Supplementary Material

Unsupervised real-time anomaly detection for streaming data

Subutai Ahmad[*], Alexander Lavin[*], Scott Purdy[*], Zuha Agha[*†]
**Numenta, Redwood City, California, USA*
† University of Pittsburgh, Pittsburgh, Pennsylvania, USA

Correspondence: Subutai Ahmad, Numenta, Inc. 791 Middlefield Road, Redwood City, CA 94063 USA. sahmad@numenta.com

S1. NAB Scoring Details

Consider the application profile A with A_{TP}, A_{FP}, A_{TN}, A_{FN} indicating the weights of corresponding detections. Let y be the position of the detection relative to the anomaly window. Then the scaled sigmoidal scoring function that defines the weight of individual detections is given by:

$$\sigma^A(y) = (A_{TP} - A_{FP})\left(\frac{1}{1+e^{5y}} - 1\right) \tag{S1}$$

The raw score for each data file is computed by summing up weights in Eq. (S1) for each individual detection and a weighted decrement for all missing detections f_d, given by the equation below:

$$S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y)\right) + A_{FN} f_d \tag{S2}$$

The final reported score is normalized as follows:

$$S_{NAB}^A = 100 \cdot \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A} \tag{S3}$$

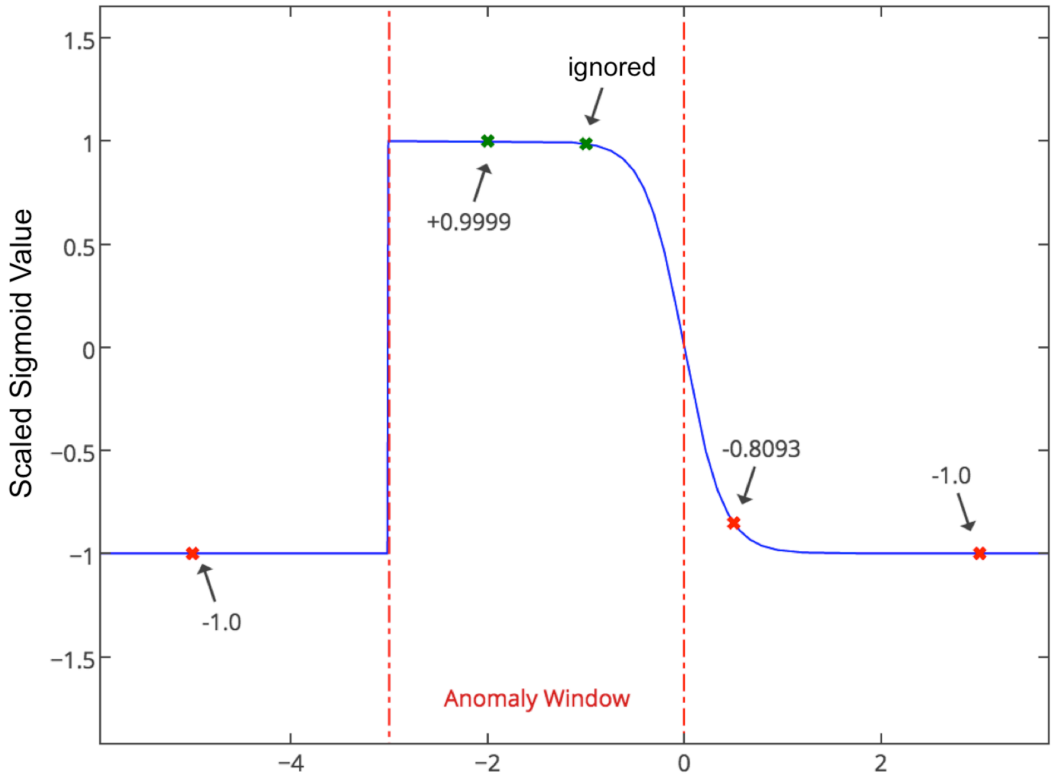

Fig. S1. NAB scoring example. The red dashed lines represent anomaly window. All detections outside the window are false positives marked with red whereas detections within the window are true positives marked with green. Only the earliest detection within the window is taken into consideration. For outside detections, as the distance of the detection from the window boundary increases, its contribution decreases as given by sigmoidal scoring function in Eq. S1.

S2. Application of HTM Anomaly Detection to Geospatial Data

The following demonstrates application of HTM to geospatial data. Sparse vectors can be created from the coordinates and speed, and be used as inputs to the HTM algorithms [1]; the encoding algorithm is included in the NuPIC repository . The same anomaly detection technique described in Section 3 then gives detections for unusual locations, speeds, or sequences of locations and speeds. Experiments using GPS coordinates from driving sequences show how HTM anomaly detection quickly learns routes that are repeated and then detects various types of unusual behaviors.

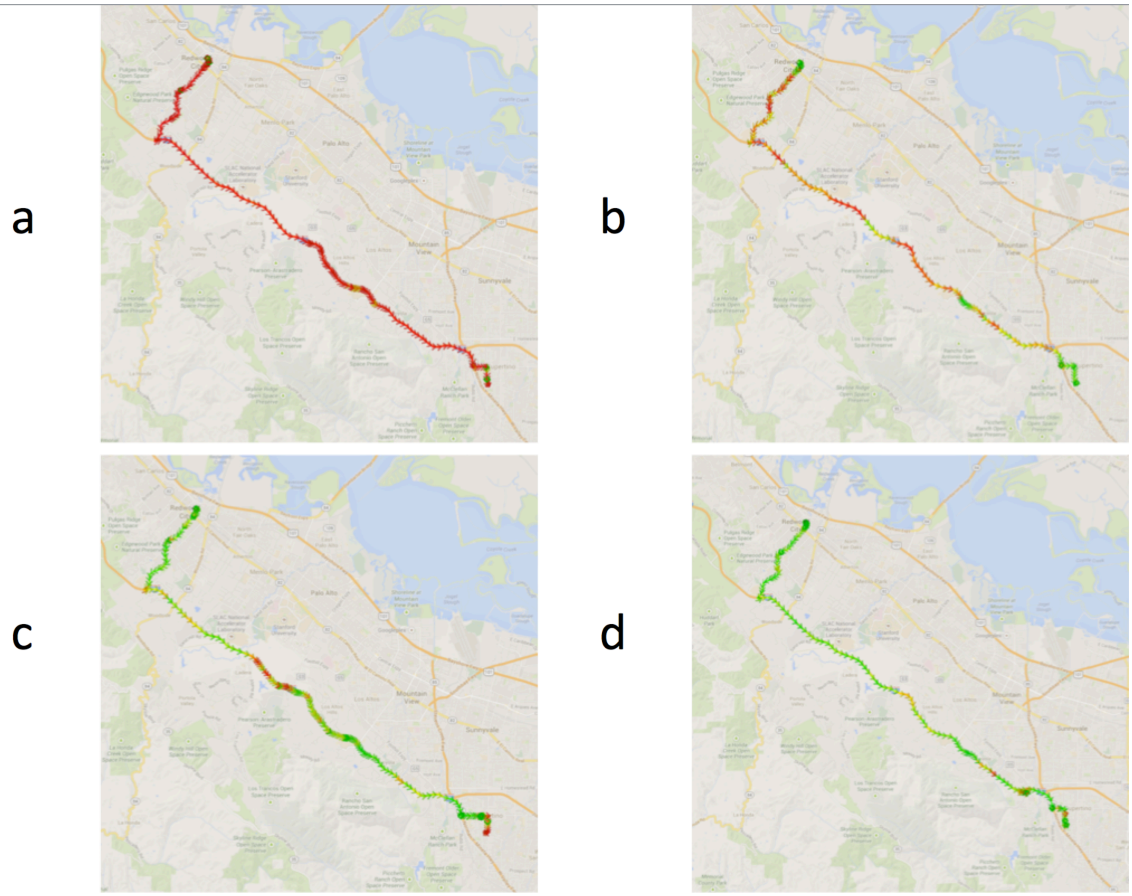

Fig. S2. (a-d) a driver's commute to work each morning over four separate days. Red areas correspond to unusual behavior, like the first day of the commute, while green areas correspond to learned paths in the world.

Each image in Fig. S2 shows the same commute path on a separate day. The paths are color coded red for very likely to be anomalous ($\epsilon = 10^{-5}$), yellow for somewhat likely ($10^{-5} < \epsilon \leq 10^{-4}$), and green for not likely to be anomalous. The route becomes less anomalous (more green) as the HTM system sees the same sequence of locations multiple days. The lingering yellow and red areas in Fig. S2(c) and Fig. S2(d) correspond to periods in which the traffic speeds differed from the previous days, resulting in different sparse codes for the coordinates.

Fig. S3. (a) a brief deviation from a learned route, and (b) a learned route in which the driver reversed direction temporarily and then resumed the usual route.

The images in Fig. S3 show two scenarios where anomalies were detected in otherwise learned paths. The left image, Fig. S3(a), shows that the driver briefly exited the highway that they were on, which had not been done on the previous drives. Once the driver returns to the usual path, the system stops detecting reporting anomalous behavior. The right image, Fig. S3(b), shows a usual route on a drive in which the driver reversed direction at some point, drove for a couple minutes, reversed direction again, and then continued on the path as usual. Even though the driver never left the usual path, the system recognized that the direction and sequence of locations were unusual. Both of these examples illustrate the power of an unsupervised approach to anomaly detection combined with online learning.

This example demonstrates the ability of HTM systems to model arbitrary types of streaming data, providing opportunity to apply the anomaly detection across industries.

S3. HTM parameters

The HTM algorithms have a number of parameters but there exists a standard set that is used in most applications. The set of model parameters in Table S1 are fixed for all models created in NAB. The complete source code for generating these models is available on Github as part of the NAB and NuPIC open source projects at https://github.com/numenta.

In NAB, the models are fed two pieces of information: the date and time, and the metric value. NuPIC's time of day encoder and scalar encoders are used to encode the timestamp and metric values, respectively, using the parameters specified below. The spatial value tolerance parameter is used to detect momentary spatial anomalies that don't trigger high likelihood scores in very noisy data streams. The parameter specifies the fraction of the range of values seen so far that new values can fall outside before being automatically given the maximum anomaly score.

Table S1
The parameter values used in the HTM models for all data files in NAB.

Parameter Name	Value
Time of day encoder width	21
Time of day encoder radius	9.49
Numeric value encoder number of buckets	130
Number of columns	2048
Number of active columns per step	40
Spatial Pooler connection threshold	0.2
Spatial Pooler permanence increment	0.003
Spatial Pooler permanence decrement	0.0005
Number of cells per column	32
Dendritic segment activation threshold	13
Maximum number of segments per cell	128
Maximum number of synapses per segment	32
Maximum number of new synapses added at each step	20
Temporal Memory initial synaptic permanence	0.21
Temporal Memory permanence increment	0.1
Temporal Memory permanence decrement	0.1
Spatial value tolerance	0.05

S4. Extending HTM with multiple models

The paper has focused on anomaly detection using a single HTM model. Industrial and real-time environments often contain a large number of sensory data streams. Such environments cannot be efficiently processed with a single complex model, as the complexity of training and inference grows much faster than linearly with the size of the input dimensionality [2]. In such scenarios, it is common to decompose a large system into a set of smaller models each of which models a subset of the environment. Smaller models lead to easier training and faster performance, but might lose potentially useful correlations across models. For example, if multiple components on a complex machine are simultaneously behaving in an erratic manner, the overall monitoring system might want to declare an anomaly even if none of the components in isolation meet the threshold. In this section we address the issue of accumulating the results of multiple individual models into a single anomaly measure.

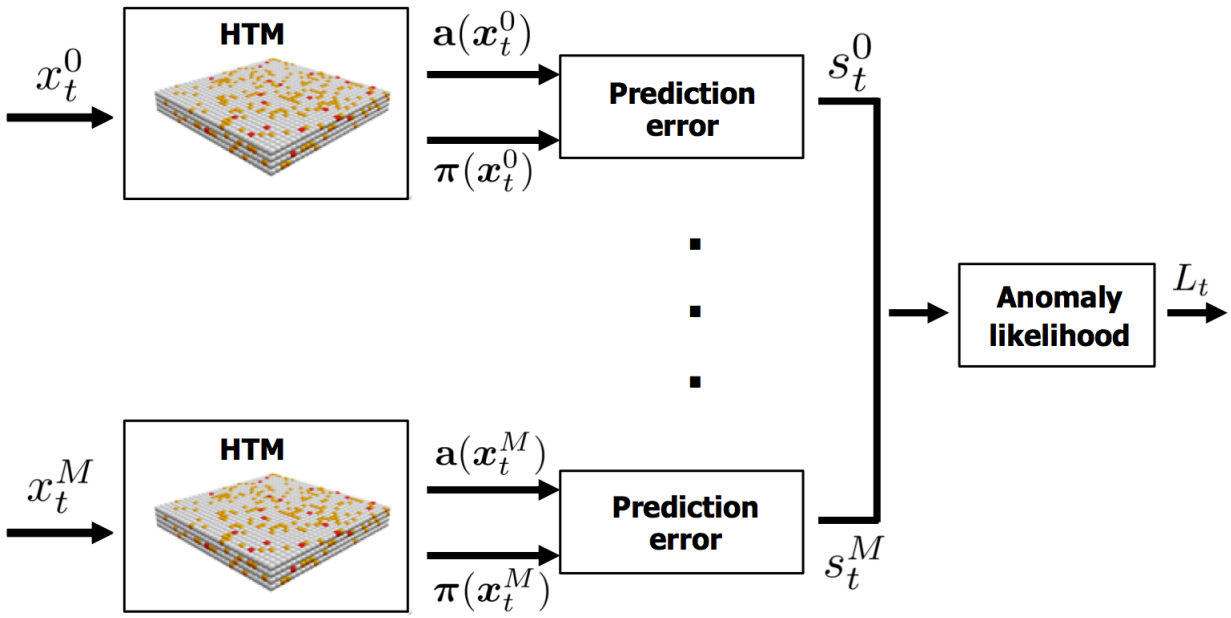

Fig. S4. Functional diagram illustrating a complex system with multiple independent HTM models.

We assume the inputs representing the system are broken up into M distinct models. Let $\boldsymbol{x}_t^m$ be the input at time t to the m'th model, and s_t^m be the prediction error associated with each model. We wish to model the distribution of prediction errors and compute the overall likelihood of an anomaly in the system (see Fig. S4).

One possible approach is to estimate the complete joint distribution $P(s_t^0, \ldots, s_t^{M-1})$ but it can be challenging and often intractable to model the complete joint distribution. We can simplify this by assuming the individual models are independent:

$$\mathrm{P}(\mathrm{s}_\mathrm{t}^0, \ldots, \mathrm{s}_\mathrm{t}^{\mathrm{M}-1}) = \prod_{\mathrm{i}=0}^{\mathrm{M}-1} \mathrm{P}(\mathrm{s}_\mathrm{t}^\mathrm{i}) \tag{S4}$$

Given this, a version of our anomaly likelihood can be computed as:

$$1 - \prod_{i=0}^{M-1} Q\left(\frac{\tilde{\mu_t^i} - \mu_t^i}{\sigma_t^i}\right) \tag{S5}$$

There is one flaw with the above methodology. In real-time dynamic scenarios, critical problems in one part of the system can cascade to other areas. Thus there are often random temporal delays built in, which can in turn lead to different temporal delays between anomaly scores in the various models [3]. A situation where multiple unusual events occur close to one another in different models is far more unlikely and unusual than a single event in a single model. It is precisely these situations that are valuable to detect and capture in a complex system.

Ideally we would be able to estimate a joint distribution of anomaly scores that go back in time, i.e. $P\left(s_{t-j}^{0}, s_{t-j}^{1}, \ldots, s_{t}^{M-2}, s_{t}^{M-1}\right)$. In theory this would capture all the dependencies, but this is even harder to estimate than the earlier joint probability. In situations where the system's topology is known, it is possible to create an explicit graphical model of dependencies, monitor expected behavior between pairs of nodes, and detect anomalies with respect to those expectations. This technique has been shown to enable very precise determination of anomalies in websites where specific calls between services are monitored [3]. However, in most applications this technique is also impractical. It may be difficult to model the various dependencies and it is often infeasible to instrument arbitrary systems to create this graph.

We would like to have a system that is fast to compute, makes relatively few assumptions, and is adaptive. We propose a simple general-purpose mechanism for handling multiple models by modifying Eq. (S5) to incorporate a smooth temporal window. A windowing mechanism allows the system to incorporate spikes in likelihood that are close in time but not exactly coincident. Let G be a Gaussian convolution kernel:

$$G(x;\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{S6}$$

We apply this convolution to each individual model to obtain a final anomaly likelihood score[1]:

$$L_t = 1 - \prod_{i=0}^{M-1} 2(G * Q)\left(\frac{\tilde{\mu_t^i} - \mu_t^i}{\sigma_t^i}\right) \tag{S7}$$

As before, we detect an anomaly if the combined anomaly likelihood is greater than a threshold $L_t \geq 1 - \epsilon$. This represents a principled yet pragmatic approach for detecting anomalies in complex real time streaming applications[2]. As before, L_t is an indirect measure, computed on top of the prediction error from each model. It reflects the underlying predictability of the models at a particular point in time and does not directly model the sensor measurements themselves.

S5. NAB implementation details for non-HTM algorithms

This section provides additional details for the algorithms that were implemented and tested on the NAB benchmark. The source code for all implementations is available in the NAB repository.

[1] Since this is a real-time system with no look-ahead, the kernel only applies to time $\leq t$. As such we use a half-normal distribution as the kernel, hence the additional factor of 2.

[2] Due to numerical precision issues with products of probabilities, in our implementation we follow common practice and use summation of log probabilities.

Multinomial Relative Entropy is a statistical technique for online anomaly detection [4]. It forms hypotheses on the fly using histograms of data over sliding windows. By using relative entropy and a chi-squared test, it measures agreement of the current hypothesis against multiple hypotheses from the past. This is a non-parametric strategy that handles some temporal anomalies as well as concept drift, without making strong assumptions regarding the underlying distribution of the data. We tuned the key parameters, window size and bin count, to obtain optimal NAB scores.

Twitter's AnomalyDetection is an open-source R package for real-time anomaly detection [5]. It uses a combination of statistical techniques to robustly detect outliers. A robust Generalized Extreme Studentized Deviate (ESD) test is combined with piecewise approximation to detect long-term trends; this is named Seasonal Hybrid ESD. The package includes two versions of the algorithm: AnomalyDetectionVec (ADVec) and AnomalyDetectionTS (ADTS). The former is intended to be more general and detects anomalies in data without timestamps but it required us to manually tune the periodicity. The latter exploits timestamps to incorporate periodicity into its detections, picking up both short-term (intra-day) and long-term (inter-day) anomalies. Unfortunately, ADTS was unable to calculate the necessary period parameters for some of the NAB data files and was excluded from the results. It is worth noting that for the data files ADTs successfully evaluated, it did worse than ADVec.

Skyline [24] is a real-time anomaly detection system originally developed by Etsy.com for monitoring its high traffic web site. The algorithm employs a mixture of simple detectors plus a voting scheme to output the final anomaly score. The detectors include deviation from moving average, deviation from a least squares estimate, and deviation from a histogram of past values. Skyline is well suited for analyzing streaming data. The internal estimates are continually adjusted and, due to the mixture of simple experts, it is relatively robust across a wide range of applications. Our results use the original parameters (additional tuning showed no significant improvement).

The Bayesian Online Changepoint algorithm attempts to detect abrupt variations in the generative parameters of a data sequence [6]. The algorithm models the previously seen data with the Student's T-distribution, and calculates the probability distribution of the length of the current run (sequence). The maximum of this distribution corresponds to the last changepoint. The technique is commonly used for anomaly detection as it is able to identify anomalies due to concept drift and shift [7]. Our implementation, a port of the author's original MATLAB code, computes an anomaly score by considering the change from a long run to be more anomalous than from a short run. Our results use the original parameters for initial estimates of the Gaussian and the hazard function (additional tuning showed no significant improvement).

EXPoSE [8] uses a kernel function to implicitly map input data onto a high dimensional feature space where similarity between pairs of data points can be estimated via inner product. Our implementation uses the decaying mechanism defined by the authors, resulting in the best performance. A decay factor weights data points based on recency

and helps the model adjust to concept drift. We tuned the decay factor and the dimensionality of feature space for projection.

We also report results of the three winners of the 2016 NAB competition, held in conjunction with IEEE WCCI. The three algorithms were CAD OSE, KNN CAD, and nab-comportex. CAD OSE [9] is a custom algorithm that stores the context for each data point and evaluates new data points with respect to that context. KNN CAD [10] is a probabilistic nearest neighbor algorithm for time-series data. nab-comportex [11] is a custom variant of HTM that uses a different set of parameters, a modified mechanism for encoding timestamps, and a modified anomaly score calculation. We have provided links to additional details including links to source code, papers and blog posts on the NAB repository.

Sliding threshold is a simple statistical technique often used in commercial anomaly detection systems. Our implementation models the data using a Gaussian distribution estimated using a sliding window. An anomaly score is computed as the distance of each data point from the mean relative to the standard deviation of the Gaussian. The size of the estimation window controls the detector's responsiveness to variations in the data stream. A larger window is more robust to noise and less reactive to abrupt changes. We tuned the window size to obtain optimal NAB scores.

It should be noted that the parameter spaces are large for most algorithms. Although an attempt was made to optimize each score, an exhaustive parameter search was not feasible.

S6. NAB file-by-file results

Table S2.
Detailed NAB scores shown for each data file across some of the algorithms with standard profile. The best and worst scores across each row of algorithms are in **bold** and *italics*, respectively. The "Counts" reflect the number of occurrences of each anomaly type in the given data file.

Source	File	Numenta HTM	CAD OSE	KNN CAD	Multinomial Relative Entropy	Twitter AdVec	Skyline	Sliding Threshold	Spatial Counts	Temporal Counts
Artificial without anomalies	art_daily_no_noise	-0.22	**0.00**	-0.11	**0.00**	*-0.88*	**0.00**	**0.00**	0	0
	art_daily_perfect_square_wave	**0.00**	-0.11	**0.00**	**0.00**	*-0.88*	**0.00**	**0.00**	0	0
	art_daily_small_noise	-0.22	-0.11	-0.44	-0.55	*-0.88*	**0.00**	**0.00**	0	0
	art_flatline	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	0	0
	art_noisy	**0.00**	**0.00**	*-0.22*	*-0.22*	**0.00**	**0.00**	**0.00**	0	0
Artificial with anomalies	art_daily_flatmiddle	0.50	**0.91**	0.62	0.89	0.53	*-1.00*	*-1.00*	1	0
	art_daily_jumpsdown	**0.75**	**0.75**	*-1.44*	0.60	0.36	-1.00	-1.00	1	0
	art_daily_jumpsup	**0.86**	**0.86**	0.64	**0.86**	0.76	0.84	*-1.00*	1	0
	art_daily_nojump	-1.28	-1.22	*-1.33*	*-1.33*	**0.34**	-1.00	-1.00	0	1
	art_increase_spike_density	**0.86**	**0.86**	0.53	*-1.00*	0.47	**0.86**	*-1.00*	0	1
	art_load_balancer_spikes	0.73	0.60	0.41	**0.78**	0.74	*-1.00*	-0.69	0	1

Online advertisement clicks	exchange-2_cpc_results	0.53	**0.83**	0.54	-0.53	*-1.00*	*-1.00*	*-1.00*	0	1
	exchange-2_cpm_results	**1.51**	-0.25	1.42	*-2.40*	-0.94	-2.00	-2.00	2	0
	exchange-3_cpc_results	**2.72**	2.49	2.36	0.50	0.72	*-3.00*	2.50	2	1
	exchange-3_cpm_results	0.75	**0.86**	0.20	0.75	**0.86**	*-1.00*	**0.86**	1	0
	exchange-4_cpc_results	0.51	**0.62**	0.25	0.22	**0.62**	*-3.00*	0.29	2	1
	exchange-4_cpm_results	1.49	**1.60**	1.11	*-2.24*	-0.37	1.28	1.27	3	1
AWS server metrics	ec2_cpu_utilization_24ae8d	1.28	**1.72**	-0.47	-0.25	-0.61	-0.47	*-1.27*	1	1
	ec2_cpu_utilization_53ea38	1.34	0.44	-1.16	**1.46**	-0.36	*-2.00*	-0.14	0	2
	ec2_cpu_utilization_5f5533	**1.72**	1.51	1.62	-0.12	-0.35	-0.35	*-2.00*	2	0
	ec2_cpu_utilization_77c1ca	**0.86**	0.18	0.60	0.23	-0.37	0.56	*-1.00*	0	1
	ec2_cpu_utilization_825cc2	0.85	0.81	0.86	0.92	0.63	**0.96**	*-1.00*	2	0
	ec2_cpu_utilization_ac20cd	**0.88**	0.86	0.77	*0.64*	0.80	0.86	0.86	1	0
	ec2_cpu_utilization_c6585a	-0.33	-0.22	-0.33	**0.00**	-0.88	**0.00**	*-1.32*	0	0
	ec2_cpu_utilization_fe7f93	**0.74**	0.53	0.42	0.29	-1.79	*-2.27*	0.64	1	2
	ec2_disk_write_bytes_1ef3de	0.59	0.53	**0.70**	0.43	-1.90	-1.63	*-4.19*	0	1
	ec2_disk_write_bytes_c0d644	**0.56**	**0.56**	0.34	0.28	*-1.76*	0.54	-1.10	2	1
	ec2_network_in_257a54	**0.86**	**0.86**	*0.53*	0.82	0.75	**0.86**	**0.86**	1	0
	ec2_network_in_5abac7	1.43	1.54	**1.61**	-2.44	0.95	0.88	*-3.52*	2	0
	elb_request_count_8c0756	1.61	1.72	**1.80**	*-0.14*	1.50	1.59	*-0.14*	1	1
	grok_asg_anomaly	**2.09**	0.63	0.64	-1.38	*-2.13*	0.68	-1.25	2	1
	iio_us-east-1_i-a2eb1cd9_NetworkIn	**-2.00**	-2.11	*-2.55*	-2.22	**-2.00**	-2.22	**-2.00**	1	1
	rds_cpu_utilization_cc0c53	-0.14	**1.72**	*-0.47*	-0.25	1.41	-0.14	-0.14	2	0
	rds_cpu_utilization_e47b3b	1.71	1.62	1.50	**1.72**	*-0.79*	**1.72**	-0.14	2	0
Miscellaneous known causes	ambient_temperature_system_failure	**-0.76**	-0.32	-0.77	-1.40	*-2.00*	*-2.00*	*-2.00*	2	0
	cpu_utilization_asg_misconfiguration	-0.26	-0.14	-1.11	**0.42**	-0.02	*-1.55*	-1.11	1	1
	ec2_request_latency_system_failure	1.71	1.99	1.93	**2.04**	2.02	0.09	*-0.07*	3	0
	machine_temperature_system_failure	**1.36**	-0.16	-2.86	-0.92	-0.24	-2.13	*-4.00*	2	2
	nyc_taxi	2.44	-1.32	0.27	**3.83**	*-5.00*	*-5.00*	*-5.00*	2	3
	rogue_agent_key_hold	-1.11	-0.96	**0.30**	-1.52	-2.22	-2.00	*-2.44*	0	2
	rogue_agent_key_updown	-1.30	-2.22	**0.43**	*-2.49*	-1.40	*-2.56*	-1.40	0	2
Freeway traffic	occupancy_6005	**0.86**	0.75	0.31	-0.37	0.75	0.76	*-1.00*	1	0
	occupancy_t4013	1.63	**1.74**	*-0.42*	1.63	1.73	**1.74**	1.73	2	0
	speed_6005	0.51	0.73	0.17	0.73	**0.84**	**0.84**	*-1.00*	1	0
	speed_7578	3.20	3.36	1.25	3.35	-0.34	**3.39**	*-4.00*	3	1
	speed_t4013	**1.98**	1.86	-0.46	1.73	1.75	1.86	*-2.00*	2	0
	TravelTime_387	0.37	**2.22**	0.54	0.18	-1.58	-0.61	*-1.80*	1	2
	TravelTime_451	0.55	**0.86**	*-1.44*	0.75	-1.00	-1.00	-1.00	0	1
Tweets volume	Twitter_volume_AAPL	1.26	1.70	**2.48**	*-0.69*	0.17	0.41	1.51	3	1
	Twitter_volume_AMZN	*-0.74*	-0.39	0.49	-0.15	**1.67**	0.43	1.56	2	2
	Twitter_volume_CRM	0.72	**2.58**	1.36	*-1.25*	1.78	1.81	0.76	2	1

	Twitter_volume_CVS	2.31	**2.71**	1.59	2.46	1.65	1.85	*0.77*	2	2
	Twitter_volume_FB	1.61	**1.72**	0.01	1.39	-0.15	*-1.46*	0.12	2	0
	Twitter_volume_GOOG	-0.26	0.05	**0.88**	0.56	0.29	*-2.07*	-0.15	1	3
	Twitter_volume_IBM	-0.10	**1.25**	0.38	1.17	0.30	*-4.39*	-0.80	2	0
	Twitter_volume_KO	**2.46**	0.72	1.03	0.61	0.71	*-2.05*	-0.91	1	2
	Twitter_volume_PFE	3.02	**3.57**	1.92	0.91	1.70	1.49	*0.27*	2	2
	Twitter_volume_UPS	1.61	**2.25**	-0.72	1.43	-1.78	*-9.63*	-2.21	4	1
	Totals	46.63	46.17	18.55	10.77	-6.82	-33.2	-44.75	74	46

Supplementary References

[1] S. Purdy, Encoding Data for HTM Systems, arXiv. (2016) 1602.05925.

[2] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.

[3] M. Kim, R. Sumbaly, S. Shah, Root cause detection in a service-oriented architecture, ACM Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS '13). (2013) 93. doi:10.1145/2465529.2465753.

[4] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, 12th IFIP/IEEE Int. Symp. Integr. Netw. Manag. (IM 2011) Work. (2011) 385–392. doi:10.1109/INM.2011.5990537.

[5] A. Kejariwal, Twitter Engineering: Introducing practical and robust anomaly detection in a time series [Online blog], (2015). http://bit.ly/1xBbX0Z.

[6] R.P. Adams, D.J.C. Mackay, Bayesian Online Changepoint Detection, arXiv.org. (2007) 7. doi:arXiv:0710.3742v1.

[7] J. Gama, Knowledge discovery from data streams, Chapman and Hall/CRC, Boca Raton, Florida, Florida, 2010.

[8] M. Schneider, W. Ertel, F. Ramos, Expected Similarity Estimation for Large-Scale Batch and Streaming Anomaly Detection, (2016). http://arxiv.org/abs/1601.06602.

[9] M. Smirnov, Contextual Anomaly Detector, (2016). https://github.com/smirmik/CAD.

[10] E. Burnaev, V. Ishimtsev, Conformalized density- and distance-based anomaly detection in time-series data, (2016). http://arxiv.org/abs/1608.04585.

[11] F. Andrews, nab-comportex, (2016). https://github.com/floybix/nab-comportex.