Improving Univariate Time Series Anomaly Detection Through Automatic Algorithm Selection and Human-in-the-Loop False-Positive Removement

Cynthia Freeman,[1 2] **Ian Beaver,**[1] **Abdullah Mueen**[2]
[1]Verint Intelligent Self-Service, [2]University of New Mexico
cynthia.freeman@verint.com, ian.beaver@verint.com, mueen@cs.unm.edu

Abstract

The existence of a time series anomaly detection method that performs well for all domains is a myth. Given a massive library of available methods, how can one select the best method for their application? An extensive evaluation of every anomaly detection method is not feasible. Many existing anomaly detection systems do not include an avenue for human feedback, essential given the subjective nature of what even is anomalous. We present a technique for improving univariate time series anomaly detection through automatic algorithm selection and human-in-the-loop false-positive removement. These determinations were made by extensively experimenting with over 30 pre-annotated time series from the open-source Numenta Anomaly Benchmark repository. Once the highest performing anomaly detection methods are selected via these characteristics, humans can annotate the predicted outliers which are used to tune anomaly scores via subsequence similarity search and improve the selected methods for their data, increasing evaluation scores and reducing the need for annotation by 70% on predicted anomalies where annotation is used to improve F-scores.

Introduction

Time series are used in almost every field, and one important use of time series is for the detection of *anomalies*, patterns that do not conform to past patterns of behavior in the series. Unfortunately, anomaly detection in time series is a notoriously difficult problem for a multitude of reasons: **(1)** What is defined as anomalous may differ based on application. The existence of a one-size-fits-all anomaly detection method that works well for all domains is a myth (Laptev, Amizadeh, and Flint 2015; Vallis, Hochenbaum, and Kejariwal 2014). **(2)** Anomaly detection often must be done on real-world streaming applications. **(3)** It is unrealistic to assume that anomaly detection systems will have access to thousands of tagged time series. **(4)** 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. **(5)** It is important to detect anomalies accurately, but minimizing false positives is also of great necessity to avoid wasted time in checking for problems when there are none. **(6)** There is a massive wealth of anomaly detection methods to choose from (Chandola, Banerjee, and Kumar 2009; Blázquez-García et al. 2020; Hodge and Austin 2004). Because of these difficulties inherent in time series anomaly detection, we make the following contributions:

- A novel, efficient, human-in-the-loop technique for the classification of time series and choice of anomaly detection method based on time series characteristics;
- An empirical study determining these methods by experimenting on over 30 pre-annotated time series from the open-source Numenta anomaly benchmark repository;
- A description of how to incorporate user feedback on predicted outliers by utilizing subsequence similarity search, reducing the need for annotation by over 70% on predicted anomalies while also increasing evaluation scores.

Related Works

In comparing with similar techniques in the literature, we considered the following three requirements for an anomaly detection framework: **1)** Easy to use for non-technical analysts rather than limited to those with intimate understanding of anomaly detection methods, **2)** Computationally efficient, and **3)** Include human feedback due to the subjective nature of anomalies and varied datasets from different domains.

Some popular anomaly detection frameworks include: LinkedIn's Luminol (LinkedIn 2018), Etsy's Skyline (Etsy 2015), Mentat Innovation's datastream.io (Ltd. 2018), and Lytics Anomalyzer (Lytics 2015), but none of these include human-in-the-loop techniques, failing requirement 3.

Yahoo EGADS (Laptev, Amizadeh, and Flint 2015) and Opprentice (Liu et al. 2015) are human-in-the-loop anomaly detection systems with similar aims to ours. However, there are some key differences: **EGADS** gives users two options: the user can choose (1) how to model the normal behavior of the time series such that a significant deviation from this model is considered an outlier or (2) which decomposition-based method to use with thresholding on the noise component. For example, the user could choose ARIMA for the forecasting component, the prediction error for the detecting component, and k-sigma for the alerting component. However, the user must know the existence of such methods and understand how to choose these components, failing require-

ment 1. The alternative is to have the user choose a large variety of methods to input to EGADS, but this would not be computationally efficient, failing requirement 2. **Opprentice** makes use of a classifier to detect anomalies where features are the results of multiple anomaly detectors ran over the same data. These detectors are expected to output a non-negative value that measures the severity of the anomaly and use a threshold to determine if the severity is high enough to be considered an anomaly. The severity levels of the detectors with human outlier labeling comprise the training data. Not only is the analyst expected to know which anomaly detectors to include, but ALL classifiers/anomaly detectors are used by Opprentice. This also does not include the time needed to train Opprentice's classifier.

We focus on the characteristics present in the time series to first discard subpar anomaly detection methods. By filtering subpar methods (the first such paper to our knowledge to use this strategy), we save users time as they can directly begin working with more promising methods.

One potential direction for choosing anomaly detection methods and parameters is Automated Machine Learning (AutoML). Unfortunately, existing AutoML approaches struggle with anomaly detection as exemplified in the ChaLearn AutoML Challenge (Hutter, Kotthoff, and Vanschoren 2019). Large class imbalance was identified as being the reason for low performance by all teams in this challenge, even more so than data sets with a large number of classes. By definition of an anomaly, non-anomalous data should occur in much greater quantities than anomalous data, presenting a challenge for AutoML systems. Our method could in fact be considered a form of AutoML specifically tailored to anomaly detection, where class imbalance is present by definition. We first use an automated, data-driven approach to filter out less performant or inapplicable methods based on characteristics of the given time series. Hyperparameter optimization is difficult as large, annotated training datasets specific to an application are unlikely to preexist. Therefore, we apply a human-in-the-loop approach where human feedback is used to tune the output of the best performing anomaly detection method, eliminating erroneous anomalies for a specific application without requiring the user to be an expert in anomaly detection.

Our human-in-the-loop technique for tuning anomaly scores is similar to (Herath et al. 2019) and (Madrid et al. 2019). The former uses matrix profiling, but our system can be applied with *any* time series anomaly detection method that outputs an anomaly score. The latter is not built for anomaly detection but for insect behavior classification.

Our Proposed Approach

We propose a filtering approach based on the characteristics a given time series possesses followed by a tuning phase. This is essential as some anomaly detection methods perform better on certain characteristics than others. For example, if the time series in a user's application exhibits concept drift but no seasonality, the user may want to consider GLiM and not Twitter AnomalyDetection (Twitter 2015).

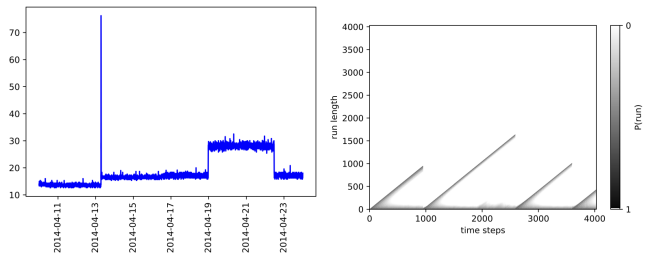

Figure 1: **Left:** A time series exhibiting concept drift. **Right:** Posterior probabilities of the run length at each time step of the left time series using a logarithmic color scale.

Time Series Characteristics

Our list of characteristics are not comprehensive but occur in many real world time series; they were present in all time series in Numenta's benchmark repository.

First, **missing time steps** make it difficult to apply certian anomaly detection methods without interpolation. However, other methods can handle missing time steps innately. The system determines the minimal time step difference in the input time series to find missing time steps, a technique employed in works such as (Xu et al. 2018a). The user can then decide if the missing time steps should be filled using some form of interpolation (e.g. linear) or if the system should limit the selection of anomaly detection methods to those that can innately deal with missing time steps.

Next, the system determines if **concept drift** (Figure 1) is present in the time series where the definition of normal behavior changes over time (Saurav et al. 2018). To detect concept drift we use an implementation of (Adams and MacKay 2007)[1] which utilizes 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 used to determine the presence of concept drifts. The user selects a threshold for the posterior probability for what is considered to be a run and also how long a run must be before it is a concept drift.

The system then determines if a time series contains **seasonality**, the presence of variations that occur at specific regular intervals. Our system makes use of the `FindFrequency` function in the R *forecast* library (Hyndman, Khandakar, and others 2007) which first removes trend from the time series if present and determines the spectral density function from the best fitting autoregressive model. By determining the frequency f that produces the maximum output spectral density value, `FindFrequency` returns $\frac{1}{f}$ as the periodicity of the time series. If no seasonality is present, 1 is returned.

Finally, the system determines if **trend** is present in the time series. Our system detects two types of trend: **1)** stochastic trend which is removed via differencing the time series and identified using the Augmented Dickey-Fuller (ADF) test (Cheung and Lai 1995) and **2)** deterministic trend

[1] `https://github.com/hildensia/bayesian_changepoint_detection`

which is removed via detrending or removing the line of best fit from the time series and identified using the Cox-Stuart test (Linden 2000).

In the case where a time series does not display any of these characteristics, we simply consider all anomaly detection methods initially. This does not provide the run time savings, but the less performant methods will quickly drop out of consideration after the first few disagreements by the human annotator.

Offline Experimentation and Guidelines

The anomaly detection methods we chose to experiment with cover a wide breadth of techniques. Some are probabilistic (Variational AutoEncoders (Xu et al. 2018b)), others are frequency-based (Anomalous (Hyndman 2018)), some rely on neural networks (Hierarchical Temporal Memory, HTMs), and others rely on decomposition of the signal itself (SARIMAX, STL (Cleveland et al. 1990)). We first performed an offline, comprehensive experimental validation on over more than 20 time series on a variety of anomaly detection methods over different time series characteristics to form guidelines.[2]

We used 10 time series for every characteristic as determined by using the techniques discussed above. Thus, every characteristic had a *corpus* of 10 time series. For example, we determined how well Facebook Prophet performs on concept drift by observing its results on 10 time series all exhibiting concept drift. Some of the time series we used came from the Numenta Anomaly Benchmark repository (Numenta 2018) which consists of 58 pre-annotated time series across a wide variety of domains and scripts for evaluating online anomaly detection algorithms. No multivariate time series are provided in Numenta's repository. Meticulous annotation instructions for Numenta's time series are available in (Numenta 2017) and (Lavin and Ahmad 2015). In cases where we did not use Numenta time series, we had undergraduate students tag the time series for anomalies following the same Numenta instructions. There were also several instances where we injected outliers, a technique employed in (Liu, Wright, and Hauskrecht 2017; Choudhary, Hiranandani, and Saini 2018). For seasonality, trend, and concept drift corpi, any missing time steps were filled using linear interpolation. For the missing time step characteristic corpus, we either chose time series with missing time steps already or we randomly removed data points from time series with originally no missing points to generate the corpus. For anomaly detection methods that involve forecasting such as Facebook Prophet, we performed grid search on the parameters to minimize the forecasting error. Otherwise, we choose models and parameters as intelligently as possible based on discovered time series characteristics.

We experimented with two different anomaly detection evaluation methods: windowed F-scores and Numenta Anomaly Benchmark (NAB) scores (see (Numenta 2018) for details on evaluation methodologies). Using these two scoring methodologies, we derived guidelines (Table 1) based on our experiments. For example, for seasonality and trend, decomposition-based anomaly detection methods such as SARIMAX (Seasonal Auto-Regressive Integrated Moving Average with eXogeneous variables), and Facebook Prophet perform the best. SARIMAX and Prophet have decomposition methods with components specifically built for seasonality and trend which might explain their performance on this characteristic. For SARIMAX, seasonal versions of the autoregressive component, moving average component, and difference are considered. The *integrated* portion of SARIMAX allows for differencing between current and past values, giving this methodology the ability to support time series data with trend. For concept drift, more complex methods are necessary such as HTMs (Hierarchical Temporal Memory networks) (Hawkins, Ahmad, and Dubinsky 2010). For missing time steps, the number of directly applicable anomaly detection methods is drastically reduced. Although interpolation is an option, this does introduce a degree of error. If no interpolation is desired, SARIMAX, STL (Seasonal decomposition of Time series by Loess), Prophet, and Generalized Linear Models (GLiMs) are options.

	Seasonality	**Trend**	**Concept Drift**	**Missing Time Steps**
Windowed Gaussian				N/A
SARIMAX	⋆ †	⋆ †	†	
Prophet	†	⋆		
Anomalous				N/A
STL				
Twitter	⋆			N/A
HOT-SAX				N/A
GLiM		†	⋆	⋆ †
HTM	⋆ †		⋆ †	N/A

Table 1: Which anomaly detection method is more promising given a time series characteristic? A ⋆ indicates the windowed F-score scheme favors the method whereas a † indicates Numenta Anomaly Benchmark scores (NAB) favors the method. If there is a N/A, it means that method is not applicable given that time series characteristic. A method is more favored in this table if, after the results of our offline experiments, the method had the best performance with time series displaying that characteristic.

We emphasize that the analyst does not select an optimal method; the analyst is not even aware multiple methods are being considered. The analyst inputs a time series, but it is the system that determines which characteristics are present and then refers to Table 1 to determine which anomaly detection methods should be used.

As there is an ever expanding library of anomaly detection methods we save users time by surfacing the most promising methods. The definition of what is an anomaly is highly subjective, so human input is essential in the decision-making process. Although we automate as much of the process as we can (determining the presence of characteristics, narrowing down the search space of anomaly detection methods), it is not advisable to completely remove the human element.

[2]See `https://s3-us-west-2.amazonaws.com/anon-share/ts2020.zip` for offline experiments.

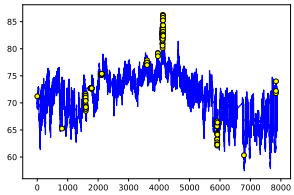

Figure 2: A time series where predicted anomalies are represented as yellow circles.

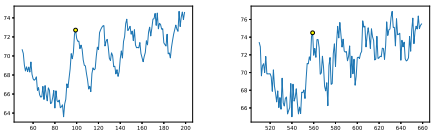

Figure 3: **Left:** A time series with a predicted anomaly (yellow circle) that the annotator disagrees with. **Right:** In the same time series, we see a similar pattern latter on.

For every selected anomaly detection method, its predicted anomalies are given to the user to annotate (Is the predicted anomaly truly an anomaly?), and based on their decision, the parameters for that method can be tuned to reduce the error. Parameter tuning is dependent on the anomaly detection method. For example, if a method produces an anomaly score $\in [0, 100]$ with an anomaly threshold of 75, the system could raise the threshold to reduce false positives. Using this feedback, the system learns to minimize false positives for the user's data. However, there is a plethora of anomaly detection methods, each with their own parameters (Chandola, Banerjee, and Kumar 2009). Determining how to tune these parameters for every possible method is not feasible, especially as the number of anomaly detection methods increases. Many methods already output an anomaly score or can be easily converted to produce such an output. Thus, we tune the anomaly scores instead of the anomaly detection parameters for the sake of generalization.

Tuning Anomaly Scores

We tune anomaly scores based on the following concepts:

Concept One: When an anomaly detection method predicts an anomaly in a time series, these predictions tend to occur in clusters like in Figure 2. To prevent alarm fatigue, we keep the first detection in a cluster but ignore remaining detections in the cluster. The user can specify how many time steps are to be ignored following a detection (*ts_affected*). Given a predicted anomaly, we multiply *ts_affected* many anomaly scores following the predicted anomaly's time step by a sigmoid function, the error function ($erf(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt$) to briefly reduce the anomaly scores of clustered anomalies.

Concept Two: In Figure 3, suppose the annotator disagrees with the predicted anomaly (yellow circle) around time step 100 (left). A very similar pattern occurs in the same time series around time step 500 (right), and the anomaly detection method predicts an anomaly in a similar location. It is likely that the annotator will also disagree with this predicted anomaly. We desire to prevent future predictions that would waste the annotator's time. Therefore, we determine "similar chunks" of the time series when given a false positive by using Mueen's Algorithm for Similarity Search (`MASS`) (Mueen et al. 2017). `MASS` takes a query subsequence (a contiguous subset of values of a time series) and a time series, *ts*. `MASS` then returns an array of normalized Euclidean distances and the indices they begin on to help users identify similar (motifs) or dissimilar (discords) subsequences in *ts* compared to the given query.

For every detected anomaly that the annotator disagrees with, a query is created by forming a subsequence of the time series of length *ts_affected* with the detection in the middle of the subsequence. We reduce the anomaly scores corresponding to these motifs by multiplying them to a sigmoid function: $y = \frac{1}{1+e^{-kx+b}}$ where $b = ln(\frac{1-min\_weight}{min\_weight}), k = \frac{ln(\epsilon)-b}{-max\_distance}$. The more similar the query is to the corresponding motif, the greater the reduction to anomaly scores. The minimum weight multiplied to the anomaly scores is set by *min_weight*, and how quickly the sigmoid function converges to 1 is determined from the max discord distance from the query, *max_distance*, also determined by virtue of using `MASS`.

Results

To fully test our framework, we randomly chose 10 pre-annotated time series from Numenta (Numenta 2018) *not* used in our previous offline experimentations.

We determined the characteristics present in each of these new time series and recorded the time in seconds taken to detect them in column $Time\ Char$ of Table 2. If a time series contained missing time steps, we did not interpolate and relied on anomaly detection methods that can innately deal with missing time steps. Based on the time series characteristics detected, we applied the best performing anomaly detection methods listed in Table 1. For example, the time series *ec2_cpu_utilization_24ae8d* displays concept drift as determined by run length posterior probabilities, so Table 1 suggests that SARIMAX, GLiM, and HTM are the best anomaly detection methods to apply. The total time to detect anomalies with these three methods is 47.81 seconds. We compare it to the time it takes to apply all anomaly detection methods in Table 1, which is 441.61 seconds. However, the method returning the best windowed F-score or NAB score is HTM (for both scoring methodologies) which was one of the predicted top performers. Thus, it would be a waste of time comparing all methods; using just the predicted methods in Table 1 would save time and effort. These best methods were in fact the highest performing for both scoring methodologies for all but one randomly chosen time series we experimented with in Table 2. In only one case was it not best performing: HOTSAX for *iio_us-east-1_i-a2eb1cd9_NetworkIn* with NAB (although the predicted GLiM is best when using windowed F-scores). This is because NAB rewards early detection of anomalies, and,

Dataset	Length	Characteristics	Time Char	# Anom	Time Opt	Time All	Best F	Best NAB	Best Method (using F/NAB)
art_load_balancer_spikes	4032	Trend, Concept Drift	4.16	1	64.98	146.60	.5	41.08	HTM/HTM
ec2_request_latency_system_failure	4032	Seasonality (3), Concept Drift, Miss	4.09	3	48.12	48.12	.86	41.77	Prophet/GLiM
iio_us-east-1_i-a2eb1cd9_NetworkIn	1243	Trend	1.30	2	5.66	26.01	.5	40.78	GLiM/HOTSAX
rogue_agent_key_hold	1882	Missing, Concept Drift	1.02	2	2.19	11.32	.25	40.93	SARIMAX/GLiM
ec2_cpu_utilization_fe7f93	4032	Seasonality (16), Trend	11.11	3	64.32	343.97	.8	41.10	HTM/HTM
ec2_cpu_utilization_24ae8d	4032	Concept Drift	4.52	2	47.81	441.61	.67	41.41	HTM/HTM
art_daily_jumpsdown	4032	Seasonality (13), Trend	11.65	1	43.44	326.63	.67	41.15	GLiM/GLiM
ec2_network_in_257a54	4032	Seasonality (42), Trend, Concept Drift, Miss	4.08	1	12.57	12.82	.67	40.72	GLiM/Prophet
exchange-4_cpc_results	1643	Concept Drift, Trend, Miss	.85	3	10.78	10.78	.46	41.35	Prophet/Prophet
exchange-4_cpm_results	1643	Concept Drift, Miss	.87	4	2.90	7.66	.47	41.64	Prophet/SARIMAX

Table 2: Optimization evaluation datasets. **Time Char** is the total time in seconds to detect all characteristics for the time series. **Time Opt** is the total time in seconds to detect anomalies using only the predetermined *best* methods from Table 1 for the characteristics present whereas **Time All** is the total time to detect anomalies using *all* methods from Table 1. These are equal in cases where some anomaly detection methods are not applicable due to seasonality and/or missing time steps. There is no human input yet in this table.

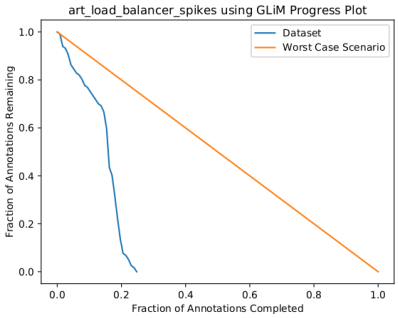

Figure 4: Progress plot (Madrid et al. 2019) for the time series *art_load_balancer_spikes* using the anomaly detection method GLiM. Only 24% of predictions need to be annotated using MASS and cluster prediction elimination.

Dataset	Before Best F	After Best F
art_load_balancer_spikes	.5	.5
ec2_request_latency_system_failure	.86	1
iio_us-east-1_i-a2eb1cd9_NetworkIn	.5	.67
rogue_agent_key_hold	.25	.31
ec2_cpu_utilization_fe7f93	.8	.8
ec2_cpu_utilization_24ae8d	.67	.8
art_daily_jumpsdown	.67	1
ec2_network_in_257a54	.67	1
exchange-4_cpc_results	.46	.55
exchange-4_cpm_results	.47	.62

Table 3: **Before Best F** shows windowed F-scores for the best performing anomaly detection method on the corresponding time series (no human feedback). **After Best F** shows F-scores for the *same method* applying MASS and prediction cluster elimination after each annotation.

in this instance, HOTSAX detected anomaly scores earlier than other anomaly detection methodologies.

Table 2 does not contain results of human input. We now experiment with Concepts 1 and 2 which involves human input.[3] As in (Madrid et al. 2019), we create *progress plots* where the x-axis is the fraction of annotations already done, and the y-axis shows the fraction of annotations left.[4] In the worst case scenario, every annotation only reduces the number of remaining annotations by 1 ($y = 1 - x$). This would mean that there are no anomaly detection clusters and no similar instances of confirmed false positives. In Figure 4, we show the progress plot for *art_load_balancer_spikes* using the anomaly detection method GLiM. Without removing clusters and applying MASS, 117 predictions would need to be reviewed by annotators. Using both concepts, only 29 annotations are needed in total, reducing the fraction of needed annotations by almost 80%.

As an example of Concept 2, the annotator disagrees with the first prediction made by GLiM. MASS determines there is a similar subsequence further along in the time series containing a prediction not yet tagged and lowers the anomaly scores corresponding to this subsequence. Thus, instead of 117 annotations being reduced to 116 after a single annotation, we have 115 remaining. In all but the worst case, as the reviewer makes annotations, the number of annotations remaining goes down in steps greater than 1.

Out of the 67 time series and anomaly detection method combinations, only 9 had worst case scenario progress plots. In total, the number of predictions that would need to be annotated across all 67 combinations without prediction cluster elimination and MASS is 1,701. Using MASS and prediction cluster elimination, the number of annotations required is 483, a 71.6% reduction in annotations. Average MASS running time after an annotation was .17 seconds across all 67 time series-method combinations. In addition, using the two concepts often increases evaluation scores due to the reduction in false positives. For our test time series, we did not see false negatives upon application of the two concepts. Either there was no change (because there are no clusters or there are no similar subsequences containing a disagreed upon anomaly) or an increase in precision due to the reduction of false positives. Table 3 displays the windowed F-scores of the best performing anomaly detection method without

[3] *ts_affected*=2% of time series length, *min_weight*=.95

[4] As the time series used are already annotated by Numenta, we "annotate" by using the ground truths provided by Numenta.

using MASS and prediction cluster elimination from Table 2 versus using MASS and prediction cluster elimination on the same method. On average, windowed F-scores increased by .14 by using MASS and prediction cluster elimination. In 8 out of 10 time series, NAB scores either stayed the same or increased in value. We suspect this is because NAB explicitly rewards early detection of anomalies, and predictions made slightly before ground truths may have been removed using the two concepts, reducing the scores (Numenta 2018).

Conclusion

Anomaly detection is a challenging problem for many reasons, one of them being method selection in an ever expanding library, especially for non-experts. Our method tackles this problem in a novel way by first determining the characteristics present in the given data and narrowing the choice down to a smaller set of promising anomaly detection methods. We determine these methods using over 20 pre-annotated time series and validate our system's ability on choosing better methods by experimenting with another 10 time series. We incorporate user feedback on predicted outliers from the methods in this smaller set, utilizing MASS and removing predicted anomaly clusters to tune these methods to the user's data, reducing the need for annotation by 70% on predicted anomalies while increasing evaluation scores.

References

Adams, R. P., and MacKay, D. J. 2007. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*.

Blázquez-García, A.; Conde, A.; Mori, U.; and Lozano, J. A. 2020. A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236*.

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3):15.

Cheung, Y.-W., and Lai, K. S. 1995. Lag order and critical values of the augmented dickey–fuller test. *Journal of Business & Economic Statistics* 13(3):277–280.

Choudhary, S.; Hiranandani, G.; and Saini, S. K. 2018. Sparse decomposition for time series forecasting and anomaly detection. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, 522–530. SIAM.

Cleveland, R. B.; Cleveland, W. S.; McRae, J. E.; and Terpenning, I. 1990. Stl: A seasonal-trend decomposition. *Journal of Official Statistics* 6(1):3–73.

Etsy. 2015. Skyline. https://github.com/etsy/skyline.

Hawkins, J.; Ahmad, S.; and Dubinsky, D. 2010. Hierarchical temporal memory including htm cortical learning algorithms. *Technical report, Numenta, Inc, Palto Alto*.

Herath, J. D.; Bai, C.; Yan, G.; Yang, P.; and Lu, S. 2019. Ramp: Real-time anomaly detection in scientific workflows.

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

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. Automated machine learning-methods, systems, challenges.

Hyndman, R. J.; Khandakar, Y.; et al. 2007. *Automatic time series for forecasting: the forecast package for R*. Number 6/07. Monash University, Department of Econometrics and Business Statistics .

Hyndman, R. J. 2018. Anomalous. https://github.com/robjhyndman/anomalous.

Laptev, N.; Amizadeh, S.; and Flint, I. 2015. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1939–1947. ACM.

Lavin, A., and Ahmad, S. 2015. The numenta anomaly benchmark (white paper). https://github.com/NAB/wiki.

Linden, M. 2000. Testing growth convergence with time series data: a non-parametric approach. *International Review of Applied Economics* 14(3):361–370.

LinkedIn. 2018. Luminol. https://github.com/linkedin/luminol.

Liu, D.; Zhao, Y.; Xu, H.; Sun, Y.; Pei, D.; Luo, J.; Jing, X.; and Feng, M. 2015. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 Internet Measurement Conference*, 211–224. ACM.

Liu, S.; Wright, A.; and Hauskrecht, M. 2017. Online conditional outlier detection in nonstationary time series. In *Proceedings of the... International Florida AI Research Society Conference. Florida AI Research Symposium*, volume 2017, 86. NIH Public Access.

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

Lytics. 2015. Anomalyzer. https://github.com/lytics/anomalyzer.

Madrid, F.; Singh, S.; Chesnais, Q.; Mauck, K.; and Keogh, E. 2019. Efficient and effective labeling of massive entomological datasets.

Mueen, A.; Zhu, Y.; Yeh, M.; Kamgar, K.; Viswanathan, K.; Gupta, C.; and Keogh, E. 2017. The fastest similarity search algorithm for time series subsequences under euclidean distance. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

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

Numenta. 2018. The numenta anomaly benchmark. https://github.com/numenta/NAB.

Saurav, S.; Malhotra, P.; TV, V.; Gugulothu, N.; Vig, L.; Agarwal, P.; and Shroff, G. 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*, 78–87. ACM.

Twitter. 2015. Anomalydetection. https://github.com/twitter/AnomalyDetection.

Vallis, O.; Hochenbaum, J.; and Kejariwal, A. 2014. A novel technique for long-term anomaly detection in the cloud. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*.

Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y.; et al. 2018a. 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*, 187–196. International World Wide Web Conferences Steering Committee.

Xu, H.; Chen, W.; Zhao, N.; Li, Z.; Bu, J.; Li, Z.; Liu, Y.; Zhao, Y.; Pei, D.; Feng, Y.; et al. 2018b. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, 187–196.