The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Experimental Comparison of Online Anomaly Detection Algorithms

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

Abstract

Anomaly detection methods abound and are used extensively in streaming settings in a wide variety of domains. But a strength can also be a weakness; given the vast number of methods, how can one select the best method for their application? Unfortunately, there is no one best way for all domains. Existing literature is focused on creating new anomaly detection methods or creating large frameworks for experimenting with multiple methods at the same time. As the literature continues to grow, extensive evaluation of every available anomaly detection method is not feasible. To reduce this evaluation burden, in this paper we present a framework to intelligently choose the optimal anomaly detection methods based on the characteristics the time series displays. We provide a comprehensive experimental validation of multiple anomaly detection methods over different time series characteristics to form guidelines. Applying our framework 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.

Introduction

An anomaly in a time series is a pattern that does not conform to past patterns of behavior in the series. It is used in a wide variety of fields such as intrusion and fraud detection, tracking KPIs, and medical sensor technologies. Early detection of anomalies is vital for ensuring undisrupted business and efficient troubleshooting.

Time series anomaly detection is a difficult problem for a multitude of reasons: **(1)** What is defined as anomalous may differ based on application. There is no one-size-fits-all method (Kejariwal 2015; Laptev, Amizadeh, and Flint 2015). **(2)** 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, computational efficiency is vital which presents a challenge. **(3)** Given the application-specific nature of anomaly detection, it is unlikely that anomaly detection systems will have access to large numbers of tagged datasets. Therefore, these systems will likely encounter behavior that was not present in the training data. **(4)** As an anomaly is a pattern that does not conform to past patterns of behavior, non-anomalous data tends to occur in significantly larger quantities than anomalous data. This guarantees the imbalanced class problem for a machine learning classifier approach to anomaly detection. **(5)** It is important to detect as many anomalies as accurately and efficiently as possible, but minimizing false positives is also desirable to avoid alarm fatigue. This demands the selected anomaly detection method be optimal to the application for success. **(6)** There is a massive wealth of anomaly detection methods to choose from.

Because of these difficulties inherent in time series anomaly detection, we present a framework for automating the classification of time series and choice of anomaly detection method based on the characteristics the time series possesses. For example, if the time series data in a user's application exhibits concept drift, the user may want to consider a forecasting RNN and not Twitter AnomalyDetection. If the time series exhibits trend, Donut's variational auto-encoder may be a good choice. 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 the performance of the methods. After obtaining the results, we derive several guidelines on method selection by time series characteristics and outline areas for future work.

Preliminaries

We discuss the time series characteristics under consideration, how to detect them, and the evaluation datasets.

Time Series Characteristics

A time series is **stationary** if the mean, variance, and autocorrelation structure are constant for all time (Natrella 2013). A typical first step for determining stationarity is conducting an Augmented Dickey-Fuller test, a statistical unit root test which can determine how strongly a time series is defined by a trend (Brownlee 2016). The Levene test is a test for nonconstant variance (Pardoe 2018). Although other such tests exist, the Levene test does not require terms to be drawn from a normal distribution.

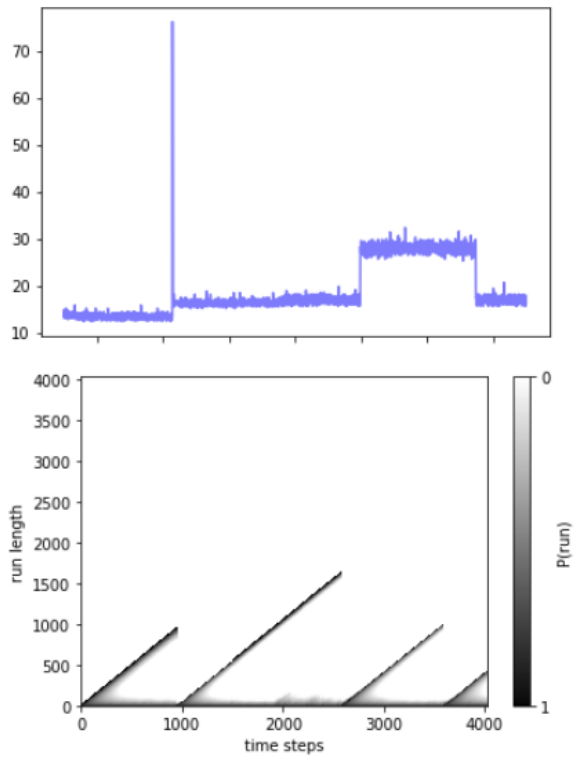

Figure 1: An example of concept drift. The posterior probability of the run length at each time step using a logarithmic color scale is included in the bottom plot.

Non-stationary time series may exhibit **seasonal** behavior. It is often necessary to analyze the autocorrelation function (ACF) which displays the correlation for time series observations with its lags (observations from previous time steps) (Keshvani 2013). A time series that exhibits seasonality will show a repetitive pattern in its autocorrelation plot. If most points are within the confidence intervals of an ACF plot, the time series may not contain seasonality. In addition to analyzing an ACF plot, one could consider using a Ljung-Box test. Other behaviors that may be exhibited by non-stationary time series include **trends** and **concept drift** where the definition of normal behavior changes over time (Saurav et al. 2018). Concept drifts can be difficult to detect especially if one does not know beforehand how many concept drifts there are. In (Adams and MacKay 2007), this number does not need to be known. An implementation of this paper is available in (Kulick 2016) using t-distributions for each 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 scale (see Figure 1).[1]

[1]https://github.com/cynthiaw2004/adclasses contains Jupyter notebooks for determining the presence of all characteristics.

Example Datasets

Almost all datasets we use come from the Numenta Anomaly Benchmark (Numenta 2018) repository which contains pre-annotated datasets across a wide variety of domains. The exception is the *ibm-common-stock-closing-prices* dataset, from (Hyndman 2018) which consists of the daily stock closing prices for IBM from 1962 to 1965. This dataset was annotated following the Numenta instructions on (Numenta 2017) and (Lavin and Ahmad 2015). Numenta instructions require 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. Missing time steps are filled using linear interpolation. Using the statistical tests listed in the previous subsection, we create corpora of behaviors: 3 datasets each for seasonality, trend, and concept drift. In Table 1, we provide a summary of all datasets under consideration.

Online Anomaly Detection Methods

We now define and discuss a selection of anomaly detection methods. Given a time series, **ARMA** models are tools for understanding and forecasting future values by regressing the variable on its own past values (**AR**) and modeling the error term as a linear combination of current and past error terms (**MA**). By differencing between current and past values, the time series can hopefully be made stationary (**ARIMA**). If seasonality is incorporated, we have a **SARIMA** model. Much manual analysis can be necessary (see Box-Jenkins method (Hoff 1983)) in choosing the parameters for a SARIMA model (Nau 2018) although there are general guidelines (Fomby 2008; SAS 2018) and libraries that can automatically determine parameters, but these methods are not perfect (Bourgeon 2016) and can be very slow. **Facebook Prophet** is an additive regression model that begins with a special time series decomposition method ($y(t) = g(t) + s(t) + h(t) + \epsilon_t$) which involves 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 and Letham 2017; Sean J. Taylor 2017). ϵ_t is the error term and is assumed to be normally distributed. Parameters are determined using MAP (maximum a posteriori) optimization. Prophet can automatically detect changes in trends by selecting change points between concept drifts. The user can also provide custom change points if desired. However, this requires knowing where change points are beforehand. Prophet was originally designed for daily time steps (Davidson-Pilon 2017). Although adjustments have been made to deal with sub-day time steps, time steps larger than a day can have unexpected behaviors (Letham 2017). Given a window of time steps in the past, **multi-step forecasting RNNs** can be trained to predict a window of time steps in the future. We use the multi-step prediction RNN with GRU units[2] developed in (Saurav

[2]GRU units adapt more quickly to concept drifts than LSTMs and are computationally more efficient (Yin et al. 2017) which is important as many anomaly detection tasks must be done online.

Name	Length	Step	Min	Max	Median	Mean	# Anomalies	# Miss	Corpora
nyc_taxi	10320	30 min	8	39197	16778	15137.569	5	0	seasonal (48)
exchange-2_cpm_results	1624	1 hr	0	1.051	0.295	0.337	2	25	seasonal (24), possible trend
ambient_temperature_system_failure	7267	1 hr	57.458	86.223	71.858	71.242	2	621	possible trend, seasonal (24)
ibm-common-stock-closing-prices	1008	1 day	306	598.50	460.625	462.818	2	452	possible trend
grok_asg_anomaly	4621	5 min	0	45.623	33.445	27.685	3	0	concept drift
rds_cpu_utilization_cc0c53	4032	5 min	5.190	25.103	6.082	8.112	2	1	concept drift
rds_cpu_utilization_e47b3b	4032	5 min	12.628	76.230	16.678	18.935	2	0	concept drift

Table 1: Summary of all datasets. Statistics were gathered before any linear interpolation if time steps are missing. **Step** is the time step size, **Min** and **Max** are the minimum and maximum values, and **# Miss** is the number of missing time steps in the dataset. **Corpora** is one or more corpora the dataset belongs to by characteristics it displays. If there is seasonality, we include the number of time steps per period in parenthesis.

et al. 2018) which can adapt to concept drifts after an initial period of training. The RNN is then trained and tested incrementally to determine anomaly scores as an average of the prediction error. Computing the score as an average is important; a very short-term anomaly will only affect the anomaly score for a short period of time. If the anomaly score is consistently large, however, this alerts the RNN to adapt to the new normal by changing its parameters accordingly. **Twitter's AnomalyDetection** (AD) (Twitter 2015; Hochenbaum, Vallis, and Kejariwal 2017) detects anomalies using a modified version of the extreme studentized deviate test (ESD). As a preprocessing step, the time series is made stationary by applying a modified version of STL (seasonal and trend decomposition using LOESS) where the median of the time series is used to represent the trend component. The authors in (Hochenbaum, Vallis, and Kejariwal 2017), argue that the mean and standard deviation are very sensitive to anomalous data and that the median is statistically more robust. 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. Unfortunately, this method may not adapt well to concept drift. Twitter counters this by stating that anomalies are *point-in-time* anomalous data points whereas concept drifts (or *breakouts*) "ramp up from one steady state to another" (Kejariwal 2015). A downside of using ESD is that an upper bound on the number of anomalies must be specified, otherwise ESD will assume that up to half of the data can be anomalous (Choudhary, Hiranandani, and Saini 2018). Despite STL's innate capability to handle missing time steps, Twitter's AD library will error out and suggest replacing NANs with interpolated values as Twitter AD uses R's default *stl* library instead of the *stlplus* library. **Donut** (Xu et al. 2018) is an unsupervised anomaly detection method based on variational auto-encoders (VAE) available in Python (Xu 2018). VAE is a deep Bayesian network, and as it is not a sequential model, a sliding window is used over the time series. Optimizing the evidence lower bound (ELBO) using Stochastic Gradient Variational Bayes (SGVB) is traditionally used to train VAEs. However, abnormal patterns and/or missing points need to be avoided as much as possible when training the VAE-based model for anomaly detection; thus, the authors use a modified version of ELBO to indicate either when an anomaly has occurred (if a supervised task, but not necessary) or if there are missing points. This modification helps Donut reconstruct missing points using a MCMC (Markov chain Monte Carlo) based missing data imputation technique. According to (Xu 2018), the *smallest* time step is used to resample the time series. The likelihood of a window can be returned by the VAE (using the last data point of the window), and a probability density can be computed, but the authors discover that this method does not work well in practice. Thus, they determine the "reconstruction probability" (An and Cho 2015) using the variational posterior. This is *not* a well-defined probability density.

Note that some popular anomaly detection "methods" were not considered because we do not wish to compare frameworks but the anomaly detection methods themselves. For example, Yahoo's EGADS (Laptev, Amizadeh, and Flint 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. Many of these components are already discussed above. Our goal is to compare the methods and not frameworks. Other popular frameworks include: LinkedIn's Luminol, Etsy's Skyline, Mentat Innovation's datastream.io, eleme's banshee, Opprentice, and Lytics Anomalyzer.

Experiment Setup

For **SARIMA** parameters, we initially use the guidelines in (SAS 2018) based on the presence of seasonality and trend. For example, if there is seasonality (with s time steps every cycle) but no trend, consider using *seasonal exponential smoothing* (SARIMA$(0, 1, s+1) \times (0, 1, 0, s)$). If there are invertibility issues, we use Pyramid's auto.arima which is usually considerably slower and more memory intensive. Pyramid determines the best set of parameters according to a given information criterion (we use 'AIC') and the stepwise algorithm (Hyndman 2008) which is less likely to overfit compared to an extensive grid search. Like (Numenta 2018), 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. For **Facebook Prophet**, we generally use default parameters, allowing Prophet to automatically determine if there are change points, etc. We use "linear" for the growth parameter as setting it to "logistic" requires knowing a maximum and min-

imum value the data will reach (van der Merwe 2018). In an online setting, this knowledge is often not known. As seasonality is a time series characteristic determined beforehand, this is the only parameter we manually adjust in Prophet. Q-functions are also used to determine anomaly scores based on the prediction error. For **RNNs**, we use 3 GRU layers ending with a dense layer. We use the Adam optimizer with a learning rate of .0005 and $\beta_1 = .5$. The number of input observations (n_{lag}) is 100, the number of output observations (n_{seq}) is 50, the number of epochs is 50, and the number of cells in each layer is 40. The initial training period matches the probationary period. Extensive grid search in an online, streaming setting with this many parameters is typically not possible. In addition, as anomalies are (and should be) rare, it can be difficult to obtain a train and test set containing both anomalous and non-anomalous behavior. Instead, our goal is to choose models and parameters as intelligently as possible based on discovered time series characteristics. Although an anomaly score is naturally returned for the RNN, this score is data dependent and not normalized. Thus, we apply the Q function *on* the anomaly score to normalize it. For **Twitter AnomalyDetection**, the direction of anomalies is set to 'both', $\alpha = .001$, and the upper bound percentage on the number of anomalies is set to $\frac{2}{\text{\# of time steps}}$ to emulate (Numenta 2018) as in a streaming setting, it is impossible to know the number of anomalies beforehand. The anomaly score *is* the label (0 or 1). For **Donut**, we use the default sliding window of 120 whenever possible (or half the default when not), and apply the Q function to the anomaly score as it is not well-defined.

Results

To evaluate and compare the anomaly detection methods on different time series behaviors, we use the standard metrics of precision, recall, and F-score on *windows* as due to class imbalance, accuracy is not a good measure, and points are too fine a granularity. For the purposes of this paper, we use the same window size as in (Numenta 2018) which is 10% of the number of time steps in the time series divided by the number of ground truth anomalies.[3] All anomaly detection method times reported were done on a laptop with an Intel Core i7-4710MQ CPU @ 2.50GHz, 16 GB of RAM, running Ubuntu 16.04. In Tables 2 and 3, every anomaly detection method considered returns an anomaly score between 0 and 1 or is adjusted to return a score (via a Q function). A threshold is set on this score to determine if the window is an anomaly. To choose this threshold, we use the same methodology as in (Numenta 2018). However, instead of choosing the threshold (in steps of .0001) based on what returns the best NAB score, we choose the threshold based on what returns the minimum number of window mistakes across all datasets comprising a corpus (where every behavior type has its own respective corpus).[4] In determining the threshold, we chose to focus on false negatives more than false positives and give a weight of 1 to false negatives ($w_{FN} = 1$) and .5 to false positives ($w_{FP} = .5$). The choice of weights is application dependent and can affect how anomaly detection methods perform. For example, SARIMA suffers with a 0 F-score for trend in Table 2. This is because it gives no predictions under $w_{FP} = .5$. However, SARIMA will actually give anomaly predictions if this weight is set lower ($w_{FP} = .1$ returns a F-score of .4 for SARIMA).[5]

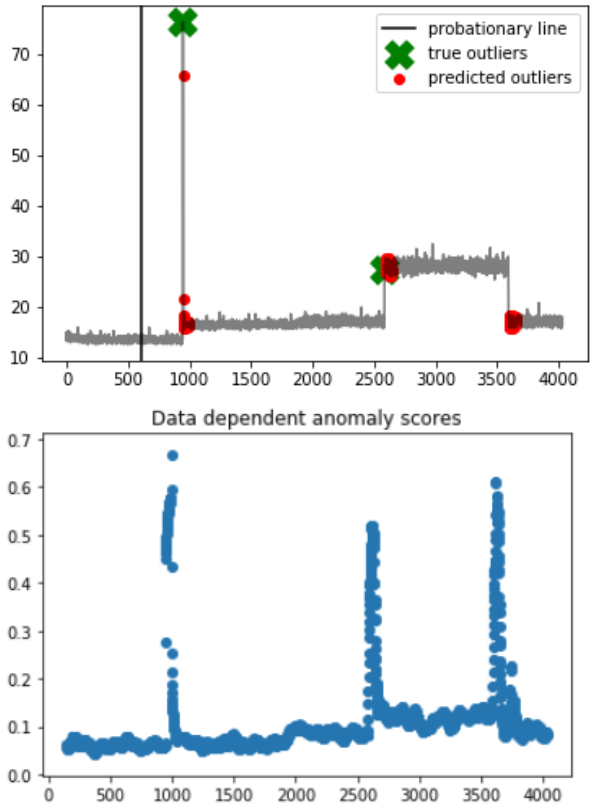

Figure 2: *Top plot:* The RNN's predicted outliers (red circles) versus the ground truth outliers (green x's) on the time series rds_cpu_utilization_e47b3b. Ground truths are provided in Numenta's repository. *Bottom plot:* The anomaly scores (not yet normalized) for the RNN. How quickly the anomaly score decreases after an anomaly depends on the size of the prediction window.

Discussion and Conclusion

The RNN in (Saurav et al. 2018) uses GRU units to adapt more quickly to concept drifts, and the RNN achieves the

[3]Like in (Numenta 2018), we do not remove the probationary period of the time series before determining window length. Also, we acknowledge that the number of ground truth anomalies is not known beforehand in an online setting. This means that the window size will most likely be an application specific user choice.

[4]Note that Twitter AD's anomaly score threshold appears very small (.0001), but recall that Twitter AD returns a label of 0 or 1, and this label *is* the anomaly score.

[5]Precision-recall curves for every behavior-method combination (15 total) are available in https://github.com/cynthiaw2004/adclasses.

	SARIMA				Prophet				RNN				Twitter				Donut			
	Threshold	P	R	F-Score	Threshold	P	R	F-Score	Threshold	P	R	F-Score	Threshold	P	R	F-Score	Threshold	P	R	F-Score
Seasonality	1	1	.22	.36	.9999	.6	.67	**.63**	.9999	.33	.78	.47	.0001	1	.22	.36	1	.33	.44	.38
Trend	1	0	0	0	1	1	.17	.29	.9999	.14	.33	.2	.0001	1	.17	.29	1	.43	.5	**.46**
Concept Drift	1	1	.43	**.6**	1	1	.29	.44	.9835	.46	.86	**.6**	.0001	.67	.29	.4	1	.5	.57	.53

Table 2: Results on behavior corpora. Each method returns an anomaly score between 0 and 1. The anomaly threshold that minimizes the number of window mistakes across *all* datasets that comprise the corpus is shown, where false positives are weighted .5 and false negatives are weighted 1. For example, a threshold of .9835 for the RNN returns the minimum number of weighted mistakes across all three concept drift datasets. The precision (P), recall (R), and F-score across all datasets making up the corpus is displayed in the table, where the best F-score for a behavior is bolded in its respective row.

		SARIMA			Prophet			RNN			Twitter			Donut		
		FP	FN	Time	FP	FN	Time	FP	FN	Time	FP	FN	Time	FP	FN	Time
Seasonality	nyc_taxi	0	3	18.23	0	2	27.79	5	0	597.88	0	4	.66	3	4	38
	exchange-2_cpm_results	0	2	1.81	9	0	8.27	0	2	69.01	0	1	.29	1	1	3.58
	ambient_temperature_system_failure	0	2	7.73	1	1	31.21	5	0	386.69	0	2	.49	3	0	25.65
Trend	exchange-2_cpm_results	0	2	1.81	0	1	8.27	0	2	69.01	0	1	.29	1	1	3.58
	ambient_temperature_system_failure	0	2	7.73	0	2	31.21	5	0	386.69	0	2	.49	3	0	25.65
	ibm-common-stock-closing-prices	0	2	.46	0	2	5.55	1	2	46.31	0	2	.20	0	2	3.51
Concept Drift	grok_asg_anomaly	0	2	5.84	0	3	26.15	3	0	312.31	1	3	.32	1	1	15.12
	rds_cpu_utilization_cc0c53	0	1	5.55	0	1	28.06	0	1	297.57	0	0	.5	2	1	14.76
	rds_cpu_utilization_e47b3b	0	1	22.44	0	1	20.18	3	0	296.54	0	2	.37	1	1	15.18

Table 3: Results on all datasets. As a dataset may appear in multiple corpora, a single dataset may appear multiple times in this table but give different results as thresholds are determined on a behavioral basis (using $w_{fp} = .5$ and $w_{fn} = 1$) and not a dataset basis. The number of false positives (FP) and false negatives (FN) as well as the time it took (in seconds) for the anomaly detection method to produce anomaly scores are shown.

highest F-score (tied with SARIMA) out of all methods for the concept drift corpus (Table 2). However, the RNN also has a total of 22 false positives (see Table 3 and Figure 2). Despite the occurrence of false positives, these false positives tend to occur very closely to the location of ground truth anomalies. We try to counteract this by using window precision and recall instead of point precision and recall, but if the window size is small, false positives can still occur. There is also a tradeoff on the choice of n_{seq}. The larger n_{seq} is, the earlier the change detection happens, but the longer false positives will persist after the occurrence of a true anomaly. The authors demonstrate this behavior by experimenting with $n_{seq} = 1, 5, 10$ in the literature. As we use $n_{seq} = 50$ for speed considerations, this increases the number of false positives. However, these false positives still occur very closely to ground truths. SARIMA's ability to handle concept drifts is surprising given that its parameters are only determined in the probationary period (whereas the concept drift occurs after the probationary period) with an average RMSE of 1.30 across all 3 concept drift datasets. Although Prophet can detect change points automatically, it was middle of the pack. Twitter AnomalyDetection performed the worst on concept drift, but this is not surprising as Twitter has stated that change points are not the focus of AnomalyDetection (Kejariwal 2015). Donut performs the best on trend which follows as Donut was built to handle "seasonal KPIs with local variations" with an explicit example of such a variation being trend (Xu et al. 2018). Prophet does well on the seasonality corpus although all datasets considered use 5 minute time steps. It would be interesting to analyze the performance of Prophet on seasonal time series with different time step sizes as Prophet's behavior on time steps larger than a day has been reported to be unexpected (Letham 2017). Prophet differs from SARIMA in that it formulates this problem as a curve-fitting exercise (Hastie and Tibshirani 1987). Its decomposition method has a component specifically built for seasonality which might explain its performance.

In this paper, we have analyzed the performance of 5 anomaly detection methods (SARIMA, Facebook Prophet, multistep forecasting RNN, Twitter AD, and Donut) on several time series characteristics (seasonality, trend, and concept drift). We create corpora of behaviors with 3 datasets for every time series characteristic. By analyzing the window precision, recall, and F-score we determine which methods tend to perform better or worse on these characteristics. We observe that **(1)** despite the prevalence of false positives, the RNN with GRU units adapts well to concept drifts whereas Twitter AD suffers, **(2)** Facebook Prophet performs well on seasonality, suggesting that decomposition-based methods with components specifically for seasonality aid in performance on this characteristic, and **(3)** although all methods suffer on trend, Donut, the variational auto-encoder, appears to perform the best on this behavior, suggesting that VAEs may be a worthwhile choice for trend. 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 missing time steps or nonuniform time step sizes. For example, Donut can innately handle missing time steps, but Twitter AnomalyDetection does not. Inclusion of contextual variables may change initial perceptions of what is anomalous. Another avenue for future work is expanding the repertoire of anomaly detection methods (see (Gupta et al. 2014; Wu 2016)). Numenta's Hierarchical Temporal Memory Network can be used for anomaly detection although it currently only works for univariate time series. Robust Principle Component Analysis (RPCA) with Netflix SURUs and HOT-SAX are other possibilities and exploring what types of time series behaviors these methods perform well (or even

not well) at would help the community. If the time series data in a user's application exhibits concept drift, the user may want to consider a RNN with GRU units and not Twitter AnomalyDetection. Instead of doing an extensive literature review and trying every anomaly detection method in a rapidly expanding library, one could just observe characteristics present in the data and narrow the choice down to a smaller class of promising anomaly detection methods.

Acknowledgements

Dr. Mueen's work is supported by NSF Grant OIA-1757207.

References

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

An, J., and Cho, S. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* 2:1–18.

Bourgeon, R. 2016. Seasonality not taken account of in auto.arima(). https://stats.stackexchange.com/questions/213201/seasonality-not-taken-account-of-in-auto-arima.

Brownlee, J. 2016. How to check if time series data is stationary with python. https://machinelearningmastery.com/time-series-data-stationary-python/.

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.

Davidson-Pilon, C. 2017. Extension to hourly components? https://github.com/facebook/prophet/issues/29.

Fomby, T. B. 2008. Exponential smoothing models. *Mannual SAS/ETS Software: Time Series Forecasting System. Version* 6:225–235.

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

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

Hochenbaum, J.; Vallis, O. S.; and Kejariwal, A. 2017. Automatic anomaly detection in the cloud via statistical learning. *arXiv preprint arXiv:1704.07706*.

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

Hyndman, R. 2008. J., and y. khandakar:"automatic time series forecasting: The forecast package for r,". *Journal of Statistical Software* 26(3).

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

Kejariwal, A. 2015. Introducing practical and robust anomaly detection in a time series. https://blog.twitter.com/engineering/en_us/a/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series.html.

Keshvani, A. 2013. How to use the autocorreation function (acf)? https://coolstatsblog.com/2013/08/07/how-to-use-the-autocorreation-function-acf/.

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

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.

Letham, B. 2017. Sub-daily data. https://github.com/facebook/prophet/blob/v0.2/notebooks/non-daily_data.ipynb.

Natrella, M. 2013. Engineering statistics handbook: Stationarity. https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm.

Nau, R. 2018. Summary of rules for identifying arima models. https://people.duke.edu/~rnau/arimrule.htm.

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.

Pardoe, I. 2018. Tests for constant error variance. https://onlinecourses.science.psu.edu/stat501/node/367/.

SAS. 2018. Equations for the smoothing models. http://support.sas.com/documentation/cdl/en/etsug/63348/HTML/default/viewer.htm#etsug_tffordet_sect014.htm.

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.

Sean J. Taylor, B. L. 2017. Prophet:forecasting at scale. https://research.fb.com/prophet-forecasting-at-scale/.

Taylor, S. J., and Letham, B. 2017. Forecasting at scale. *URL: https://facebookincubator. github. io/prophet*.

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

van der Merwe, R. 2018. Implementing facebook prophet efficiently. https://towardsdatascience.com/implementing-facebook-prophet-efficiently-c241305405a3.

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

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

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

Yin, W.; Kann, K.; Yu, M.; and Schütze, H. 2017. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*.