Neurocomputing 262 (2017) 134–147

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Unsupervised real-time anomaly detection for streaming data

Subutai Ahmad[a,*], Alexander Lavin[a], Scott Purdy[a], Zuha Agha[a,b]

[a] Numenta, Redwood City, CA, USA
[b] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

ARTICLE INFO

Article history:
Received 9 August 2016
Revised 19 April 2017
Accepted 22 April 2017
Available online 2 June 2017

Keywords:
Anomaly detection
Hierarchical Temporal Memory
Streaming data
Unsupervised learning
Concept drift
Benchmark dataset

ABSTRACT

We are seeing an enormous increase in the availability of streaming, time-series data. Largely driven by the rise of connected real-time data sources, this data presents technical challenges and opportunities. One fundamental capability for streaming analytics is to model each stream in an unsupervised fashion and detect unusual, anomalous behaviors in real-time. Early anomaly detection is valuable, yet it can be difficult to execute reliably in practice. Application constraints require systems to process data in real-time, not batches. Streaming data inherently exhibits concept drift, favoring algorithms that learn continuously. Furthermore, the massive number of independent streams in practice requires that anomaly detectors be fully automated. In this paper we propose a novel anomaly detection algorithm that meets these constraints. The technique is based on an online sequence memory algorithm called Hierarchical Temporal Memory (HTM). We also present results using the Numenta Anomaly Benchmark (NAB), a benchmark containing real-world data streams with labeled anomalies. The benchmark, the first of its kind, provides a controlled open-source environment for testing anomaly detection algorithms on streaming data. We present results and analysis for a wide range of algorithms on this benchmark, and discuss future challenges for the emerging field of streaming analytics.

1. Introduction

With sensors pervading our everyday lives, we are seeing an exponential increase in the availability of streaming, time-series data. Largely driven by the rise of the Internet of Things (IoT) and connected real-time data sources, we now have an enormous number of applications with sensors that produce important data that changes over time. Analyzing these streams effectively can provide valuable insights for any use case and application.

The detection of anomalies in real-time streaming data has practical and significant applications across many industries. Use cases such as preventative maintenance, fraud prevention, fault detection, and monitoring can be found throughout numerous industries such as finance, IT, security, medical, energy, e-commerce, agriculture, and social media. Detecting anomalies can give actionable information in critical scenarios, but reliable solutions do not yet exist. To this end, we propose a novel and robust solution to tackle the challenges presented by real-time anomaly detection.

Consistent with [1], we define an *anomaly* as a point in time where the behavior of the system is unusual and significantly different from previous, normal behavior. An anomaly may signify a negative change in the system, like a fluctuation in the turbine rotation frequency of a jet engine, possibly indicating an imminent failure. An anomaly can also be positive, like an abnormally high number of web clicks on a new product page, implying stronger than normal demand. Either way, anomalies in data identify abnormal behavior with potentially useful information. Anomalies can be *spatial*, where an individual data instance can be considered anomalous with respect to the rest of data, independent of where it occurs in the data stream, like the first and third anomalous spikes in Fig. 1. An anomaly can also be *temporal*, or *contextual*, if the temporal sequence of data is relevant; i.e., a data instance is anomalous only in a specific temporal context, but not otherwise. Temporal anomalies, such as the middle anomaly of Fig. 1, are often subtle and hard to detect in real data streams. Detecting temporal anomalies in practical applications is valuable as they can serve as an early warning for problems with the underlying system.

1.1. Streaming applications

Streaming applications impose unique constraints and challenges for machine learning models. These applications involve analyzing a continuous sequence of data occurring in real-time. In contrast to batch processing, the full dataset is not available. The

* Corresponding author.
E-mail address: sahmad@numenta.com (S. Ahmad).

http://dx.doi.org/10.1016/j.neucom.2017.04.070

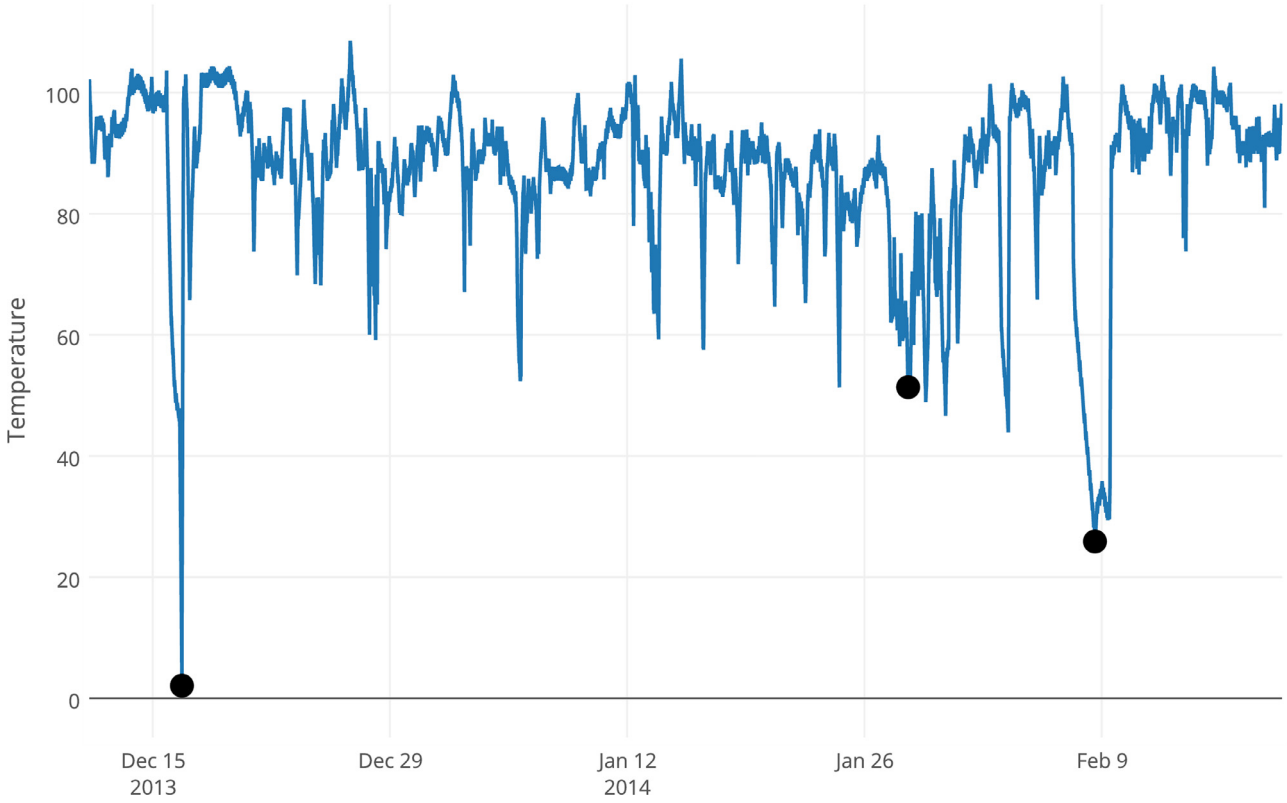

Fig. 1. The figure shows real-world temperature sensor data from an internal component of a large industrial machine. Anomalies are labeled with circles. The first anomaly was a planned shutdown. The third anomaly was a catastrophic system failure. The second anomaly, a subtle but observable change in the behavior, indicated the actual onset of the problem that led to the eventual system failure. The anomalies were hand-labeled by an engineer working on the machine. This file is included in the Numenta Anomaly Benchmark corpus [2].

system observes each data record in sequential order as they arrive and any processing or learning must be done in an online fashion. Let the vector $\boldsymbol{x}_t$ represent the state of a real-time system at time t. The model receives a continuous stream of inputs:

$$\ldots, \boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \ldots$$

Consider for example, the task of monitoring a datacenter. Components of $\boldsymbol{x}_t$ might include CPU usage for various servers, bandwidth measurements, latency of servicing requests, etc. At each point in time t we would like to determine whether the behavior of the system is unusual. The determination must be made in real-time, before time $t+1$. That is, before seeing the next input ($\boldsymbol{x}_{t+1}$), the algorithm must consider the current and previous states to decide whether the system behavior is anomalous, as well as perform any model updates and retraining. Unlike batch processing, data is not split into train/test sets, and algorithms cannot look ahead.

Practical applications impose additional constraints on the problem. Typically, the sensor streams are large in number and at high velocity, leaving little opportunity for human, let alone expert, intervention; manual parameter tweaking and data labeling are not viable. Thus, operating in an unsupervised, automated fashion is often a necessity.

In many scenarios the statistics of the system can change over time, a problem known as *concept drift* [3,4]. Consider again the example of a production datacenter. Software upgrades and configuration changes can occur at any time and may alter the behavior of the system (Fig. 2). In such cases models must adapt to a new definition of "normal" in an unsupervised, automated fashion.

In streaming applications early detection of anomalies is valuable in almost any use case. Consider a system that continuously monitors the health of a cardiac patient's heart. An anomaly in the data stream could be a precursor to a heart attack. Detecting such an anomaly minutes in advance is far better than detecting it a few seconds ahead, or detecting it after the fact. Detection of anomalies often gives critical information, and we want this information early enough that it's actionable, possibly preventing system failure. There is a tradeoff between early detections and false positives, as an algorithm that makes frequent inaccurate detections is likely to be ignored.

Given the above requirements, we define the ideal characteristics of a real-world anomaly detection algorithm as follows:

1. Predictions must be made online; i.e., the algorithm must identify state $\boldsymbol{x}_t$ as normal or anomalous before receiving the subsequent $\boldsymbol{x}_{t+1}$.
2. The algorithm must learn continuously without a requirement to store the entire stream.
3. The algorithm must run in an unsupervised, automated fashion—i.e., without data labels or manual parameter tweaking.
4. Algorithms must adapt to dynamic environments and concept drift, as the underlying statistics of the data stream is often non-stationary.
5. Algorithms should make anomaly detections as early as possible.
6. Algorithms should minimize false positives and false negatives (this is true for batch scenarios as well).

Taken together, the above requirements suggest that anomaly detection for streaming applications is a fundamentally different problem than static batch anomaly detection. As discussed further below, the majority of existing anomaly detection algorithms

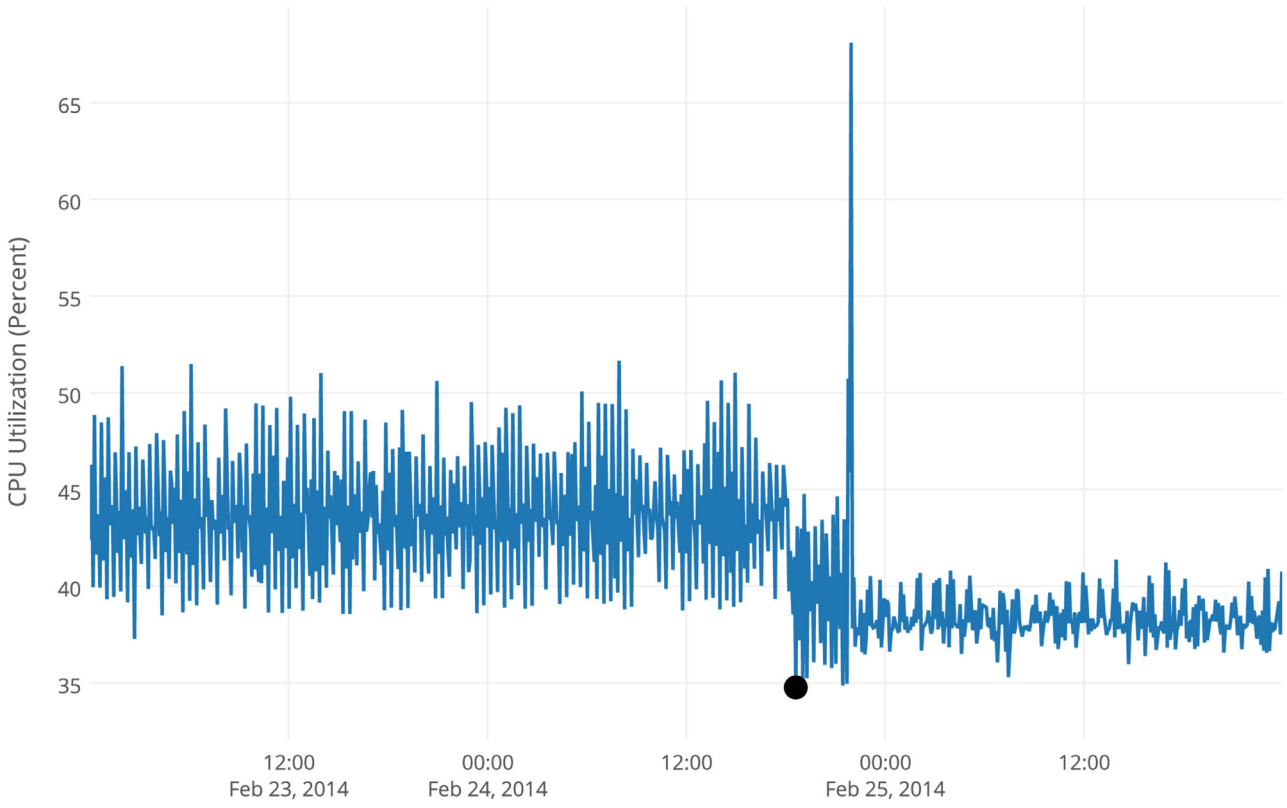

Fig. 2. CPU utilization (percent) for an Amazon EC2 instance (data from the Numenta Anomaly Benchmark [2]). A modification to the software running on the machine caused the CPU usage to change. The initial anomaly represents a changepoint, and the new system behavior that follows is an example of concept drift. Continuous learning is essential for performing anomaly detection on streaming data like this.

(even those designed for time-series data) are not applicable to streaming applications.

1.2. Related work

Anomaly detection in time-series is a heavily studied area of data science and machine learning, dating back to [5]. Many anomaly detection approaches exist, both supervised (e.g. support vector machines and decision trees [6]) and unsupervised (e.g. clustering), yet the vast majority of anomaly detection methods are for processing data in batches, and unsuitable for real-time streaming applications. Examples from industry include Netflix's robust principle component analysis (RPCA) method [7] and Yahoo's EGADS [8] both of which require analyzing the full dataset. Likewise, Symbolic Aggregate Approximation (SAX) [9] involves decomposing the full time series to generate symbols prior to anomaly detection. Other recent techniques include [10,11]. Although these techniques may work well in certain situations, they are traditional batch methods, and the focus of this paper is on methods for online anomaly detection. For reviews of anomaly detection in general we recommend [1,6,12,13]. For prior work on data stream mining and concept drift in general see [3,14–17].

Some anomaly detection algorithms are partially online. They either have an initial phase of offline learning, or rely on look-ahead to flag previously-seen anomalous data. Most clustering-based approaches fall under the umbrella of such algorithms. Some examples include Distributed Matching-based Grouping Algorithm (DMGA) [18], Online Novelty and Drift Detection Algorithm (OLINDDA) [19], and MultI-class learNing Algorithm for data Streams (MINAS) [20]. Another example is self-adaptive and dynamic k-means [21] that uses training data to learn weights prior to anomaly detection. Kernel-based recursive least squares (KRLS) proposed in [22] also violates the principle of no look-ahead as it resolves temporarily flagged data instances a few time steps later to decide if they were anomalous. However, some kernel methods, such as EXPoSE [23], adhere to our criteria of real-time anomaly detection (see evaluation section below).

For streaming anomaly detection, the majority of methods used in practice are statistical techniques that are computationally lightweight. These techniques include sliding thresholds, outlier tests such as extreme studentized deviate (ESD, also known as Grubbs') and k-sigma (e.g., [24,25]), changepoint detection [26], statistical hypotheses testing, and exponential smoothing such as Holt–Winters [27]. Typicality and eccentricity analysis [28,29] is an efficient technique that requires no user-defined parameters. Most of these techniques focus on spatial anomalies, limiting their usefulness in applications with temporal dependencies.

More advanced time-series modeling and forecasting models are capable of detecting temporal anomalies in complex scenarios. ARIMA is a general purpose technique for modeling temporal data with seasonality [30]. It is effective at detecting anomalies in data with regular daily or weekly patterns. Extensions of ARIMA enable the automatic determination of seasonality [31] for certain applications. A more recent example capable of handling temporal anomalies is the technique in [32] based on relative entropy.

Model-based approaches have been developed for specific use cases, but require explicit domain knowledge and are not generalizable. Domain-specific examples include anomaly detection in aircraft engine measurements [33], cloud datacenter temperatures [34], and ATM fraud detection [35]. Kalman filtering is a common technique, but the parameter tuning often requires domain knowledge and choosing specific residual error models [12,36–38]. Model-based approaches are often computationally efficient but their lack of generalizability limits their applicability to general streaming applications.

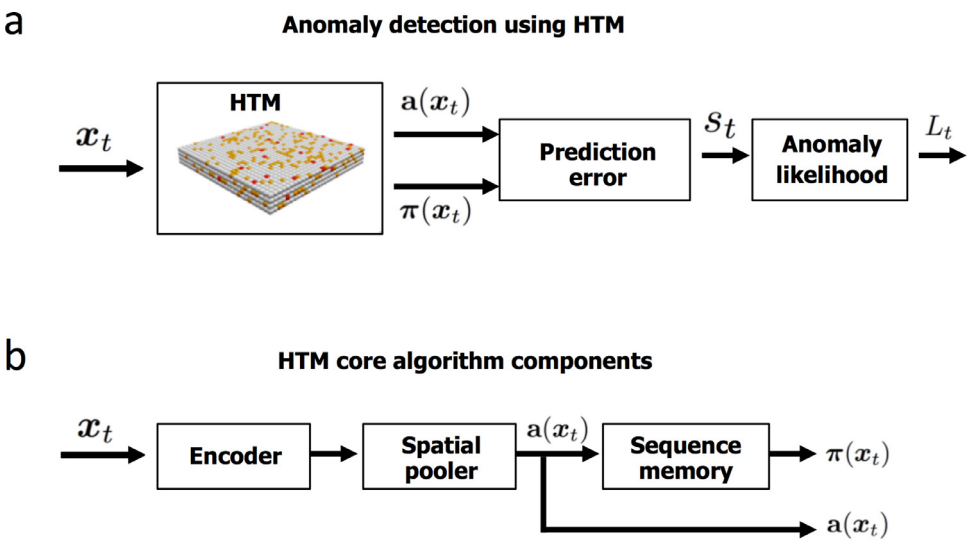

Fig. 3. (a) A block diagram outlining the primary functional steps used to create a complete anomaly detection system based on HTM. Our process takes the output of an HTM system and then performs two additional post-processing steps: computing the prediction error followed by computing an anomaly likelihood measure. (b) Breakdown of the core algorithm components within an HTM system.

There are a number of other restrictions that can make methods unsuitable for real-time streaming anomaly detection, such as computational constraints that impede scalability. An example is Lytics Anomalyzer [39], which runs in $O(n^2)$, limiting its usefulness in practice where streams are arbitrarily long. Dimensionality is another factor that can make some methods restrictive. For instance online variants of principle component analysis (PCA) such as osPCA [40] or window-based PCA [41] can only work with high-dimensional, multivariate data streams that can be projected onto a low dimensional space. Techniques that require data labels, such as supervised classification-based methods [42], are typically unsuitable for real-time anomaly detection and continuous learning.

Additional techniques for general purpose anomaly detection on streaming data include [9,43,44]. Twitter has an open-source method based on Seasonal Hybrid ESD [45]. Skyline is another popular open-source project, which uses an ensemble of statistical techniques for detecting anomalies in streaming data [24]. We include comparisons to both of these methods in our Results section.

1.3. Outline

The contributions of this paper are twofold: a novel anomaly detection technique built for real-time applications, and a comprehensive set of results on a benchmark designed for evaluating anomaly detection algorithms on streaming data. In Section 2 we show how to use Hierarchical Temporal Memory (HTM) networks [46–48] to robustly detect anomalies on a variety of data streams. The resulting system is efficient, extremely tolerant to noisy data, continuously adapts to changes in the statistics of the data, and detects subtle temporal anomalies while minimizing false positives. The HTM implementation and documentation are available as open-source.[1] In Section 3 we review the Numenta Anomaly Benchmark (NAB) [2], a rigorous benchmark dataset and scoring methodology we created for evaluating real-time anomaly detection algorithms. In Section 4 we present results comparing NAB on ten algorithms, many of which are commonly used in industry and academia. Section 5 concludes with a summary and directions for future work.

[1] HTM implementation is available at https://github.com/numenta/nupic.

2. Anomaly detection using HTM

Based on known properties of cortical neurons, Hierarchical Temporal Memory (HTM) is a theoretical framework for sequence learning in the cortex [46]. HTM implementations operate in real-time and have been shown to work well for prediction tasks [47,49]. HTM networks continuously learn and model the spatiotemporal characteristics of their inputs, but they do not directly model anomalies and do not output a usable anomaly score. In this section we describe our technique for applying HTM to anomaly detection.

Fig. 3(a) shows an overview of our process. At each point in time, the input data $\boldsymbol{x}_t$ is fed to a standard HTM network. We perform two additional computations on the output of the HTM. We first compute a measure of prediction error, s_t. Then, using a probabilistic model of s_t, we compute L_t, a likelihood that the system is in an anomalous state. A threshold on this likelihood determines whether an anomaly is detected. In the following subsections, we provide an overview of HTM systems and then describe our techniques for the additional steps of computing the prediction error and anomaly likelihood. Taken together, the algorithm fulfills the requirements for streaming applications outlined in Section 1.1.

2.1. Overview of HTM

Fig. 3(b) shows the core algorithm components and representations within a typical HTM system [49]. The current input, x_t, is fed to an encoder [50] and then a sparse spatial pooling process [51,52]. The resulting vector, $a(x_t)$, is a sparse binary vector representing the current input. The heart of the system is the sequence memory component. This component models temporal patterns in $a(x_t)$ and outputs a prediction in the form of another sparse vector $\pi(x_t)$. $\pi(x_t)$ is thus a prediction for $a(x_{t+1})$.

HTM sequence memory consists of a layer of HTM neurons organized into a set of columns (Fig. 4). The network accepts a stream of inputs encoded as sparse vectors. It models high-order sequences (sequences with long-term dependencies) using a composition of two separate sparse representations. The current input, $\boldsymbol{x}_t$ and the previous sequence context, $\ldots \boldsymbol{x}_{t-3}$, $\boldsymbol{x}_{t-2}$, $\boldsymbol{x}_{t-1}$, are simultaneously encoded using a dynamically updated sparse distributed representation. The network uses these representations to make predictions about the future in the form of a sparse vector.

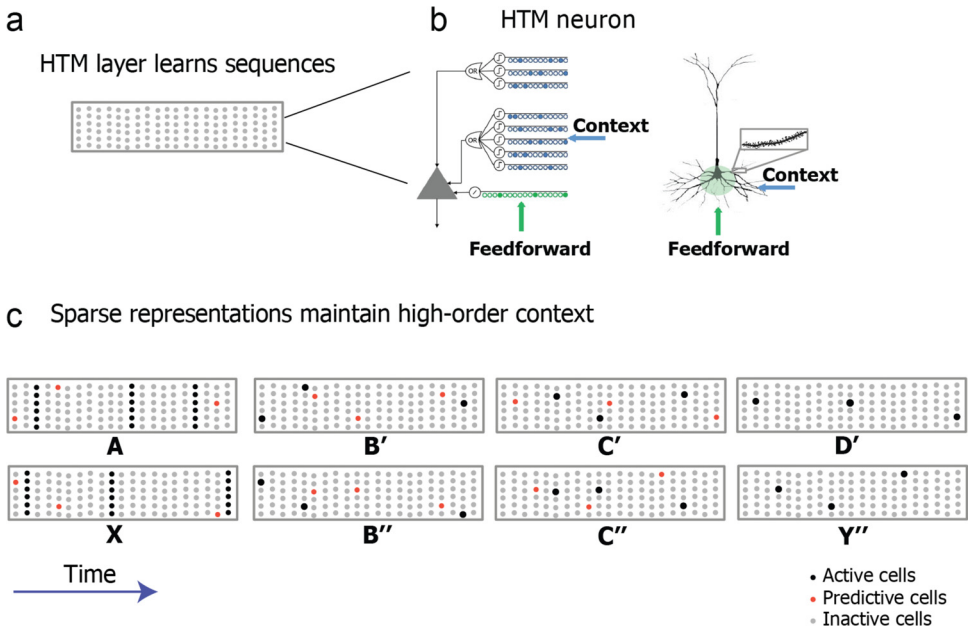

Fig. 4. The HTM sequence memory. (a) HTM sequence memory models one layer of cortex. The layer consists of a set of mini-columns, with each mini-column containing multiple neurons. (b) An HTM neuron (left) models the dendritic structure of pyramidal neurons in cortex (right). An HTM neuron models dendrites as an array of coincident detectors, each with a set of synapses. Context dendrites receive lateral input from other neurons within the layer. Each dendrite represents one transition in a sequence. Sufficient lateral activity on a context dendrite will cause the cell to enter a predicted state. (c) Representing high-order Markov sequences with shared subsequences (ABCD vs. XBCY). Each sequence element invokes a sparse set of cells within mini-columns. Cells that are predicted through lateral connections prevent other cells in the same column from firing through intra-column inhibition resulting in a highly sparse representation. As shown in the figure, such a representation can maintain past context. Because different cells respond to "C" in the two sequences (C' and C"), they can then invoke the correct prediction of either D or Y depending on the input from two time steps ago.

Fig. 4(c) shows how the sparse representations are used to represent temporal patterns and disambiguate sequences with long-term dependencies. When receiving the next input, the network uses the difference between predicted input and the actual input to update its synaptic connections. Learning happens at every time step but since the representations are highly sparse only a tiny percentage of the synapses are updated.

The details of the HTM learning algorithm and the properties of its representation are beyond the scope of this paper but are described in depth in [46,53]. In our implementation we use the standard HTM system [49] and a standard set of parameters (See Supplementary Section S3 for the complete list).

2.2. Computing the prediction error

Given the current input, $\boldsymbol{x}_t$, $\boldsymbol{a}(\boldsymbol{x}_t)$ is a sparse encoding of the current input, and $\boldsymbol{\pi}(\boldsymbol{x}_{t-1})$ is the sparse vector representing the HTM network's internal prediction of $\boldsymbol{a}(\boldsymbol{x}_t)$. The dimensionality of both vectors is equal to the number of columns in the HTM network (we use a standard value of 2048 for the number of columns in all our experiments). Let the *prediction error*, s_t, be a scalar value inversely proportional to the number of bits common between the actual and predicted binary vectors:

$$s_t = 1 - \frac{\boldsymbol{\pi}(\boldsymbol{x}_{t-1}) \cdot \boldsymbol{a}(\boldsymbol{x}_t)}{|\boldsymbol{a}(\boldsymbol{x}_t)|} \tag{1}$$

where $|\boldsymbol{a}(\boldsymbol{x}_t)|$ is the scalar norm, i.e. the total number of 1 bits in $\boldsymbol{a}(\boldsymbol{x}_t)$. In Eq. (1) the error s_t will be 0 if the current $\boldsymbol{a}(\boldsymbol{x}_t)$ perfectly matches the prediction, and 1 if the two binary vectors are orthogonal (i.e. they share no common 1 bits). s_t thus gives us an instantaneous measure of how well the underlying HTM model predicts the current input $\boldsymbol{x}_t$.

Changes to the underlying statistics are handled automatically due to the continuous learning nature of HTMs. If there is a shift in the behavior of the system, the prediction error will be high at the point of the shift, but will automatically degrade to zero as the model adapts to the "new normal". Fig. 5 shows an example stream and the behavior of the prediction error s_t. Shifts in the temporal characteristics of the system are handled in addition to spatial shifts in the underlying metric values.

An interesting aspect of this metric is that branching sequences are handled correctly. In HTMs, multiple predictions are represented in $\boldsymbol{\pi}(\boldsymbol{x}_t)$ as a binary union of each individual prediction. Similar to Bloom filters, as long as the vectors are sufficiently sparse and of sufficient dimensionality, a moderate number of predictions can be represented simultaneously with exponentially small chance of error [53,54]. The above error handles branching sequences gracefully in the following sense. If two completely different inputs are both possible and predicted, receiving either input will lead to a 0 error. Any other input will generate a positive error.

2.3. Computing anomaly likelihood

The prediction error described above represents an instantaneous measure of the predictability of the current input stream. As shown in Fig. 5, it works well for certain scenarios. In some applications however, the underlying system is inherently very noisy and unpredictable and instantaneous predictions are often incorrect. As an example, consider Fig. 6(a). This data shows the latency of a load balancer in serving HTTP requests on a production web site. Although the latency is generally low, it is not unusual to have occasional random jumps, leading to corresponding spikes in prediction error as shown in Fig. 6(b). The true anomaly is actually later in the stream, corresponding to a sustained increase in the frequency of high latency requests. Thresholding the prediction error directly would lead to many false positives.

To handle this class of scenarios, we introduce a second step. Rather than thresholding the prediction error s_t directly, we model

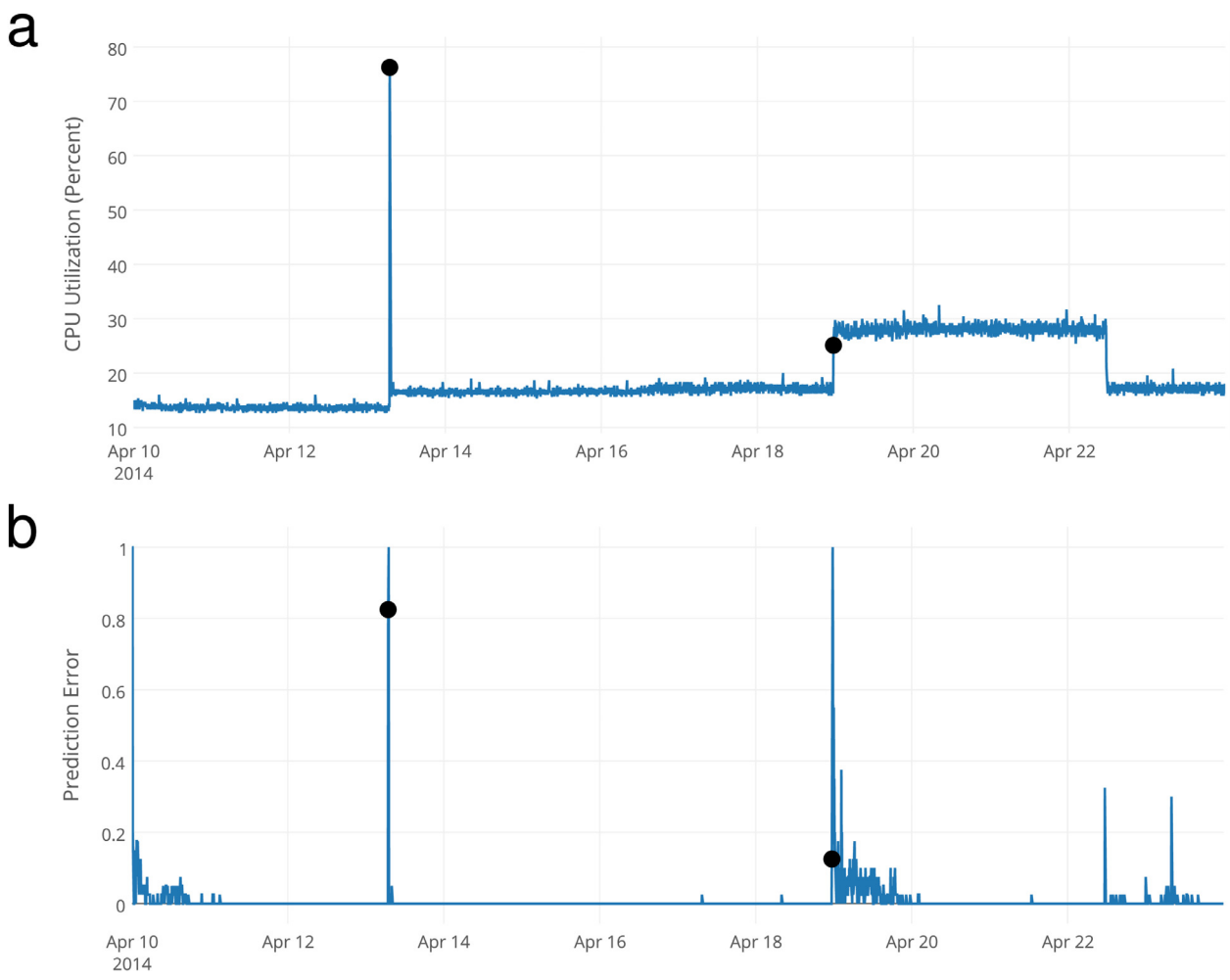

Fig. 5. An example stream along with prediction error. (a) This plot shows CPU usage on a database server over time. There are two unusual behaviors in this stream, the temporary spike up to 75% and the sustained shift up to 30% usage. B. This plot shows the prediction error while the HTM trains on this stream. Early during training the prediction error is high while the HTM model learns the data. There is a spike in prediction error corresponding to the temporary spike in CPU usage that quickly drops once usage goes back near normal. Finally there is an increase in prediction error corresponding to the sustained shift, which drops after the HTM has learned the new behavior.

the distribution of error values as an indirect metric, and use this distribution to check for the likelihood that the current state is anomalous. The *anomaly likelihood* is thus a probabilistic metric defining how anomalous the current state is based on the prediction history of the HTM model. To compute the anomaly likelihood we maintain a window of the last W error values. We model the distribution as a rolling normal distribution[2] where the sample mean, μ_t, and variance, σ_t^2, are continuously updated from previous error values as follows:

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W} \tag{2}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{W-1} \tag{3}$$

We then compute a recent short term average of prediction errors, and apply a threshold to the Gaussian tail probability (Q-function [55]) to decide whether or not to declare an anomaly.[3] We define the *anomaly likelihood* as the complement of the tail probability:

$$L_t = 1 - Q\left(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}\right) \tag{4}$$

where:

$$\tilde{\mu}_t = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{W'} \tag{5}$$

W' here is a window for a short term moving average, where $W' \ll W$, the duration for computing the distribution of prediction errors.[4] We threshold L_t based on a user-defined parameter ϵ to report an anomaly:

$$\text{anomaly detected}_t \equiv L_t \geq 1 - \epsilon \tag{6}$$

Since thresholding L_t involves thresholding a tail probability, there is an inherent upper limit on the number of alerts and a

[2] The distribution of prediction errors is not technically a normal distribution. We have also attempted to model the errors using a number of other distributions and distribution free bounds, such as Chebyshev's inequality. Modeling errors as a simple normal distribution worked significantly better than these other attempts. It is nevertheless still possible that modeling another distribution could improve results.

[3] The tail probability is the probability that a variable will be larger than x standard deviations above the mean.

[4] In practice we find that system performance is not sensitive to W as long as it is large enough to compute a reliable distribution. We use a generous value of $W = 8000$, and $W' = 10$.

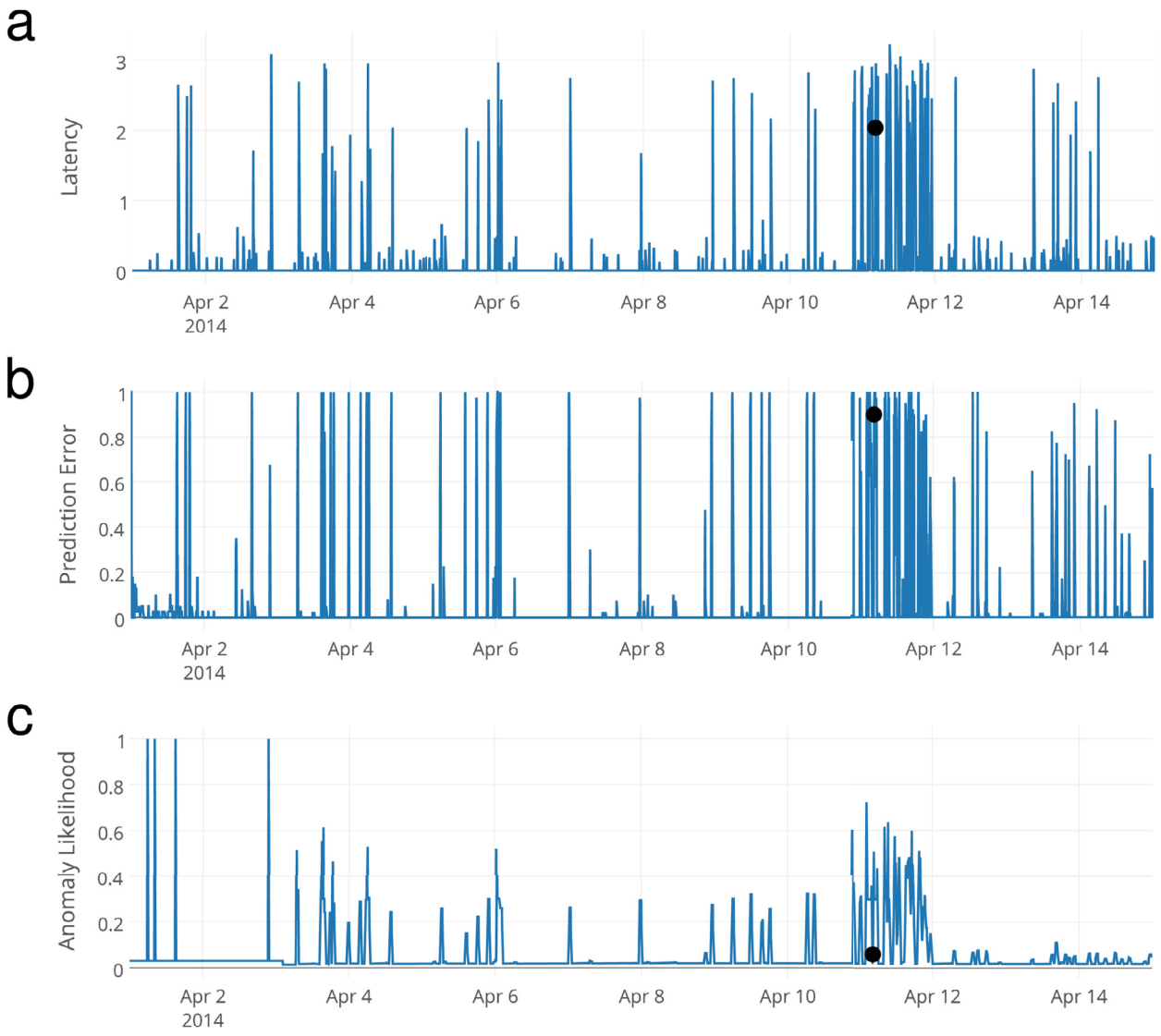

Fig. 6. A very noisy, unpredictable stream. (a) The data shows the latency (in seconds) of a load balancer on a production website. The anomaly (indicated by the dot) is an unusual sustained increase in latencies around April 10. (b) The prediction error from an HTM model on the latency values. The unpredictable nature of the latencies results in frequent spikes in the prediction error that cannot be distinguished from the true positives. The fact that the unpredictable metric values are spikes and the rest of the latencies are close to zero results in the coincidental similarity between the latencies and resulting prediction error. (c) A log-scale plot of the anomaly likelihood computed from the prediction error. Unlike the prediction error plot, there is a clear peak right around the real anomaly.

corresponding upper bound on the number of false positives. With ϵ very close to 0 it would be unlikely to get alerts with probability much higher than ϵ. In practice we have found that $\epsilon = 10^{-5}$ works well across a large range of domains and the user does not normally need to specify a domain-dependent threshold.

Fig. 6(c) shows an example of the anomaly likelihood, L_t, on noisy load balancer data. The figure demonstrates that the anomaly likelihood provides clearer peaks in extremely noisy scenarios compared to pure prediction error.

It is important to note that L_t is based on the distribution of prediction errors, not on the distribution of underlying metric values $\boldsymbol{x}_t$. As such, it is a measure of how well the model is able to predict, relative to the recent history. In clean predictable scenarios L_t behaves similarly to s_t. In these cases, the distribution of errors will have very small variance and will be centered near 0. Any spike in s_t will similarly lead to a corresponding spike in L_t. However, in scenarios with some inherent randomness or noise, the variance will be wider and the mean further from 0. A single spike in s_t will not lead to a significant increase in L_t but a series of spikes will. Importantly, a scenario that goes from wildly random to completely predictable will also trigger an anomaly.

2.4. Extensions

Modeling multiple streams simultaneously can enable the system to detect anomalies that cannot be detected from one stream alone. In Supplementary Section S4, we discuss this scenario and describe an extension for performing anomaly detection across multiple streams. We show how to combine independent models while accounting for temporal drift. This is particularly useful when there are many sensors and the combinations that enable detection are unknown.

Our algorithm is agnostic to the data type, as long as the data can be encoded as a sparse binary vector that captures the semantic characteristics of the data. We present an interesting extension with streaming geospatial data in Supplementary Section S2, demonstrating the applicability in diverse industries.

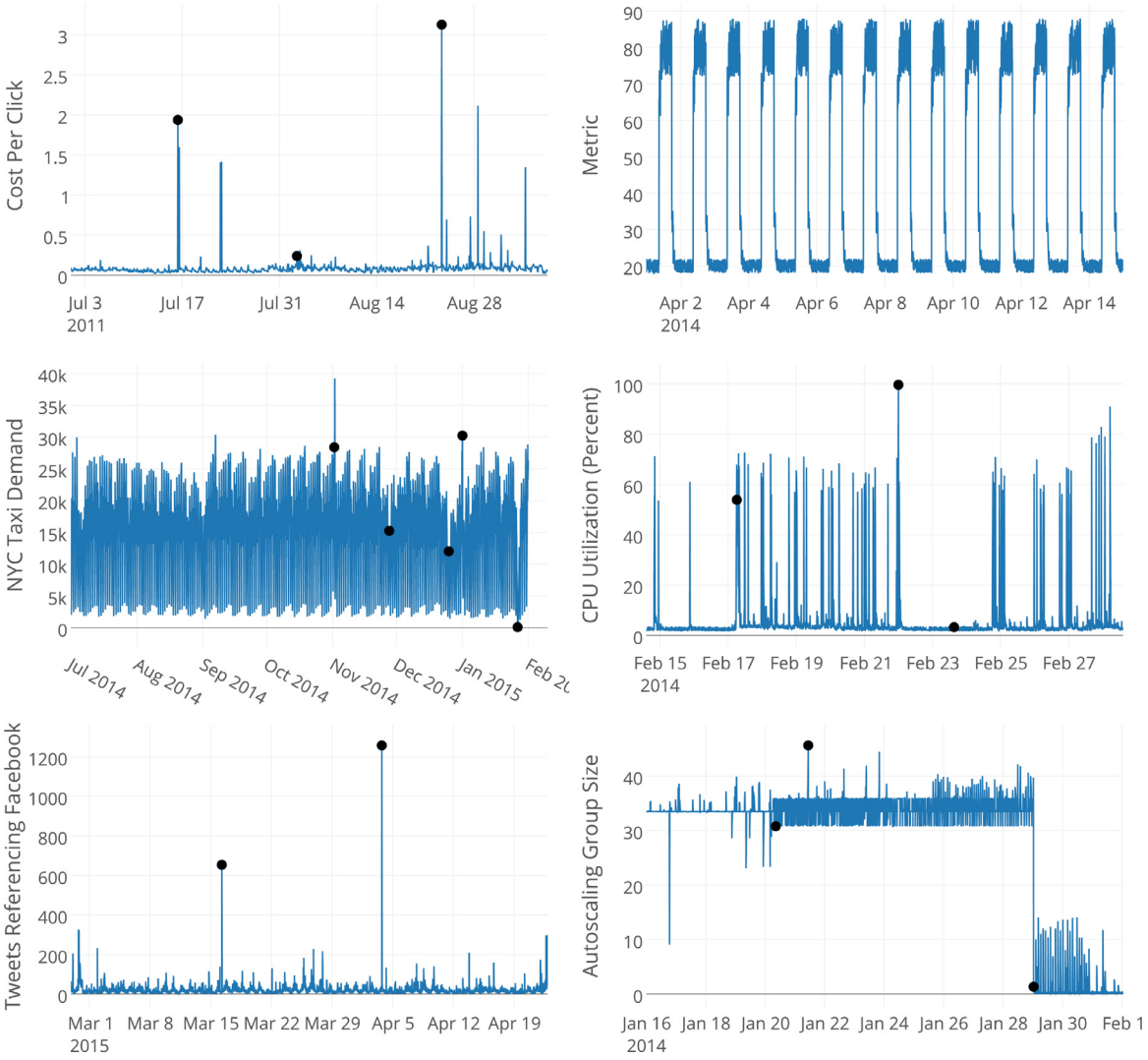

Fig. 7. Several data streams from the NAB corpus, showing a variety of data source and characteristics. From top left proceeding clockwise: click-through prices for online advertisements, an artificial stream with some noise but no anomalies, AWS Cloudwatch CPU utilization data, autoscaling group data for a server cluster, a stream of tweet volumes related to FB stock, and hourly demand for New York City taxis.

3. Evaluation of streaming anomaly detection algorithms

Numerous benchmarks exist for anomaly detection [1,56] but these benchmarks are generally designed for static datasets. Even benchmarks containing time-series data typically do not capture the requirements of real-time streaming applications. It is also difficult to find examples of real-world data that is labeled with anomalies. Yahoo released a dataset for anomaly detection in time-series data [8], but it is not available outside of academia and does not incorporate the requirements of streaming applications. As such we have created the Numenta Anomaly Benchmark (NAB) with the following goals:

1. Provide a dataset of labeled data streams from real-world streaming applications.
2. Provide a scoring methodology and set of constraints designed for streaming applications.
3. Provide a controlled open repository for researchers to evaluate and compare anomaly detection algorithms for streaming applications.

We briefly describe each of these below (full details can be found in [2]).

3.1. Benchmark dataset

The aim of the NAB dataset is to present algorithms with the challenges they will face in real-world scenarios, such as a mix of spatial and temporal anomalies, clean and noisy data, and data streams where the statistics evolve over time. The best way to do this is to provide data streams from real-world use cases, and from a variety of domains and applications. The data currently in the NAB corpus represents a variety of sources, ranging from server network utilization to temperature sensors on industrial machines to social media chatter.

NAB version 1.0 contains 58 data streams, each with 1000–22,000 records, for a total of 365,551 data points. Also included are some artificially-generated data files that test anomalous behaviors not yet represented in the corpus's real data, as well as several data files without any anomalies. All data files are labeled, either because we know the root cause for the anomalies from the provider of the data, or as a result of the well-defined NAB labeling procedure.[5] These labels define the ground truth anomalies used in the NAB scoring process. Fig. 1 is an example of noisy sensor data with spatial and temporal anomalies. Fig. 2 shows two related anomalies preceding a shift in the underlying statistics of the stream. Fig. 7 shows several data streams from the benchmark

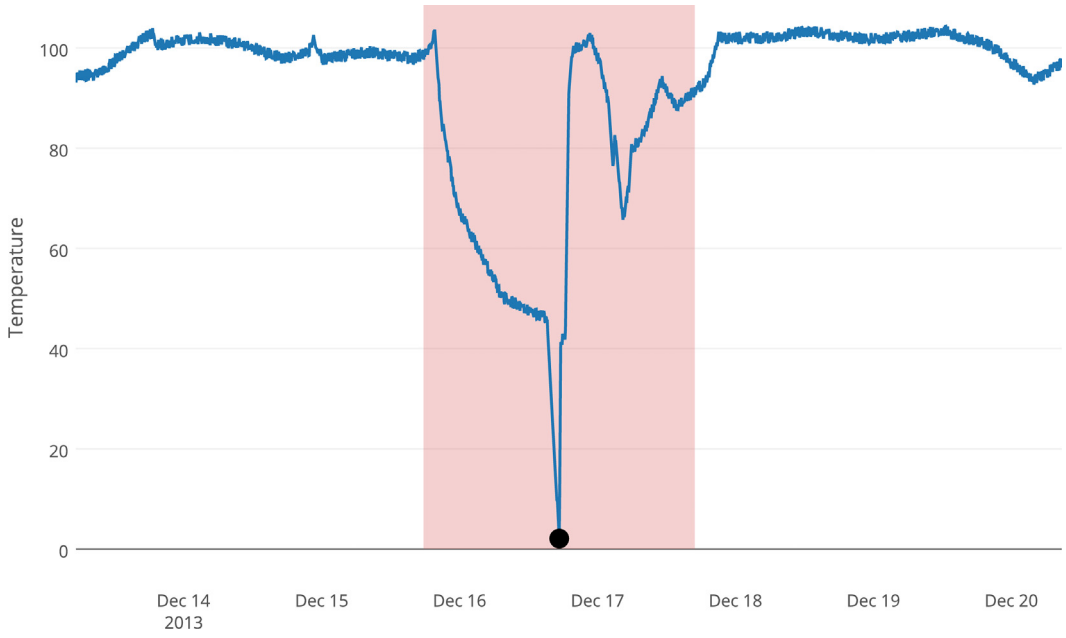

Fig. 8. A zoomed in view showing the anomaly window around the first anomaly in Fig. 1. The window size is set large enough to reward early detections of the anomaly, yet small enough such that poor detections count as false positives.

dataset, sourced from a variety of domains and exhibiting diverse characteristics such as temporal noise and short and long-term periodicities.

3.2. NAB scoring

The NAB scoring system formalizes a set of rules to determine the overall quality of streaming anomaly detection relative to the ideal, real-world anomaly detection algorithm that we defined earlier. There are three key aspects of scoring in NAB: anomaly windows, the scoring function, and application profiles. These are described below, and detailed discussion can be found in [2].

To incorporate the value of early detection into scoring, we define *anomaly windows* that label ranges of the data streams as anomalous, and a *scoring function* that uses these windows to reward early detections (and penalize later detections). When an algorithm is tested on NAB, the resulting detections must be scored. After an algorithm has processed a data file, the windows are used to identify true and false detections, and the scoring function is applied relative to each window to give value to the resulting true positives and false positives. Detections within a window correctly identify anomalous data and are true positives (TP), increasing the NAB score. If a detection occurs at the beginning of a window, it is given more value by the scoring function than a detection towards the end of the window. Multiple detections within a single window identify the same anomalous data, so we only use the earliest (most valuable) detection for the score contribution. Detections falling outside the windows are false positives, giving a negative contribution to the NAB score. The value of false positives is also calculated with the scoring function such that a false positive (FP) that occurs just after a window hurts the NAB score less than a FP that occurs far away from the window. Missing a window completely, or a false negative (FN), results in a negative contribution to the score. Refer to Supplementary Section S1 for details on scoring equations. All the scoring code and documentation is available in the repository.[5]

How large should the windows be? Large windows promote early detection of anomalies, but the tradeoff is that random or unreliable detections would be regularly reinforced. Using the underlying assumption that anomalous data is rare, the anomaly window length is defined to be 10%[6] the length of a data file, divided by the number of anomalies in the given file. This scheme provides a generous window to reward early detections and gives partial credit for detections slightly after the true anomaly, yet small enough such that poor detections are likely to be counted negatively (Fig. 8). Note that the streaming algorithms themselves have no information regarding the windows or the data length. Anomaly windows are only used as part of the benchmark and scoring system to gauge end performance.

NAB also includes a mechanism to evaluate algorithms on their bias towards false positives or false negatives. Depending on the application, a FN may be much more significant than a FP, such as detecting anomalies in ECG data, where a missed detection could be fatal. These scenarios are formalized in NAB by defining *application profiles* that vary the relative values of these metrics. For example, a datacenter application would be interested in the "Reward Low FP" profile, where false positives are weighted more heavily than in the other profiles.

The combination of anomaly windows, a smooth temporal scoring function, and application profiles allows researchers to evaluate online anomaly detector implementations against the requirements of the ideal detector. The NAB scoring system evaluates real-time performance, prefers earlier detection of anomalies, penalizes "spam" (i.e. FPs), and provides realistic costs for the standard classification evaluation metrics TP, FP, TN, and FN.

3.3. NAB is open-source

With the intent of fostering innovation in the field of anomaly detection, NAB is designed to be an accessible and reliable framework for all to use. Included with the open-source data and code is extensive documentation and examples on how to test algorithms.

[5] The NAB repository contains all the open-source data, code, and extensive documentation, including the labeling procedure and examples for running anomaly detection algorithms on NAB: https://github.com/numenta/NAB.

[6] We tested a range of window sizes (between 5% and 20%) and found that, partly due to the scaled scoring function, the end score was not sensitive to this percentage.

Table 1
NAB scoreboard showing results of each algorithm on v1.0 of NAB. Note the Random detector scores reflect the mean over a range of random seeds. HTM AL denotes the HTM algorithm using prediction error plus anomaly likelihood, as described in Section 2.2. HTM PE denotes the HTM algorithm using prediction error only. [+]Algorithms that were winners of the WCCI NAB competition.

Detector	Standard Profile	Reward Low FP	Reward Low FN
Perfect	100	100	100
HTM AL	70.1	63.1	74.3
CAD OSE[+]	69.9	67.0	73.2
nab-comportex[+]	64.6	58.8	69.6
KNN–CAD[+]	58.0	43.4	64.8
Relative Entropy	54.6	47.6	58.8
HTM PE	53.6	34.2	61.9
Twitter ADVec	47.1	33.6	53.5
Etsy Skyline	35.7	27.1	44.5
Sliding Threshold	30.7	12.1	38.3
Bayesian Changepoint	17.7	3.2	32.2
EXPoSE	16.4	3.2	26.9
Random	11	1.2	19.5
Null	0	0	0

Table 2
Comparison of properties for algorithms we implemented and tested on NAB (based on published information, excludes competition entries). Latency measures the time taken to process a single data point for anomaly detection. Latency time reported is an average over three runs on a single large data file from NAB, consisting of 22,695 data records. Timing was performed on an eight-core laptop with a 2.6 GHz Intel core i7 processor. NAB Score reflects the standard profile scores.

Detector	Latency (ms)	Spatial Anomaly	Temporal Anomaly	Concept Drift	Non Parametric	NAB Score
HTM	11.3	✔	✔	✔	✔	70.1
Relative Entropy	0.05	✔	✔	✔	✔	54.6
Twitter ADVec	3.0	✔	✔	✔	✗	47.1
Etsy Skyline	414.2	✔	✗	✗	✗	35.7
Sliding Threshold	0.4	✔	✗	✗	✗	30.7
Bayesian Changepoint	3.5	✔	✗	✔	✗	17.7
EXPoSE	2.6	✔	✔	✔	✔	16.4

The NAB repository contains source code for commonly used algorithms for online anomaly detection, as well as some algorithms submitted by the community.

4. Results & discussion

In this section we discuss NAB results and the comparative performance of a collection of real-time anomaly detection algorithms drawn from industry and academia. Our goals are to evaluate the performance of our HTM anomaly detection algorithm and, through a detailed discussion of the findings, to facilitate further research on unsupervised real-time anomaly detection algorithms.

4.1. Tested algorithms and parameter tuning

We considered a number of anomaly detection methods but the list was heavily filtered based on the criteria discussed in Sections 1.1 and 1.2. The algorithms evaluated include HTM, Twitter's Anomaly Detection, Etsy's Skyline, Multinomial Relative Entropy [32] EXPoSE [23], Bayesian Online Changepoint detection [57], and a simple sliding threshold. Some of these algorithms have open-source implementations and we implemented the rest based on their respective papers. We performed extensive parameter tuning for each algorithm; the resulting parameters are optimal to the best of our knowledge. Most of the algorithms also involve setting thresholds. The parameters were kept constant across all streams, and a single fixed threshold for each algorithm was used for the entire dataset, as required by NAB. Additional implementation details for these algorithms are included in Supplementary Section S5.

In addition, during the summer of 2016 we ran a NAB competition in collaboration with IEEE WCCI[7] to encourage additional algorithm testing. We include the result of the three competition winners below.

We use the HTM algorithm as described in Section 2. The core HTM algorithm by its nature is not highly sensitive to parameters. We used the architecture shown in Fig. 3 and the standard HTM parameter set (see Supplemental Section S3). The parameters specific to anomaly detection, ε, W, and W', were set to 10^{-5}, 8000, and 10, respectively. Timestamps and metric values in the NAB dataset were encoded using standard HTM datetime and scalar encoders [50]. In the results below, we show two variations. We show NAB results obtained by using prediction error only (i.e. thresholding s_t directly). We also show results obtained by using anomaly likelihood, as defined in Section 2.3.

For transparency and reproducibility, we have incorporated the source code and parameter settings for all of the above algorithms into the NAB repository.

4.2. Comparison of results

Table 1 summarizes the NAB scores for each algorithm across all application profiles including the three NAB competition winners. In addition to the algorithms described above, we also use three control detectors in NAB. A "null" detector runs through the dataset passively, making no detections, accumulating all false negatives. A "perfect" detector is an oracle that outputs detections that would maximize the NAB score; i.e., it outputs only true positives at the beginning of each window. The raw scores from these two detectors are used to scale the score for all other algorithms between 0 and 100. We also include a "random" detector that

[7] http://www.wcci2016.org.

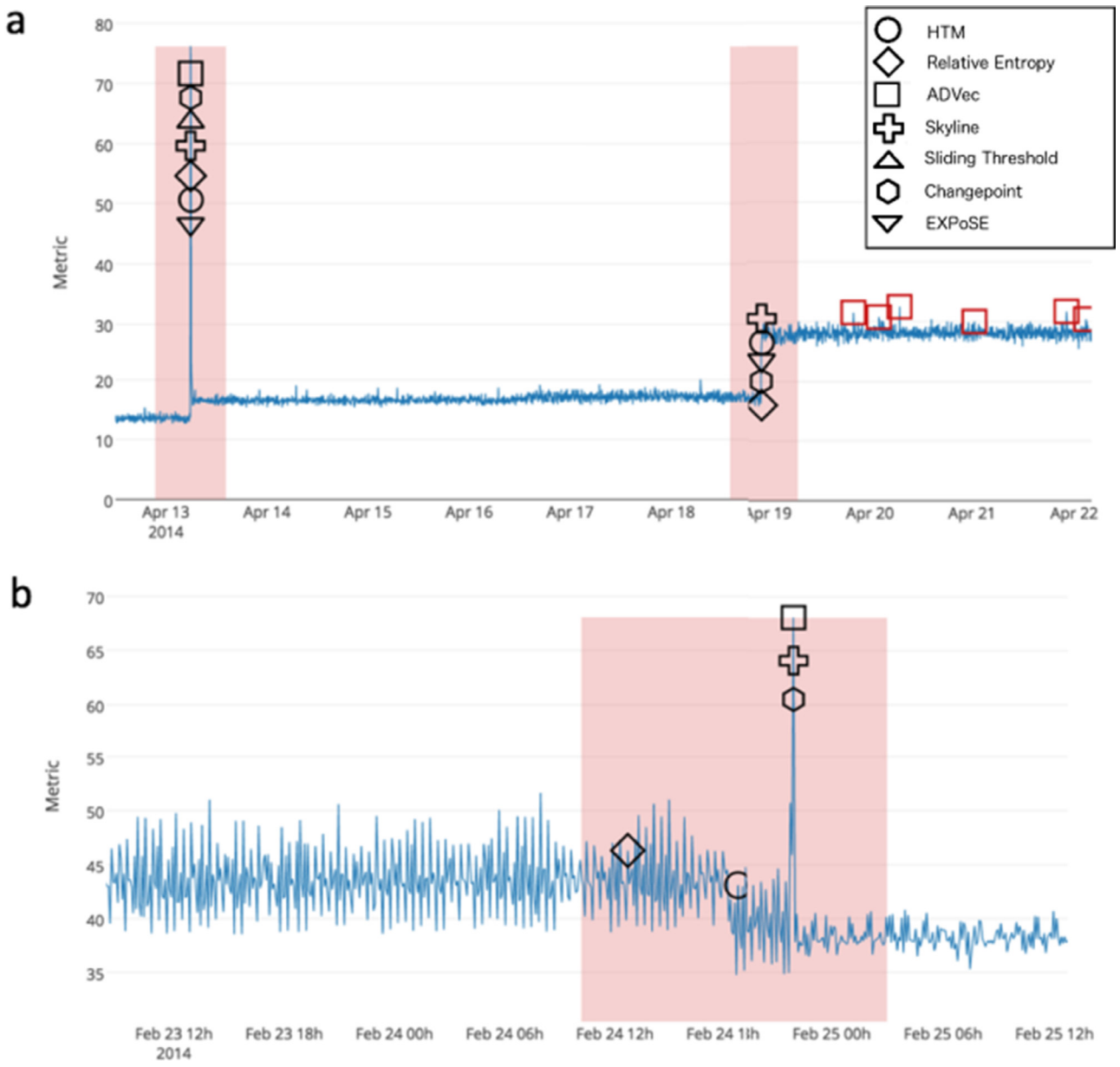

Fig. 9. These plots show detector results for two example NAB data streams. In both cases we show a subset of the full data stream. The plotted shapes correspond to the detections of seven different detectors: HTM, Multinomial Relative Entropy, Twitter ADVec, Skyline, Sliding Threshold, Bayesian Online Changepoint, and EXPoSE. Shapes that correspond to the same data point have been spaced vertically for clarity. For a given detector, true positive detections within each window (red shaded regions) are labeled in black. All false positive detections are colored red. (a) Detection results for a production server's CPU metric. The second anomaly shows a sustained shift that requires algorithms to adjust to a new normal behavior. (b) Results for the data stream shown in Fig. 2. Here we see a subtle temporal anomaly that preceded a large, obvious spike in the data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

outputs a random anomaly probability for each data instance, which is then thresholded across the dataset for a range of random seeds. The score from this detector offers some intuition for chance-level performance on NAB.

Overall we see that the HTM detector using anomaly likelihood gets the best score, followed by CAD-OSE, nab-comportex, KNN–CAD, and the Multinomial Relative Entropy detector. The HTM detector using only prediction error performs moderately well, but using the anomaly likelihood step significantly improves the scores. Most of the algorithms perform much better than a random detector. There is a wide range of scores but none are close to perfect suggesting there is still significant room for improvement.

Table 2 summarizes the various algorithmic properties of each of the algorithms we implemented. Each algorithm is categorized based on its ability to detect spatial and temporal anomalies, handle concept drift, and automatically update parameters; these characteristics are based on published information, which may or may not reflect the actual performance. We also list the measured latency of processing each data point. Several algorithms claim to have all of the listed properties but their actual anomaly detection performance on the benchmark varies significantly. In general there is a rough correlation between the number of properties satisfied and the NAB ranking (with the exception of EXPoSE, see discussion below).

Based on a more detailed analysis of the results we highlight four factors that were important for achieving good performance on NAB: concept drift, ability to detect temporal anomalies, assumptions regarding distribution, and assumptions regarding the number of data points. We discuss each of these qualitatively below as well as more detailed file-level comparisons.

Table 3
Comparison of average NAB scores for some algorithms across data from different kinds of sources provided by the benchmark. For each source, average scores are shown separately for data files that contain only spatial anomalies, only temporal anomalies and a mixture of spatial and temporal anomalies. The average score is given by $avg(a_s) = \frac{\sum_{i \in D(a_s)} score_i}{\sum_{i \in D(a_s)} counts_i}$ where $D(a_s)$ denotes data files from a given source s consisting of anomalies of the given type a, $score_i$ denotes the NAB score for the i^{th} file and $counts_i$ denotes the number of occurrences of the given anomaly type in i^{th} file. See Supplementary Section S6 for the scores and anomaly counts for each individual file. The best average scores across each row of algorithms are shown in **bold**. Note that scores are computed over standard profile with equal weights assigned to false positives and false negatives.

Source	Anomaly Type	Numenta HTM	CAD OSE	KNN CAD	Relative Entropy	Twitter AdVec	Skyline	Sliding Threshold
Artificial	Spatial only	0.70	**0.84**	−0.06	0.78	0.55	−0.39	−1.00
	Temporal only	0.11	0.08	−0.13	−0.52	**0.52**	−0.38	−0.90
	Spatial + Temporal	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Online advertisement clicks	Spatial only	**0.75**	0.21	0.54	−0.55	−0.03	−1.00	−0.38
	Temporal only	0.53	**0.83**	0.54	−0.53	−1.00	−1.00	−1.00
	Spatial + Temporal	**0.47**	**0.47**	0.37	−0.15	0.10	−0.47	0.41
AWS server metrics	Spatial only	0.61	**0.74**	0.54	0.11	0.28	0.40	−0.42
	Temporal only	**0.70**	0.29	0.03	0.53	−0.66	−0.77	−1.33
	Spatial + Temporal	**0.29**	0.20	0.01	−0.23	−0.45	−0.14	−0.34
Miscellaneous known causes	Spatial only	0.19	**0.33**	0.23	0.13	0.00	−0.38	−0.41
	Temporal only	−0.60	−0.79	**0.18**	−1.00	−0.91	−1.14	−0.96
	Spatial + Temporal	**0.32**	−0.15	−0.34	0.30	−0.48	−0.79	−0.92
Freeway traffic	Spatial only	0.83	0.85	−0.06	0.62	0.85	**0.87**	−0.38
	Temporal only	0.55	**0.86**	−1.44	0.75	−1.00	−1.00	−1.00
	Spatial + Temporal	0.51	**0.80**	0.26	0.50	−0.27	0.40	−0.83
Tweets volume	Spatial only	0.38	**0.74**	0.10	0.64	0.04	−1.46	−0.17
	Temporal only	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Spatial + Temporal	0.34	**0.43**	0.29	0.13	0.20	−0.25	0.05
Total Average		**0.40**	0.39	0.16	0.10	−0.03	−0.28	−0.36

The ability of each algorithm to learn continuously and handle concept drift is one of the contributing factors to obtaining a good NAB score. Fig. 9(a) demonstrates one example of this. This file shows CPU usage on a production server over time, and contains two anomalies. The first is a simple spike that is easily detectable by all algorithms. The second is a sustained shift in the usage, where the initial change in sequence is an anomalous changepoint, but the new behavior persists. Most algorithms detect the change and then adapt to the new normal. However, the Twitter ADVec algorithm fails to detect the changepoint and continues to generate anomalies for several days. The NAB corpus contains several similar examples, where an anomaly sets off a change in sequence that redefines the normal behavior. This is representative of one of the key challenges in streaming data. An inability to handle concept drift detections effectively results in a greater number of false positives, lowering the NAB score.

The ability to detect subtle temporal anomalies, while limiting false positives, is a second major factor in obtaining good NAB scores. In practical applications, one major benefit of detecting temporal patterns is that it often leads to the early detection of anomalies. Fig. 1 shows a representative example. The temporal anomaly in the middle of this figure precedes the actual failure by several days. In Fig. 2, the strong spike is preceded by a subtle temporal shift several hours earlier. Fig. 9(b) shows detection results on that data stream. The Twitter ADVec, Skyline, and Bayesian Online Changepoint algorithms easily detect the spike, but there are subtle changes in the data preceding the obvious anomaly. The Multinomial Relative Entropy and HTM detectors both flag anomalous data well before the large spike and thus obtain higher scores. It is challenging to detect such changes in noisy data without a large number of false positives. Both the HTM and Multinomial Relative Entropy perform well in this regard across the NAB dataset.

The third major factor concerns assumptions regarding the underlying distribution of data. A general lesson is that algorithms making fewer assumptions regarding distribution perform better. This is particularly important for streaming applications where algorithms must be unsupervised, automated, and applicable across a variety of domains. Techniques such as the sliding threshold and Bayesian Online Changepoint detectors make strong assumptions regarding the data and suffer as a result. Note that the Gaussian used in our anomaly likelihood technique is used to model the distribution of prediction errors, not the underlying metric data. As such it is a non-parametric technique with respect to the data.

Another interesting factor is demonstrated by the performance of EXPoSE. Theoretically EXPoSE possesses all the properties in Table 2, however it performs poorly on the benchmark. One of the reasons for this behavior is that EXPoSE has a dependence on the size of the dataset and is more suitable for large-scale datasets with high-dimensional features [58]. The technique computes an approximate mean kernel embedding and small or moderate data sets do not provide a sufficiently good proxy for this approximation. The average NAB data file contains 6300 records and is representative of real streaming applications. This issue highlights the need to output reliable anomalies relatively quickly.

4.3. Detailed NAB results

A breakdown of the algorithms performance on the benchmark is shown in Table 3. Results have been aggregated across data sources ranging from artificially generated streams to real streams from advertisement clicks, server metrics, traffic data and twitter volume. Data streams are characterized by spatial anomalies, temporal anomalies or a combination thereof. Grouping the streams by their anomaly types in Table 3 helps us inspect the characteristics of the algorithms identified earlier in Table 2.

Results show that both HTM and CAD-OSE yield the best overall aggregate scores on almost all data sources and anomaly types, with the exceptions of Twitter AdVec on artificial temporal streams and KNN–CAD on miscellaneous known causes. The difference between aggregate scores for HTM and CAD-OSE for the majority of the data streams is less than 0.20. For some stream types, HTM significantly outperforms CAD-OSE such as spatial advertisement streams, temporal server metric streams and spatial/temporal miscellaneous streams. In particular, the results show HTM performing well on server metrics and online advertisements data while CAD-OSE performs well on traffic and twitter streams.

In addition, the results in Table 3 also demonstrate that statistical techniques with assumptions on data distribution such as sliding threshold, Twitter AdVec and Skyline may be able to

capture spatial anomalies (e.g. Skyline on spatial traffic streams) but are not effective enough for capturing temporal anomalies. This is reflected by the negative scores for most temporal and spatial/temporal anomaly streams for these algorithms. This further reinforces the correlation between non-parametric techniques and detection of temporal anomalies.

5. Conclusion

With the increase in connected real-time sensors, the detection of anomalies in streaming data is becoming increasingly important. The use cases cut across a large number of industries. We believe anomaly detection represents one of the most significant near-term applications for machine learning in IoT.

In this paper we have discussed a set of requirements for unsupervised real-time anomaly detection on streaming data and proposed a novel anomaly detection algorithm for such applications. Based on HTM, the algorithm is capable of detecting spatial and temporal anomalies in predictable and noisy domains. The algorithm meets the requirements of real-time, continuous, online detection without look ahead and supervision.

We also reviewed NAB, an open benchmark for real-world streaming applications. We showed results of running a number of algorithms on this benchmark. We highlighted three key factors that impacted performance: concept drift, detection of temporal anomalies, and assumptions regarding distribution and size of data.

There are several areas for future work. The error analysis from NAB indicates that the errors across various algorithms (including HTM) are not always correlated. An ensemble-based approach might therefore provide a significant increase in accuracy. The current NAB benchmark is limited to data streams containing a single metric plus a timestamp. Adding real-world multivariate data streams labeled with anomalies, such as the data available in the DAMADICS dataset [59], would be a valuable addition.

Acknowledgments

We thank the anonymous reviewers for their feedback and helpful suggestions. We also thank Yuwei Cui, Jeff Hawkins, and Ian Danforth for many helpful comments, discussions, and suggestions.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.neucom.2017.04.070.

References

[1] V. Chandola, V. Mithal, V. Kumar, Comparative evaluation of anomaly detection techniques for sequence data, in: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 743–748, doi:10.1109/ICDM.2008.151.

[2] A. Lavin, S. Ahmad, Evaluating real-time anomaly detection algorithms – the Numenta anomaly benchmark, in: Proceedings of the 14th International Conference on Machine Learning Application, Miami, Florida, IEEE, 2015, doi:10.1109/ICMLA.2015.141.

[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (2014) 1–37, doi:10.1145/2523813.

[4] M. Pratama, J. Lu, E. Lughofer, G. Zhang, S. Anavatti, Scaffolding type-2 classifier for incremental learning under concept drifts, Neurocomputing 191 (2016) 304–329, doi:10.1016/j.neucom.2016.01.049.

[5] A.J. Fox, Outliers in time series, J. R. Stat. Soc. Ser. B. 34 (1972) 350–363.

[6] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. 41 (2009) 1–72, doi:10.1145/1541880.1541882.

[7] Wong J. Netflix Surus GitHub, Online Code Repos https://github.com/Netflix/Surus 2015

[8] N. Laptev, S. Amizadeh, I. Flint, Generic and Scalable Framework for Automated Time-series Anomaly Detection, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, 2015, pp. 1939–1947.

[9] E. Keogh, J. Lin, A. Fu, HOT SAX: Efficiently finding the most unusual time series subsequence, in: Proceedings of the IEEE International Conference on Data Mining, ICDM, 2005, pp. 226–233, doi:10.1109/ICDM.2005.79.

[10] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, Eur. Symp. Artif. Neural Netw. (2015) 22–24.

[11] H.N. Akouemo, R.J. Povinelli, Probabilistic anomaly detection in natural gas time series data, Int. J. Forecast. 32 (2015) 948–956, doi:10.1016/j.ijforecast.2015.06.001.

[12] J. Gama, Knowledge Discovery from Data Streams, Chapman and Hall/CRC, Boca Raton, Florida, 2010.

[13] M.A.F. Pimentel, D.A. Clifton, L. Clifton, L. Tarassenko, A review of novelty detection, Signal Process. 99 (2014) 215–249, doi:10.1016/j.sigpro.2013.12.026.

[14] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams, ACM SIGMOD Rec. 34 (2005) 18.

[15] M. Sayed-Mouchaweh, E. Lughofer, Learning in Non-Stationary Environments: Methods and Applications, Springer, New York, 2012.

[16] M. Pratama, J. Lu, E. Lughofer, G. Zhang, M.J. Er, Incremental learning of concept drift using evolving Type-2 recurrent fuzzy neural network, IEEE Trans. Fuzzy Syst (2016) 1, doi:10.1109/TFUZZ.2016.2599855.

[17] M. Pratama, S.G. Anavatti, M.J. Er, E.D. Lughofer, pClass: an effective classifier for streaming examples, IEEE Trans. Fuzzy Syst 23 (2015) 369–386, doi:10.1109/TFUZZ.2014.2312983.

[18] P.Y. Chen, S. Yang, J.A. McCann, Distributed real-time anomaly detection in networked industrial sensing systems, IEEE Trans. Ind. Electron 62 (2015) 3832–3842, doi:10.1109/TIE.2014.2350451.

[19] E.J. Spinosa, A.P.D.L.F. De Carvalho, J. Gama, OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams, in: Proceedings of the 2007 ACM Symposium on Applied Computing, 2007, pp. 448–452, doi:10.1145/1244002.1244107.

[20] E.R. Faria, J. Gama, A.C. Carvalho, Novelty detection algorithm for data streams multi-class problems, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 795–800, doi:10.1145/2480362.2480515.

[21] S. Lee, G. Kim, S. Kim, Self-adaptive and dynamic clustering for online anomaly detection, Expert Syst. Appl. 38 (2011) 14891–14898, doi:10.1016/j.eswa.2011.05.058.

[22] T. Ahmed, M. Coates, A. Lakhina, Multivariate online anomaly detection using kernel recursive least squares, in: Proceedings of the 26th IEEE International Conference on Computing Communication, 2007, pp. 625–633, doi:10.1109/INFCOM.2007.79.

[23] M. Schneider, W. Ertel, F. Ramos, Expected Similarity estimation for large-scale batch and streaming anomaly detection, Mach. Learn. 105 (2016) 305–333, doi:10.1007/s10994-016-5567-7.

[24] A. Stanway, Etsy Skyline, Online Code Repos. (2013). https://github.com/etsy/skyline.

[25] A. Bernieri, G. Betta, C. Liguori, On-line fault detection and diagnosis obtained by implementing neural algorithms on a digital signal processor, IEEE Trans. Instrum. Meas 45 (1996) 894–899, doi:10.1109/19.536707.

[26] M. Basseville, I. V Nikiforov, Detection of Abrupt Changes, 1993.

[27] M. Szmit, A. Szmit, Usage of modified holt-winters method in the anomaly detection of network traffic: case studies, J. Comput. Networks Commun. (2012), doi:10.1155/2012/192913.

[28] P. Angelov, Anomaly detection based on eccentricity analysis, in: Proceedings of the 2014 IEEE Symposium Evolving and Autonomous Learning Systems, 2014, doi:10.1109/EALS.2014.7009497.

[29] B.S.J. Costa, C.G. Bezerra, L.A. Guedes, P.P. Angelov, Online fault detection based on typicality and eccentricity data analytics, in: Proceedings of the International Joint Conference on Neural Networks, 2015, doi:10.1109/IJCNN.2015.7280712.

[30] A.M. Bianco, M. García Ben, E.J. Martínez, V.J. Yohai, Outlier detection in regression models with ARIMA errors using robust estimates, J. Forecast. 20 (2001) 565–579.

[31] R.J. Hyndman, Y. Khandakar, Automatic time series forecasting : the forecast package for R Automatic time series forecasting : the forecast package for R, J. Stat. Softw 27 (2008) 1–22.

[32] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, in: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, 2011, pp. 385–392, doi:10.1109/INM.2011.5990537.

[33] D.L. Simon, A.W. Rinehart, A model-based anomaly detection approach for analyzing streaming aircraft engine measurement data, in: Proceedings of Turbo Expo 2014: Turbine Technical Conference and Exposition, ASME, 2014, pp. 665–672, doi:10.1115/GT2014-27172.

[34] E.K. Lee, H. Viswanathan, D. Pompili, Model-based thermal anomaly detection in cloud datacenters, in: Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems, 2013, pp. 191–198, doi:10.1109/DCOSS.2013.8.

[35] T. Klerx, M. Anderka, H.K. Buning, S. Priesterjahn, Model-based anomaly detection for discrete event systems, in: Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, IEEE, 2014, pp. 665–672, doi:10.1109/ICTAI.2014.105.

[36] F. Knorn, D.J. Leith, Adaptive Kalman filtering for anomaly detection in software appliances, in: Proceedings of the IEEE INFOCOM, 2008, doi:10.1109/INFOCOM.2008.4544581.

[37] A. Soule, K. Salamatian, N. Taft, Combining filtering and statistical methods for anomaly detection, in: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement, 4, 2005, p. 1, doi:10.1145/1330107.1330147.
[38] H. Lee, S.J. Roberts, On-line novelty detection using the Kalman filter and extreme value theory, in: Proceedings of the 19th International Conference on Pattern Recognition,, 2008, pp. 1–4, doi:10.1109/ICPR.2008.4761918.
[39] A. Morgan, Lytics Anomalyzer Blog, (2015). https://www.getlytics.com/blog/post/check_out_anomalyzer.
[40] Y.J. Lee, Y.R. Yeh, Y.C.F. Wang, Anomaly detection via online oversampling principal component analysis, IEEE Trans. Knowl. Data Eng 25 (2013) 1460–1470, doi:10.1109/TKDE.2012.99.
[41] A. Lakhina, M. Crovella, C. Diot, Diagnosing network-wide traffic anomalies, ACM SIGCOMM Comput. Commun. Rev 34 (2004) 219, doi:10.1145/1030194.1015492.
[42] N. Görnitz, M. Kloft, K. Rieck, U. Brefeld, Toward supervised anomaly detection, J. Artif. Intell. Res 46 (2013) 235–262, doi:10.1613/jair.3623.
[43] U. Rebbapragada, P. Protopapas, C.E. Brodley, C. Alcock, Finding anomalous periodic time series : An application to catalogs of periodic variable stars, Mach. Learn. 74 (2009) 281–313, doi:10.1007/s10994-008-5093-3.
[44] T. Pevný, Loda: Lightweight on-line detector of anomalies, Mach. Learn 102 (2016) 275–304, doi:10.1007/s10994-015-5521-0.
[45] A. Kejariwal, Twitter Engineering: Introducing Practical and Robust Anomaly Detection in a Time Series [Online blog], (2015). http://bit.ly/1xBbX0Z.
[46] J. Hawkins, S. Ahmad, Why neurons have thousands of synapses, a theory of sequence memory in neocortex, Front. Neural Circuits. 10 (2016) 1–13, doi:10.3389/fncir.2016.00023.
[47] D.E. Padilla, R. Brinkworth, M.D. McDonnell, Performance of a hierarchical temporal memory network in noisy sequence learning, in: Proceedings of the International Conference on Computational Intelligence and Cybernetics, IEEE, 2013, pp. 45–51, doi:10.1109/CyberneticsCom.2013.6865779.
[48] D. Rozado, F.B. Rodriguez, P. Varona, Extending the bioinspired hierarchical temporal memory paradigm for sign language recognition, Neurocomputing 79 (2012) 75–86, doi:10.1016/j.neucom.2011.10.005.
[49] Y. Cui, S. Ahmad, J. Hawkins, Continuous online sequence learning with an unsupervised neural network model, Neural Comput 28 (2016) 2474–2504, doi:10.1162/NECO_a_00893.
[50] S. Purdy, Encoding Data for HTM Systems, arXiv. (2016) arXiv:1602.05925 [cs.NE].
[51] J. Mnatzaganian, E. Fokoué, D. Kudithipudi, A Mathematical Formalization of hierarchical temporal memory's spatial pooler, Front. Robot. AI. 3 (2017) 81, doi:10.3389/frobt.2016.00081.
[52] Y. Cui, S. Ahmad, J. Hawkins, The HTM Spatial Pooler: a neocortical algorithm for online sparse distributed coding, bioRxiv, 2016, doi:http://dx.doi.org/10.1101/085035.
[53] S. Ahmad, J. Hawkins, Properties of sparse distributed representations and their application to Hierarchical Temporal Memory, 2015, arXiv:1503.07469 [q-NC].
[54] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM. 13 (1970) 422–426, doi:10.1145/362686.362692.
[55] G.K. Karagiannidis, A.S. Lioumpas, An improved approximation for the Gaussian Q-function, IEEE Commun. Lett 11 (2007) 644–646.
[56] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv (2009) 1–72.
[57] R.P. Adams, D.J.C. Mackay, Bayesian Online Changepoint Detection, 2007, arXiv:0710.3742 [stat.ML].
[58] M. Schneider, W. Ertel, G. Palm, Constant Time expected similarity estimation using stochastic optimization, (2015) arXiv:1511.05371 [cs.LG].
[59] M. Bartyś, R. Patton, M. Syfert, S. de las Heras, J. Quevedo, Introduction to the DAMADICS actuator FDI benchmark study, Control Eng. Pract. 14 (2006) 577–596, doi:10.1016/j.conengprac.2005.06.015.

Subutai Ahmad received his A.B. in Computer Science from Cornell University in 1986, and his PhD in Computer Science from the University of Illinois at Urbana-Champaign in 1991. He is the VP of Research at Numenta. His research interests are in computational neuroscience, machine learning, computer vision, and real-time systems.

Alexander Lavin received his B.S. in Mechanical Engineering from Cornell University in 2012, and M.S. in Mechanical Engineering from Carnegie Mellon University in 2014, specializing in spacecraft engineering. He is currently a Senior Research Engineer at Vicarious, building general artificial intelligence for robotics. His main research interests are computational neuroscience, computer vision systems, and robotics.

Scott Purdy received his B.S. and M.Eng. degrees in Computer Science in 2010 and 2011, respectively, from the College of Engineering at Cornell University. He is Director of Engineering at Numenta. Scott's research interests are computational neuroscience, machine learning, and robotics.

Zuha Agha received her B.S in Computer Science from Lahore University of Management Sciences Pakistan in 2014, and her M.S in Computer Science from University of Pittsburgh in 2017. She was formerly an Algorithms Intern at Numenta, and will join Apple in the summer of 2017. Her research interests include data science, machine learning, and computer vision.