Using Machine Learning to Detect Traffic Anomalies

Jim Goodrich
Senior Sales Engineer | Splunk
During the course of this presentation, we may make forward-looking statements regarding future events or plans of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results may differ materially. The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only, and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Splunk, Splunk>, Turn Data Into Doing, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2019 Splunk Inc. All rights reserved.
This session is a step-by-step tutorial on how to use machine learning to detect time series anomalies.

Based on real world experiences from a global content delivery network provider
The Scenario

Detecting Anomalies in Network Traffic Patterns
E-commerce Example

- Commerce site hosted by CDN
- Traffic is analyzed by multiple security toolsets
- Single highly enriched network log generated

Customer Goal

- Use ML to detect anomalies in specific network traffic patterns (i.e. bot traffic, DoS attacks, false positive/negative conditions)
Why Use Machine Learning?
Technique #1

Simple Spike Detection via Last Hour Analysis
Develop Your Base Search – Part 1

Search the last 60 minutes of logs for the specific condition
Set your latest to “-1m@m” to prevent measuring a partial minute

index=mydata condA=0 condB=1 earliest=-61m@m latest=-1m@m
Group events into buckets (e.g. “bins”) of time

A short span (i.e. 1m) will be highly sensitive to short duration spikes

A long span (i.e. 15m) will be less sensitive to short duration spikes

index=mydata condA=0 condB=1 earliest=-61m@m latest=-1m@m | bin _time span=1m
Develop Your Base Search – Part 3

Count the number of events

Split by _time and a field that differentiates entities

Limit your search to one entity. This is useful for visualization and tuning

```
index=mydata condA=0 condB=1 earliest=-61m@m latest=-1m@m
| bin _time span=1m
| stats count by _time endpoint
| search endpoint="12345"
```
What Does This Look Like?

<table>
<thead>
<tr>
<th>_time</th>
<th>count</th>
<th>endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-07-01 00:00:00</td>
<td>291</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:01:00</td>
<td>294.5</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:02:00</td>
<td>303</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:03:00</td>
<td>301.5</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:04:00</td>
<td>295</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:05:00</td>
<td>301.5</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:06:00</td>
<td>300</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:07:00</td>
<td>312.5</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01 00:08:00</td>
<td>293</td>
<td>12345</td>
</tr>
</tbody>
</table>
12345? Amazing, I have the same combination on my luggage!
Simple Spike Detection = Success
Operationalize the Alert

Use the “Show SPL” button and grab the new code
Search for only outliers
Schedule this an an alert

```
index=mydata condA=0 condB=1 earliest=-61m@m latest=-1m@m
| bin _time span=1m
| stats count by _time endpoint
| search endpoint="12345"
| streamstats window=20 current=true avg("count") as avg stdev("count") as stdev by "endpoint"
| eval lowerBound=(avg-stdev*exact(3)), upperBound=(avg+stdev*exact(3))
| eval isOutlier=if('count' < lowerBound OR 'count' > upperBound, 1, 0)
| search isOutlier=1
```
Technique #2

Detecting Anomalies with Probability Density Functions (PDF)
We are now looking for events that are **abnormal** based on history.

Use each endpoints past to learn “normal” on an individual basis.
Understanding the Machine Learning Process

Training
The “learning” part
Scheduled once per week

Testing
The “alerting” part
Scheduled periodically

Tuning
Visualize and tweak
Optimize the models
Probability Density Function Basics

Visualizing a Normal Distribution
The Empirical Rule
The Left & Right Boundaries
Training the Model Based on Time Variables

Time variables are used to train the model.
These allow the model to isolate time into different buckets.
You need a minimum of 5 training points per buckets (30 - 50 recommended)

<table>
<thead>
<tr>
<th>Example Time Variable</th>
<th>How it Trains The Model</th>
<th>Minimum Training History</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_minutebin (i.e. 15m)</td>
<td>Allows the training to see 0, 15, 30, and 45 minutes as separate slices of time</td>
<td>5 hours</td>
</tr>
<tr>
<td>date_hour</td>
<td>Allows the training to know that hour 1 and hour 14 are different</td>
<td>5 days</td>
</tr>
<tr>
<td>date_wday</td>
<td>Allows the training to differentiate between Monday and Saturday</td>
<td>5 weeks</td>
</tr>
</tbody>
</table>
Training the Model – Part 1

Adjust the earliest and latest to achieve cardinality

Consider a longer span. This is to smooth/average the training data

```
index=mydata condA=0 condB=1 earliest=-36d@d latest=-1d@d
| bin _time span=15m
```
Training the Model – Part 2

Add your time variables

```bash
index=mydata condA=0 condB=1 earliest=-36d@d latest=-1d@d
| bin _time span=15m
| eval date_minutebin=strftime(_time, "%M")
| eval date_hour=strftime(_time, "%H")
| eval date_wday=strftime(_time, "%A")
```
Add back your stats command.

Split by all time variables (as well as endpoint)

We’ll call this the base search on later slides

```bash
index=mydata condA=0 condB=1 earliest=-36d@d latest=-1d@d
| bin _time span=15m
| eval date_minutebin=strftime(_time, "%M")
| eval date_hour=strftime(_time, "%H")
| eval date_wday=strftime(_time, "%A")
| stats count by _time date_minutebin date_hour date_wday endpoint
```
## What Does This Look Like?

<table>
<thead>
<tr>
<th>date</th>
<th>count</th>
<th>date_minutebin</th>
<th>date_hour</th>
<th>date_wday</th>
<th>endpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-07-01</td>
<td>487</td>
<td>00</td>
<td>18</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>461.5</td>
<td>15</td>
<td>18</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>500</td>
<td>30</td>
<td>18</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>519.5</td>
<td>45</td>
<td>18</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>503</td>
<td>00</td>
<td>19</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>487.5</td>
<td>15</td>
<td>19</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>493</td>
<td>30</td>
<td>19</td>
<td>Monday</td>
<td>12345</td>
</tr>
<tr>
<td>2019-07-01</td>
<td>541.5</td>
<td>45</td>
<td>19</td>
<td>Monday</td>
<td>12345</td>
</tr>
</tbody>
</table>
Training the Model – Part 4

Use the fit command to train your data and store the model
Allow 5% of the area to be considered anomalous (similar to 2 stdev)
Manually set the distribution when you have a small training sample

<base search>
| fit DensityFunction count by "date_minutebin,date_hour,date_wday,endpoint"
into mydensitymodel threshold=0.05 dist=norm
Testing the Model

Search a recent time range
Apply the model and search for outliers
Schedule as an alert

<base search>
| apply mydensitymodel
| search “IsOutlier(count)”=1
### Visualization and Tuning – Part 1

#### LowerBound

<table>
<thead>
<tr>
<th>_time</th>
<th>count</th>
<th>date_hour</th>
<th>date_minutebin</th>
<th>date_wday</th>
<th>BoundaryRanges</th>
<th>IsOutlier(count)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-08-01 05:30:00</td>
<td>244.0</td>
<td>5</td>
<td>30</td>
<td>Thursday</td>
<td>-Infinity:13.1509:5e-06</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>224.2039:Infinity:5e-06</td>
<td></td>
</tr>
</tbody>
</table>

#### UpperBound

```<your base search>```
```
<table>
<thead>
<tr>
<th>apply mydensitymodel</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval leftRange=mvindex(BoundaryRanges,0)</td>
</tr>
<tr>
<td>eval rightRange=mvindex(BoundaryRanges,1)</td>
</tr>
<tr>
<td>rex field=leftRange &quot;-Infinity:([^:]*):&quot;</td>
</tr>
<tr>
<td>rex field=rightRange &quot;([^:]*):Infinity&quot;</td>
</tr>
<tr>
<td>fields _time, count, lowerBound, upperBound, &quot;IsOutlier(count)&quot;, *</td>
</tr>
</tbody>
</table>
```
Visualization and Tuning – Part 2

Select the Outliers Chart visualization
Key Takeaways

Best Practices from Real World Testing
1. Implemented an accelerated data model

2. Replaced base search with TSTATS

3. Achieved 99.7X (not percent) query performance increase

```
| tstats count prestats=true FROM datamodel=MYD.MYD WHERE MYD.condA=0 MYD.condB=1 earliest=-61m @m latest=-1m @m BY _time MYD.endpoint span=15m |
| eval date_minutebin=strftime(_time, "%M") |
| eval date_hour=strftime(_time, "%H") |
| eval date_wday=strftime(_time,"%A") |
| stats count by _time date_minutebin date_hour date_wday endpoint |
```
Quality

Getting Actionable Results

1. Implemented multiple anomaly detection techniques (simple spike & PDF)
2. Stored all detections in an anomaly index
3. Used aggregate analysis to determine when scale of anomalies was actionable
Thank You!

Go to the .conf19 mobile app to RATE THIS SESSION