2020

Hierarchical Anomaly Detection for Time Series Data

Ryan E. Sperl
Wright State University

HIERARCHICAL ANOMALY DETECTION FOR TIME SERIES DATA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

By

RYAN E. SPERL
B.S.C.S., Wright State University, 2019

2020
Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

4/28/2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Ryan E. Sperl ENTITLED Hierarchical Anomaly Detection for Time Series Data BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

_____
Soon M. Chung, Ph.D.
Thesis Director

_____
Mateen Rizki, Ph.D.
Chair, Department of Computer Science and Engineering

Committee on Final Examination

_____
Soon M. Chung, Ph.D.

_____
Nikolaos Bourbakis, Ph.D.

_____
Vincent A. Schmidt, Ph.D.

_____
Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Sperl, Ryan E. Department of Computer Science and Engineering, Wright State University, 2020. Hierarchical Anomaly Detection for Time Series Data.

With the rise of Big Data and the Internet of Things, there is an increasing availability of large volumes of real-time streaming data. Unusual occurrences in the underlying system will be reflected in these streams, but any human analysis will quickly become out of date. There is a need for automatic analysis of streaming data capable of identifying these anomalous behaviors as they occur, to give ample time to react. In order to handle many high-velocity data streams, detectors must minimize the processing requirements per value. In this thesis, we have developed a novel anomaly detection method which makes use of a diverse set of detectors in a hierarchical structure.

The composite detector follows a filtration paradigm to mark each value in the series. The base model, chosen to be fast potentially at the expense of precision, identifies candidate anomalies in the series as each value arrives. Models higher in the hierarchy verify the candidates from their immediate predecessor, potentially rejecting some as false alarms. Our experiments show that this hierarchical method can achieve similar performance to state-of-the-art detectors using computationally simple models with lower processing requirements, enabling better scalability.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. Soon Chung for giving me clear direction to my early efforts. I would also like to express gratitude to my thesis committee members, Drs. Nikolaos Bourbakis and Vincent A. Schmidt. My deepest gratitude goes to my parents, who supported me throughout college, and to my older brother, Leland, who originally sparked my interest in computer science. I would not be where I am today without them.

CHAPTER 1
INTRODUCTION

Data is a key element of today's highly connected world, but that interconnectedness leads to interdependence, where a failure in one critical system can cascade failure to many others. Patterns in system metrics, as collected by a variety of sensors, may provide an early warning, enabling an issue to be identified and mitigated before much harm can be done [7]: A persistent drop in electricity usage may be due to a blackout; a sudden increase in the number of users on a service may indicate an oncoming DDoS attack; an overheating CPU may damage itself and nearby components if not allowed to cool. Such systems require constant, careful monitoring to identify problems as soon as possible [5]. Even non-critical systems benefit from this diligence: An abnormally high stock price may indicate a time to sell; altered levels of traffic congestion may be due to a holiday.

Many systems in today's interconnected world generate large quantities of time-varying data, such as from sensors in Internet of Things devices or the high-volume streams involved in Big Data [1]. This data is often noisy, unstructured, and arrives in real-time [11]. Already far too much for a person, or group of experts, to analyze in a timely manner, the volume of data generated is expected to grow over time [3], requiring automatic analysis solutions to be scalable to increasing data velocities [11].

In this thesis, we have implemented a new anomaly detection approach, using multiple streaming detectors in a hierarchical structure. We also implemented its (computationally simple) component models, and compared the performance of each detector against that of two state-of-the-art algorithms, Hierarchical Temporal Memory and Extensible Generic Anomaly Detection System.

This thesis is organized as follows: Chapter 2 discusses the definition of anomalies and what qualities are desired in detectors. Chapter 3 defines the anomaly detection heuristics and details the tested algorithms. Chapter 4 discusses the benchmark data sets used for testing and their properties. Chapter 5 contains the test results and comparisons on the precision and runtimes of each detector. In Chapter 6, we discuss possible future work and list our references.

CHAPTER 2
BACKGROUND

2.1 Anomaly Definition

An anomaly is an unusual value, behavior which significantly differs from previously observed patterns; as abnormal occurrences, they are rare in real-world data [5]. Anomalies do not always indicate harmful occurrences; an increase in the number of users is beneficial but should be met with increased capacity [3]. Spatial anomalies fall outside the expected range of values and can be detected easily [1]. Temporal anomalies are contextual in nature, only anomalous when considered in relation to the past behavior [3]; these are subtle and difficult to detect, particularly in noisy data [7]. A sudden change to the series attributes may be expected as part of seasonal variation [3], or the anomalous change may persist long enough to be expected in the future [1].

2.2 Prior Work

Most studied anomaly detection methods are batch methods, requiring all of the dataset or significant portions to analyze. Real-time data sources cannot satisfy this requirement; future values are unavailable, and a data stream with no defined end will require arbitrarily large amounts of storage. Past studies generally don't consider the use case of large data flows [1]. Clustering-based algorithms learn offline, setting aside a training portion before labeling online [7]. Streaming detectors use statistical techniques such as hypothesis testing, exponential smoothing, Grubbs test, or k-sigma [3]. Generally, detectors which model the data are often domain-specific and suffer many false alarms if

brought to new domains [5], requiring model retraining. Neural network models have been shown to accurately learn long-term behavior patterns but are often slow to adapt to changes therein [10].

Batch training methods can learn continuously by maintaining a buffer of past data values, representing a sliding window which the model periodically retrains on. However, the retraining step may be computationally expensive, and the window size poses multiple problems. Small windows do not allow the model to know long-term patterns, while large windows increase the storage requirements for the data buffer and limit the model's ability to quickly adapt to behavioral changes in the data [10].

2.3 Desired Qualities

Real-time data presents several challenges to potential detectors. A model cannot know how a series will evolve in the future, limiting the effectiveness of pretraining. Transfer learning, training a model on a dataset to apply on a related set, may alleviate this issue. Most real-world data is not stationary; their statistics may change suddenly (change points) or gradually (concept drift). Shifting data patterns require a detector to continuously adapt to new notions of normality [1]. As real-time data contains no ground truth labels to evaluate against and is often noisy, this learning must be unsupervised and noise-tolerant [7]. Furthermore, lookahead is not possible and cannot be depended upon [1]. Large, high-velocity data flows deny human intervention in the learning process, and each value must be processed as fast as possible to maximize scalability to these large streams [5]. Values should be labeled online, as they arrive, to minimize response time

[1]. Detectors should not store large portions of the stream, or memory bandwidth and available storage may become limiting factors [7].

Due to the potential consequences of anomalies in critical systems, an ideal detector should identify anomalies as early as possible [1]. Unfortunately, early detection increases the number of false alarms, resulting in a detector whose alerts will often be ignored [7]. In practice, a detector must balance detection time and false positive rate. Nevertheless, a missed anomaly may be far more costly than multiple false alarms, though the relative importance of each mistake will vary depending on the system [1].

CHAPTER 3
DETECTION METHODS

3.1 Anomaly Heuristic

Model-based detectors attempt to forecast series values. A fixed threshold on the prediction error is not usable in most cases except for highly regular data, as the expected variance may change over time. Any metrics used to determine a dynamic threshold must necessarily be sample metrics, as a detector cannot view all of a real-time series, only past values [7]. To account for changes in nonstationary series statistics, these metrics should only include the most recent values to limit the influence of old data; this will also improve the processing time by limiting the amount of data to be calculated and stored.

Variance, which considers the square of the deviation, is sensitive to outliers and may be significantly distorted by even one past anomaly. The regular shifts in seasonal data also tend to increase the variance, reducing the model's sensitivity. Thresholds using the standard deviation assume that the series is normally distributed, which is unknowable at the start and uncommon in practice [3]. The median absolute deviation (MAD) is a robust alternative:

$$MAD = median(|X_i - median(X)|)$$

where X is the portion of the input series in a rolling sample window. The median does not consider the magnitude of the residuals, only the ordering thereof, enabling the MAD to tolerate up to 50% of the sample window being anomalous, highly unlikely due to the rarity of anomalies [3]. While the mean can theoretically be used as the central measure, it is more sensitive to outliers, and both can be calculated in linear time [6]. A

constant factor applied to the MAD or other threshold metric provides a means to tune the model's sensitivity [3].

Absolute error is strongly influenced by the magnitude of the series values, and thus is a poor error metric for a series whose values vary widely. Relative error normalizes the influence on the magnitude, allowing a fair comparison across different data scales [5]. The overall anomaly metric is given by:

$$AnomalyDetected = |\varepsilon_t| > k * MAD$$

where ε_t is the error metric and k is the sensitivity constant.

3.2 SARIMA

Autoregressive Integrated Moving-Average, hereafter denoted ARIMA, is a computationally simple forecasting model which predicts future values based on a weighted sum of the most recent data and an unpredictable error term, on the premise that more recent values are closer in time to the forecast and therefore are more informative than older values. These prior data points are the lags of the series and are accessed using the backshift operator (B), which shifts the values of the series backward one timestep:

$$X_{t-k} = B^k X_t$$

where X_t is the value of a time series X at time t, and k is the number of shifts applied to X. The sequence of residuals for a well-fit model corresponds to white noise, with no bias towards too large or too small predictions. ARIMA models consist of three main components.

Autoregression (AR) is a regression on the self, predicting a linear dependence on the closest lags of the modeled series:

$$X_t = c + \varepsilon_t + \sum_{k=1}^{p} a_k B^k X_t$$

where a is the sequence of p model parameters, ε_t is the stochastic error term, equal to the prediction error at time t, and c is a constant term. The order p of the model is the farthest lag which the model considers relevant, requiring that an AR(p) model store p lags in addition to the constant term and p coefficients. In practice, p will be minimized to avoid overfitting the data, providing a small memory footprint for low-order models.

Moving-average (MA) forecasts future values as a weighted sum of lagged residuals:

$$X_t = \mu + \varepsilon_t + \sum_{k=1}^{p} b_k B^k \varepsilon_t$$

where b is the sequence of q model parameters, and μ is the series mean. MA models weakly stationary processes, whose statistical properties do not change over time; thus μ is expected to be constant. The series mean can be approximated by a sample mean, but the sample size need not be large; the estimation of an unchanging mean may be refined as new values arrive without the need to store old values. Thus, the memory required is comparable to autoregression. AR and MA models are components of the more general autoregressive moving-average model, denoted ARMA(p,q), which is the sum of an AR(p) and MA(q) model:

$$X_t = c + \varepsilon_t + \sum_{k=1}^{p} a_k B^k X_t + \sum_{k=1}^{q} b_k B^k \varepsilon_t$$

If the data is not stationary, it must be transformed to become stationary. Trends in the series may be removed by differencing, which replaces each value with its immediate predecessor:

$$\nabla X_t = X_t - X_{t-1} = (1 - B) X_t$$

where ∇X is a first-order differenced time series, representing how the data is currently changing. Second-order differencing extracts the current curvature of the original data by differencing ∇X; the operation is analogous to a discretized derivative. The original series may then be restored by integrating the differenced series:

$$X_t = \nabla X_t + X_{t-1}$$

The order of integration *d* defines the minimum differences necessary to obtain a stationary series. If d is finite, a series is I(d); an already stationary series is I(0). Differencing a cyclic trend in a series will not produce a stationary series, and may exacerbate the variations. If such a pattern has a fixed period of length *m*, it may be removed by differencing against an older lag, known as seasonal differencing:

$$\nabla_m X_t = X_t - X_{t-m} = (1 - B^m)X_t$$

An ARIMA(p,d,q) model forecasts an I(d) series by integrating the predictions of an ARMA(p,q) model fitted to the differenced series $\nabla^d X$. Seasonal ARIMA, SARIMA$(p,d,q)_m$, fits an ARMA model to the seasonally differenced series $\nabla^d{}_m X$. Consequently, SARIMA is equivalent to ARIMA when *m*=1. As there is no prior data to difference against until the beginning of the second cycle, a first-order seasonal differenced time series must start *m* time steps ahead of its integration, requiring batch fitting methods to gather larger quantities of data to fit the underlying ARMA model. Though SARIMA models can model a wide variety of series, they cannot adapt to data with a variable season length.

3.3 Hierarchical Detection

Complex models are capable of accurately modeling a wide variety of data but run the risk of overfitting. Simple models may process input data much faster, achieving greater scalability at the cost of accuracy. Such models may be supplemented by multiple detection phases using increasingly accurate methods, such that the slower detectors verify the labels of the simpler detectors. However, blindly applying multiple detectors to verify every label will simply increase the processing time; to minimize the increased time cost, only the rarer anomalous labels should be checked, correcting any false positives. This introduces a desire for a bias towards anomalous labels in the base detector to minimize missed anomalies. We expect the processing time to be less than the sum of each detector in the hierarchy. At worst, the base detector marks every value as anomalous, requiring a verification on each value; at best, the base detector perfectly labels the data. We expect that there will be few enough false positives to limit the processing time to be similar to the verification model with better precision.

We denote a hierarchical set of detectors using an ordered pair; i.e. (MA, ARIMA). Moving-average models are computationally simple and have been shown to rarely miss anomalies at the cost of a large number of false positives [11]. We propose a two-step detector, using a simple, high-sensitivity model such as moving-average as a base detector and a slower, more accurate model such as ARIMA as the hierarchical detector. In the presence of seasonal data, SARIMA may yield more accurate results than ARIMA. These models require the series of residuals for the MA terms, which will be as sparse as the true labels of the base detector. To remedy the gaps in the series, we reuse

the residuals from the base model. If extending the hierarchy to more than two levels, care must be taken to choose models which carry less state between predictions.

3.4 EGADS

Extensible Generic Anomaly Detection System (EGADS) is a modular framework for anomaly detection, built with the goals of extensibility and scalability to data flows on the order of millions of data points per second.

EGADS separates the detection of outliers, values which differ significantly from past behavior, and change points, where the series statistics abruptly change. The time series modeling module (TMM) provides the expected behavior by forecasting the data series, storing values in a Hadoop cluster. The anomaly detection module (ADM) detects outliers based on the resulting deviations; as no statistic will be optimal for all possible series, the system monitors multiple error metrics and allows for users to add custom metrics. EGADS determines thresholds based on the error metric: three-sigma provides a straightforward threshold in normally-distributed residuals, while local outlier factor (LOF) compares the density of the deviation distribution [5].

EGADS differentiates absolute change point detection techniques, which compare the series behavior in separate sliding windows, and relative methods, which apply the absolute methods to compare the behavior of the residuals. This enables the model to ignore non-anomalous change points, which were expected by the TMM [5].

Not all identified anomalies may be desired for some use cases. The alerting module (AM) uses machine learning techniques to determine the relevancy of each

anomaly, based on user feedback. Naturally, the AM requires user interaction during training.

In most real-world data, a "one size fits all" approach is misguided; a model which works well in one domain will yield many false positives if brought to another. EGADS does not attempt to select the best algorithm for a given dataset, rendering model selection to its users. All selected models are trained on the batched training period data, and kept in memory to minimize disk latency.

3.5 HTM

Hierarchical Temporal Memory is a neural network based model developed by Numenta, which attempts to mimic the cortical neurons in the human brain to internally represent spatiotemporal patterns [7]. The system tolerates highly noisy data and constantly learns from its input data [11]. Components of an HTM network learn by forming synaptic connections to other components. The overall functionality can be generally divided into three main parts

The encoder deterministically translates input data into a Sparse Data Representation (SDR) format, a bit vector typically with less than 5% of the bits set. Comparisons between SDRs are accomplished by computing the bitwise intersection and counting the active bits; this fuzzy matching allows for partial matches, granting high noise-tolerance. The low density of SDRs reduces the chance of a false match to near zero. The union of multiple SDRs is itself an SDR which represents all of its component values, enabling fast set membership tests by the same similarity test, retaining the

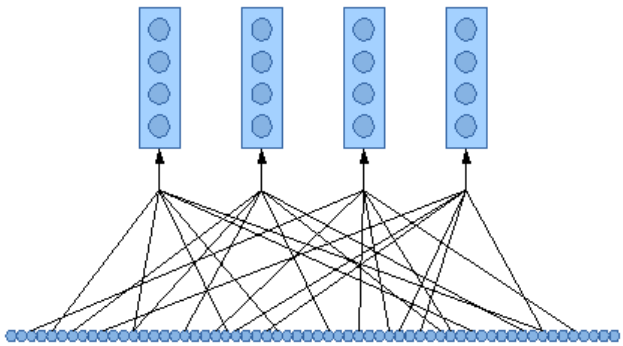

Figure 1: Proximal synapses, connected to neurons in the previous layer

robustness despite the increased density. The encoder assigns meaning to the bits in a way that preserves the similarity of the inputs [2].

The spatial pooler learns spatial patterns in the data, recognizing patterns irrespective of context. Each layer is composed of several columns, each with a set of potential proximal synapses, connections to the previous layer, forming its receptive field. A threshold on each synapse's permanence value, initialized randomly, determines if it is currently connected. As new data arrives in the network, the spatial pooler computes the overlap for each column's connected synapses against the input SDR and activates columns with the best match score, thereby encoding the input value with a consistent sparseness. As only active columns update their synaptic connections, the network boosts the match score for highly inactive columns and hinders strongly expressed columns, preventing any columns from dominating the output [2].

Temporal memory enables the network to recognize temporal patterns in the data and predict its future evolution. Each column in the spatial pooler consists of multiple cells, each with a set of potential distal synapses which connect to other cells in the same layer. An active column will have at least one active cell; the pattern of these cells

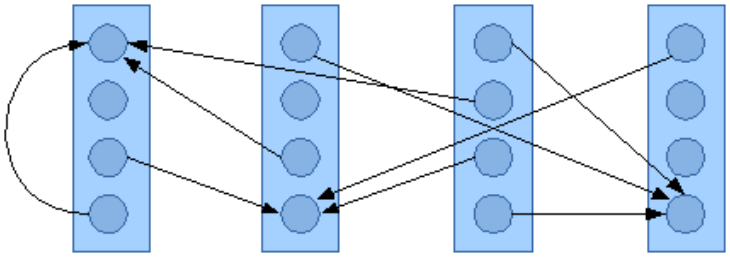

Figure 2: Distal synapses, connected to neurons in the same layer

encodes the current value's context. Each cell uses the current context visible in its receptive field to predict its column's next activation state, thereby forecasting the next input value. Active columns which were not predicted activate all of their cells and choose one each to represent this new context, potentially expanding its receptive field via new potential synapses if necessary [2]. The sparseness of active cells enables the network to remember many sequences for long periods of time without using a buffer of input data [10].

An HTM model's anomaly score is the prediction error, based on the match score of its predicted columns and the actual active columns:

$$\varepsilon_t = 1 - \frac{\pi(X_{t-1}) * a(X_t)}{|a(X_t)|}$$

where $a(X_t)$ is the encoded value at time t, $|a(X_t)|$ is the number of bits set (the scalar norm), and $\pi(X_t)$ is the prediction for the next timestep. An anomaly score of 1 indicates no overlap, while 0 is a perfect match. As a consequence of the union property of SDRs, ambiguous contexts will predict all known branches, resulting in an anomaly score of 0 if any of the branches was correct [7]. To limit the number of false positives due to noisy data, the detector thresholds the anomaly likelihood score, a measure of how

well the network predicts the data, calculated as the Gaussian tail probability of the current error value:

$$L_t = 1 - Q(\frac{\mu'_t - \mu_t}{\sigma_t})$$

where μ_t and σ_t are the mean and standard deviation of a rolling sample window of past anomaly scores, μ'_t is the short-term rolling average with a much smaller sample window, and Q is the Gaussian tail probability function. The aggregation smooths out the influence of sudden spikes in the data; moreover, any significant change to the error distribution may be marked, even for changes to the noise level [7]. The system quickly adapts to changes in the data and is robust in the presence of noise [10].

While the network initially learns the normal behavior patterns, it does not know enough about the series to make accurate predictions, resulting in high initial likelihoods. During this training period, we ignore its anomaly labels [7].

3.6 SSA

Singular Spectrum Analysis is a model-free technique based on linear algebra. Rather than forecasting the data, SSA decomposes the input series into multiple sub-series representing periodic oscillations, underlying trend, and unpredictable noise, then recombines them according to the information they carry.

An input series *X* is first embedded as a matrix Y_{LxK}, commonly known as the trajectory matrix [13]. Each column of Y is an L-lagged vector of X, a collection of L consecutive lags of X. Y is a Hankel matrix, where each antidiagonal is constant. The

Singular Value Decomposition (SVD) of the trajectory matrix represents Y as a sum of rank-1 elementary matrices Y_i, such that

$$Y_i = \sqrt{\lambda_i} U_i V_i^T$$

where λ is the sequence of eigenvalues of YY^T in descending order, and U and V are the corresponding left and right singular vectors, respectively. The right singular vectors are also known as the principal components, and the variance described by each is proportional to the corresponding eigenvalue [12].

The set of matrices is then partitioned into disjoint subsets and summed to produce matrices Y_I, where I is the set of indices included in the group. Each matrix Y_I represents a component of the original series; its subseries can be extracted by Hankelizing the matrix, averaging each antidiagonal, with the sequence of averages as the reconstructed subseries. Partitions should be chosen such that each does not mix trend, harmonic, and noise components. Separability between components is measured by a weighted inner product of the reconstructions and normalized by their magnitudes, treating each subseries as a vector. Typically, trend components appear early in the elementary matrix set and are separable from other components, while harmonic components consist of two elementary matrices which are highly separable from all but each other [13].

In its basic form, SSA evaluates a series as a whole, unsuitable for real-time processing. Even if the entire series was available, series with many values result in a large matrix Y due to the number of lag vectors, potentially slowing the SVD step significantly. In streaming scenarios, SSA evaluates using a sliding window, limiting the amount of data considered at a time. The window should be large enough to include the

longest expected period. While tracking the statistics is useful, most implementations employ a comparison between two separate sliding windows. The results are compared component-wise using a distance metric such as Euclidean distance or cosine similarity (treating each subseries as a vector) and aggregated over the subseries. Not all subseries need to be considered in this aggregation; users may choose the *m* subseries with the largest eigenvalues, explaining the most variation [13]. The result is the anomaly score; should it exceed some specified threshold, an anomaly has likely occurred. This approach naturally favors detecting change points, though it is unlikely to identify the change point immediately.

CHAPTER 4
DATASETS

Historically, benchmark datasets have been insufficient for drawing comparisons or modeling real-world use cases [1]. Many partition data into training and testing sets with similar statistics, which has no continuous learning analogue [7] and does not accurately capture the behavior of streaming data, which is often noisy and constantly changing [10]. With the rise of Big Data, algorithms which can handle large, complex data streams are desired, but this use case is often not taken into account [11].

To verify the efficacy of the aforementioned algorithms, we tested them on four data sets from the Numenta Anomaly Benchmark (NAB), a collection of labeled univariate time series collected from multiple application domains. We believe the varied behavior of the chosen series will provide a strong benchmark for comparison. A few series have known causes for their anomalies and are labeled accordingly. All others were hand-labeled by multiple labelers following a shared set of procedures; these labels were then algorithmically combined to minimize human error. The labeling procedures and combiner source code are publicly available [1].

Table 1: Overview of the data sets

	CPU Utilization 825cc2	CPU Utilization 5f5533	NYC Taxi	GoogleTweets
Time Interval	5 min	5 min	30 min	5 min
Season Interval	20 min	40 min	1 week	1 day
Total Anomalies	2	2	5	4
Total values	4032	4032	10,320	15,842

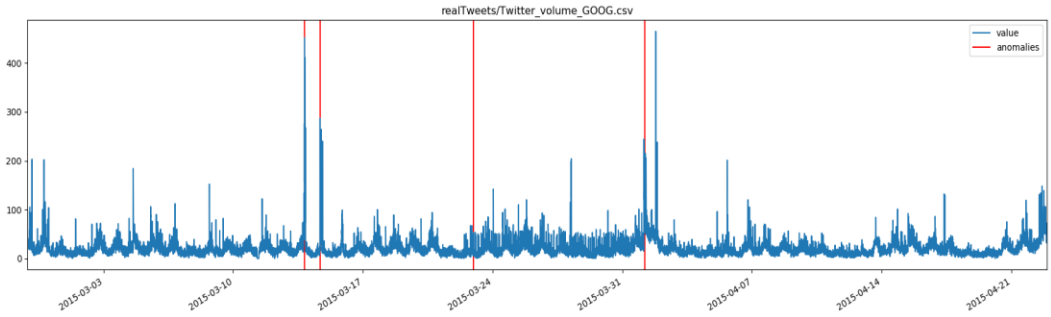

Figure 3: Google stock mentions on Twitter

The NycTaxi data set tracks the number of taxi passengers in NYC, aggregated into half-hour buckets. Five anomalies occur due to various holidays, a marathon, and a snowstorm. The data shows clear seasonality with daily and weekly patterns, matching typical work schedules. Due to differing patterns on weekends, we found a period of one day to be unsuitable.

The GoogleTweets data set tracks mentions of google stock on Twitter, identified by its ticker symbol, aggregated into 5-minute buckets. Three of the four anomalies are abnormally high attention given to this particular stock. Though the data is noisy, the distribution shows a daily rise and fall which varies in magnitude.

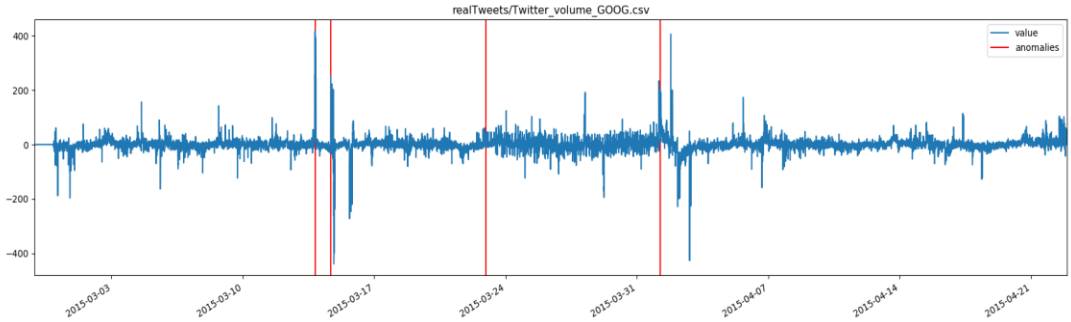

Figure 4: Google stock mentions, after differencing

We used two datasets from the AWS server metrics, each of which monitors the percentage of ec2 compute units in use by an instance [4]. The first, denoted ec2_cpu utilization_825cc2, is fairly noisy but stays close to 90%. The data shows an abrupt drop which lasts for a significant portion of a day, preceded by a less severe downward spike. The change does not last long enough to be expected in the future; the return to its typical mean is not considered anomalous. We found that a period of 20 minutes eliminates most of the trend. The second, denoted ec2_cpu utilization_5f5533, likewise shows significant noise levels. Its anomalies are two change points which reduce the mean and range, with the first being much less pronounced compared to the second. A period of 40 minutes removes the seasonal component hidden amid the noise.

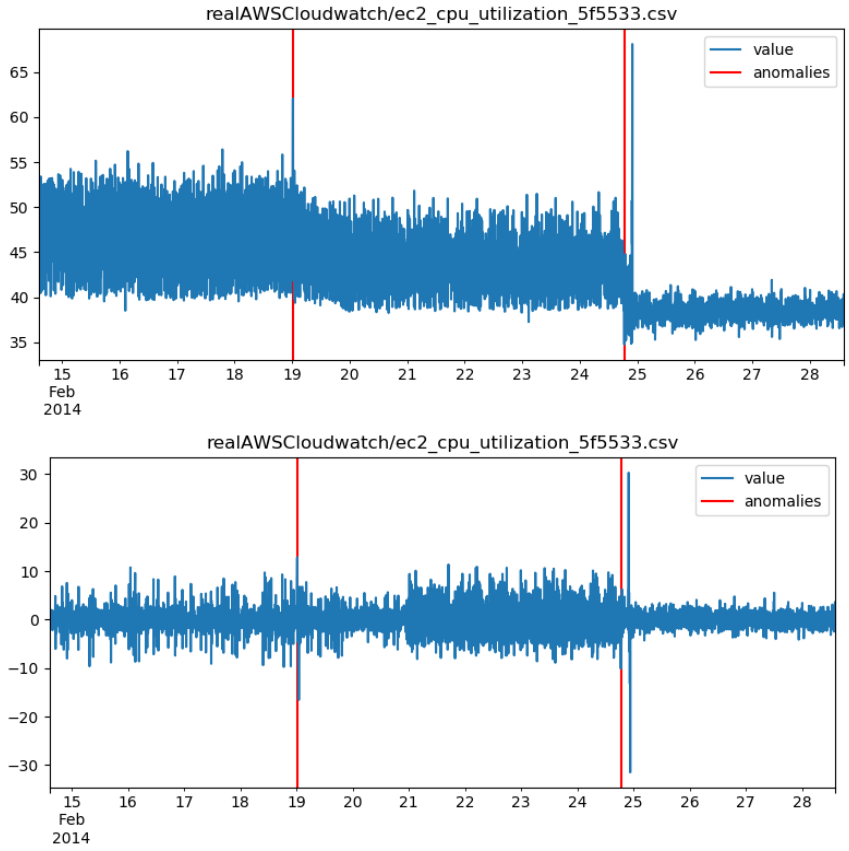

Figure 5: CPU Utilization 5f5533, before and after seasonal differencing.

CHAPTER 5

EXPERIMENTAL RESUTLS

We implemented SARIMA and its component models in Python 3.6 and executed them on a Windows 10 laptop. We also implemented two 2-step hierarchical detectors, (MA, SARIMA) and (SIMA, SARIMA), where SIMA is a moving-average model applied to seasonally differenced data. We determined the optimal model orders via grid search, limiting the maximum for each component to avoid overfitting, and fit to the first three days of each series using Python's statsmodels package [8]. Due to its longer seasons, the models were fit to the first three weeks of the NYC Taxi dataset.

We used the open-source implementations for HTM, EGADS, and SSA, as provided by their authors. Numenta's HTM is implemented in Python 2.7, which has recently reached end-of-life [9]. We allow it a learning period consisting of the same length of fitting data given to the SARIMA models, during which its labels are ignored. EGADS does not allow users to specify a training time, so any anomaly labels output during this period are ignored. Furthermore, it does not automatically choose the optimal model for the TMM or ADM modules, necessitating a test against each available model to find the optimal choice. As a result, the runtimes will vary depending on the dataset. Due to its dependencies, we executed EGADS in a Linux environment on the same machine. SSA was implemented in Python 3 and ran under the same environment as the hierarchical detectors. The results of all of our runs are shown in table 2, with the most precise results for each data set highlighted. We used the built-in time library to track the processing time for each model except EGADS, including time taken to fit the model. We timed each EGADS test using the "time" command line tool.

Table 2: Experimental Results

	CPU usage: 825cc2				CPU usage: 5f5533				NYC Taxi				Google Tweets			
Detector	**TP**	**FP**	**FN**	**Time (s)**	**TP**	**FP**	**FN**	**Time (s)**	**TP**	**FP**	**FN**	**Time (s)**	**TP**	**FP**	**FN**	**Time (s)**
MA	2	121	0	1.81	2	23	0	1.21	2	975	3	2.07	3	213	1	2.98
SIMA	2	5	0	1.25	2	22	0	1.53	4	865	1	2.38	3	207	1	3.34
SARIMA	2	9	0	2.33	1	1	1	2.83	2	1292	3	2.50	3	232	1	3.52
MA, SARIMA	2	11	0	6.18	2	0	0	4.64	2	471	3	3.92	3	139	1	3.38
SIMA, SARIMA	2	3	0	3.71	2	17	0	1.98	4	392	1	16.17	3	197	1	3.39
HTM	2	11	0	12.45	1	7	1	18.34	4	178	1	59.35	3	433	1	64.97
EGADS	2	42	0	16.46	2	31	0	0.94	2	72	3	2.54	3	451	1	8.24
SSA	2	60	0	2.76	1	56	1	2.52	2	388	3	12.17	2	257	2	18.01

All detectors successfully identified every anomaly in the CPU utilization 825cc2 dataset. The hierarchical model (SIMA, SARIMA) and its individual components each yielded fewer false alarms compared to the HTM model, in much less time. EGADS did not perform well, taking longer than any other model to process the data, with a worse precision than all but moving-average and SSA. On the other CPU usage dataset, all but three models identified both anomalies. Interestingly, both SARIMA and HTM models failed to identify the second change point, despite it representing a greater change to the mean and variance; SSA failed to detect the first change. Moreover, the hierarchical method (MA, SARIMA) appears to be a perfect detector, which may indicate overfitting. This was also the only dataset for which relative error yielded higher precision overall; all other datasets performed better using absolute error.

No model identified all anomalies in the NYC Taxi dataset, though the anomalies found varied. In particular, only the HTM model identified the first anomaly (a marathon) but missed the fourth (New Year's), which all other models identified. Similarly, all models except SARIMA recognized the last anomaly (a snowstorm). All models show

numerous false positives with the exception of EGADS, which may be partially responsible for the abnormally large runtime for (SIMA, SARIMA). Despite having the best precision and recall, HTM took nearly a full minute to process the data, far more than any other detector. In the Google Tweets dataset, all detectors except for HTM and SSA missed only the last anomaly. The HTM found all anomalous data spikes but missed the much subtler third anomaly, the only one which is not spatial. SSA failed to detect the second and third anomalies, despite the last anomaly being a larger outlier than the second; this was also the only detector to find only two anomalies. (MA, SARIMA) shows the best precision, with an overall runtime less than its component, SARIMA.

The two-step hierarchical methods consistently yield fewer false alarms compared to their component models, often with equivalent recall and comparable runtime. In particular, (SIMA,SARIMA) on the CPU usage 5f5533 dataset labeled the data faster than its component SARIMA, while the same model was remarkably slow on the NYC Taxi data.

HTM consistently generates reasonably precise labels, though the hierarchical methods yield similar or better results for all but the NYC Taxi set. EGADS never performed well compared to the rest of the models, yielding numerous false positives with lengthy runtimes. While the runtime varies due to model selection, it is generally only faster than the HTM. SSA generally resulted in poor precision compared to the other models, only outperforming MA with any consistency. Its runtime is comparable to, and in some cases better than, the hierarchical detectors, but noticeably increases as the size of the data grows; some of this is due to choice of window size and number of subseries.

CHAPTER 6
CONCLUSIONS AND FUTURE TOPICS

Anomaly detection is an important problem in a variety of domains, for which human analysis is insufficient. We have developed a hierarchical detector capable of identifying most of the anomalies in a given time series with minimal false alarms. For a robust threshold, we used Median Absolute Deviation. We used absolute and relative prediction error as an error metric in the hierarchical detectors and their component models. We compared their precision and performance against open-source implementations of Singular Spectrum Analysis, the Extensible Generic Anomaly Detection System, as well as the state-of-the-art Hierarchical Temporal Memory models, using four time series datasets with varied behavior.

Advantages of our detector include:

- It is precise, yielding similar or better precision to the HTM on most of the data
- It is efficient, consistently processing the data much faster than the HTM and usually faster than EGADS
- It is extensible. Any model is accepted for the base model, though a fast, high-sensitivity model such as MA is preferred. Any model is accepted for the verification portion, as long as parameters from past labels can be retrieved from the base model. New models can be inserted at any level in the hierarchy, though adding too many models may drastically reduce its efficiency.

We do not claim that our hierarchical detector is perfect, nor that it is generally optimal. Time series data varies greatly in its behavior; no one model will always be the

most optimal. We have provided a precise, efficient detector which processes input data quickly while successfully identifying most anomalies, allowing for reasonable detection in high-velocity data flows.

REFERENCES

[1] A. Lavin and S. Ahmad, “Evaluating Real-Time Anomaly Detection Algorithms — the Numenta Anomaly Benchmark,” Proc. of 14th Int’l Conf. on Machine Learning and Applications, 2015, pp. 38−44.

[2] J. Hawkins, et al. 2016-2020. Biological and Machine Intelligence. Release 0.4. Accessed at https://numenta.com/resources/biological-and-machine-intelligence/.

[3] J. Hochenbaum, O. S. Vallis, and A. Kejariwal, “Automatic Anomaly Detection in the Cloud via Statistical Learning,” Published in ArXiv, 2017.

[4] List the Available CloudWatch Metrics for Your Instances, docs.aws.amazon.com

[5] N. Laptev, S Amizadeh, and I. Flint. "Generic and Scalable Framework for Automated Time-Series Anomaly Detection," Proc. of the 21th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, 2015, pp. 1939-1947.

[6] numpy.median, numpy.partition, docs.scipy.org

[7] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, “Unsupervised Real-Time Anomaly Detection for Streaming Data,” Neurocomputing, vol. 262, 2017, pp. 137−147.

[8] statsmodels v0.11.1, statsmodels.org

[9] Sunsetting Python 2, python.org

[10] Y. Cui, S. Ahmad, and J. Hawkins. "Continuous Online Sequence Learning with an Unsupervised Neural Network Model." Neural Computation, vol. 28, no. 11, Nov. 2016, pp. 2454-2504.

[11] Z. Hasani, “Robust Anomaly Detection Algorithms for RealTime Big Data: Comparison of Algorithms,” Proc. of the 6th Mediterranean Conference on Embedded Computing, 2017.

[12] H. Hassani, "Singular Spectrum Analysis: Methodology and Comparison." Journal of Data Science, vol. 5, 2007, pp. 239-257.

[13] Q. Dong, Z. Yang, Y. Chen, X. Li, and K. Zeng. "Exploration of Singular Spectrum Analysis for Online Anomaly Detection in CRNs." EAI Endorsed Transactions, 2017.