anodot

ULTIMATE GUIDE TO BUILDING A MACHINE LEARNING **OUTLIER DETECTION SYSTEM**

Part III: Correlating Abnormal Behavior

INTRODUCTION

Many high-velocity online business systems today have reached a point of such complexity that it is impossible for humans to pay attention to everything happening within the system. There are simply too many metrics and too many data points for the human brain to discern. Most online companies already use data metrics to tell them how the business is doing, and detecting outliers in the data can lead to saving money or creating new business opportunities. Thus, it has become imperative for companies to use machine learning in large-scale systems to analyze patterns of data streams and look for outliers.

Consider an airline pricing system that calculates the price for each and every seat on all of its routes in order to maximize revenue. Seat pricing can change multiple times a day based on thousands of factors, both internal and external to the company. The airline must consider those factors when deciding to increase, decrease or hold a fare steady. An outlier in any given factor can be an opportunity to raise the price of a particular seat to increase revenue, or lower the price to ensure the seat gets sold.

Automated outlier detection is a technique of machine learning, and it is a complex endeavor. Anodot is using this series of white papers to help explain and clarify some of the sophisticated decisions behind the algorithms that comprise an automated outlier detection system for large scale analytics. In Part I of this white paper series, we outlined the critical design principles of an outlier detection system. In Part II we continued the discussion with information about how systems can learn what "normal behavior" looks like in order to identify abnormal behavior. We recommend first reading parts 1 and 2 to gain the foundational information necessary to comprehend this document.

Here in Part III, the final document of our white paper series, we will cover the processes of identifying, ranking and correlating abnormal behavior. Many of the aspects we discuss in this document are unique to Anodot, such as ranking and scoring outliers and correlating metrics together. Most other vendors that provide outlier detection solutions for do not include these steps in their analysis, and we believe them to be a real differentiator and a major reason why Anodot's solution goes beyond merely bringing accurate outliers to light with minimum false positives and negatives, but puts them into the context of the full story to provide actionable information.

There are five steps necessary to learn and identify outliers:

1. **METRICS COLLECTION – UNIVERSAL SCALE TO MILLIONS**
2. **NORMAL BEHAVIOR LEARNING**
3. **ABNORMAL BEHAVIOR LEARNING**
4. **BEHAVIORAL TOPOLOGY LEARNING**
5. **FEEDBACK-BASED LEARNING**

Steps 1 and 2 were covered in detail in the previous two white papers. This document covers steps 3 and 4. Step 5 is not in the scope of this white paper series.

ABNORMAL BEHAVIOR LEARNING AND SCORING

The objective of any outlier detection system is to, well, detect outliers. But not all outliers are equal. Some are more significant than others, and the reaction an outlier causes might depend upon how significant it is.

In our earlier documents, we used the example of the human body as a complex system with many metrics and data points for each metric. Body temperature is one of those metrics; an individual's body temperature typically changes by about a half to one degree between its highest and lowest points each day. A slight temperature rise to, say, 37.8 °C (100.0 °F), would be irregular but not a cause for great concern, as taking an aspirin might help lower the temperature back to normal. However, an anomalous rise to 40 °C (104.0 °F) will certainly warrant a trip to the doctor for treatment. These are both outliers, but one is more significant than the other in terms of what it means within the overall system.

In a complex business system, how do we understand which outlier is more significant than another? Let's consider this at the individual metric level, as shown in **Figure 1a** below. **Figure 1a** shows a set of outliers — some are small, some are big, some last longer, some are shorter in duration. Though not shown in this illustration, some outliers might have a pattern to them, and some patterns could be a square or a linear increase or decrease. Looking at the chart with the human eye, one could posit what is more or less significant based on intuition, and this method can be encoded into an algorithm.

For every outlier found in a metric, there is a notion of how far it deviates from normal as well as how long the outlier lasts. These notions are called deviation and duration, respectively.

For every outlier found in a metric, there is a notion of how far it deviates from normal as well as how long the outlier lasts. These notions are called deviation and duration, respectively. As for the outliers seen in **Figure 1a**, some of them contain many data points, which means that the data series was abnormal for quite a while (i.e., had a longer duration), and some of them have fewer data points (i.e., the outlier had a shorter duration). In some cases, the peak of the data points is higher (i.e., a greater deviation from normal), and for others, the peak is lower (i.e., less of a deviation from normal). There are other conditions around both duration and deviation, and they all need to be considered in the statistical model. In the case of Anodot, the input is a set of statistics related to each outlier, and the output is a score (on a scale of 0 to 100) of how significant the outlier is.

Having such a score provides the ability to filter -outliers based on their significance. In some cases, the user would want to be alerted only if the score or the significance is very high; and in other cases, the user would want to see all outliers. For example, if a business is looking at a metric that represents the company's revenue, then the user would probably want to see outliers pertaining to anything that happens, even if they are very small. But if the same business is looking at the number of users coming into its application from a specific location like Zimbabwe – assuming the company doesn't do a lot of business in Zimbabwe – then maybe the user only wants to see the big outliers; i.e., highly significant outliers. In the Anodot system, this is configured using a simple slider as seen in **Figure 2a**.

The user needs this input mechanism because all the outlier detection is unsupervised, and the system has no knowledge of what the user cares about more.

Note that the significance slider in the Anodot system does not adjust the baseline or the normal behavior model; it only defines which outliers the user chooses to consume. This helps users focus on what is most important to them, preventing alert fatigue. If there are too many alerts, such as one for every single outlier, the alerts eventually become overwhelming and meaningless.

Scoring occurs through machine learning since the scores are relative to past outliers of that metric, not an absolute value.

Consider the outliers shown in **Figure 1b**. Even without looking at the assigned numbers for some of the outliers, a person looking at the signal would probably come up with similar scores. How? It is not based on each outlier's amount of deviation from normal, rather, it is based on the fact that the high peak outliers deviated a lot more, and the smaller ones deviated less than the bigger ones. Even for human eyes, it is all relative.

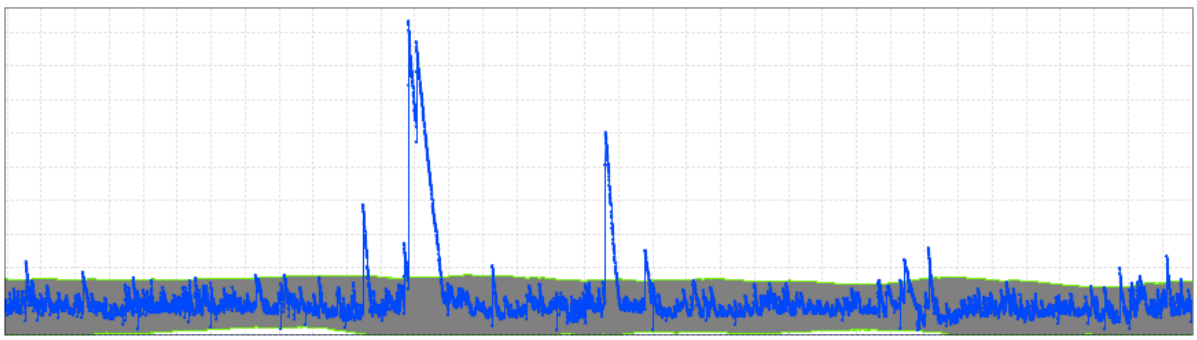

Figure 1a. *A single metric with several instances of abnormal behavior.*

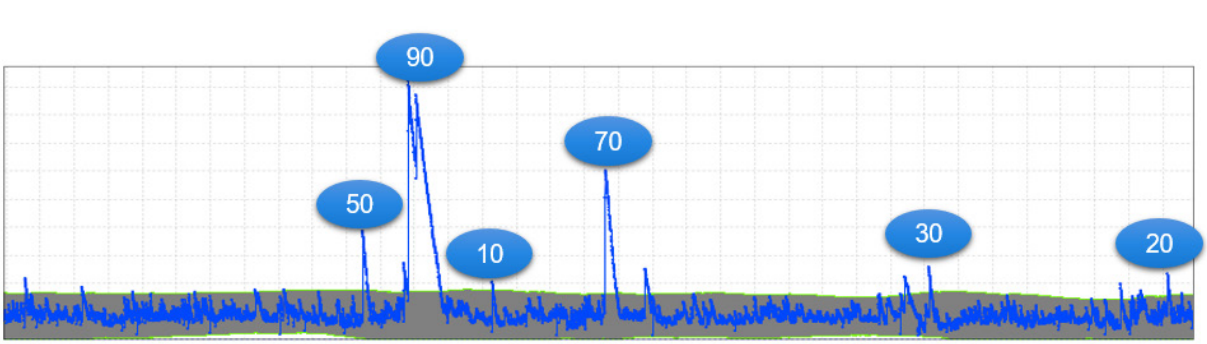

Figure 1b. *Outliers ranked by significance.*

Now suppose the big spike – the one labeled “90” – was not there. Without a significant outlier to compare to, the other outliers would look bigger, more significant. In fact, we would probably change the scale of the graph.

This is an important distinction because there are other scoring mechanisms that look at the absolute deviation without context of what happened in the past. Anodot initially took this approach but we saw quickly, from a human perspective, that when people look at a long history of a time series and see the outliers within it, in their minds they consider the outliers relative to each other as well as relative to normal. Anodot’s algorithms now mimic this human thought process using probabilistic Bayesian models.

In the screenshot in **Figure 2b**, the significance slider is set to 70, meaning that only the orange outliers would be alerted on, and not the gray ones, which fall below that score.

Figure 2a. *The significance slider in the Anodot system lets users select the level of outliers to be alerted on.*

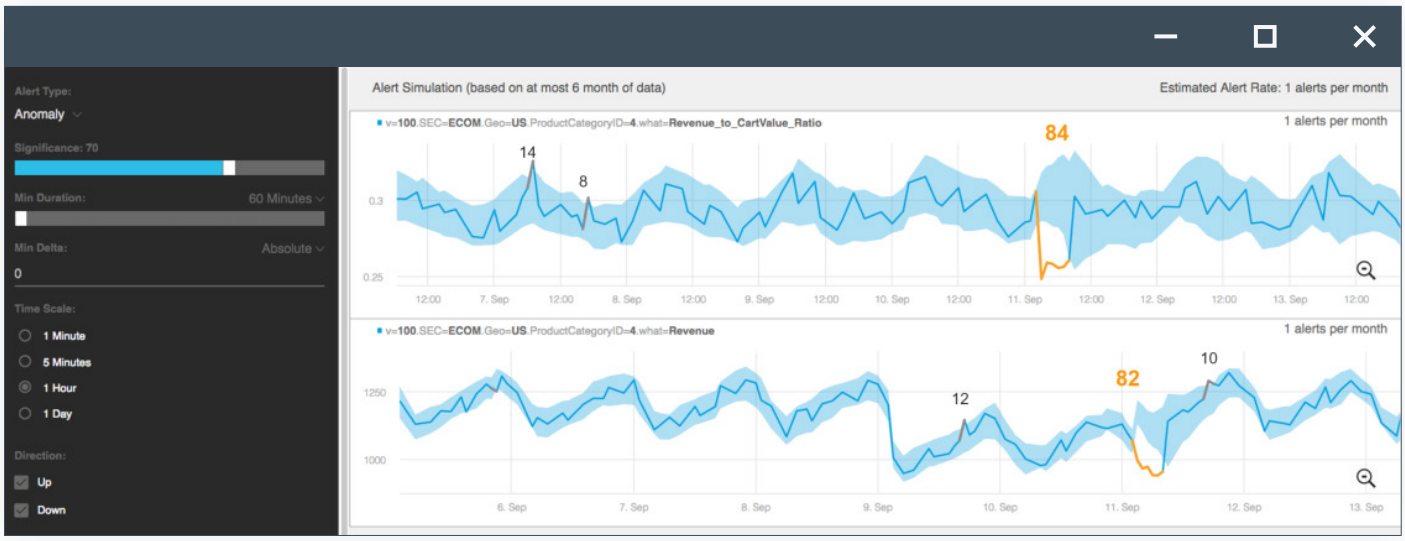

Figure 2b. *With significance set at 70, users would be alerted on the two orange alerts that are above 70, but not the smaller gray alerts below 70.*

BEHAVIORAL TOPOLOGY LEARNING

The next step in the overall process of learning and identifying outliers in a system is behavioral topology learning. In the first document of this white paper series, we discussed learning system design principles and covered the conciseness of outliers. Conciseness refers to idea that the system considers multiple metrics simultaneously, to view what is happening holistically.

If there are many outliers at the single metric level and they are not combined into a story that describes the whole incident, then it is very hard to understand what is going on. However, combining them into a concise story requires an understanding of which metrics are related, because otherwise the system runs the risk of combining things that are completely unrelated. The individual metrics could be anomalous at the same time just by chance.

Behavioral topology learning provides the means to learn the actual relationships among different metrics. This type of learning is not well-known; consequently, many solutions do not work this way. Moreover, finding these relationships at scale is a real challenge. If there are millions of metrics, how can the relationships among them be discovered efficiently?

As shown in **Figure 3**, there are several ways to figure out which metrics are related to each other.

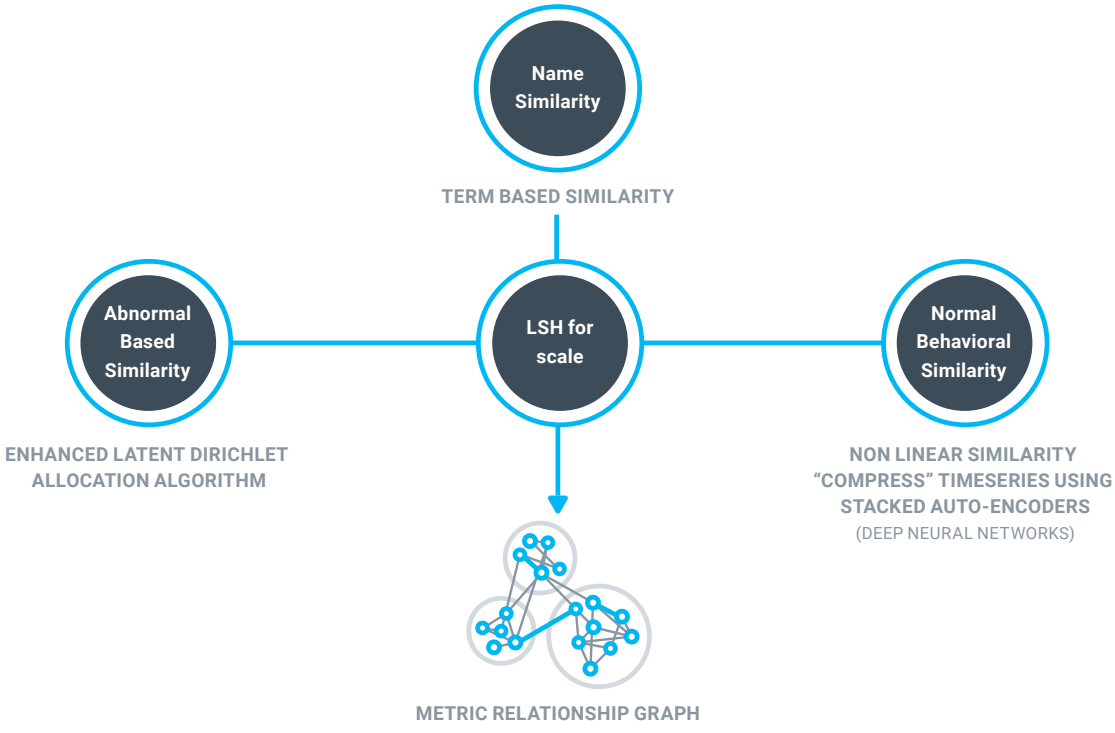

Figure 3. *Methods of relating metrics to each other.*

ABNORMAL BASED SIMILARITY

The first method of relating metrics to each other is abnormal based similarity. Intuitively, human beings know that when something is abnormal, it will typically affect more than one key performance indicator (KPI). In the other papers in this series, we have been using the example of the human body. When someone has the flu, the illness will affect his or her temperature, and possibly also heart rate, skin pH, and so on. Many parts of this system called a body will be affected in a related way.

When an automatic outlier detection system takes in these measurements, it does not know that the temperature, heart rate and skin pH are from the same person (unless someone tells the system that fact). However, if the person gets the flu several times, several of his or her vital signs will become irregular at the same time, thus there is a high likelihood that some of the outliers on their measurements will overlap.

The more often metrics are anomalous at similar times, the more likely it is that they are related.

The chance of two metrics having a single concurrent outlier is high if you are measuring many things. If we were to simply rely on outliers happening together to determine that they are related, it would cause many mistakes. But the probability of them being anomalous twice at the same time is much lower. Three times, even lower. The more often the metrics are anomalous at similar times, the more likely it is that they are related.

The metrics don't always have to be anomalous together. A person's temperature could increase but his or her heart rate might not increase at the same time, depending on the illness. But we know that many illnesses do cause changes to the vital signs together.

Based on these intuitions, one can design algorithms that find the abnormal based similarity between metrics. One way to find abnormal based similarity is to apply clustering algorithms. One possible input to the clustering algorithm would be the representation of each metric as anomalous or not over time (vectors of 0's and 1's); the output is groups of metrics that are found to belong to the same cluster. There are a variety of clustering algorithms, including K-means, hierarchical clustering and the Latent Dirichlet Allocation algorithm (LDA). LDA is one of the more advanced algorithms, and Anodot's abnormal based similarity processes have been developed on LDA with some additional enhancements.[i]

The advantage that LDA has over other algorithms, is that most clustering algorithms would allow a data point - or a metric in this case - to belong to only one group. There could be hundreds of different groups, but in the end, a metric will belong to just one. Often, it is not that clear-cut. For example, on a mobile app, its latency metric could be in a group with the metric related to the application's revenue, but it could also be related to the latency of that app on desktops alone. By using clustering algorithms that force a choice of just one group, the system might miss out on important relationships. LDA clusters things in such a way that they can belong to more than one group, i.e. "soft" clustering, as opposed to "hard" clustering.

Another advantage of LDA is that most clustering algorithms have some distance function between what is being measured that is similar. The LDA algorithm allows a metric to be partially similar to the other metrics. This comes back to the “softness” of the algorithm—it allows partial similarity for a metric to still belong to a group. In the context of learning metric relationships, this is an important feature because, for example, application latency doesn’t always have to be anomalous when the revenue is anomalous. It is not always the case that latency goes up anomalously and revenue goes down, and there can be times when the revenue becomes abnormal but the latency does not go up or down accordingly. The outlier detection system must be able to take that partiality into account.

The primary issue with abnormal based similarity is that it does not scale well – we discuss scaling later in the paper. In addition, it requires seeing enough historical data containing outliers so it can capture these relationships. Are there additional types of information that can help capture the metric topology with less (or no) history? We will discuss two additional methods of capturing relationships between metrics next.

NAME SIMILARITY

Another method for determining relationships among metrics is name similarity. Every metric in a system must be given a name that is not just free-form text. In the industry of data handling, there are recommended naming conventions for metrics, typically comprised of key value pairs describing what is being measured and its source. For example, say we are measuring the revenue of an app for Android in the US. For simplicity's sake, we will call this app XYZ. The key value pair describing what we are measuring and the source would be XYZ together with US. Thus, the revenue metric might have a name like appName=XYZ. Country=US.what=revenue.

This particular app is also available in Germany, so the name for the metric that measures revenue in there might be something like appName=XYZ. Country=Germany.what=revenue. By looking at the similarity between these two metric names, we have a measure of how similar they are. If they are very similar, then we say they should be grouped because they probably describe the same system. It is reasonable to associate metrics using this method; it is essentially based on term similarity, by comparing terms to see whether they are equal and how much overlap they have.

NORMAL BEHAVIOR SIMILARITY

A third method of determining relationships among metrics is normal behavior similarity, which looks at the metrics under normal circumstances as opposed to the abnormal based similarities. This method asks questions like, “Do the metrics have the same shape?” and “Do they look the same when the signal is normal?” What they look like when abnormal does not matter. For example, if we look at revenue for the XYZ app on different platforms such as Android and iPhone, they will probably look quite similar; the signals for these two metrics will most likely have the same shape. However, if we compare the application latency on Android to the app’s revenue on that platform, they won’t be similar.

Normal behavior similarity looks at the metrics under normal circumstances as opposed to the abnormal based similarities.

Normal behavior similarity is the weakest method of the three discussed in this document because it is always possible to find correlations if one looks hard enough. The question is how to do it intelligently without getting a lot of false positives.

The most commonly used method of performing normal behavior similarity comparisons is with linear correlation. Here people use measures such as the Pearson correlation coefficient, which is a measure of the linear dependence (correlation) between two variables (metrics). This method requires some caution. For example, it is necessary to de-trend the data, meaning that if there is a linear line constantly going up or down, it must be subtracted from the original time series before computing the Pearson correlation. Otherwise, any metric that is trending up will be correlated with anything else that is trending up, resulting in a lot of false positives.

It is also necessary to remove seasonal patterns from the metrics; otherwise anything with a seasonal pattern will be correlated with anything else that has the same seasonal pattern. If two metrics both have a 24-hour seasonal pattern, the result will be a very high similarity score regardless of whether they are related or not. In fact, many metrics do have the same seasonal patterns but they are not related at all. For instance, we could have two online apps that are not related, but if we look at the number of visitors to both apps throughout the day, we will see the same pattern because both apps are primarily used in the US and have the same type of users. It could be the XYZ app and a totally unrelated news application.

Unlike abnormal based similarity which creates very few false positives but is dependent on outliers happening (which occurs rarely), thus more time to pass, normal behavior similarity requires much less data in order to be computed. However, if not done right – e.g., if the data patterns are not de-trended and de-seasonalized – this method could create many false positives.

The Pearson correlation is a simple algorithm and is quite easy to implement, but there are better approaches that are less prone to false positives, such as the pattern dictionary based approach. Suppose each time series metric can be partitioned into segments, where each segment is classified to one of N prototypical patterns that are defined in a "dictionary" of known patterns like a daily sine wave, a saw tooth, a square wave-like pattern, or other classifiable shapes. Once the user has a dictionary of typical shapes, he or she can describe each metric based on what shapes appeared in it at each segment.

As an example, from 8 AM to 12 PM, the metric had shape number 3 from the dictionary of shapes, and from 12 PM to 5 PM, it had shape number 10. This changes how the time series is represented with a more compressed representation, which also describes attributes at a high level, rather than just the values. From there, it is relatively easy to do clustering or any type of similarity grouping based on the new representation. It is also easier to discount the weight of very common shapes in the dictionary by using techniques from document analysis (such as TF-IDF weights[11]). It is safe to assume that things are correlated if at every point in time, they have similar shapes.

The main challenge in the shape dictionary based approach is how to create the dictionary. A variety of algorithms can be employed for learning the dictionary, but they all follow a similar approach: Given a (large) set of time series metric segments, apply a clustering technique (or soft clustering technique such as LDA) on all the segments, and then use the representations of the clusters as the dictionary of shapes. Given a new segment of a metric, find the most representative cluster in the dictionary and use its index as the new representation of the segment.

One of the most promising algorithms tested at Anodot for creating such a dictionary is a Neural-Network based approach (Deep Learning), namely, Stacked Autoencoders. Stacked autoencoders are a multi-layer Neural Network designed to discover a high-level representation of the input vectors in the form activation of the output nodes. Training stacked autoencoders is done with a set of segments of the time series; the activated nodes at the output of the network are the dictionary representing prototypical shapes of the input segments. The details of implementing this deep learning technique to accomplish this task are out of the scope of this white paper.

USER INPUT

There are additional methods of establishing relationships among metrics that do not require sophisticated algorithms; one is direct user input. If a user says that all XYZ app metrics are related, this fact can be encoded into the learning model. It is a technical process, not an algorithmic one, but this type of direct input can be useful if the user can provide it.

The second method is indirect input, in which the user manipulates the metrics to create new metrics out of them. If there is revenue of XYZ app in multiple countries, the user can now create a new metric by calculating the sum of the revenue from all the countries. It can be assumed that if it makes sense to create a composite metric of multiple metrics, then the individual metrics are likely related to each other.

Anodot uses both methods, depending on what information is available.

11 • Ultimate guide to building a machine learning outlier detection system. Part III.

A MATTER OF SCALE

Of the various methods discussed above, one of the major challenges is scale. How can these comparisons be applied at very large scale? The algorithm-based methods are computationally expensive when there are a lot of metrics to work with. It either requires a lot of machines or a lot of time to get results. How can it be done efficiently on a large scale, such as a billion metrics?

One method is to group the metrics. We would start with one billion metrics sorted into 100 different groups that are roughly related to each other. We can go into each group and perform the heavy computation because now the number of groups is small, and each group has its own order. If we have a group of one million metrics, and then we separate them into 10 groups, we end up with 10 groups of 100k metrics each, which is a much smaller, more manageable number. A mechanism is needed to enable fast and accurate partitioning.

How can this be done without knowing what things are similar? A locality sensitive hashing (LSH) algorithm can help here. For every metric a company measures, the system computes a hashtag that determines which group it belongs to. Then, additional algorithms can be run on each group separately. This breaks one big problem into a lot of smaller problems that can be parsed out to different machines for faster results. This methodology does have a certain probability of false positives and false negatives; however, the algorithm can tune the system, depending on how many false positives and false negatives users are willing to tolerate.

In this case, “false positive” means that two things are grouped together, despite not exhibiting characteristics that would cause them to be grouped together. “False negative” means that two things are put into separate groups when they should be in the same group. The tuning mechanism allows the user to specify the size of the groups based on the total number of metrics, as well as the tolerance of false positives and false negatives that he or she is willing to accept. One way to reduce the number of false negatives is to run the groups through the algorithms a few times, changing the size of the group each time. If the groups are small enough, they can run rapidly while not being computationally expensive.

THE IMPORTANCE OF EACH STEP

The real goal of performing outlier detection in a business system is not to merely identify unusual things that are happening within that system, but to use the insights about when and where the outliers happen to understand the underlying cause(s) and hopefully uncover opportunities to improve the business. A large-scale business system can have hundreds of thousands or even millions of metrics to be measured. A well-known social network that is used by billions of people around the world is estimated to have 10 billion metrics.

Any large-scale system with a high number of metrics will yield many outliers—perhaps too many for the business to investigate in a meaningful time frame. This is why all of the steps discussed across our series of three white papers are important. Each step helps reduce the number of outliers to a manageable number of truly significant insight. This is illustrated in **Figure 4** below.

This chart illustrates the importance of all the steps in an outlier detection system: normal behavior learning, abnormal behavior learning, and behavioral topology learning. Consider a company that is tracking 4 million metrics. Out of this, we found 158,000 single metric outliers in a given week, meaning any outlier on any metric. This is the result of using our system to do outlier detection only at the single metric level, without outlier scoring and without metric grouping. Without the means to filter things, the system gives us all the outliers, and that is typically a very large number. Even though we started with 4 million metrics, 158,000 is still a very big number—too big to effectively investigate; thus, we need the additional techniques to whittle down that number.

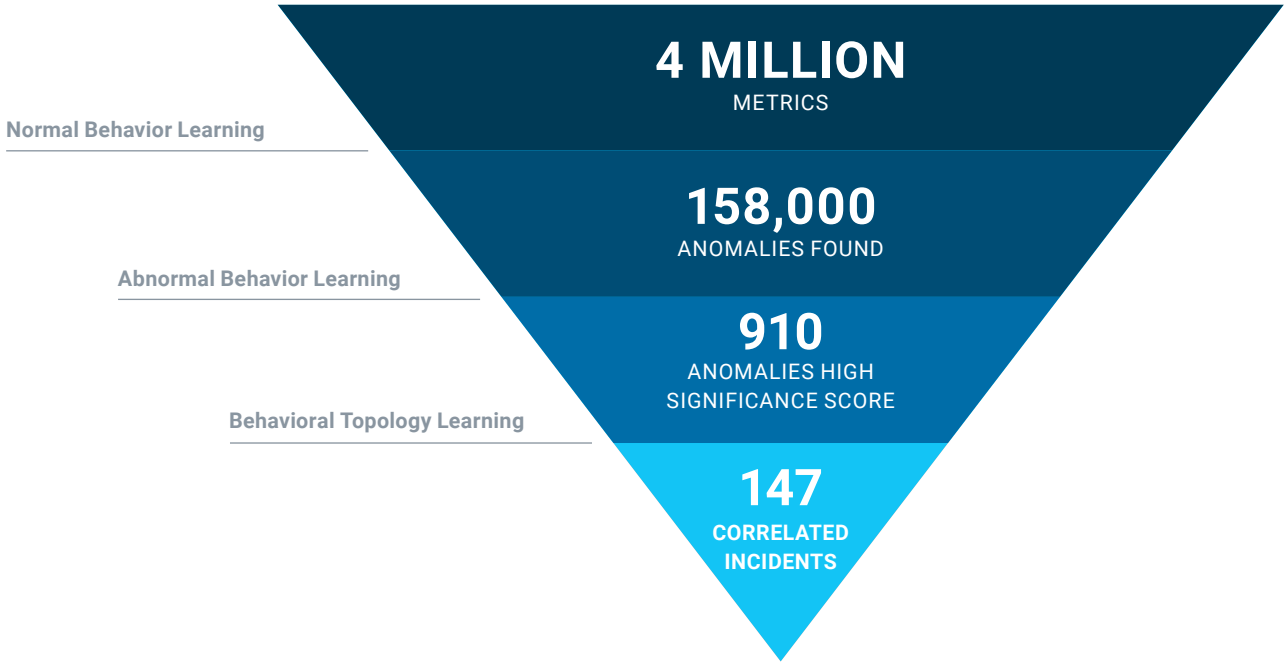

Figure 4. *The importance of all steps in an outlier detection system.*

If we look at only the outliers that have a high significance score – in this case a score of 70 or above – the number of outliers drops off dramatically by an order of magnitude to just over 910. This is the number of significant outliers we had for single metrics out of 4 million metrics for one week — 910 of them. Better, but still too many to investigate thoroughly.

The bottom of the funnel shows how many grouped outliers with high significance we end up after applying behavioral topology learning techniques. This is another order of magnitude reduction — from 910 to 147. This number of outliers is far more manageable to investigate. Any organization with 4 million metrics is large enough to have numerous people assigned to dig into the distilled number of outliers, typically looking at those outliers that are relevant to their areas of responsibility.

Figure 4 does not necessarily show the accuracy of the outliers; rather, it shows why all these steps are important; otherwise the number of outliers can be overwhelming. Even if they are "good" outliers – they found the right things – it would be impossible to investigate everything in a timely manner. Users would simply stop paying attention because it would take them a long time to understand what is happening. This demonstrates the importance of grouping — really reducing the forest of 158,000 outliers into 147 grouped outliers per week. This goes back to the notion of conciseness covered in the design principles white paper (Part I of this series). Concise outliers help to tell the story of what is happening without being overwhelming, enabling a human to investigate more quickly. Then the business can take advantage of an opportunity that might be presented through the outlier, or take care of any problem that the outlier has highlighted.

THE ARCHITECTURE OF AN OUTLIER DETECTION SYSTEM

In a generic sense, any large-scale outlier detection system should follow the design principles we outlined in the first part of this white paper series. In **Figure 5** below, we use Anodot's system as an example to describe the architecture and components of a typical system. Where possible, we will point out how the Anodot system might differ from others.

The most important requirement of this architecture is that it be scalable to a very large number of metrics. Anodot achieves this by performing most of the normal behavior learning as the data flows into our system. We perform machine learning on the data stream itself. This is shown in the central part of the illustration, Anodotd, labeled "Online Baseline Learning."

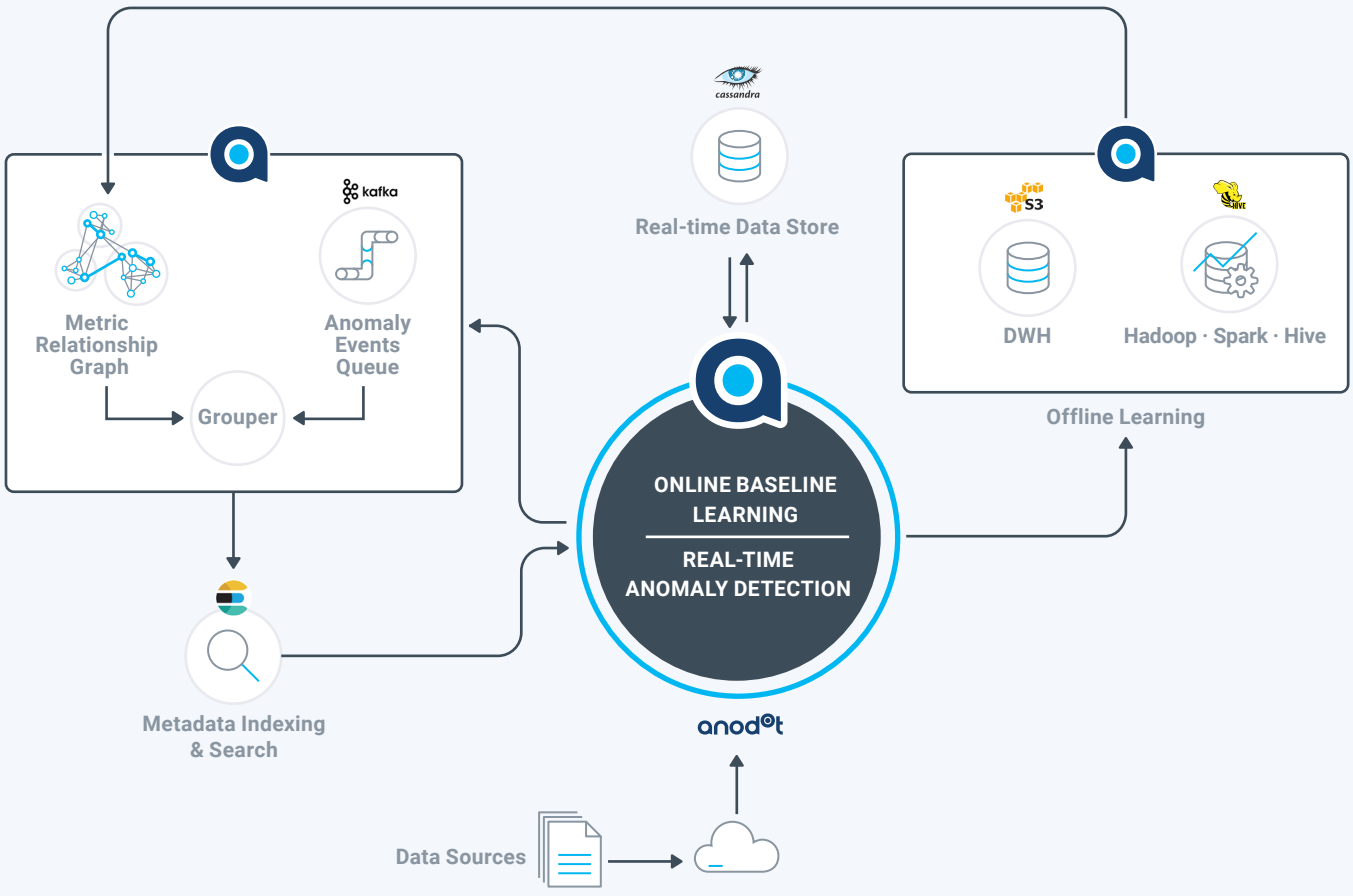

Figure 5. *The architecture of Anodot's large-scale outlier detection system.*

The flow of data comes from Customer Data Sources, as shown at the bottom of the illustration, into what we call Anodotd, or Anodot Daemon, which does the learning. When a data point comes in from a metric, the system already has the pre-constructed normal model for that metric in its memory. If there is an outlier, it scores it using the abnormal model and sends it to the "Outlier Events Queue" (we use Kafka) on the left side of the illustration. If there is no outlier, Anodotd simply updates the model that it has so far and stores that model in the database.

Many machine learning systems do not work this way. They pull data from a database, do their learning and then push the data back to a database. However, if you want the system to scale and find outliers on 100% of the metrics – because it is unknown which metrics are important – then the learning must be done on all the samples that come in. If the data has already been stored in a database and then must be pulled out in order to do the learning, the system will not be able to scale up. There is no database system in the world that both read efficiently and write rapidly. Enlarging the database system is a possibility, but it will increase costs significantly. Certainly, to get the system to scale, learning must be done on the data stream itself.

The components on the left side of the illustration perform the processes of identifying and correlating abnormal behavior. Outlier events on single metrics pass through the queue to the Grouper component which checks whether to group single metric outliers based on the information from the metric relationship graph.

All information about the outliers is passed to the Metadata Indexing and Search Engine. Any changes to an outlier will be updated in this Engine. Anodot uses Elastic Search for this process, which does not affect the design of the system.

Figure 6. *Anodot in numbers, per day.*

On the right side of **Figure 5** is Hadoop/Spark HIVE offline learning. There are some processes that Anodot runs offline, for example, the behavioral topology learning or seasonality detection can be run offline; we do not have to run this process on the data stream itself. Discovering that one metric is related to another is not something that will change from data point to data point. Finding that something has a weekly seasonal pattern does not have to be detected on every data point that comes in for that metric. There is a price to pay when processes run on the data stream, often in the form of accuracy. With online learning, there is no luxury of going back and forth; thus, Anodot performs these activities offline. This combination of online and offline learning optimizes accuracy and efficiency.

Not all outlier detection systems have all these components, but Anodot believes they are all important to yield the best results in an efficient and timely manner.

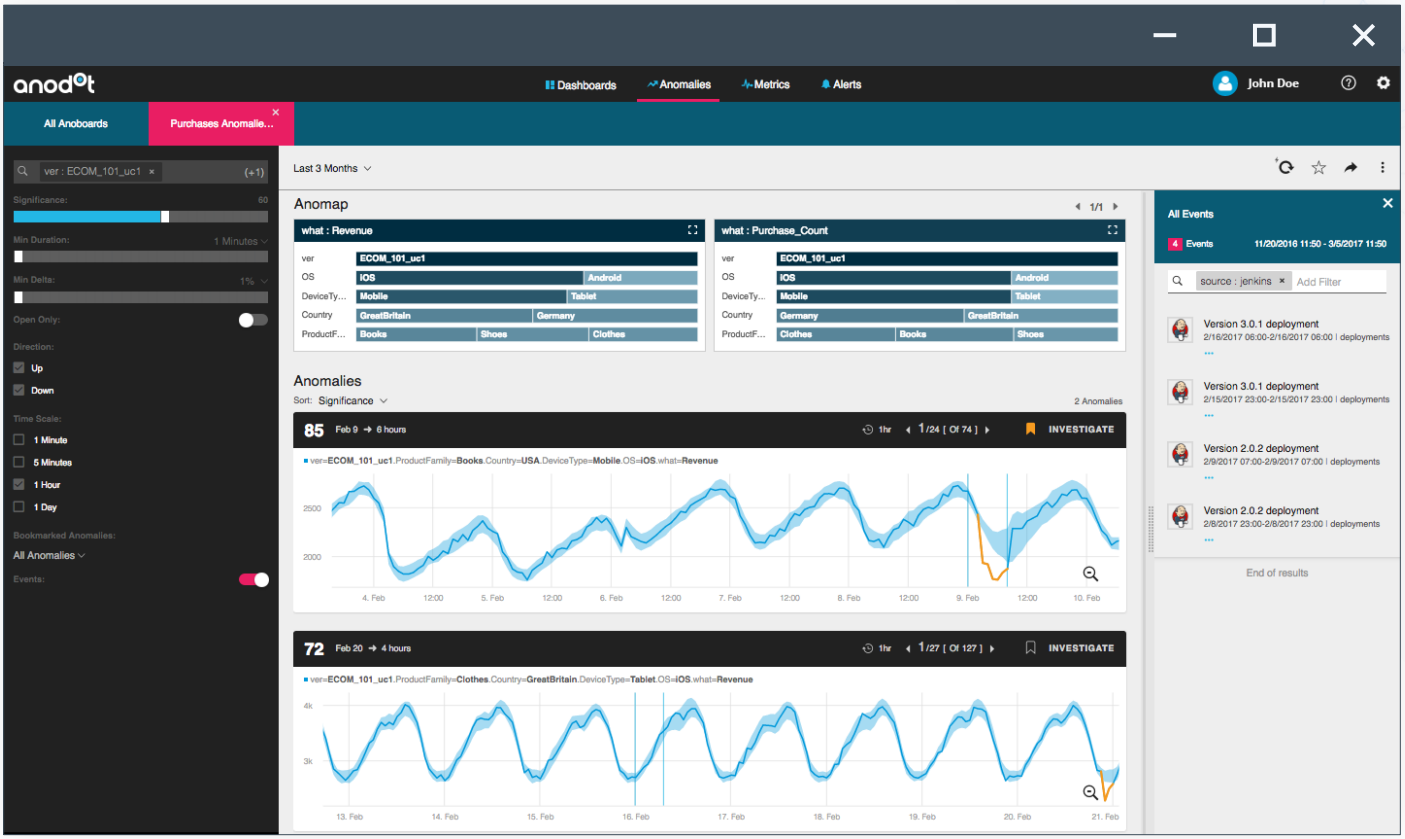

Figure 7. *A screenshot of the Anodot "Anoboard" Dashboard showin outliers detected in time series data.*

THE HUMAN ELEMENT

This white paper series has focused on the technical elements necessary to build an outlier detection system, a recipe, as it were. But what about the chef? It is not enough to pick up the ingredients at the market; someone has still must cook the meal. This brings us to the human factor of an outlier detection system -- the team needed to build the system.

At a minimum, you will need a team of data scientists with a specialty in time series data and online machine learning. Just as chefs and doctors have their own specialties, data scientists do as well. While there is a shortage of data scientists in the market in general, the scarcity is even more acutely felt when searching for particular specialties such as time series, and you may find yourself in competition for talent with companies such as Google, Facebook and other industry giants.

Besides the data scientists, you need a team of developers and other experts to build a system around the algorithms which is efficient at scalable stream processing and developing backend systems and has an easy-to-use user interface. At the bare minimum, you would need backend developers creating data flows, storage, and management of the large scale backend system, in addition to UI experts and developers, QA and product management.

Note that this team not only has to develop and deploy the system, but maintain it over time.

While one might be tempted to skimp on UI for an internally-developed solution, this is a mistake. An early investment in UI means that the eventual outlier detection system will be able to be used widely in the organization by multiple teams, with multiple needs. If the UI is not simple enough for everyone to learn easily, the data science and business intelligence teams will forever find themselves as the frustrating bottleneck, providing retroactive reports and alerts.

The more people using an outlier detection system within the organization (and the more metrics being analyzed), the more powerful the insights it can provide.

Conversely, the more people using the outlier detection system within the organization (and the more metrics being analyzed), the more powerful the insights it can provide. For example, at Anodot, we have customers that have hundreds of people on dozens of different teams – from sales to executive management to BI to monitoring to devops – using the Anodot system to alert them to outliers relevant to their areas of responsibility.

Based on our own experience and discussions with our customers who have faced the “build or buy” decision, we estimate that it would take a minimum of 12 human years (a team of data scientists, developers, UI and QA) to build even the most rudimentary outlier detection system. And this basic system could still encounter various technical issues that are far beyond the scope of this paper.

SUMMARY

Across this series of three white papers, we have covered the critical processes and various types of learning of a large-scale outlier detection system.

- In Part I, we discussed what an outlier is, and why a business would want to detect outliers. We outlined the five main design considerations when building an automated outlier detection system: timeliness, scale, rate of change, conciseness, and definition of incidents. And finally, we discussed supervised and unsupervised machine learning methods.

- In Part II, we detailed the processes of learning the normal behavior of time series data. After all, we need to know what is normal for a business system in order to identify what is not normal—an outlier. We talked about creating data models, uncovering seasonality, and the importance of online adaptive learning models.

- In Part III, this document, we discussed how to identify and correlate abnormal behavior to determine the significance of outliers. This process is critical for distilling the total number of discovered outliers into a much smaller number of only the most important outliers. Without this distillation process, there would be too many alerts to investigate in a timely and cost effective manner.

Hopefully these documents have given the reader some insight to the complexity of designing and developing a large-scale outlier detection system. The Anodot system has been carefully designed using sophisticated data science principles and algorithms, and as a result, we can provide to our customers truly meaningful information about the outliers in their business systems.

anodot

Looking for another useful read? We recommend our ebook "Automated Time Series Analysis for Anomaly Detection on Amazon Web Services

Download eBook Now

I. Anodot uses an enhanced version of the latent Dirichlet allocation (LDA) algorithm in a unique way to calculate abnormal based similarity. In natural language processing, LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word's creation is attributable to one of the document's topics.

In LDA, each document may be viewed as a mixture of various topics, where each document is considered to have a set of topics that are assigned to it via LDA. In practice, this results in more reasonable mixtures of topics in a document.

For example, an LDA model might have topics that can be classified as CAT_related and DOG_related. A topic has probabilities of generating various words, such as "milk," "meow" and "kitten," which can be classified and interpreted by the viewer as CAT_related. Naturally, the word "cat" itself will have high probability given this topic. The DOG_related topic likewise has probabilities of generating each word: "puppy," "bark" and "bone" might have high probability. Words without special relevance, such as "the" will have roughly even probability between classes (or can be placed into a separate category). A topic is not strongly defined, neither semantically nor epistemologically. It is identified on the basis of supervised labeling and (manual) pruning on the basis of their likelihood of co-occurrence. A lexical word may occur in several topics with a different probability, however, with a different typical set of neighboring words in each topic. (Wikipedia)

II. TF-IDF is short for "term frequency–inverse document frequency." It is a numerical statistic intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval and text mining. (Wikipedia)

For more information, please contact Anodot:

North America
669-600-3120
info.us@anodot.com

International
+972-9-7718707
info@anodot.com

Anodot was founded in 2014, and since its launch in January 2016 has been providing valuable business insights through outlier detection to its customers in financial technology (fin-tech), ad-tech, web apps, mobile apps, eCommerce and other data-heavy industries. Over 40% of the company's customers are publicly traded companies, including Microsoft, VF Corp, Waze (a Google company), and many others. Anodot's real-time business incident detection uses patented machine learning algorithms to isolate and correlate issues across multiple parameters in real time, supporting rapid business decisions.
Learn more at http://www.anodot.com/.