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*

A R T I C L E I N F O

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

A B S T R A C T

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

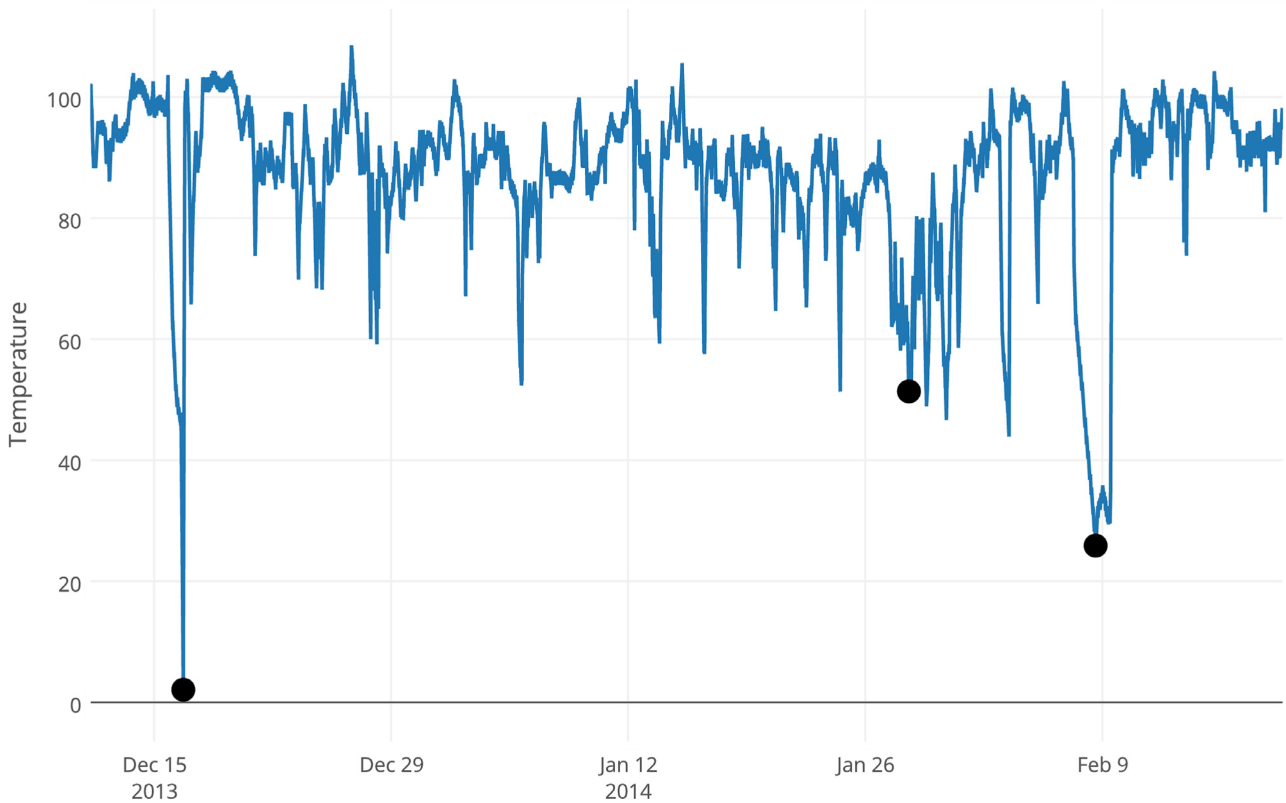

Fig. 1. The figure shows real-world temperature sensor data from an internal component of a large industrial machine. Anomalies are labeled with circles. The first anomaly was a planned shutdown. The third anomaly was a catastrophic system failure. The second anomaly, a subtle but observable change in 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