MTR180294

MITRE TECHNICAL REPORT

MITRE

Machine Learning for Anomaly Detection on VM and Host Performance Metrics

Use machine learning techniques to reduce the number of false alerts sent to IT system operators. Discover alert conditions not detected by conventional IT system monitoring

Author(s): Monica-Ann Mendoza
Henry R. Amistadi

June 2018

Sponsor: ECIS Innovation Research Program
Dept. No.: R105

Bedford, MA

Abstract

IT operations needs an improved approach to warnings and alerts. Currently, most IT monitoring software uses static performance thresholds i.e. <80% CPU usage. This project explores use of machine learning algorithms for dynamic thresholds, based on time series anomaly detection. We developed a procedure that: 1) Determines the periodicity using the autocorrelation function (ACF). 2) Uses Kalman filters for that periodicity, to learn the behavior of IT performance metrics and forecast values based on time of day, etc. Compare the actual to the forecast to check for anomalies or violation of dynamic thresholds. 3) Since Kalman filters continue to learn on new data, we needed another algorithm (DBSCAN) to check for week to week degradation or abnormal behavior and prevent the Kalman algorithm from learning from "bad performance" data and corrupting the calculation of the dynamic threshold. 4) Work included examining time series data for many virtual machines metrics and identified frequently occurring patterns. The algorithms (1-3) were successfully tested on examples of all the patterns. The research only calculates dynamic thresholds for single independent performance metric at a time.

This page intentionally left blank.

Preface

Although this technical paper is being published in June 2018, The work was completed in August of 2017. Higher priority work delayed the final review and preparation of the paper. Work has continued in the interim period that hasn’t been published.

Acknowledgments

Authors would like to thank Chris Teixeira for his evaluation of our methodology and his helpful and careful review of the paper.

Table of Contents

List of Figures

List of Tables

This page intentionally left blank.

1 Background

1.1 The problem

Monitoring of Virtual Machine (VM) and host performance is necessary to ensure the efficient allocation of resources. Unexpected spikes or drop in performance—such as for CPU usage—may be a sign of resource constraints. Such problems need to be addressed by Ops professionals in timely manner; otherwise, applications associated with the VMs and hosts may stall or fail.

Anomalies are instances when performance metrics deviate from normal, expected behavior. With MITRE's current ECIS monitoring system, anomalies are flagged based on static thresholds. Static thresholds are constant values that provide upper limits of normal behavior. For example, VM CPU usage is considered anomalous when the value rises above 75%. When anomalies are detected, alerts are sent out for Ops professionals to inspect.

The problem with static thresholds is that they provide an overly simplistic measure of what should be considered "normal." Different metrics have their own patterns of "normal" behavior. We can expect their values to fluctuate based on time-of-day or weekday, in accordance with application usage or backup schedules. For example, a VM tied to a business application such as the MII should have higher usage during business hours, when users are accessing the application frequently.

For a given time-of-day or weekday, a static threshold may be too low, causing a stream of false alarms. This can become an annoyance, so Ops professionals may come to ignore the alerts or disable them entirely. On the other hand, a static threshold that is too high will cause anomalies to be missed. When this is the case, Ops professionals may not realize the system has a problem until it is too late.

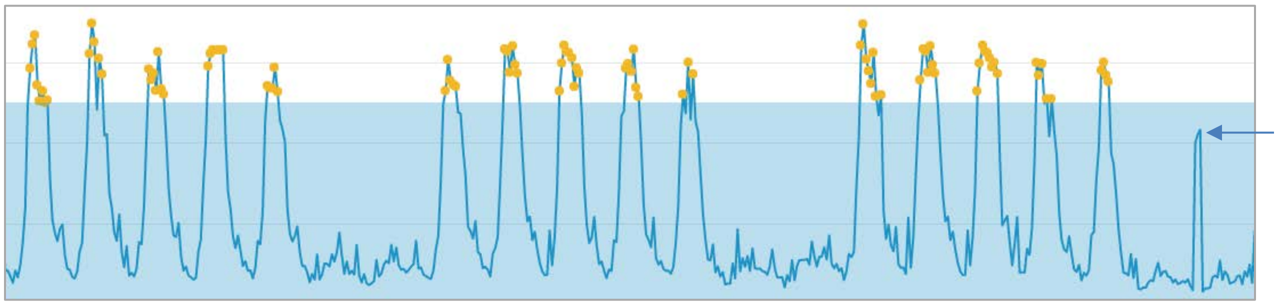


Figure 1 Static thresholds don't truly identify anomalies

This time chart plots a metric's value over 3 weeks. The value spikes during business hours. With this static threshold, business hours are always flagged as anomalous, even though this should be considered normal behavior. The static threshold also misses the indicated spike. Its value is low respective to the overall time series, but it is abnormally high given that it occurs on the weekend.

More accurate VM and hosts performance thresholds should reflect how their workload changes over time. However, setting these thresholds manually is infeasible for the Ops team. MITRE's vSphere environment includes hundreds of VMs, which are monitored with dozens of metrics (for CPU, memory, network, etc.). Each VM and metric might require a different threshold value for each time-of-day and weekday, and normal behavior might change over longer periods of time (for example, what is normal one quarter might be different from the next). Thus, manually configuring thresholds would be an incredibly time-consuming and perhaps futile task.

MITRE needs an adaptive monitoring system that can work on a large scale. It should handle all the vSphere VMs and metrics, automatically determining the patterns of "normal" behavior and using these patterns to set accurate anomaly detection thresholds.

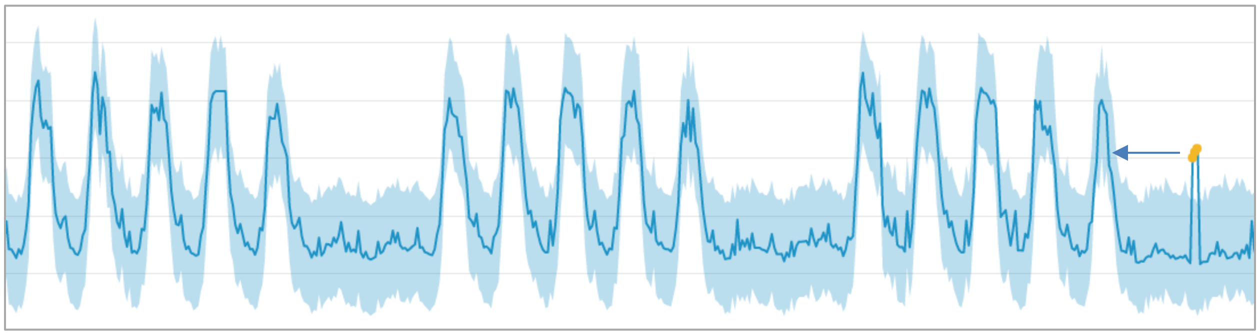


Figure 2 Dynamic thresholds truly identify anomalies

1.2 Why machine learning?

Machine learning (ML) is a subfield of computer science in which computers learn from data without being explicitly programmed. ML algorithms automatically detect patterns in data, build statistical models, and make predictions, all without human intervention.

ML is a key component of AIOps (Algorithmic IT Operations), a term coined by the IT research company Gartner to describe the use of advanced analytics and big data technologies for IT management. AIOps uses ML to automate the process of detecting and repairing problems in IT environments. With machine-assisted root cause analysis, AIOps enables monitoring on a scale that far surpasses human management. This is especially important due to the high velocity and volume of IT data.

AIOps is a rapidly-growing field in IT. According to a July 2016 Gartner report, "By 2020, approximately 50% of enterprises will actively use AIOps platforms to provide insight into both business execution and IT operations, which is an increase from fewer than 10% today."[1]

For anomaly detection, we can use ML to:

1. Learn a statistical model of normal behavior
2. Use the model to forecast future values
3. Identify anomalies by comparing the forecasted values to the actual values, as new data is collected in real-time

An advantage of ML is that it can learn over time. As new data is collected, ML models can update themselves automatically. For example, a model of "normal behavior" could update each week to incorporate new data points, and thus it could output an updated set of forecasts each week as well. With this adaptive algorithm, we can account for data patterns that change over time, rather than basing all our forecasts on a model built solely on our data's original pattern. These updates can be done without human intervention.

[1] Gartner Data Center, Infrastructure & Operations Management Conference. Dec 5-8 2016. Las Vegas, NV.

1.3 Design considerations

When designing our anomaly detection system, we decided upon a ML-based solution and chose our specific ML algorithms to meet the following specifications:

- **Automated**- Building models of normal behavior should require little to no manual tuning. What is considered normal should be learned from the data, rather than defined by an Ops professional.

- **Complex**- An anomaly threshold often requires more complexity than a single, static value (such as 75% CPU usage). Our models should reflect fluctuating patterns (such as daily or weekly cycles) and set thresholds based on these patterns.

- **Adaptive**- The models should update themselves as new data is collected over time.

- **Flexible**- Our algorithm for anomaly detection should generalize well across different patterns. For example, we should be able to use the same algorithm for metrics with different cycle lengths or for metrics with no cycles at all.

- **Streaming**- We should be able to detect anomalies in (near) real-time, rather than retroactively. This is necessary so that Ops professionals can respond to problems soon after they occur.

1.4 Our solution

Kalman filters adapt with new data and recent data has more influence on forecasts. If performance is degrading over time Kalman will learn the new pattern and not detect it as an anomaly. To detect faulty situations like this, our method has 2 stages, one that detects short term (point) and one longer term (collective) anomalies. Point anomalies can be transient, where collective anomalies are more likely to persist and indicate a behavior change.

Kalman filter forecasting to detect abnormal spikes in a metric's value

- We use the Kalman filter to learn a model of normal behavior from a few weeks of historical data. We then forecast a week of future values, and as new data is collected over the week, we flag a data point as an anomaly if its actual value is far from the forecasted value.
- The Kalman filter can handle cyclic patterns. We use the autocorrelation function (ACF) to detect the length of the cycles.
- In anomaly detection literature, detecting such spikes is known as point anomaly detection[2].

[2] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41, no. 3 (2009): 15.

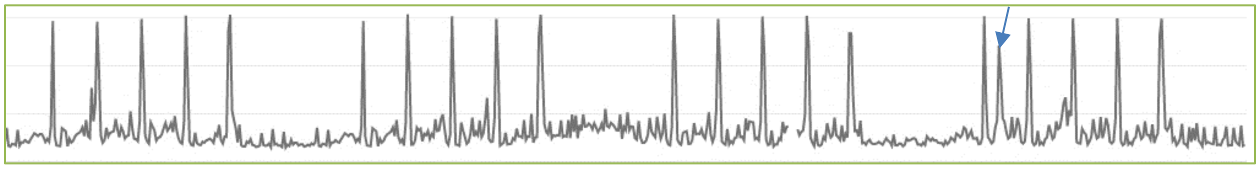


Figure 3 An example of a point anomaly that can be detected with our Kalman filter method

DBSCAN clustering to detect extended periods of abnormal behavior for a metric

- A benefit of the Kalman filter is that it can adapt to changing patterns in our data. However, not all pattern changes should be automatically considered “normal,” as they may reflect an undesirable and unexpected system change that an Ops professional should investigate.
- We use DBSCAN to compare sequences of metric values. For example, we can compare the most recent week’s pattern to each of the previous several weeks’ patterns. DBSCAN calculates a distance metric that determines whether the most recent week is anomalous compared to the previous weeks.
- In anomaly detection literature, detecting an anomalous data sequence is known as collective anomaly detection[2].

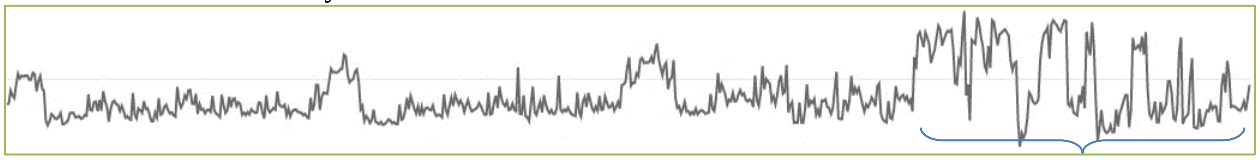


Figure 4 An example of an abnormal period that can be detected with our DBSCAN method

2 Methodology

2.1 Splunk Applications

To collect and analyze our VMware data, we used the following Splunk apps:

<u>Splunk Add-on for VMware (v3.3.2)</u>
The add-on ingests data from our VMware vSphere environment into a Splunk index. Metrics are collected at both the host and VM level, at 20-second intervals.

<u>Search & Reporting (v6.6.2)</u>
We used this app to query our data to generate summary statistics and visualizations. It also allowed us to set alerts and to create reports and dashboards.

<u>Machine Learning Toolkit (v2.3.0)</u>
The MLTK provides commands and visualizations for ML analysis of our data. Algorithms are grouped into categories of analytics, and the MLTK includes "assistants"—dashboards that guide users through the ML workflow—for each category. We used the following three assistants:

- Detect numeric outliers
- Forecast time series
- Cluster numeric events

Other available assistants are:

- Predict numeric fields
- Predict categorical fields
- Detect categorical outliers

The MLTK includes over two dozen algorithms from Python's scientific and ML libraries, such as scikit-learn and statsmodels. It is possible for users to extend the toolkit by importing other Python algorithms, but we did not experiment with this option.

2.2 Identifying high-workload VMs

Our vSphere environment includes hundreds of VMs and dozens of hosts. We began our analysis by determining the highest-workload VMs, which we expected might have performance problems and thus should be monitored more closely. Additionally, we expected that many of the highest-workload VMs would be used for MITRE's well-known applications, such as Corporate file and our intranet portal. It is especially important to monitor these VMs because poor performance would impact a large proportion of the MITRE community.

We analyzed performance over a 2-week period (6/7/17 – 6/22/17). For each of the four metrics (CPU, memory usage % and network receive / transmit rates), we ranked VMs based on their 95th percentile values.

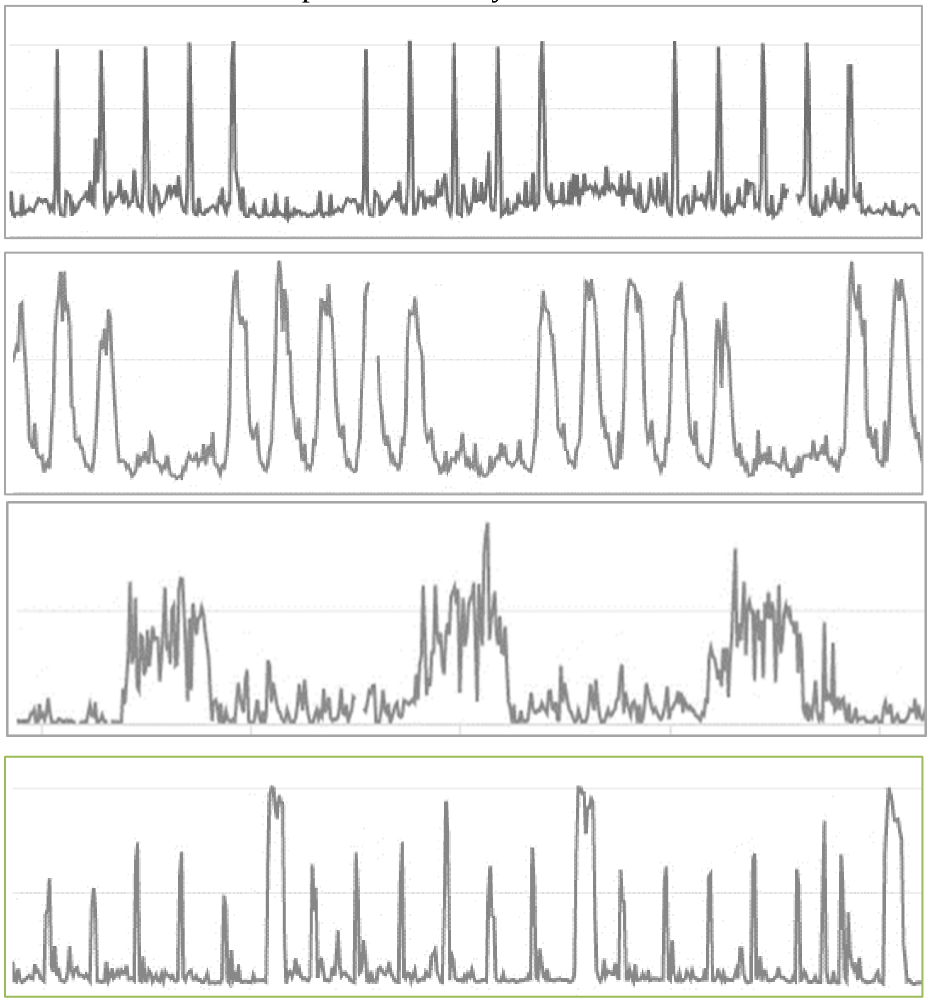
2.3 Workload patterns

The workloads of our VMs vary based on application usage and scheduled backups. With each workload corresponding to a different time series pattern, it is difficult to configure thresholds to monitor them all. (See charts below for some example patterns.) We aimed to develop an anomaly detection solution that was flexible enough to handle all the patterns, in an automated manner.

Our anomaly detection solution is based primarily on the *shape* of the data. Thus, for our patterns, we do not need to be concerned about which VM and metric are depicted.

Time series charts for a 3-week period are shown below. All data are at an hourly granularity.

Weekly patterns
Workloads rise and drop on certain days and hours of the week.



Spike on weeknights. Flat weekends

Spike during business hours (M-F 8am-6pm).

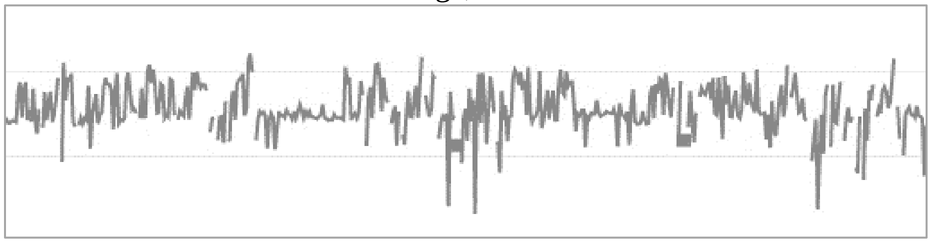
Spike on weekends

Spike every night, but the largest spike occurs on one night each week (in this case, Sunday)

Figure 5 Weekly Patterns

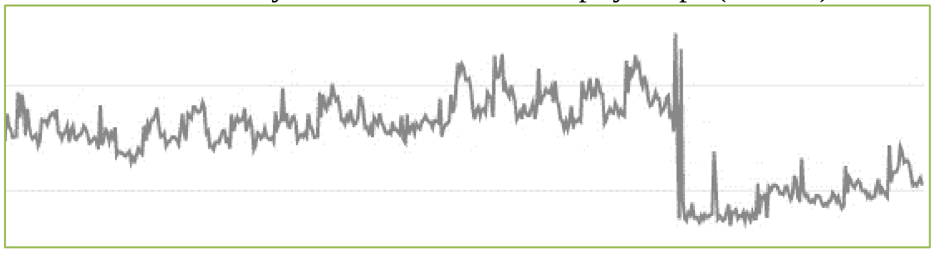
Constant value with noise

The value hovers around an average, and the variation has no detectable trend.



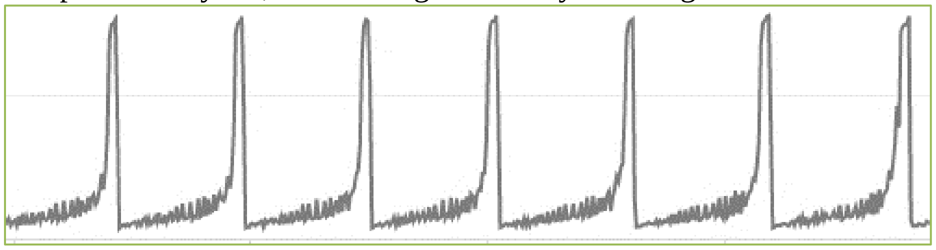
Step change

The value is relatively constant but then abruptly drops (or rises).



Changing periodicity

The pattern is cyclic, but the length of the cycle changes over time.



The cycle is initially 60 hours long but gradually lengthens to 70 hours.

Deviating pattern

The workload pattern is consistent for a certain period but then changes.

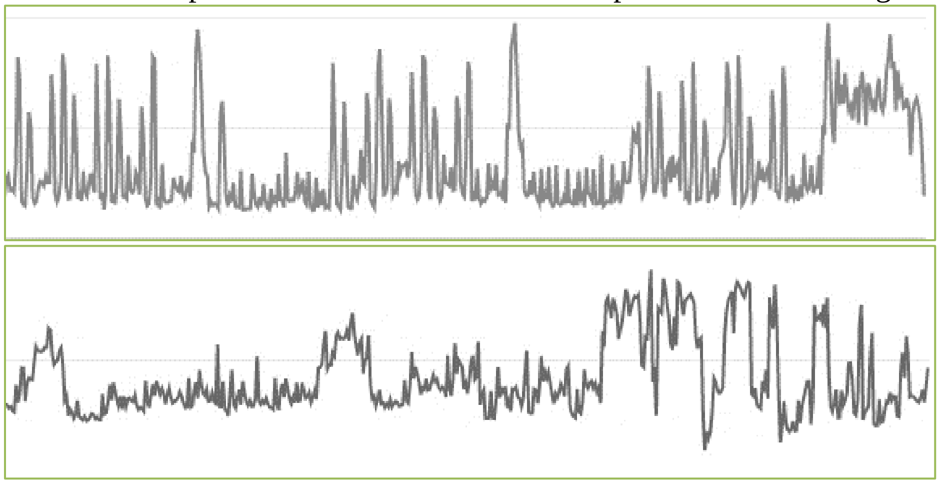


Figure 6 Non-weekly patterns

Kalman filters work best with regular pattern time series and constant periodicity. Kalman works well for "weekly patterns" and "constant with noise". DBscan compares last week to previous weeks, chunk to chunk, and is good at detecting "step changes" or "deviating patterns". Both algorithms based on weekly periodicity (168 hours) so algorithms don't work well for "changing periodicity" pattern.

2.4 Literature review of time series anomaly detection techniques

The workload patterns documented above are time series data—data that have a temporal ordering. Thus, in developing our anomaly detection solution, we relied upon principles of time series analysis to examine and model our performance metrics.

Time series analysis is an area of active research, and it is one that spans numerous fields beyond IT (e.g. economics, engineering, biology). Many of the analysis and anomaly detection techniques are domain-agnostic, meaning that they can be applied to time series data regardless of the field. There are also companies and solutions that focus specifically on monitoring and anomaly detection for IT.

We reviewed both areas (domain-agnostic and IT-specific) in search of different solutions. Our literature review included academic papers, business whitepapers, and conference presentations.

We sought methods for locating two types of anomalies

- Point anomaly- one event (i.e. a single point in time) is abnormal compared to the typical workload. This appears as a sudden spike—in either the positive or negative direction—in a time series chart. A point anomaly can be either
    - Global- abnormal with respect to the entire dataset
    - Local- abnormal given a specific context (for example, given the event's weekday and hour)
- Collective anomaly- a collection of events (i.e. a time span) is abnormal. We considered step changes and deviating patterns to be collective anomalies.

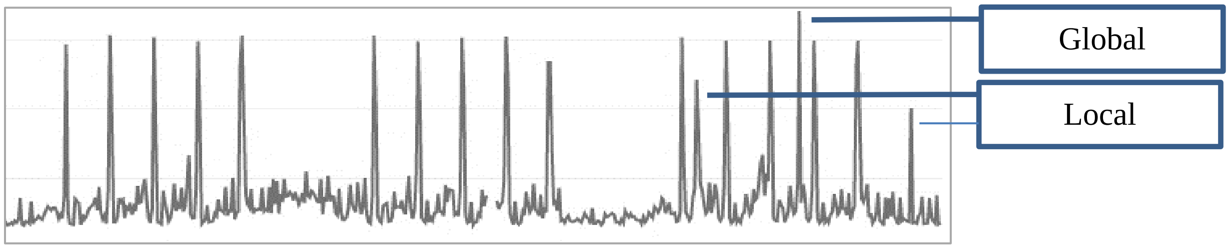


Figure 7 Global vs. local anomalies

Some of the common approaches involved

- Descriptive statistics
- Sliding windows
- Regression
- Forecasting
- Clustering

2.5 Algorithm selection and implementation

After surveying various methods for time series anomaly detection, we determined the set of algorithms that we wanted to apply to our data. Our decisions were based on the following factors

- Fit our design considerations (outlined in Section 1.3)
- Identified multiple types of anomalies (e.g. both global and local)
- Included in Splunk's MLTK or was open-source
  - We wanted to avoid purchasing proprietary solutions
  - We also wanted to utilize pre-built libraries and packages, rather than coding algorithms entirely from scratch. We emphasized this to ensure that any Ops professional—regardless of statistical or coding background—could implement our solution.

We sought advanced methods that used ML to model normal workload patterns and create anomaly thresholds. We based our solution on algorithms built-in to Splunk's MLTK. This allowed all steps of our analytics process—data collection, modeling, anomaly detection, and alerting—to be conducted within the Splunk environment.

Splunk's MLTK includes a "Detect Numeric Outliers" assistant that can be run on time series data. These methods are limited to descriptive statistics, such as standard deviation across a set of data points, which did not consider the weekly patterns in our data. Thus, we could only locate global, but not local anomalies.

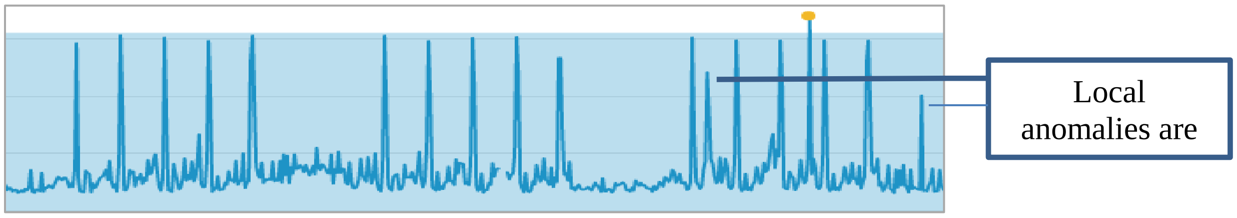


Figure 8 Anomaly threshold (blue band) based on standard deviation of the time series

This assistant behaves in a similar fashion to how IT performance monitoring and alerting software work. For this reason, it provided a baseline against which we compared to more advanced techniques.

To build custom anomaly detection solutions in Splunk, we investigated other MLTK assistants, such as "Forecast Time Series" and "Cluster Numeric Events." These algorithms allowed us to model the fluctuations in our metrics' workload patterns.

We compared the results of our Splunk anomaly detection solution to different open-source algorithms, such as those available as Python or R packages. Similar results suggested that our Splunk anomaly detection was appropriate.

Table 1 Algorithms we selected to test on our data

Software	Point Anomalies	Collective Anomalies
Within Splunk	-Kalman Filter -ARIMA	-DBSCAN
Open-source	-Twitter's AnomalyDetection R package	- Twitter's BreakoutDetection R package - HOT SAX from R package jmotif - Rare Rule Algorithm from R package jmotif

See Appendix A for details about each algorithm and links to their documentation.

For testing, we ran each algorithm on all our identified workload patterns. We evaluated the algorithms based on whether they identified anomalies accurately (i.e. equivalent to our human judgment) and in the context of the different patterns.

3 Results

3.1 Kalman filter for point anomaly detection

Overview

For each metric's time series, we applied the Kalman filter to learn the workload pattern and then forecast a week of values into the future. We then used the forecast to create an adaptive anomaly threshold. This threshold flagged an event as an anomaly when the actual value was far from the forecasted value.

Details

The Kalman filter is included in Splunk MLTK's "Forecast Time Series" assistant. (See Appendix C for details). The algorithm uses historical data to forecast future values, which fluctuate in accordance with the workload pattern. This functionality is built into the assistant.

To create our custom anomaly detector, we used these forecasts to build an anomaly threshold. We set our thresholds to be 2 standard deviations (SD) above and below each forecasted value. Because the forecasted values fluctuated to match the workload pattern, our thresholds did as well.

We can detect anomalies in (near) real-time by comparing each incoming data point with its threshold. If the actual value is beyond the threshold, an alert should be sent. To reduce the number of alerts, the thresholds can be increased from 2 SD to 3 SD above and below the forecasted value.

Automated periodicity detection

The Kalman filter can handle metrics with cyclical patterns (e.g. daily, weekly). For each model, the length of the cycle (i.e. the periodicity) should be specified; otherwise, the algorithm tries to detect it. Manually specifying the periodicity provides more accurate results, but this would be time-consuming and impractical for Ops professionals. Thus, to automate this process, we determine the periodicity using the autocorrelation function (ACF), in which the time of the greatest significant ACF value indicates the periodicity. (See Appendix A for details.) One example is below:

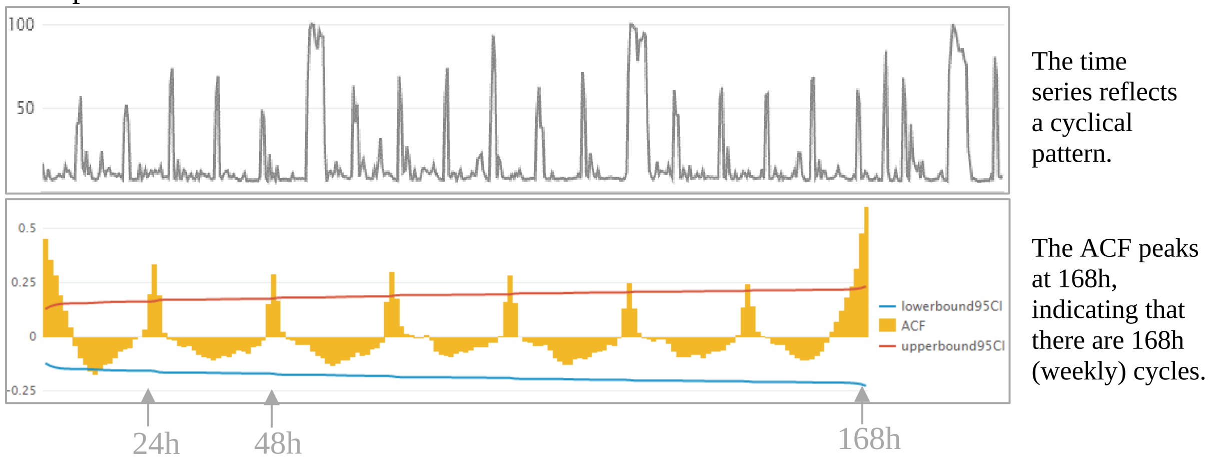


Figure 9 Automated periodicity detection using Autocorrelation function

Test results

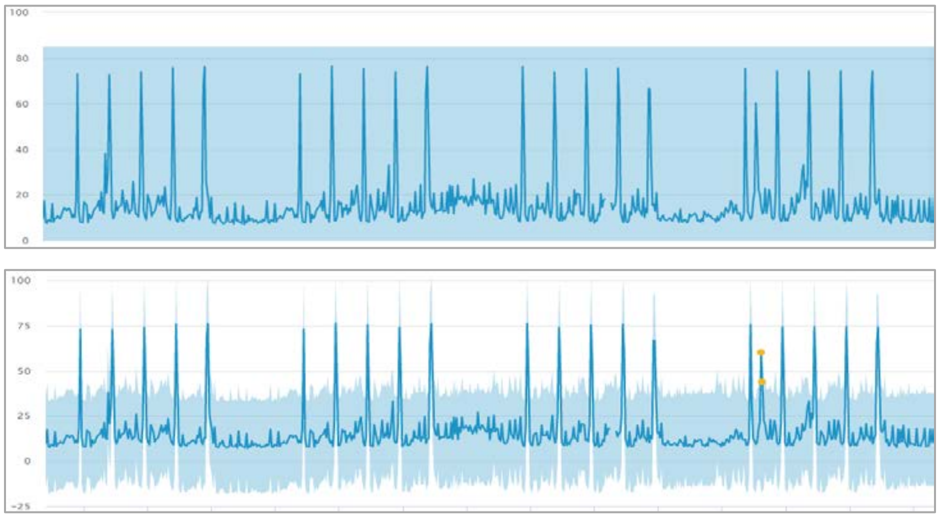
We tested our Kalman filter anomaly detection solution on our identified workload patterns (Section 2.3). We trained each model on 3 weeks of historical data, and we forecasted 1 week of values for anomaly detection. We compared our results to the number and types of outliers detected by static thresholds.

We found that our Kalman filter method successfully detected point anomalies—both local and global—on metrics with various weekly patterns or with constant values. Our method was superior to static thresholds, which often created false alerts or missed alerts. Our method did not work well for a pattern of changing periodicity, since the Kalman filter requires periodicity to be constant.

Examples

Weekly pattern- spike on weeknights

VM File share Memory usage (%)



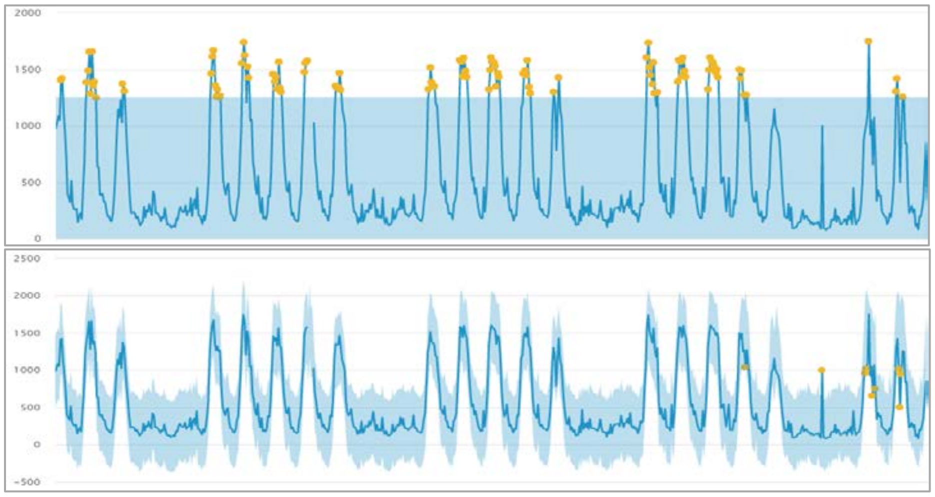
Static threshold
All values are within the threshold.
0
outliers

Adaptive threshold
Local anomalies are detected.
2
outliers

Figure 10 Weekly pattern- spike on weeknights

Weekly pattern- spike during business hours

VM abc1 - Network receive rate (KB/s)



<u>Static threshold</u>
Business hours deemed anomalous. Local anomaly is missed.

101
outliers

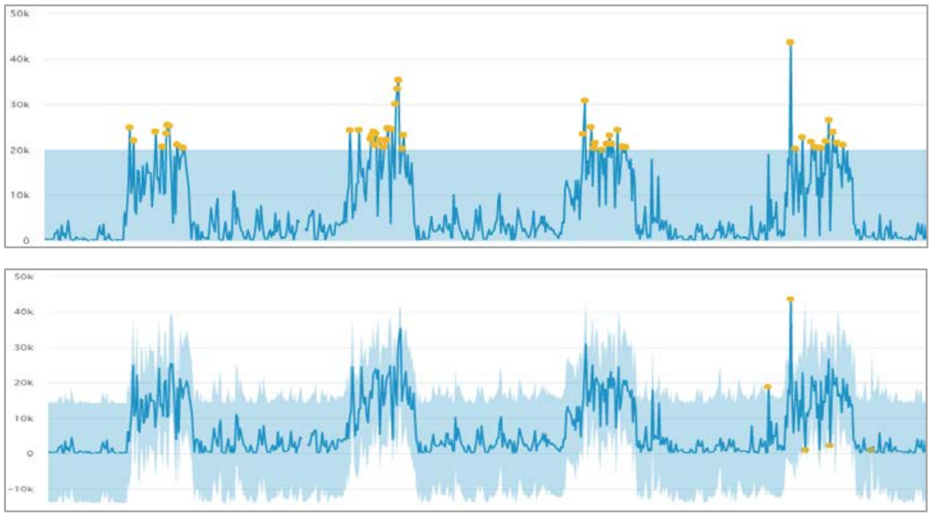
<u>Adaptive threshold</u>
Local anomalies (both positive and negative) are detected.

9
outliers

Figure 11 Weekly pattern- spike during business hours

Weekly pattern- spike on weekends

VM Sharepoint Network receive rate (KB/s)



<u>Static threshold</u>
Weekends deemed anomalous. Local anomalies are missed.

50
outliers

<u>Adaptive threshold</u>
Detects global and local (positive and negative) anomalies.

5
outliers

Figure 12 Weekly pattern- spike on weekends

Weekly pattern- spike on weeknights (largest spike on Sundays)

VM File share - CPU utilization (%)

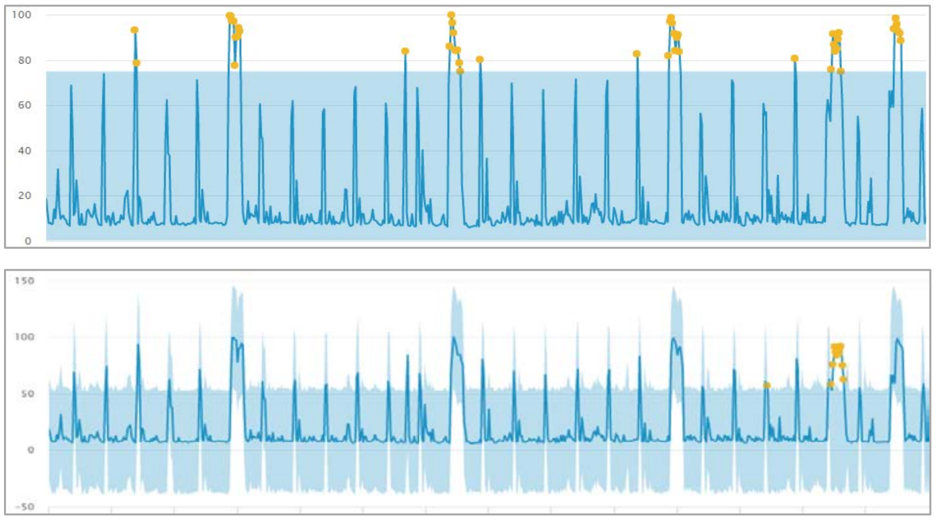


Figure 13 Weekly pattern- spike on weeknights (largest spike on Sundays)

Constant value with noise

VM abc2 - CPU utilization (%)

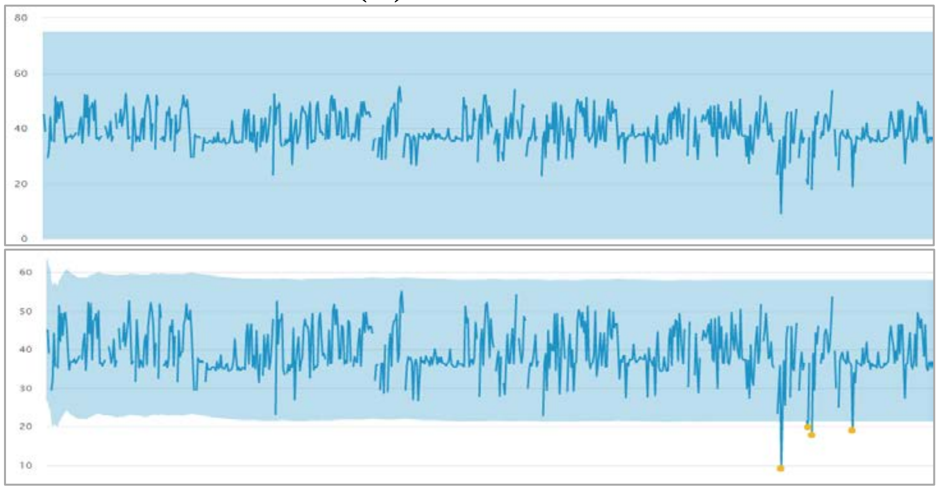
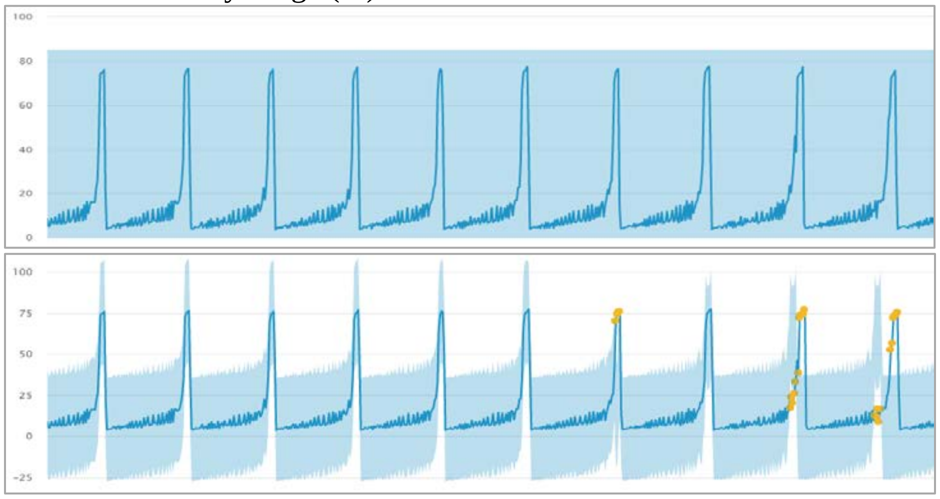


Figure 14 Constant value with noise

When there is no periodicity, the Kalman Filter forecast is a constant value.
The anomaly threshold is based on 2 SD of the *actual* value (since the *forecast* has no SD).

Changing periodicity- 60h cycles → 71h cycles

VM abc3 Memory usage (%)



Static threshold
All values are below the threshold.

0
outliers

Adaptive threshold
The Kalman filter cannot fit the data properly.

so

Figure 15 Changing periodicity- 60h cycles to 71h cycles

The Kalman filter cannot fit the data well, since it expects the periodicity to remain constant. In this example, the forecasted periodicity is shorter than the actual periodicity. For the last week of data, the chart shows that the anomaly threshold spikes before the actual metric value does.

Additional Notes

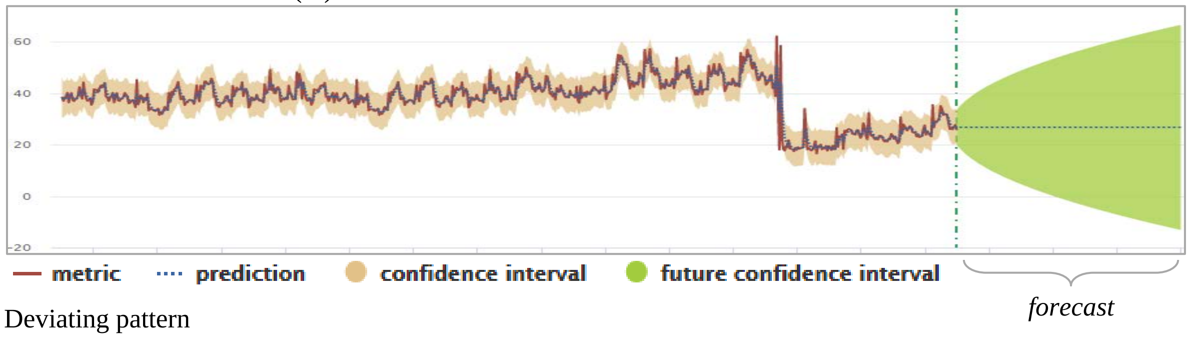
Model validation
Before we try to detect anomalies, we can validate that our forecasts match the pattern of normal behavior. Details of this process are in Appendix A.

Robustness
The Kalman filter is not robust to outliers. This means that outliers in the training data can be considered normal behavior and thus influence the forecasts. Because of this, we need to ensure that our training data is anomaly-free. This may require manual inspection.

Adapting to changing patterns
As new data is collected over time, the Kalman filter can adjust to any step changes or deviating patterns. It can automatically update its model of "normal" behavior and then generate updated forecasts.

In each of the examples below, the first three weeks are similar, but there is a change during the 4th week. The forecast reflects the new pattern.

Step change
VM abc4- CPU utilization (%)



Deviating pattern
Host esx-blade-z- CPU utilization (%)

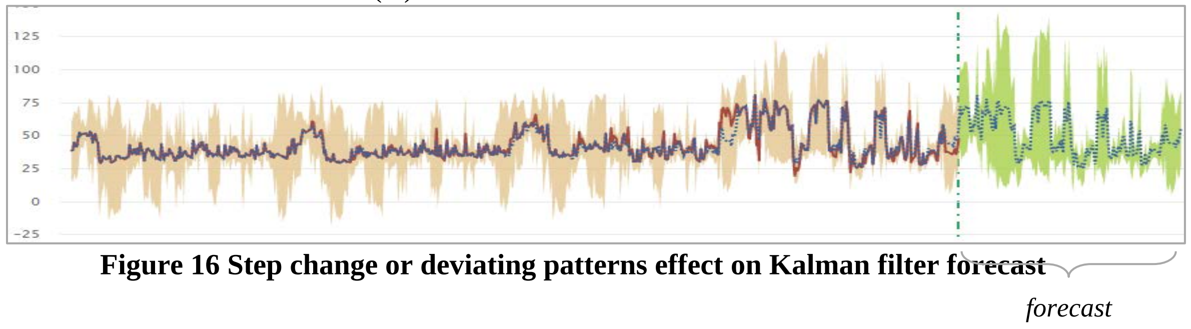


Figure 16 Step change or deviating patterns effect on Kalman filter forecast

3.2 DBSCAN for collective anomaly detection

<u>Overview</u>

We use DBSCAN to identify an anomalous week of data—for example, due to a step change or a workload pattern deviation. Using DBSCAN based on Euclidean distance, a common distance metric for time series, we determine whether the latest week's pattern is dissimilar to each of the previous weeks' patterns.

<u>Details</u>

DBSCAN is a clustering algorithm that we use to build a custom anomaly detection solution. It is included within Splunk MLTK's "Clustering numeric events" assistant. We consider each week to be a data point, and we use DBSCAN to cluster the weeks. Weeks that have similar patterns should cluster together. A week that is dissimilar will not be assigned a cluster, and it will be flagged as an anomaly.

While we primarily used DBSCAN to compare week-over-week patterns, it is also possible to compare sequences of a different length. For example, if a metric had a 100-hour periodicity, we could compare 100-hour time windows to one another.

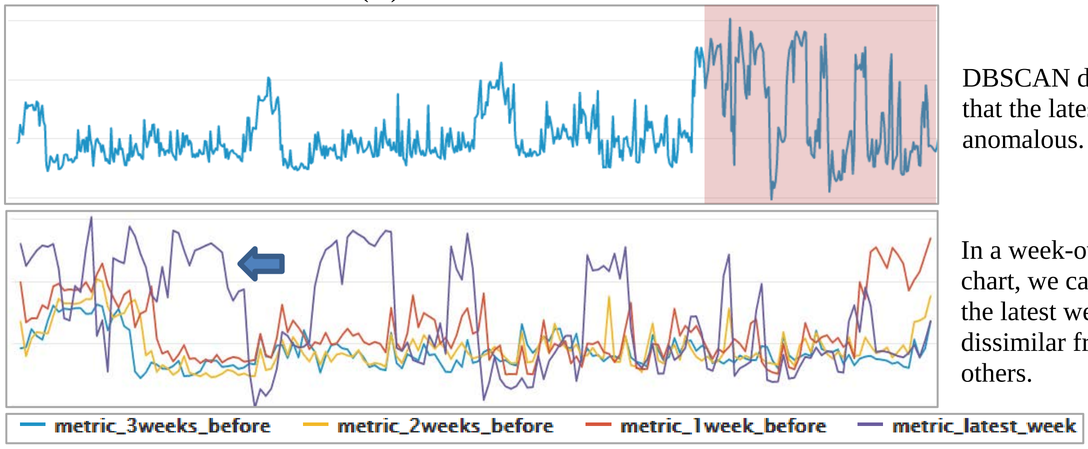
See Appendix A for further details.

<u>Test results</u>

We tested our DBSCAN anomaly detection on time series with a similar pattern for the first 3 weeks but then a pattern change for the 4th week. For our two examples, DBSCAN determined that the 4th week was anomalous.

<u>Examples</u>

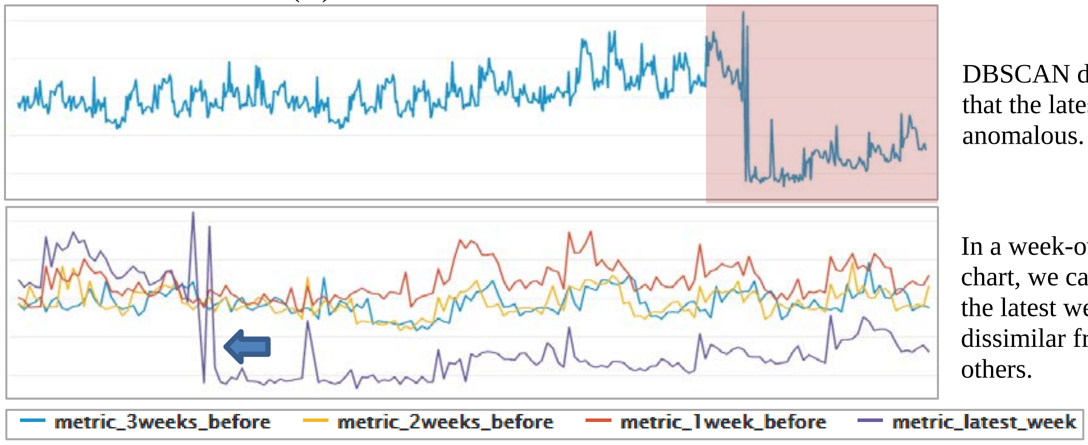
Deviating pattern
Host esx-blade z CPU utilization (%)



DBSCAN determines that the latest week is anomalous.

In a week-over-week chart, we can see that the latest week is dissimilar from the others.

Step change
VM abc4- CPU utilization (%)



DBSCAN determines that the latest week is anomalous.

In a week-over-week chart, we can see that the latest week is dissimilar from the others.

Figure 17 Step change or deviating patterns detection with DBSCAN

3.3 Comparison to other techniques

To validate the accuracy and flexibility of our Kalman filter and DBSCAN anomaly detection, we compared our results to other Splunk algorithms and different open-source R packages.

While there are several R packages built specifically for anomaly detection on time series, we selected those that seemed to be the newest and most popular (according to academic citations and in discussions on the website Cross Validated – Stack Exchange). We expected that these packages would provide the highest benchmark for open-source anomaly detection.

We tested the R packages on each of our workload patterns to see if different anomalies would be detected. We also evaluated how the R packages compared to our solutions in regard to our design considerations (Section 1.3).

We concluded that our Kalman filter and DBSCAN anomaly detection provided results that were comparable to popular open-source solutions. We found that our solutions often identified the same anomalies. We also considered our solutions to be similarly flexible, adaptive, and automated.

Because our solutions use algorithms built-in to Splunk, they are easy to implement. We do not have to export vSphere data for analysis with another program, nor do we have import any non-MLTK algorithms into Splunk. With our solutions, we can conduct our entire analytics process—data collection, modeling, anomaly detection, and alerting—within the Splunk environment.

See Appendix A for details about each algorithm and links to their documentation.

3.3.1 Point anomaly detection

ARIMA

The Splunk MLTK "Forecast Time Series" assistant provides a second algorithm, Autoregressive Integrated Moving Average (ARIMA). In theory, we could have used ARIMA rather than the Kalman filter to generate our forecasts. However, the MLTK implementation of ARIMA has several limitations that make the Kalman filter preferable. Notably, the MLTK Kalman filter is more automated and flexible, and it can model more complex patterns. From a theoretical standpoint, statisticians also agree that the Kalman filter is a superior algorithm. More details are in Appendix A.

Twitter's AnomalyDetection R Package

Twitter provides an open-source R package for detecting point anomalies, both local and global. Twitter's algorithm is called Seasonal Hybrid ESD (S-H-ESD).

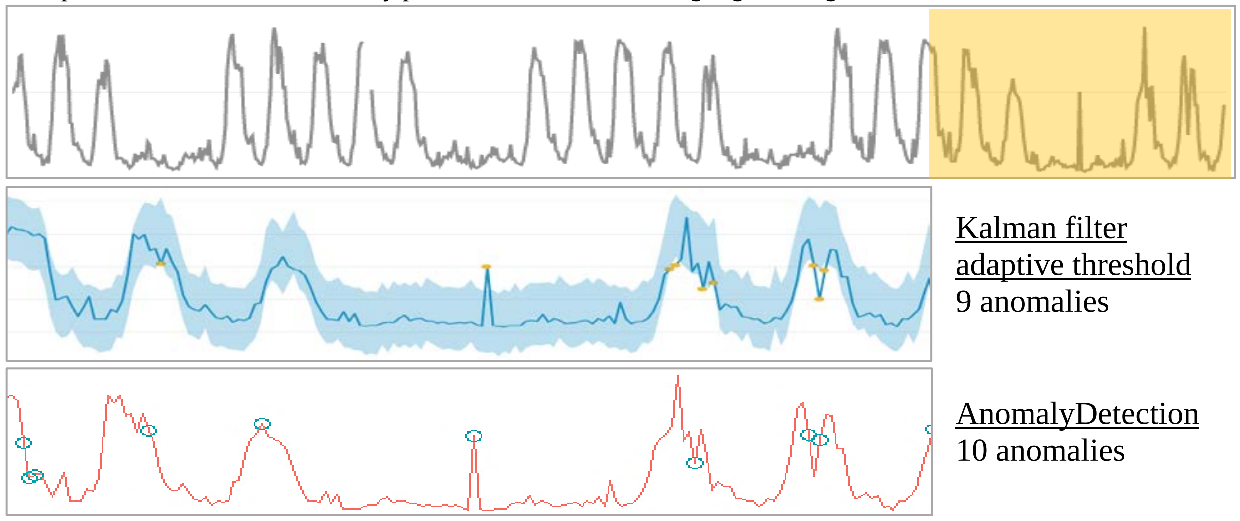
After testing, we found that AnomalyDetection successfully detected local and global anomalies on our workloads with weekly patterns. It also detected anomalies in a non-periodic, constant-valued time series. Like our Kalman filter approach, it could not be applied to a time series with changing periodicity, since it requires the user to specify a single periodicity.

We judged that our Kalman filter adaptive thresholds worked just as well as AnomalyDetection. The methods both seemed to be equally flexible, automated, and complex (Section 1.3). We note that AnomalyDetection is intended for retroactive time series analysis, while Kalman filter approach can detect anomalies in (near) real-time.

As another concern, we found that AnomalyDetection detected a higher number of anomalies, many of which we considered false positives. This occurred with AnomalyDetection's default parameters. It is possible to change the parameters so that fewer anomalies are returned.

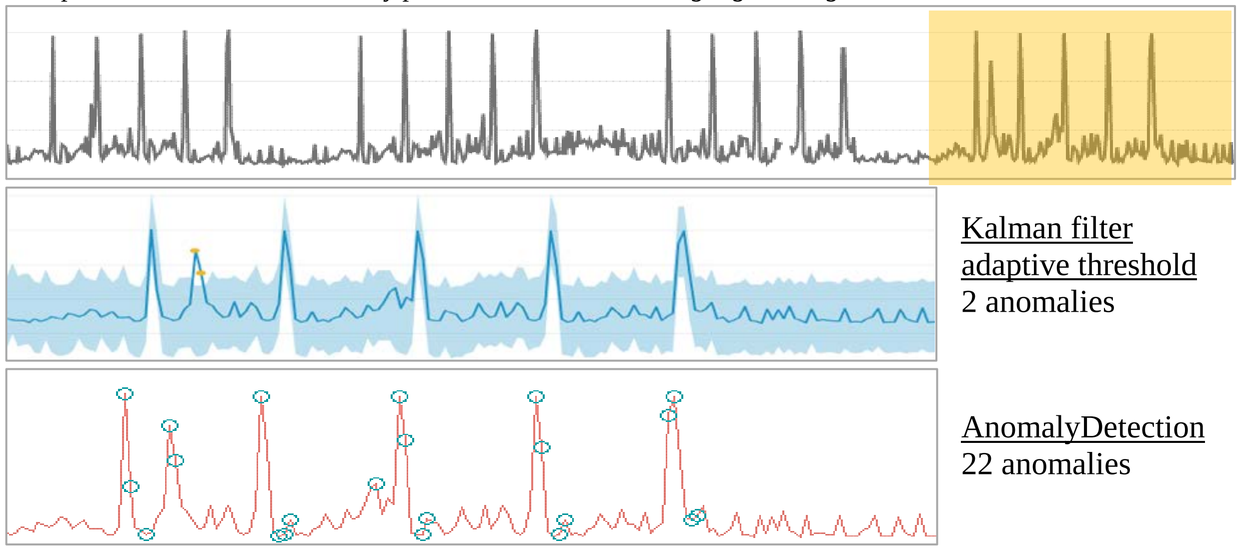
A couple test examples are below. We return only anomalies from the last week of data. We maintained the default parameters of AnomalyDetection, except for allowing the detection of both positive and negative anomalies (the default is to detect positive anomalies only).

Example 1: time series with a weekly pattern. The last week is highlighted in gold



Both methods detect local anomalies well. There are 4 data points that are identified by both methods.

Example 2: time series with a weekly pattern. The last week is highlighted in gold



Both methods appropriately detected 2 local anomalies, but AnomalyDetection returned multiple other locations, which we did not agree were anomalous.

Figure 18 Compare Kalman filter & AnomalyDetection with a weekly pattern

3.3.2 Collective anomaly detection

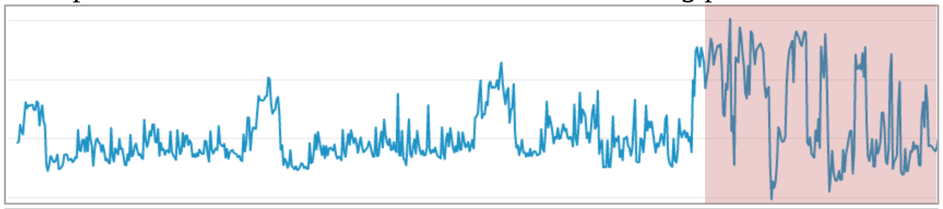
Twitter's BreakoutDetection R Package

Twitter also has an open source package for detecting breakouts. Twitter defines two main types of breakouts: (1) a mean shift (i.e. step change) and (2) a ramp up (i.e. gradual increase). The underlying algorithm is called E-Divisive with Medians (EDM).

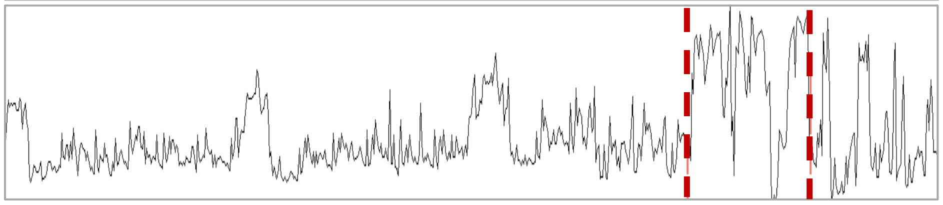
During testing, we found that BreakoutDetection was able to identify deviating patterns and step changes. Whereas our DBSCAN approach flagged collective sequences (e.g. a week) as anomalous, BreakoutDetection was able to pinpoint exact locations of the pattern changes.

BreakoutDetection worked well with its default parameters. We changed only a single parameter, which allowed us to detect multiple anomalies across the time series, rather than just one.

Example 1: time series whose 4th week has a deviating pattern

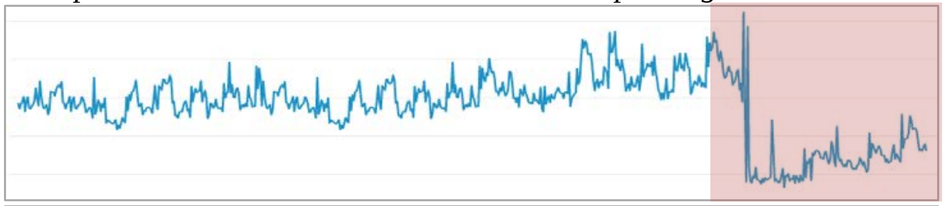


DBSCAN
Week in red is anomalous.

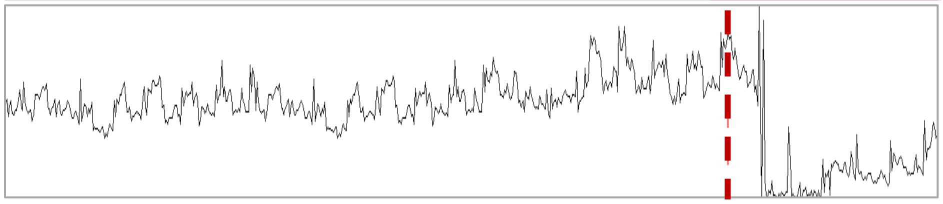


BreakoutDetection
2 anomalies are detected.

Example 2: time series whose 4th week has a step change



DBSCAN
Week in red is anomalous.



BreakoutDetection
1 anomaly is detected.

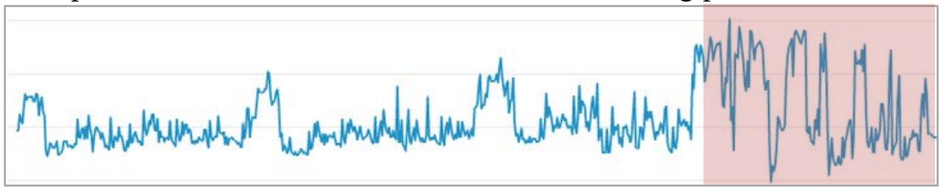
Figure 19 Compare DBSCAN & BreakoutDetection on weekly pattern change

HOT SAX and RRA, implemented in the R package "jmotif"

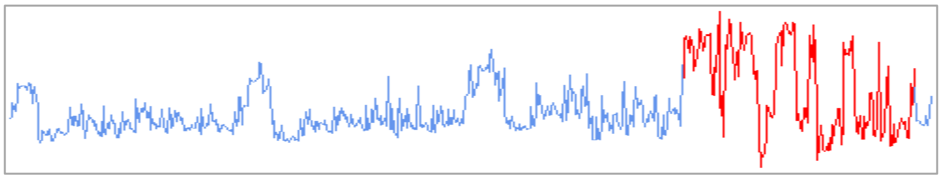
Jmotif is a time series analysis package based on Symbolic Aggregate Approximation (SAX). We tested two of jmotif's algorithms for identifying time series "discords" (i.e. collective anomalies): HOT SAX and RRA.

We found that HOT SAX and RRA could accurately detect step changes and deviating patterns in our time series. Compared to DBSCAN, an advantage of HOT SAX and RRA is that they analyze sliding windows and thus can identify anomalies anywhere along the series. When we want to return the most anomalous week of data, HOT SAX and RRA can identify a 7-day time frame with any start date. In contrast, DBSCAN breaks the time series up into non-overlapping windows. The collective anomaly must match one of these pre-defined time frames.

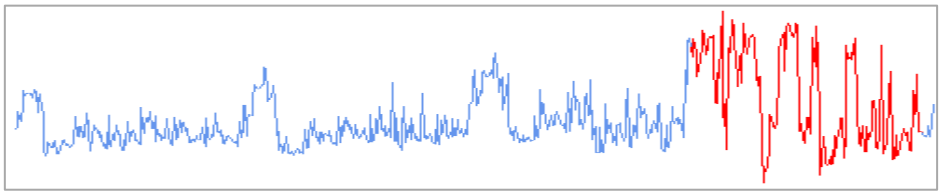
Example 1: time series whose 4th week has a deviating pattern



DBSCAN
Week in red is flagged as anomalous.

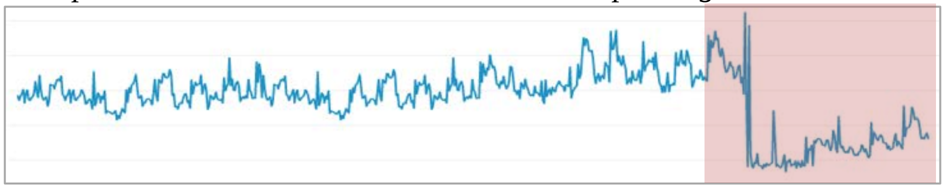


HOT SAX
Week in red is flagged as anomalous.

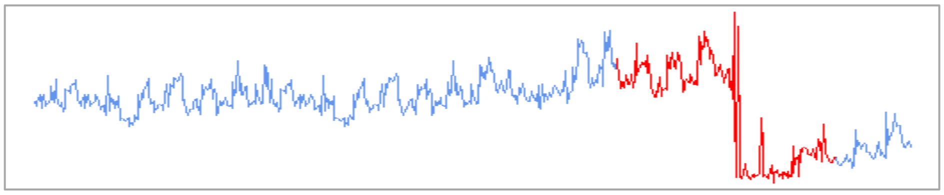


RRA
Week in red is flagged as anomalous.

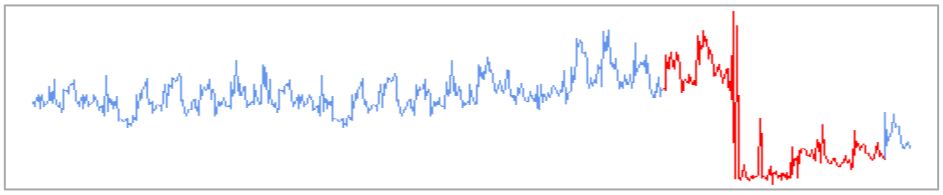
Example 2: time series whose 4th week has a step change



DBSCAN
Week in red is flagged as anomalous.



HOT SAX
Week in red is flagged as anomalous.



RRA
Week in red is flagged as anomalous.

Figure 20 Compare DBSCAN, HOT SAX & RRA on weekly pattern change

3.4 Proposed workflow

Our complete anomaly detection solution combines our Kalman Filter and DBSCAN methods. It allows us to detect both point and collective anomalies in near-real time.

To ensure our models reflect the most up-to-date pattern of "normal" behavior, we retrain our models every week. For retraining, we use only the most recent data (including the data that were newly-collected). This allows our system to adapt to workload patterns that change over time.

We detail the weekly workflow below, using one metric as an example:

1. Gather the most recent 3 weeks of data
   This data provides a baseline of "normal" behavior. The series should be anomaly-free.

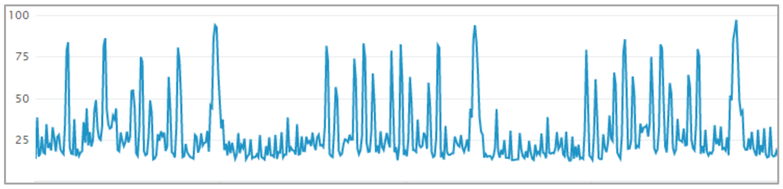


Figure 21 Three weeks of normal baseline behavior

2. Determine the cycle length with ACF
   Use the ACF to detect whether the metric has a cyclical pattern, and if so, what the periodicity is.

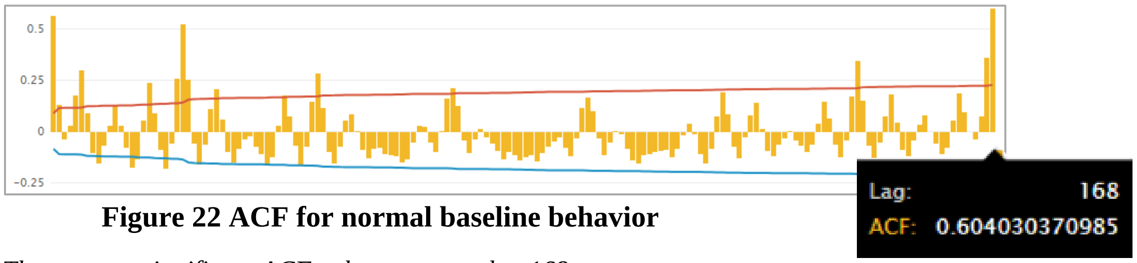


Figure 22 ACF for normal baseline behavior

The greatest significant ACF value occurs at lag 168.
This indicates that the pattern has a weekly periodicity.

3. Train the Kalman Filter model and forecast a week of values
   The results from ACF determine the appropriate periodicity to use for the Kalman Filter. Train the Kalman Filter on the historical data, and use its learned pattern of "normal" behavior to predict a week of future values.

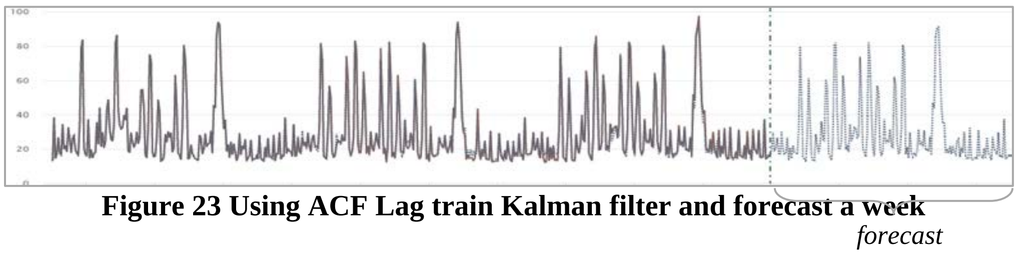


Figure 23 Using ACF Lag train Kalman filter and forecast a week

4. Create the anomaly threshold
   Set a threshold that is 2 standard deviations above and below the forecast. We can increase the threshold (e.g. 3 standard deviations) if we wish to flag fewer anomalies.

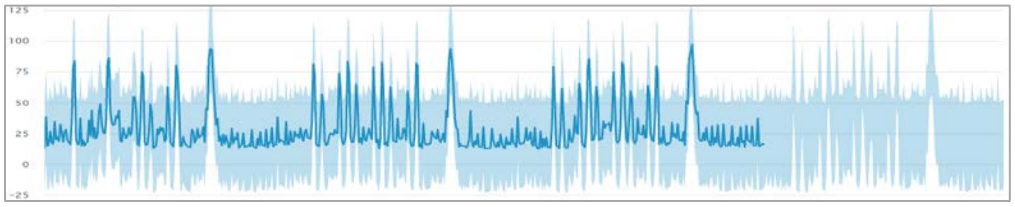


Figure 24 Create the anomaly threshold

The dark blue line depicts the historical data. The light blue band depicts the anomaly threshold.

5. Detect anomalies throughout the week
   As data are collected in near-real time, send an alert if a value falls outside the threshold

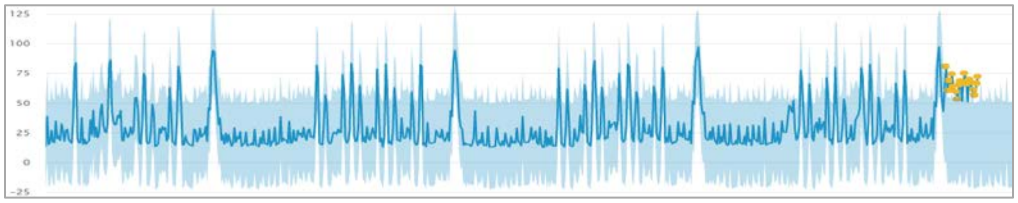
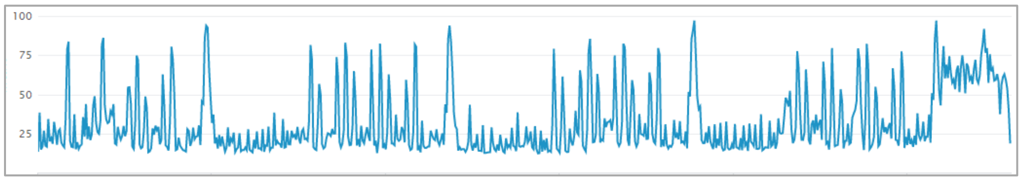


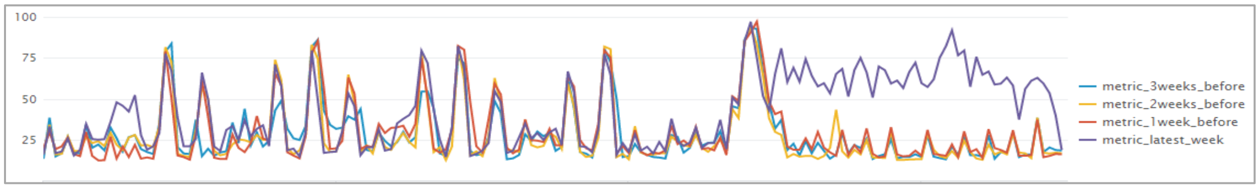
Figure 25 Detect anomalies throughout the week

Mid-week results of the anomaly detection system. New data points have been collected, and some values have fallen above the threshold.

6. Run DBSCAN to determine whether the week had an anomalous pattern
   With individual real-time alerts, we might not realize that altogether, multiple alerts may signify a collective pattern change. We thus use DBSCAN to analyze the week retroactively.



Time series at the end of the week. The last week has a pattern that is dissimilar from the previous three.



Week-over-week patterns. DBSCAN will flag the latest week as anomalous.

Figure 26 With DBSCAN determine if last week was anomalous

7. <u>Remove the effect of anomalies for the next round of predictions</u>
   If the week was flagged as anomalous, we must determine whether the pattern change was either
   (A) long-lasting and expected
   OR (B) temporary and unexpected

   For example, case (A) might have occurred during a planned reconfiguration of the vSphere environment by the Ops team. In this case, we believe that the new pattern should continue and thus become the new baseline for “normal” behavior. When we retrain the Kalman Filter, it automatically adapts to this new pattern.

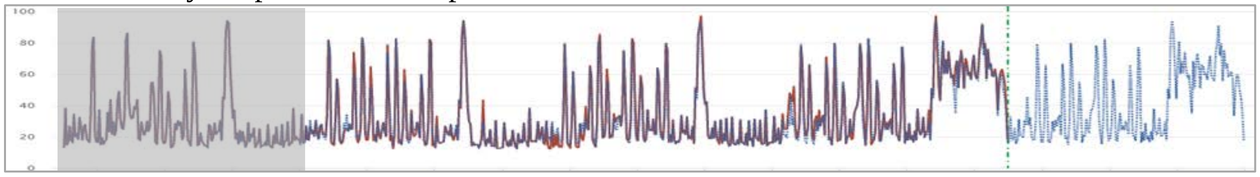


Figure 27 Last week long lasting or expected - add to Kalman filter

When we retrain the Kalman Filter (using only the 3 most recent weeks of data), it automatically adapts to the pattern change.
Note: the grey band denotes data that are no longer used for retraining

Case (B) represents a true anomaly, such as one caused by a system fault. In this case, we do not want to use this pattern to represent “normal” behavior. We want to prevent this anomaly from distorting our Kalman Filter model. When we retrain the Kalman Filter, we replace the week’s actual values with its predicted values, so that the training data is anomaly-free.

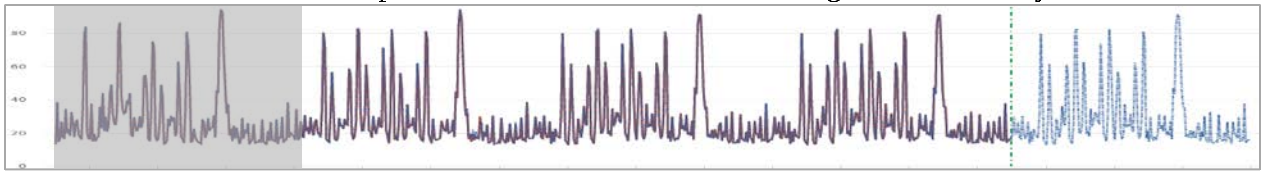


Figure 28 Last week was anomalous - Kalman filter uses forecast

We replace the anomalous week with its predicted values before retraining the Kalman Filter (using only the 3 most recent weeks of data).
Note: the grey band denotes data that are no longer used for retraining

Even if the week was not flagged as anomalous, we still need to replace anomalous data points with their predicted values. We can then retrain the Kalman Filter.

Some considerations

- Splunk’s Kalman Filter only requires 2 cycles of historical data to generate a forecast. We base our forecast on 3 weeks of data to increase our confidence in the model.
  - It is possible to increase the length of historical data (i.e. to 4+ weeks). However, this may be unnecessary because the Kalman Filter will nevertheless base its forecast more heavily on the more recent weeks. Additionally, we expect our metrics in our vSphere environment to change over long timeframes (e.g. month-to-month), so data from several weeks ago might not match the current workload.
- We can expect certain timeframes to be anomalous—for example, holidays or planned system outages. We can turn off alerts for these timeframes. At the end of the week, we need to replace their actual values with their predicted values before retraining the Kalman Filter.

4 Conclusion

In this report, we present an anomaly detection workflow for the monitoring of VM and host performance metrics. Our solution includes two machine learning algorithms: the Kalman filter for point anomaly detection and DBSCAN for collective anomaly detection. These algorithms are packaged as part of the Splunk MLTK.

4.1 Strengths and limitations of our solution

Strengths
We built an anomaly detection solution that fit our design considerations (Section 1.3):

Automated- Our Kalman Filter and DBSCAN methods can run without any manual tuning. They automatically learn patterns of “normal” behavior and then identify instances that are atypical.

Complex- We create adaptive anomaly-detection thresholds that reflect varying workload patterns (e.g. based on time-of-day or weekday)**.** Our solution can also detect and adapt to pattern changes.

Adaptive- We retrain our models weekly to utilize newly collected data. We use DBSCAN to detect when a metric’s workload pattern changes, and the Kalman Filter models update themselves automatically to reflect the most recent pattern

Flexible- We verified our solution on several workload patterns. We believe that we can extend our solution to VMs and hosts across our vSphere environment.

Streaming- With the Kalman Filter, we can detect point anomalies in (near) real-time. With DBSCAN, we can detect collective anomalies week-by-week.

Moreover, because our solution uses algorithms built-in to the Splunk MLTK, we can execute our full analytics process—data collection, modeling, anomaly detection, and alerting—in our Splunk environment.

Limitations
We expect that our solution can extend across VMs and hosts, for a variety of performance metrics. However, our testing is non-exhaustive, since there are an infinite number of potential workload patterns. When we begin to implement our solution across our vSphere environment, we expect to encounter a few patterns that cannot be modeled (such as the workload with changing periodicity). Ops professionals will need to handle these on a case-by-case basis.

Furthermore, while our solution can detect anomalies, it is still up to Ops professionals to determine whether the anomalies are problematic. Human judgment and domain knowledge are still required for proper system management and root-cause analysis.

5 Next Steps

There are opportunities to test our solution further and to expand its functionality. We detail potential next steps below:

5.1 Further research

5.1.1 Experiment with time granularities

The Splunk VMware add-in collects metric data at 20-second intervals. Our results above involve time series in which the 20-second data points are averaged into 1-hour aggregates. This reduced the dimensionality of our data, dramatically reducing computational processing time.

We would like to investigate other time granularities for forecasting and anomaly detection:

Option 1: Forecast with aggregated hourly data

For real-time alerting, after we build a model and forecast based on 1-hour timespans, we can flag anomalies by:

1) Creating 1-hour aggregates of the incoming data and comparing these values to the 1-hour forecasts

OR

2) Checking whether each incoming 20-second data point matches its corresponding 1-hour forecast. Alternatively, we could aggregate the 20-second data points into sub-hourly timespans (e.g. 10 minutes, 30 minutes).

The first choice may require fewer computational resources and may result in a smaller, more manageable number of alerts. However, it may prevent Ops professionals from responding to problems in the timeliest manner, and some anomalies (e.g. a spike that only lasts for 20 seconds) may be smoothed over in 1-hour aggregates. We need to determine which option is more appropriate for our VM environment and for our Ops team.

Option 2: Forecast with sub-hourly data

It is also possible to build models and forecast on timespans that are less than 1 hour. This may be desirable for certain patterns (e.g. if spikes normally occur in the last 15 minutes of every hour, any spikes that occur in the first 45 minutes should be flagged as anomalous). Again, we must determine the appropriate tradeoffs between model complexity, computational processing speed, and alert frequencies and counts.

5.1.2 Detect when anomalies occur simultaneously for related metrics

Although our results only considered high-workload VMs and their patterns for four metrics (CPU usage, memory usage, network receive and transmit rates), our anomaly detection workflow can be used for other performance metrics, dozens of which are tracked by vSphere. While we could create models and detect anomalies for all of these metrics, this might result in an unmanageably high number of alerts for Ops professionals to sift through.

Rather than triggering an alert for every single metric anomaly, it may be more effective to trigger composite alerts, for when there are simultaneous anomalies across related metrics. For example, the following network problems are related

1. Low receive rates
2. High number of dropped packets
3. High latency

Rather than triggering multiple alerts—one for each metric—we could aggregate them into one alert, potentially titled "network problem."

5.1.3 Analyze end-to-end application performance

Many of our VMs run business applications, such as Corporate file share and intranet portal . We can import application data from BCO into Splunk, allowing us to obtain end-to-end visibility into our virtualization system.

Application usage
We could expect VM workloads to increase in tandem with application usage. For example, if application usage suddenly spikes, CPU usage could as well. In this case, the CPU spike might not be indicative of a problem requiring further investigation. In contrast, if application usage is normal but CPU usage spikes, there may be an issue with vSphere's resource allocation.

Application response times
Proper VM and host performance is necessary for ensuring employees' access to business-critical applications. Using our anomaly detection workflow, we can detect when applications have abnormally slow response times. To assist Ops professionals with determining the root-cause of a slowdown, we can report which performance metrics have simultaneous anomalies. This could indicate, for example, that one delay was caused by CPU problems, while another was caused by the network.

IT Ops Performance Model
The two examples above reflect how VMs and their applications can be monitored concurrently. The goal of end-to-end analysis is to connect all components of the IT Ops Performance Model, as shown below. ECIS must be able to understand and model these relationships to enable proper management and problem resolution.

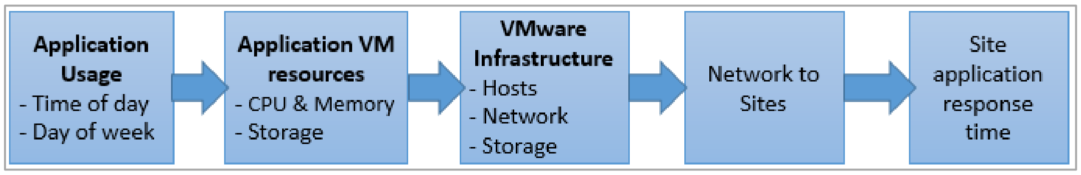


Figure 29 End to end application performance

5.1.4 Compare our results to Splunk ITSI anomaly detection

Splunk IT Service Intelligence (ITSI) is a product that uses ML for real-time IT monitoring and troubleshooting. It can be connected to the Splunk Add-On for VMware. Like our solution, its anomaly detection system employs ML to determine normal behavior, identify anomalies, and set adaptive thresholds.

5.2 Implementation in a production environment

The results of this report were generated from historical data on a select few metrics and VMs. As a next step, we would like to test our anomaly detection workflow in a production environment, where we detect anomalies in (near) real-time.

We would like to determine how well our Splunk environment can handle (near) real-time monitoring. Because our Splunk environment has limited resources, it may be infeasible to monitor the entire vSphere system, which includes hundreds of VMs and dozens of metrics. We may need to limit the number of VMs and metrics under analysis, and we may need to adjust the granularity of our data (e.g. 1-hour aggregates rather than 20-second data points).

To manage the scope of our initial implementation, we could begin by monitoring VMs tied to the most business-critical applications (i.e. Tier 1 applications). If successful, we could then scale up further, while checking whether increased monitoring leads to Splunk performance degradations.

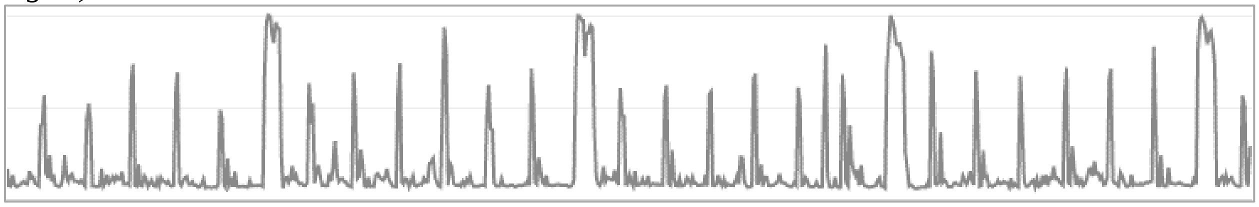
Appendix A Algorithm Descriptions

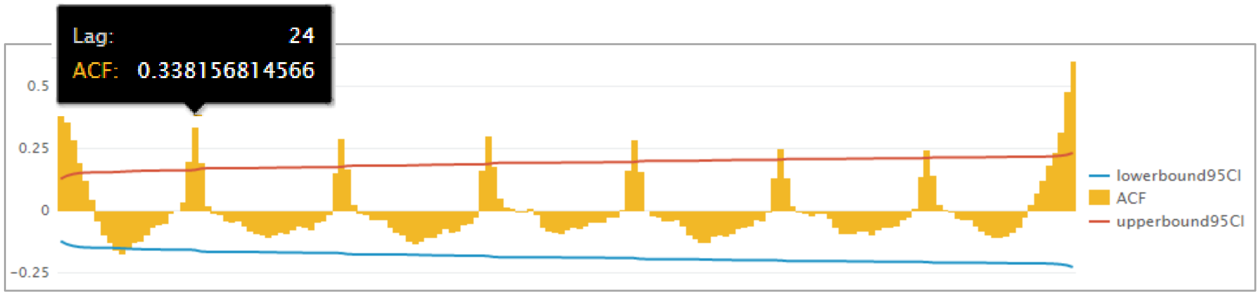
A.1 Autocorrelation Function (ACF)

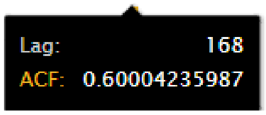
The autocorrelation function (ACF) computes the correlation between a series and a lagged version of itself. For example, we can determine the correlation between a value X_t and its previous lags X_{t-1}, X_{t-2}, X_{t-3}, etc. On an ACF chart, the x-axis plots the lag number, while the y-axis plots the autocorrelation value. Confidence bands indicate which autocorrelation values are statistically significant.

In time series analysis, the ACF is used to detect temporal patterns. We specifically use the ACF to determine a series' periodicity (i.e. cycle length). A series with a cyclical pattern should have alternating positive and negative autocorrelation values. We assume that the periodicity is the lag at which the greatest positive, significant autocorrelation occurs. (We ignore the first set of positive lags, since we assume that the first few lags are always highly correlated, regardless of whether there is a pattern).

Example #1: time series with weekly periodicity
The value spikes every night, with the highest spike occurring once a week (in this case, Sunday nights).

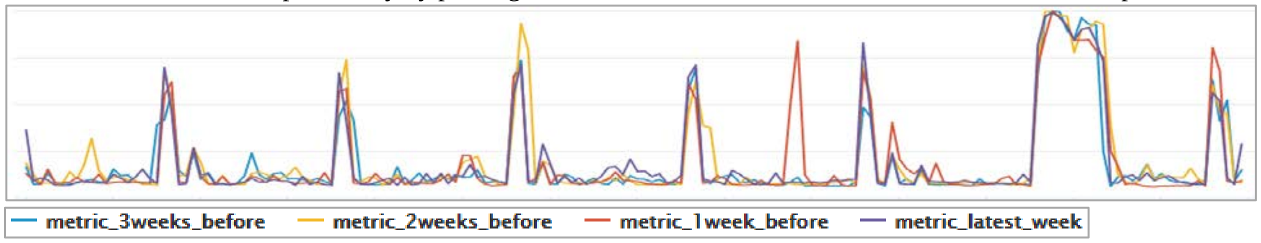




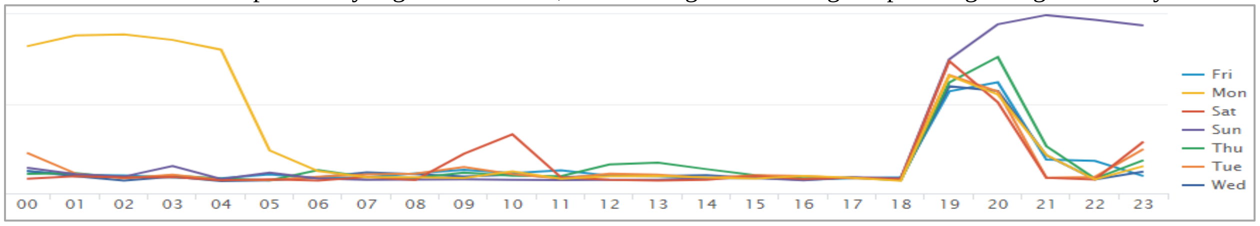


In a chart of ACF from lags 1 through 168, the first positive spike occurs at 24. There continue to be significant positive spikes for every multiple of 24h (e.g. 48h, 72h). With a 24h periodicity, we capture our metric's time-of-day pattern (e.g. nightly spikes every day of the week). However, the greatest positive spike occurs at lag 168, so we consider this to be the best periodicity. With a 168h periodicity, we identify both a time-of-day and day-of-week pattern. We can now capture how the metric has its highest spikes on Sunday nights and smaller spikes on all other nights.

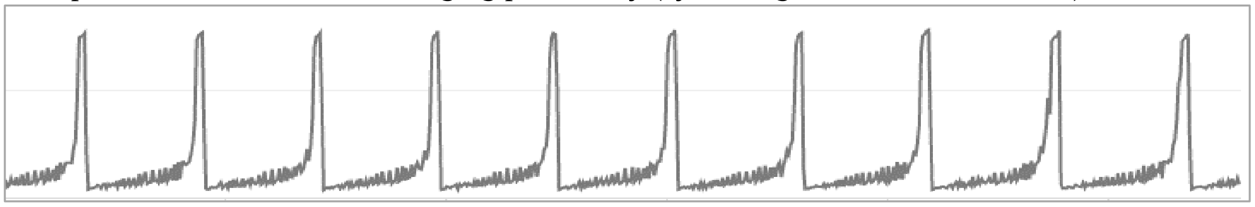
We can validate our 168h periodicity by plotting a week-over-week time chart. All weeks have a similar pattern.

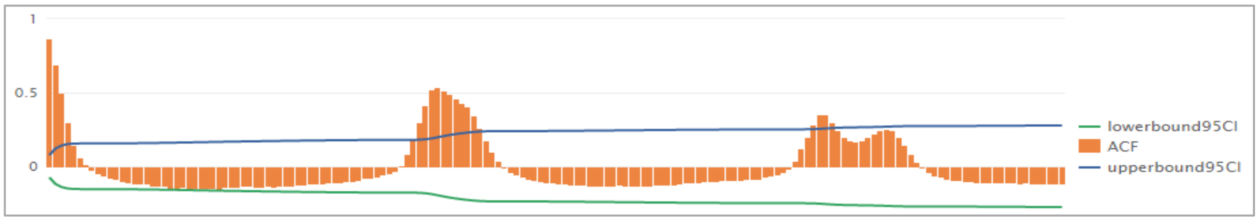


To determine the specific weekly pattern, we can plot the mean values for each hour and day of the week. Here, we confirm that there is a spike every night of the week, with the highest and longest spike beginning on Sunday.



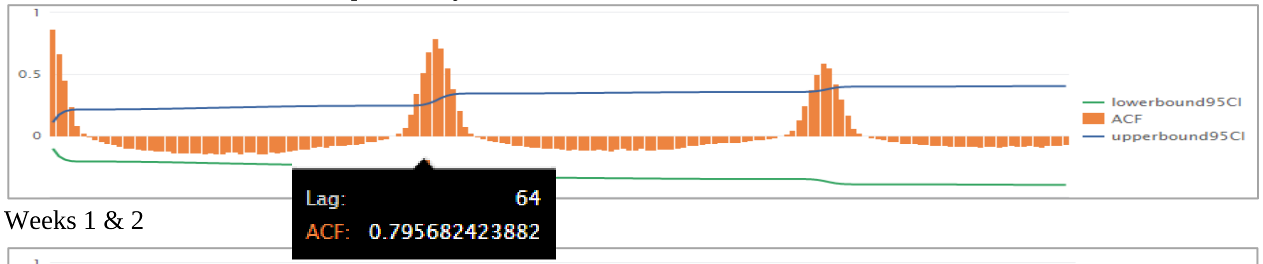
Example #2: time series with changing periodicity (cycle length increases over time)



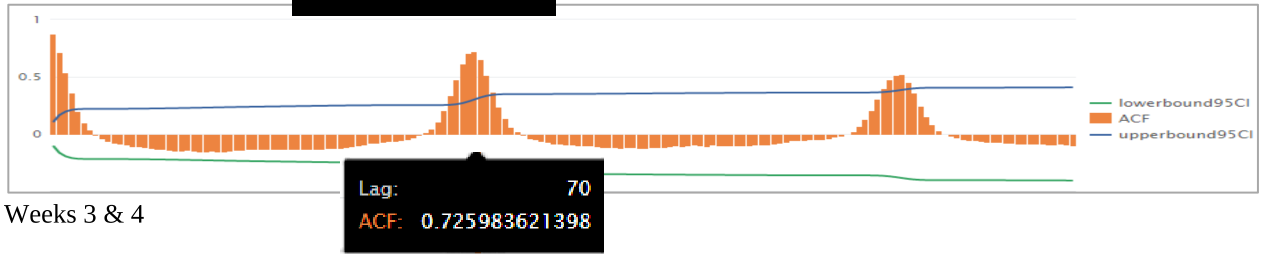


The plot above shows the ACF for lags 1 through 168, across all four weeks of the time series.

If we create separate ACF plots for weeks 1-2 and weeks 3-4, we see that the autocorrelation spikes at different values. This means that the periodicity changes over time. Thus, this series will not work well with the Kalman filter, which assumes a constant periodicity.

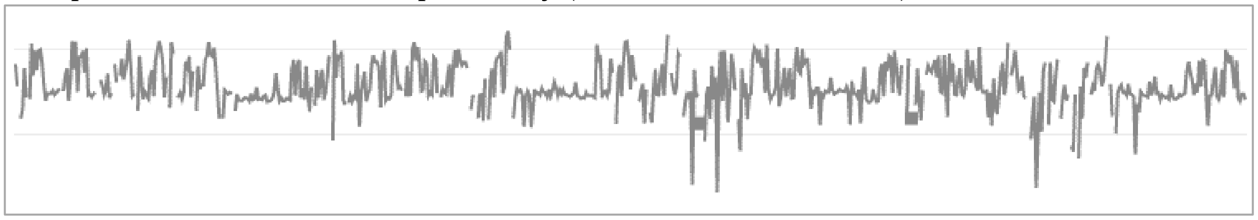


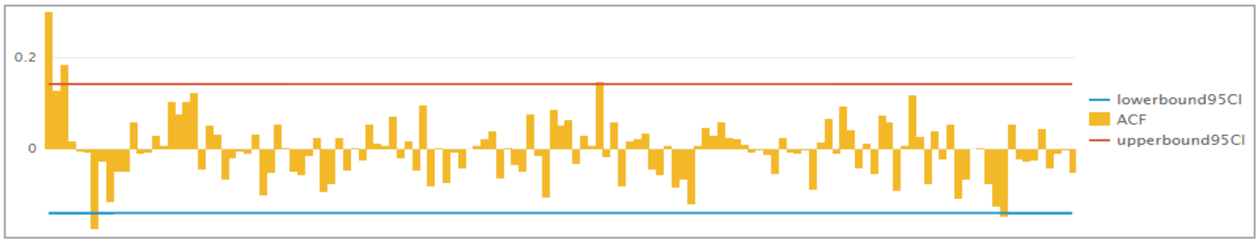
Weeks 1 & 2



Weeks 3 & 4

Example #3: time series with no periodicity (constant value with noise)





The plot above shows the ACF for lags 1 through 168. There are no significant autocorrelation values (beyond the first few), so we determine that this metric has no periodicity.

Documentation
Splunk implementation

- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/Algorithms#Utility_Algorithms
- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/ForecastTimeSeries#ACF:_Autocorrelation_function_chart

A.2 Kalman Filter

The Kalman filter is a state-space method that can be applied to time series data. Given a noisy data stream, the Kalman Filter determines the “state” of the system; for time series analysis, the “state” refers to the series’ level, trend, and seasonality. The Kalman filter involves a recursive mathematical process, in which estimates of the state are updated with each new data point. Once the Kalman filter determines the historical data’s state, it can forecast future values.

Parameters
The Splunk implementation requires the following parameters

1. Type of model
   - LL (local level)- for constant-valued metrics
   - LLT (local level trend)- for metrics that either increase or decrease over time
   - LLP (seasonal local level)- for metrics with a cyclical pattern
   - LLP5 (combines LLT and LLP)
2. Length of period- the expected cycle length; for use with LLP and LLP5

Missing data
The Kalman filter can create forecasts even if there are missing data in the training set. The Kalman filter can also be used to fill in missing data.

Verifying model fit

We detect anomalies by comparing our forecasted values to their actual values. However, this process depends on how accurate the forecasts are. The Kalman filter must create well-fitting models (i.e. learn the pattern of "normal" behavior). Otherwise, the forecasts will be invalid.

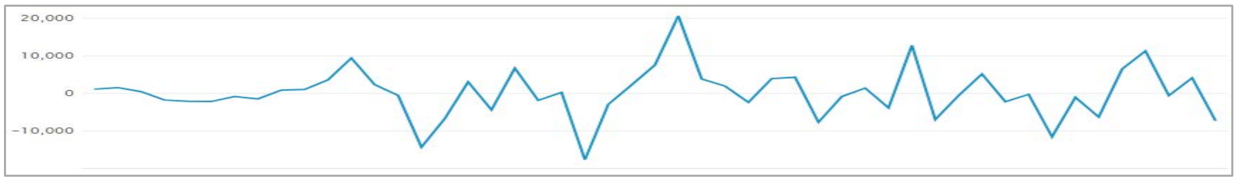
When testing our anomaly detection method (Section 3.1), for each time series, we trained our model on 3 weeks of historical data and then forecasted 1 week of values. If we had an additional week of historical data, we could split our data into 3 parts:

1. Training set- 3 weeks of anomaly-free historical data, which are used to learn the pattern of "normal" behavior
    - Note: The length of the training set can be extended to include historical data that goes further back in time. This may increase the accuracy and precision of the state estimates
2. Validation set- 1 week of anomaly-free historical data. The Kalman filter first creates a forecast for this week. Since we already know the week's actual values, we can determine the forecast accuracy (error = actual value – forecasted value).
    - If the forecast is *accurate*, the Kalman filter learned the correct pattern.
    - If the forecast is *inaccurate*, the Kalman filter could not learn the correct pattern. We should not use our Kalman filter anomaly detection for this metric.
        - For example, the forecast would be inaccurate for the workload pattern whose periodicity increases.
3. Forecast- 1 week of future values. Used for (near) real-time anomaly detection.

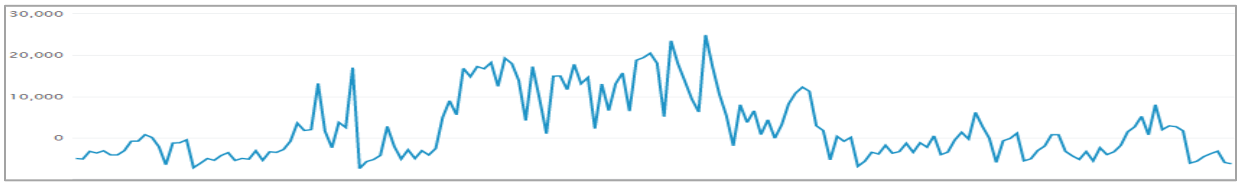
In the MLTK "Forecast Time Series" assistant, there are two measures to quantify model fit. They are based on the validation set's error.

1. R^2 - On a scale of 0 to 1, with 0 representing no fit and 1 representing perfect fit. Many of our models had R^2 values of > 0.8.
2. RMSE (root mean squared error) – Smaller RMSE values reflect better fit. RMSE is scale-dependent, so it can be used to compare different models on the same time series (e.g. LLP vs. LL on the same data). However, it should not be used to compare across series of different scales (e.g. percent vs. KB/s)

Another method to check model fit involves plotting the validation set's distribution of error values. A well-fitting model should have errors that are not correlated over time. The errors should be randomly distributed around a mean of zero. While this plot is not built-in to the "Forecast Time Series" assistant, we can create them with a custom Splunk Search. Two examples are below:



Plot of errors over time for a model with good fit. The errors are randomly distributed around 0, and they have no detectable pattern.



Plot of errors over time for a model with poor fit. In the middle of the week, the error is only positive, indicating that the model is *underestimating* the value during this timeframe.

Documentation

Splunk implementation:

- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/ForecastTimeSeries
- http://docs.splunk.com/Documentation/SplunkCloud/6.6.0/SearchReference/Predict

Theory:

- https://global.oup.com/academic/product/an-introduction-to-state-space-time-series-analysis-9780199228874

A.3 DBSCAN

DBSCAN is a density-based clustering algorithm. It clusters data points that are close together (in an n-dimensional space, where *n* is any positive integer). A point that is far away from others is not assigned to a cluster; it is considered an outlier.

Parameters

- eps- maximum distance between two points for them to assigned the same cluster
- min_samples- minimum number of points needed to comprise a cluster
    - Note: Users cannot change this parameter in Splunk MLTK v2.3.0. The default value (from the underlying sci-kit learn code) is 5.

DBSCAN for collective anomaly detection

We use DBSCAN to determine if the pattern of a series changes. Specifically, we determine whether the last week of data is notably different from the previous few weeks. Our methodology is based on an algorithm used by the data-monitoring company Datadog. Weeks with similar patterns should cluster together, while a week that is different will not be assigned a cluster. Our process is as follows:

1. Begin with a time series consisting of *n* weeks (one week = 168 hours). Consider each week to be a data point in 168-dimensional space (one dimension per hour).
2. Compute the median time series. Do so by calculating the median (across all *n* weeks) for each of the 168 hours.
3. Calculate the Euclidean distance between the median series and each of the *n* weeks.
4. Set "eps" = median(Euclidean_distances) x 2
5. Cluster all *n* weeks in the 168-dimensional space.
6. The first (n-1) weeks should cluster together, as long as their patterns are similar. If the last week is not assigned a cluster, its pattern is dissimilar. Consider it to be a collective anomaly.

Note: Because the "min_samples" parameter is set at 5 in Splunk MLTK, we currently must run DBSCAN on a minimum of 6 weeks of data. (This allows the first 5 weeks of data to be clustered together, while the last week can either be included or excluded from the cluster.) We ideally would like to set "min_samples" to be 3, so that we can run DBSCAN to determine if the 4th week is anomalous.

The process above detects weeklong anomalies, so it is best for metrics with a weekly periodicity. We can alter the process for other periodicities (for example, if we wanted to compare 48h timeframes). Rather than clustering in 168-dimensional space, our space would have d-dimensions, where“d” is the periodicity.

Documentation
Splunk implementation of DBSCAN:

- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/Algorithms#DBSCAN

Scikit-learn documentation (underlying code of Splunk MLTK):

- http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Datadog algorithm:

- https://www.datadoghq.com/blog/outlier-detection-algorithms-at-datadog/#dbscan
- https://www.youtube.com/watch?v=mG4ZpEhRKHA

A.4 ARIMA

The Autoregressive Integrated Moving Average (ARIMA) model is another approach to time series forecasting. It is also known as the Box-Jenkins method.

Parameters
(Reference Documentation for further explanation)

- p- order of autoregressive lags
- d- degree of differencing
- q- order of the moving-average model

Limitations of Splunk ARIMA implementation

- There is no seasonal ARIMA (SARIMA) option in Splunk. SARIMA is necessary for capturing our cyclical day-of-week and time-of-day patterns.
- ARIMA requires the users to specify three parameters (p, d, q) to achieve proper model fit. The MLTK assistant provides graphs (ACF and PACF) to aid users in selecting these values, but this nevertheless is a manual, time-consuming process. The Kalman filter requires less manual tuning.
- ARIMA cannot handle missing data, while the Kalman filter can.

Kalman filter vs. ARIMA forecasting
For time series forecasting, the advantages of using the Kalman filter (KF) instead of ARIMA include:

- KF is adaptive, allowing for changes in time series’ structure over time. ARIMA is homogenous through time
    - For example, KF can automatically adjust to step changes and pattern changes.
- KF can model more complex processes. Any ARIMA model can be represented in KF form, but only simple processes can be represented in ARIMA form.
- KF can handle missing data.
- KF is more transparent. It structures separate components of the series (level, trend, seasonality), whereas ARIMA is more of a “black-box.”

Source: *Time Series Analysis by State Space Methods* by James Durbin & Siem Jan Koopman

Documentation

Splunk implementation:

- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/ForecastTimeSeries
- https://docs.splunk.com/Documentation/MLApp/2.3.0/User/Algorithms#ARIMA

Statsmodels documentation (underlying code of Splunk MLTK):

- http://www.statsmodels.org/devel/generated/statsmodels.tsa.arima_model.ARIMA.html

A.5 Twitter's AnomalyDetection R package

Twitter released this package for point anomaly detection in January 2015. Twitter's algorithm is called Seasonal Hybrid ESD (S-H-ESD). S-H-ESD builds upon the Generalized ESD test, which is a statistical test for detecting anomalies, combining it with STL time series decomposition to extract the seasonal component (e.g. time-of-day, day-of-week patterns). S-H-ESD can detect both local and global anomalies.

Parameters

The algorithm requires the user to specify the periodicity of the data.

The user can also change parameters that determine the number and type of anomalies that are detected:

1. Max_anoms- max # of anomalies (as a % of the data) that should be detected. Default is 10%.
2. Direction- direction of anomalies. Options: "pos" (default), "neg," "both"
3. Alpha- level of statistical significance with which to accept or reject anomalies. Default is 0.05
4. Threshold- to report only positive anomalies above the specified threshold. Options: "None" (default), "med_max" (median of the daily max values), "p95" (95th percentile of the daily max values), "p99" (99th percentile)

We identified a few major pros and cons of AnomalyDetection:

(+) Can be applied to time series with different patterns

(+) Extracts seasonal patterns (e.g. day-of-week, time-of-day) so that both global and local anomalies can be detected

(+) Works well with little tuning

(+) Can change parameters to control the number and type of anomalies that are detected

(-) Can't handle missing data

Documentation

GitHub (code and documentation):

- https://github.com/twitter/AnomalyDetection

Release notes:

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

Academic paper (contains walkthrough of algorithm):

- https://www.usenix.org/system/files/conference/hotcloud14/hotcloud14-vallis.pdf

A.6 Twitter’s BreakoutDetection R package

Twitter released this package for breakout detection in October 2014. Twitter defines two main types of breakouts: (1) mean shifts and (2) gradual ramp-ups. The underlying algorithm is called E-Divisive with Medians (EDM), which uses energy statistics and robust statistical metrics.

Parameters

We found that BreakoutDetection worked well with its default parameters. The 2 main parameters are

- Min.size- the minimum number of observations between change points. Default is 30.
- Method- How many change points to return. Either “amoc” (at most one change; default) or “multi” (multiple changes).

The user can tweak other parameters depending on the choice of “amoc” or “multi.” We did not experiment with these, since the default settings provided accurate results.

We identified a few major pros and cons of BreakoutDetection:

(+) Can be applied to time series with different patternsvg
(+) Can identify breakouts even in the presence of point anomalies
(+) Works well with little tuning
(+) Can detect multiple change points
(-) Can’t handle missing data

Documentation

GitHub (code and documentation):

- https://github.com/twitter/BreakoutDetection

Release notes:

- https://blog.twitter.com/engineering/en_us/a/2014/breakout-detection-in-the-wild.html

Academic paper (contains walkthrough of algorithm):

- https://courses.cit.cornell.edu/nj89/docs/edm.pdf

A.7 HOT SAX and RRA, implemented in R package “jmotif”

Jmotif is a time series analysis package based on Symbolic Aggregate Approximation (SAX). SAX converts a real-valued time series into a discrete-valued symbolic representation. This process involves dimensionality reduction with piecewise aggregate approximation (PAA). SAX can be used for a variety of time series analytics, including clustering and classification.

HOT SAX and RRA use SAX for identifying time series discords (i.e. collective anomalies). HOT SAX was created in 2005. HOT SAX first converts subsequences into symbolic representations (SAX). It then flags the subsequence that has the furthest Euclidean distance from the others. RRA is devised in 2015 to improve upon HOT SAX. RRA uses grammatical inference instead of Euclidean distance to determine anomalies. This makes RRA much faster than HOT SAX.

In 2016, SAX researchers presented a new algorithm called SLADE-TS. It also identified anomalous time series, but it had the advantage of requiring less manual tuning than HOT SAX and RRA. An open-source implementation of SLADE-TS is not yet available.

Parameters

1. w_size- the sliding window size. This is the expected length of anomalous subsequences. We set this to be 168, equivalent to a week
2. paa_size- the PAA size; used for dimensionality reduction
3. a_size- alphabet size; used for SAX discretization
4. n_threshold- the normalization threshold
5. discords_num- the number of discords to report. We set this as 1, so that we would only flag the most anomalous week.

According to the documentation, HOT SAX works best with lower values of “paa_size” and “a_size.” RRA works best with higher values.

We identified a few major pros and cons of HOT SAX and RRA:

(+) Dimensionality reduction makes data processing much more efficient. This is necessary for long or high-granularity time series, which may be extremely noisy or take up too much memory.

(+) Uses sliding windows, so anomalies can start at any point along the series. (Whereas with DBSCAN, the series is split into non-overlapping windows, so there are pre-determined start points.)

(+) Can be applied to time series with different patterns

(+) Works well with little tuning

(+) Can handle missing data

(-) SLADE-TS is the most automated algorithm, since it does not require the user to specify the length of the expected anomaly.

Documentation

GitHub (code and documentation):

- https://github.com/jMotif/jmotif-R

Reference manual:

- https://cran.r-project.org/web/packages/jmotif/jmotif.pdf

Academic paper for HOT SAX:

- http://www.cs.ucr.edu/~eamonn/HOT%20SAX%20%20long-ver.pdf

Academic paper for RRA:

- https://openproceedings.org/2015/conf/edbt/paper-155.pdf

Academic paper for SLADE-TS:

- http://dl.acm.org/citation.cfm?id=2983344&dl=ACM&coll=DL

This page intentionally left blank.