irregularly sampled time series.

7. The variance of AUC differences between different Gaussian window sizes is not large. However, out of all methods, Prophet is the most affected by small Gaussian window sizes.

8. There are anomaly detection method library idiosyncrasies. Although STL can innately handle missing time steps, different implementations may or may not be able to. For example, R's *stl* cannot handle missing time steps but STLPLUS can. However, STLPLUS requires periodicity to be at least 4. Twitter AD TS only allows for 1 minute, 1 hour, and 1 day time step sizes and will automatically determine a seasonality parameter versus AD VEC which does not have such time step size restrictions but the user must input the periodicity.

There are many avenues for future work. In addition to considering more datasets for each behavior, we could look at more behaviors themselves such as different types of concept drift, irregular sampling, or multiple seasonalities. Unfortunately, many anomaly detection methods can only take into account a single periodicity for input although time series can display multiple. Inclusion of contextual variables may also change initial perceptions of what is anomalous. There may be more innovative ways to generate anomaly scores given these methods instead of using a sliding window and Q-functions.

Instead of performing an extensive literature review and trying every anomaly detection method in a rapidly expanding library (Gupta et al., 2014; Wu, 2016), we observe characteristics present in the data and narrow the choice down to a smaller class of promising anomaly detection methods.

Time series are being created at an unprecedented scale (Keogh, 2006) and are used in a wide variety of domains. This huge increase in available data makes it difficult to detect

anomalies, especially as the number of anomaly detection methods increases every year. In our view, although it is important to generate new anomaly detection methods, this is daunting for those who want to choose from the assortment of existing methods, especially as a one-size-fits-all method is a myth. It is our hope that this survey and experimental comparison will serve those individuals.

References

Adams, R. P., & MacKay, D. J. (2007). Bayesian online changepoint detection..

Ahmad, S., et al. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, *262*, 134–147.

Anava, O., et al. (2015). Online time series prediction with missing data. In *International Conference on Machine Learning*, pp. 2191–2199.

Banerjee, A., et al. (2008). Anomaly detection: A tutorial. In *Tutorial SIAM Conf. on Data Mining*.

Barbosa, S. M. (2011). Testing for deterministic trends in global sea surface temperature. *Journal of climate*, *24*(10), 2516–2522.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Campos, G. O., et al. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, *30*(4), 891–927.

Cancino, F., & Benschoten, A. V. (2020). matrixprofile-ts. `https://github.com/target/matrixprofile-ts`.

Chandola, V., et al. (2009a). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 15.

Chandola, V., et al. (2009b). Detecting anomalies in a time series database. *Computer Science and Engineering*.

Choudhary, S., et al. (2018). Sparse decomposition for time series forecasting and anomaly detection. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pp. 522–530. SIAM.

Cleveland, R. B., et al. (1990). Stl: A seasonal-trend decomposition. *Journal of Official Statistics*, *6*(1), 3–73.

Cook, A., et al. (2019). Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, *7*(Jan), 1–30.

Dumoulin, J. (2014). Using multiple classifiers to improve intent recognition in human chats. In *Proceedings of the 25th Modern Artificial Intelligence and Cognitive Science Conference 2014*, pp. 10–21.

Eleme (2018). Banshee. `https://github.com/eleme/banshee`.

Emmott, A., et al. (2015). A meta-analysis of the anomaly detection problem..

Etsy (2015). Skyline. `https://github.com/etsy/skyline`.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Feremans, L., et al. (2019). Pattern-based anomaly detection in mixed-type time series. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 240–256. Springer.

Fluss, R., et al. (2005). Estimation of the youden index and its associated cutoff point. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, *47*(4), 458–472.

Gupta, M., et al. (2014). Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *26*(9), 2250–2267.

Hafen, R. (2016). stlplus. `https://github.com/hafen/stlplus`.

Hastie, T., & Tibshirani, R. (1987). Generalized additive models: some applications. *Journal of the American Statistical Association*, *82*(398), 371–386.

Hawkins, J., et al. (2010). Hierarchical temporal memory including htm cortical learning algorithms..

Haykin, S. (2002). *Adaptive Filter Theory (Fourth Edition)*. Pearson Education, Inc.

Hochenbaum, J., et al. (2017). Automatic anomaly detection in the cloud via statistical learning..

Hodge, V., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial intelligence review*, *22*(2), 85–126.

Hoff, J. C. (1983). *A practical guide to Box-Jenkins forecasting*. Lifetime Learning Publications.

Hyndman, R. (2008). Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, *26*(3).

Hyndman, R. (2018a). Time series data library. `https://datamarket.com/data/list/?q=provider:tsdl`.

Hyndman, R. J. (2018b). Anomalous. `https://github.com/robjhyndman/anomalous`.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J., et al. (2015). Large-scale unusual time series detection. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, pp. 1616–1619. IEEE.

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, *26*(3), 1–22.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, *82*(1), 35–45.

Keogh, E. (2006). A decade of progress in indexing and mining large time series databases. In *Proceedings of the 32nd international conference on Very large data bases*, pp. 1268–1268. VLDB Endowment.

Kowalska, K., & Peel, L. (2012). Maritime anomaly detection using gaussian process active learning. In *2012 15th International Conference on Information Fusion*, pp. 1164–1171. IEEE.

Kulick, J. (2016). Bayesian changepoint detection. `https://github.com/hildensia/bayesian_changepoint_detection`.

Laptev, N., et al. (2015). Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1939–1947. ACM.

Li, S. C.-X., & Marlin, B. M. (2016). A scalable end-to-end gaussian process adapter for irregularly sampled time series classification. In *Advances In Neural Information Processing Systems*, pp. 1804–1812.

Liu, D., et al. (2015). Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 Internet Measurement Conference*, pp. 211–224. ACM.

Liu, S., et al. (2017). Online conditional outlier detection in nonstationary time series. In *Proceedings of the International Florida AI Research Society Conference. Florida AI Research Symposium*, Vol. 2017, p. 86. NIH Public Access.

Ltd., M. I. (2018). datastream.io. `https://github.com/MentatInnovations/datastream.io`.

Lytics (2015). Anomalyzer. `https://github.com/lytics/anomalyzer`.

Makridakis, S., et al. (2018). Statistical and machine learning forecasting methods concerns and ways forward. *PloS one*, *13*(3).

Numenta (2017). Anomaly labeling instructions. `https://drive.google.com/file/d/0B1_XUjaAXeV3YlgwRXdsb3Voa1k/view`.

Numenta (2018a). Htm studio. `https://numenta.com/machine-intelligence-technology/htm-studio/`.

Numenta (2018b). The numenta anomaly benchmark. `https://github.com/numenta/NAB`.

Oppenheim, A. V., et al. (2001). *Discrete-time signal processing. Vol. 2.* Upper Saddle River, NJ: Prentice Hall.

Pateiro-Lopez, B. (2008). *Set estimation under convexity type restrictions.* Ph.D. thesis, Universidade de Santiago de Compostela.

Saurav, S., et al. (2018). Online anomaly detection with concept drift adaptation using recurrent neural networks. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp. 78–87. ACM.

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, Vol. 57, p. 61. SciPy society Austin.

Singh, N., & Olinsky, C. (2017). Demystifying numenta anomaly benchmark. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 1570–1577. IEEE.

Smith, T. G. (2018). Pyramid. `https://github.com/tgsmith61591/pyramid`.

Tan, S. C., et al. (2011). Fast anomaly detection for streaming data. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Tatbul, N., et al. (2018). Precision and recall for time series. In *Advances in Neural Information Processing Systems*, pp. 1922–1932.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, *72*(1), 37–45.

Twitter (2015). Anomalydetection. `https://github.com/twitter/AnomalyDetectionc`.

Vallis, O., et al. (2014). A novel technique for long-term anomaly detection in the cloud.. In *HotCloud*.

Wang, X., et al. (2013). Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, *26*(2), 275–309.

Wu, H.-S. (2016). A survey of research on anomaly detection for time series. In *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pp. 426–431. IEEE.

Xu, H. (2018). Donut. `https://github.com/haowen-xu/donut`.

Xu, H., et al. (2018). Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 187–196. International World Wide Web Conferences Steering Committee.

Yeh, C.-C. M., et al. (2018). Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Mining and Knowledge Discovery*, *32*(1), 83–123.