The Importance of Features for Statistical Anomaly Detection

David Goldberg
eBay

Yinan Shan
eBay

Abstract

The theme of this paper is that anomaly detection splits into two parts: developing the right features, and then feeding these features into a statistical system that detects anomalies in the features. Most literature on anomaly detection focuses on the second part. Our goal is to illustrate the importance of the first part. We do this with two real-life examples of anomaly detectors in use at eBay.

1 Introduction

eBay has a very large number of servers, running a layered set of software. The hardware and software layers of these servers are monitored, logging cpu load, memory usage, and many other *monitored signals* at frequent intervals. Like every large cloud-based system, eBay has occasionally suffered *disruptions*, where some of its services were degraded or even completely unavailable to customers. A post-mortem analysis of these disruptions has always revealed that at least one monitored signal exhibited unusual behavior before the disruption began to significantly affect end-users, but these anomalies failed to be detected.

Thus the problem addressed by this paper: how to monitor signals in way that can detect disruptions before they affect users, and do so with few false positives. We need to decide when a signal is exhibiting *anomalous* behavior that is likely to indicate a site disruption. Then we can scan all the monitored signals, and raise an alert when we detect an anomaly

Summarizing the terminology, our goal is to define properties of *monitored signals* that are highly likely to indicate a site *disruption*. We call such properties an anomaly, and when we see one we raise an alert.

To illustrate our approach, it helps to compare with the methodology developed for machine learning problems. It is well established that many learning problems can best be tackled by splitting them into two parts: developing features, and then finding a function that maps features to a target. For example, suppose the problem is to recognize the scanned image of a digit, and produce a target that is a numeral from 0 to 9. It will be very difficult to get a high recognition rate if the raw bitmap is fed directly into a machine learning algorithm. The problem becomes much easier using good features, for example normalize the image and take each pixel as a feature. This is a straightforward set of features, other problems require more ingenuity. (For deep learning, see the Discussion Topics section).

Back to anomaly detection. The theme of this paper is that detecting disruptions also splits into two parts: developing the right features, and then feeding these features into a statistical system that detects anomalies in the features. If done correctly, the detected anomalies will have a high correlation with site disruptions and can be used to create alerts with a low false positive rate. Most literature on anomaly detection focuses on the statistical part (for example [1], [4], [3]). However, it is well-known that feature selection is key in real-life applications (e. g. [2]). Our goal is to illustrate this importance in the context of anomaly detection.

2 A Warm-up Problem (Muninn)

As a warm-up problem, consider a signal from the highest software layer: the number of searches done on the US site ebay.com. This is part of a system called Muninn. The number of searches (srp) is recorded every 2 minutes. Figure 1 shows the value of srp on Mar 25. The horizontal axis ranges from midnight to midnight, PDT. Circled in blue are two suspicious blips, occurring around 4:00 am and 10:30 pm. Even the most sophisticated statistical method can not reliably determine if these should be considered anomalies, that is, if they indicate a site disruption.

But now consider Figure 2 which shows the values of srp on two additional days. Based on this data it seems likely that the blip at 4:00 am on Mar 25 should be categorized as an anomaly, but not the blip at 10:30 pm.

One approach to distinguish these cases would be to use a long-term time series and check for a periodic pattern, which is the method of Vallis et. al. [4]. But to illustrate our method, consider another way of thinking about this problem.

Construct a simple feature, a vector containing the current value of srp with its value 24 hours ago, 48 hours ago, 72 hours ago, etc. Using this feature the circled value on the left in Figure 1 will show up as an anomaly because the vector will have one entry that is out-of-whack, but the circled on the right will have a feature that appears quite ordinary.

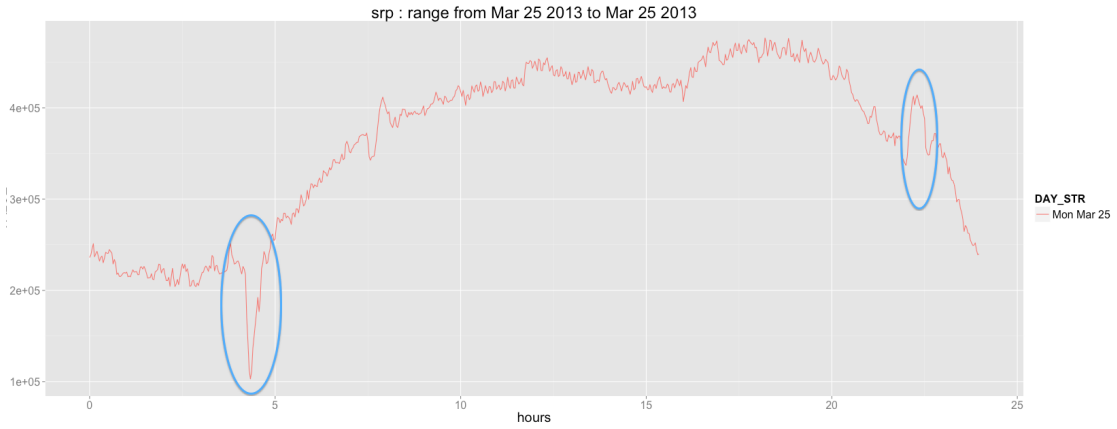

Figure 1: The y-axis is the number of searches/2 minutes. The x-axis covers a 24 hour period on Mar 25. The circled regions look suspicious.

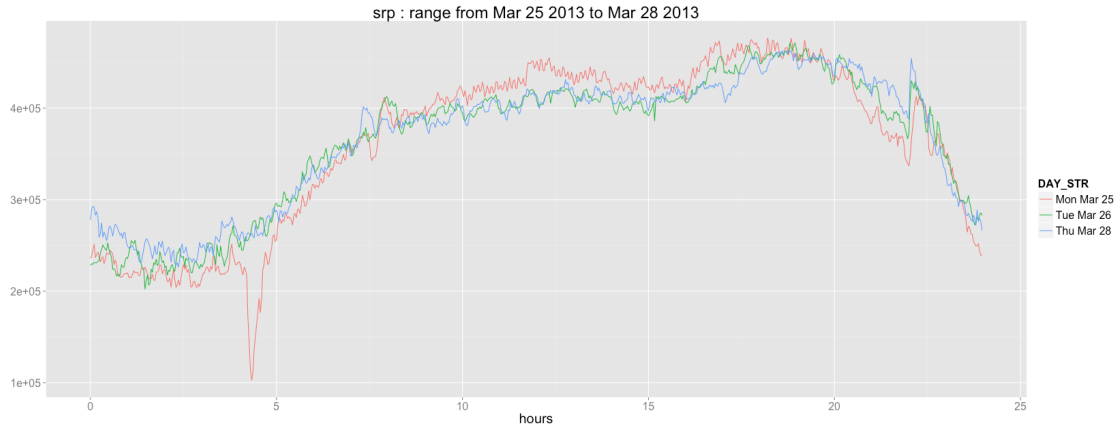

Figure 2: The y-axis is the number of searches/2 minutes. The x-axis covers a 24 hour period. The three curves represent three different days: Mar 25, Mar 26, and Mar 28.

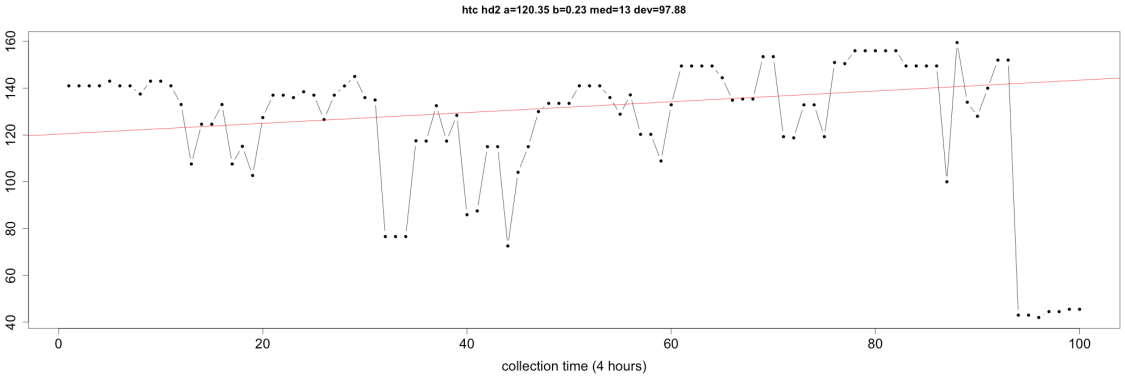

Figure 3: Data for the query *htc hd2*. The y-axis is the value of the metric median sale price. The x axis is the time the query was issued. The leftmost point is 12:00 pm Nov 11, the rightmost point is 4:00 am Nov 28. The dots are 4 hours apart.

3 A more Complex Feature (Atlas)

Here is a more interesting example of monitored signals where the time-series approach does not easily apply, and a feature based approach is more natural. It again concerns searching on eBay. eBay has a monitoring system that periodically issues a query and reports metrics on the resulting item set, such as the number of items returned, the average list price, the fraction of items that are auctions, etc. The monitoring system repeatedly cycles through a fixed set of 3000 queries. This is part of the Atlas system. As mentioned in the previous section, the goal is to detect a problem with the search software (a disruption) before users do. In this case the signal is identical to what users see, so the system can't literally find the disruption before a user. But it can detect and report the problem much more quickly than waiting for a user to phone customer support, having customer support recognize the call as a site problem, and report it to engineering.

Figure 3 is a plot of a (metric, query) pair. The horizontal axis is time, with the right-hand side the most recent data point. The vertical axis is the value of the metric median sale price on the query *htc hd2* (this is the name of a smartphone). The signal is very noisy, but it appears to have unusual behavior in the region $95 \le t \le 100$. But this should not be considered an anomaly because there's no disruption: this type of behavior is perfectly normal, as we now show. Although the points near $t = 100$ are not an anomaly, they are surprising, and we quantify the surprise by fitting all the points to a line (shown in red), and computing the (unsigned) deviation of each point from the line. The surprise of a point is its deviation divided by the median deviation of all the points. For the most recent value (rightmost point at $t = 100$), this is $97.9/13.4 \approx 7.30$.

Should a surprise value of 7.3 be considered an anomaly? Figure 4 shows a histogram of surprise values of median sale price for 2300 queries. A surprise of 7 is not unusual. Because the histogram is so bunched up, a histogram of log(surprise) is also shown. Since $\log(7.3) \approx 2$, it is more clear that this value of surprise is not all that unusual. Quantitatively, the percentile of the surprise for the query *htc hd2* is about 96%.

Incidentally, this example shows the difficulty in getting a low false positive rate. There are about 3000 queries, 40 metrics, and 6 collection periods per day. So there are $3000 \times 40 \times 6 = 720{,}000$ graphs just like Figure 3 each day. To drop as low as one false positive per day would require only triggering on graphs with a surprise so high that it happens 0.00014% of the time. Such a stringent cutoff is likely to overlook many real disruptions.

The way around this is aggregation. Since there will always be a few queries with a high surprise, we construct a feature that captures the number of queries with a high surprise. A sudden change might be a good indicator of a disruption. This is done separately for each metric. To make this quantitative, instead of counting the number of queries with a high surprise, we examine a quantile (specifically 0.9) of the surprise values of all the queries of a metric. Heres how it works.

The surprise of the most recent value, as computed in Figure 3, depends on three things: the metric, the query and the 4-hour collection window of the latest value. The quantile is computed by picking a metric and collection window, gathering up the surprise of all the queries, and then taking the 90% quantile of those numbers. So there is a quantile for each (metric, collection-period) pair. Every four hours the following process is performed for each metric M: recompute the surprise in M for each query and determine the 90% quantile of these numbers. This gives a new value for the 90% quantile for M. The quantiles when M is median sale price over a two-week period are shown in Figure 5.

On the left is a histogram of their values, showing they are clustered near 3.6, with a range from 3.0 to 4.6. The right hand displays the same data in a time series, suggesting there's no strong correlation. This shows the feature is fairly smooth, and might be candidate for anomaly detection. But is it too smooth? Will it actually show an anomaly during a disruption? We went back over historical data and discovered that for each known disruption at least one metric had an anomaly using this feature. And there was only one anomaly that did not correspond to a known disruption (although we never determined if this was an unknown disruption or a false positive).

Here is an example of an anomaly that corresponds to a genuine disruption. While Figure 5 shows the data from up to 07:00 on Nov 28, Figure 6 adds four more time points. Now this is clearly an anomaly. Figure 5 is vaguely normal with a mean of 3.7 and a standard deviation of 0.3. The anomaly in Figure 6 is 29.6, which is $|29.63.7|/0.3 \approx 86$ standard deviations from the mean. So the historical value of the quantile appears to be a great feature.

4 The Statistical Test

As we said in the introduction, we split anomaly detection into two parts. We've just discussed feature construction, what about part two, the statistical test? Combined with good features, even ultra-simple statistical tests already give excellent results.

For the warm-up problem (Muninn), the feature was a vector containing the value of srp 24 hours ago, 48 hours ago, 72 hours, etc. The statistical test is a robust version of $|x - \text{mean}|/\text{sd}$ applied to the vector. The mean is

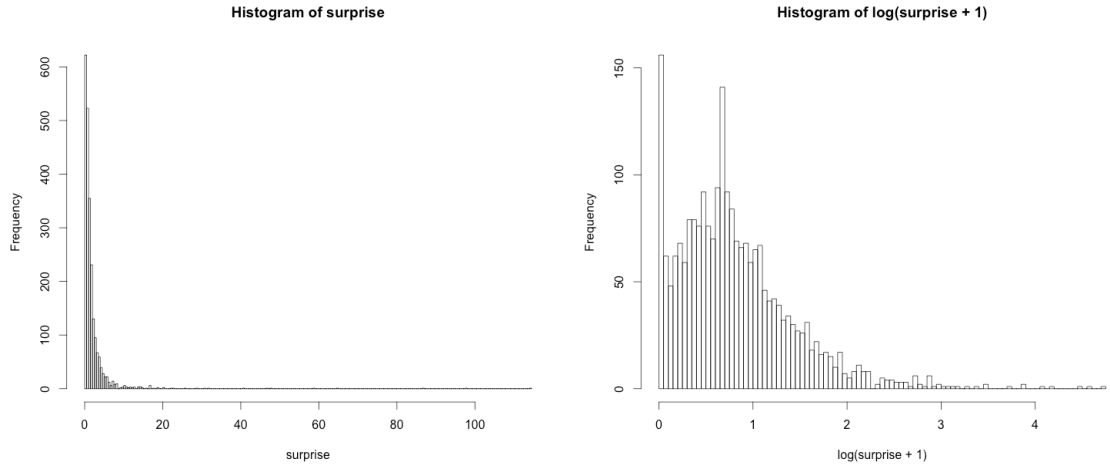

Figure 4: Histograms of surprise values for 2300 queries. Because the histogram is so bunched up near 0, a histogram of log(surprise+1) is also shown.

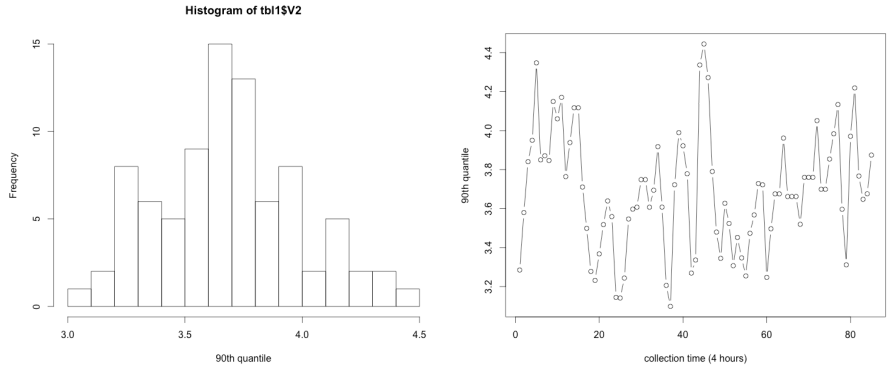

Figure 5: 90th quantile of surprise for median sale price from Nov 14 to Nov 28. The 100 values of the quantile are shown as a histogram on the left, and plotted individually in time-sorted order as a time-series on the right

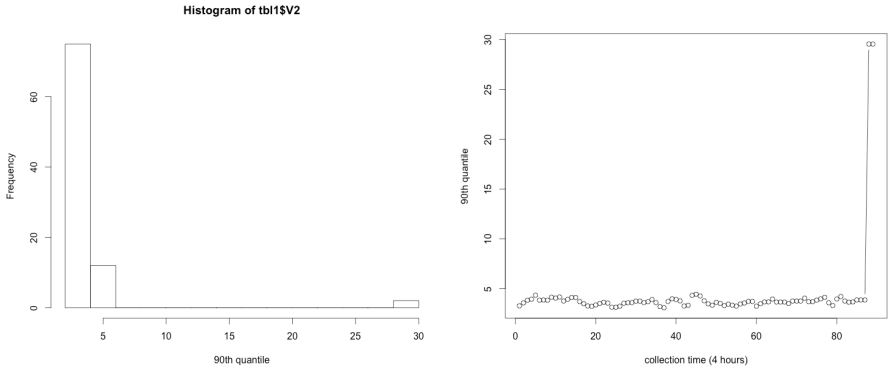

Figure 6: 90% quantile of surprise from Nov 14 to Nov 28, but going 16 additional hours (4 collection times) past Figure 5. Shown as a histogram on the left, and a time series on the right

replaced by the median, and the standard deviation is replace by the median absolute deviation (MAD). MAD is computed just like it sounds: compute the median, then for each value compute the absolute value of distance to the median, and then take the median of these distances. Summarizing: every two minutes when a new value of srp is recorded, set x to be the latest srp, compute the median and MAD over the past 30 srps (not including the most recent) and declare an anomaly if $|x - \text{median}|/\text{MAD}$ is large.

For the second problem (Atlas), the feature was the historical values of the 90th quantile of the surprise values of all the queries. We noted this is a fairly smooth signal, so in this case rather than the robust test using median and MAD, we use the more traditional $|x - \text{mean}|/\text{sd}$ to test for anomalies.

5 Results

The second example (Atlas) has been implemented and replaces an earlier rule-based system. After running for several weeks, Atlas raised its first alert, which identified a real disruption that was not detected by the rule based system. It was subtle: it appeared that only 102 out of 3K queries were affected. But Atlas easily detected this disruption. 28 metrics had more than a 5-sigma deviation of the 0.9 quantile feature. In fact the median value of $|x - \text{mean}|/\text{sd}$ over all 28 metrics was 85. This shows the power of well-constructed features.

In 2014, there were 50 Atlas alerts, of which 35 were traced to a site change or bug, and 15 could not be explained. In other words, about 70% of Atlas alerts are meaningful, which we find to be a very satisfactory hit rate.

6 Systems Aspects

There is a systems aspect of statistical anomaly detection. One of the alerts from the Atlas system described in the last section was caused by a regular update to a system table that affects search results. This was not a disruption. On the other hand, the update did cause a large site change and it is a good sign that Atlas detected it. This suggests that statistical alerting should be part of a larger system that knows about site changes and other external events, and can suppress the alerts they cause. Related is the question of Christmas. Many people's reaction to statistical alerting on an e-commerce site is to ask how the system will learn about unusual shopping days like Christmas. Ideally signals will be archived for several years and so a feature using history would automatically accommodate a yearly pattern like Christmas. However, this is probably not possible for all signals, and so the best system design for this is not to tweak the features and statistics to handle this very special case, but rather to embed it into the larger system. Even if all signals had archived values going back several years, there would still be 1-off events like the British Royal wedding (which caused a large decrease in srp) that would need to handled at a higher level.

7 Summary

To be effective, an alerting system must catch most alerts, and do it with very few false positives. We argue that processing monitored signals into features is a more fruitful way to build such systems than devising ever more complex statistical tests. This is in contrast to the traditional approaches that presume a specific type of feature (e.g. points in the k-space for [3] and time series for [4]) and build sophisticated methods for detecting outliers in this feature.

Our experience with Atlas supports the importance of features. A subtle disruption that was not detected by an earlier rule-based system was easily spotted by Atlas, even though it uses an extremely simple statistical test: $|x - \text{mean}|/\text{sd}$.

8 Acknowledgments

Thanks to Jing Hua who helped coordinate this project, and Tim Converse for comments on the writeup.

9 Discussion Topics

Our primary claim is that high-quality alerting systems (meaning low-false positive rate) require domain-specific knowledge of the signals being monitored, and the construction of features that exploit this knowledge. In contrast, academic papers often imply that there is a very general technique that works almost out-of-the box for most problems. We are hoping for feedback from audience members who have grappled with building low-false positive alert systems. Does our proposal for focusing on feature construction resonate?

It's possible that most people will report great success with standard statistical methods, in which case our idea becomes less interesting.

The main unaddressed issue is whether our technique generalizes, since we presented a single case study. This splits into three parts. Are there other alerting problems amenable to feature construction? Is building custom features a requirement for low-false positive features? And even if good features exist in principle, is it practical to discover them?

Another possible objection is that building features is just building a statistical procedure in disguise. We don't agree with that, and think the features is a more natural way to get alerts that are very high true-positive and very low false-positive rates. It is also something that can be done by domain experts who do not have deep statistical expertise.

Finally, what about deep learning, which promises to automate the process of feature engineering. Our view is that deep learning works well, but only for a subset of problems. However, if you have monitoring data where neural nets can automatically discover good features for alerting, then we would consider that an exciting further step in the approach advocated in this paper.

If nothing else, we hope to get a sense of the types of anomaly problems out there.

References

[1] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly detection: A survey. *Computing Surveys 41*, 3 (2009).

[2] KANDEL, S., PAEPCKE, A., HELLERSTEIN, J. M., AND HEER, J. Enterprise Data Analysis and Visualization: An Interview Study. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012).

[3] PAPADIMITRIOU, S., KITAGAWA, H., GIBBONS, P. B., AND FALOUTSOS, C. Loci: Fast outlier detection using the local correlation integral. U. Dayal, K. Ramamritham, and T. M. Vijayaraman, Eds., IEEE Computer Society.

[4] VALLIS, O., HOCHENBAUM, J., AND KEJARIWAL, A. A novel technique for long-term anomaly detection in the cloud. M. A. Kozuch and M. Yu, Eds., USENIX Association.