anodot

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

Part II:

Learning Normal Time Series Behavior

INTRODUCTION

Outlier detection helps companies determine when something changes in their normal business patterns. When done well, it can give a company the insight it needs to investigate the root cause of the change, make decisions, and take actions that can save money (or prevent losing it) and potentially create new business opportunities.

High-velocity online businesses need real-time outlier detection; waiting for days or weeks after the outlier occurs is simply too late to have a material impact on a fast-paced business. This puts constraints on the system to learn to identify outliers quickly, even if there are a million or more relevant metrics and the underlying data patterns are complicated .

Automated outlier detection is a technique of machine learning, and it is a tremendously complex endeavor. In this series of white papers, Anodot aims to help people understand some of the sophisticated decisions behind the algorithms that comprise an automated outlier detection system for large scale analytics, especially in the monitoring of time series data. In Part I of this white paper series, we outlined the various types of machine learning and the critical design principles of an outlier detection system. We highly recommend reading Part I to get the foundational information necessary to comprehend this document.

In Part II, we will continue the discussion with information about how systems can learn what "normal behavior" in time series data looks like, in order to identify abnormal behavior. Part III of our white paper series will cover the processes of identifying and correlating abnormal behavior. In each of the documents, we discuss the general technical challenges and Anodot's solutions to these challenges.

Outlier detection is an imperative for online businesses today, however building an effective system in-house is a complex task. It is a particular challenge to first learn the normal behavior of time series data in order to identify events that differ from the norm, defined here as outliers.

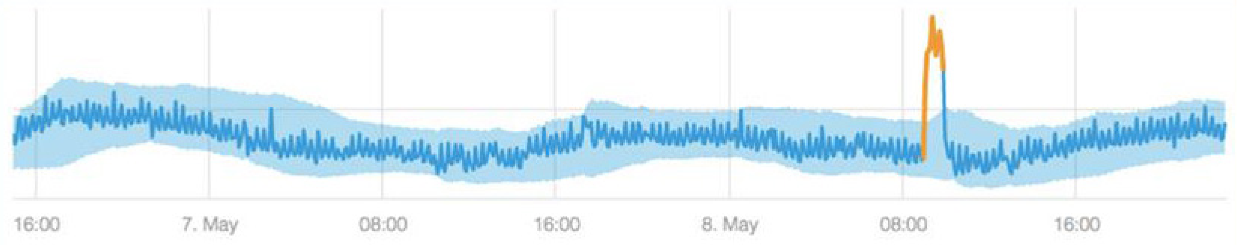
The techniques described within this paper are well grounded in data science principles and have been adapted or utilized extensively by the mathematicians and data scientists at Anodot. The veracity of these techniques has been proven in practice across hundreds of millions of metrics from Anodot's large customer base. A company that wants to create its own automated outlier detection system would encounter challenges like those described within this document.

High velocity online businesses need real-time outlier detection; waiting for days or weeks after the outlier occurs is simply too late to have a material impact on a fast-paced business.

A GENERAL FRAMEWORK FOR LEARNING NORMAL BEHAVIOR

The general process of any outlier detection method is to take data, learn what is normal, and then apply a statistical test to determine whether any data point for the same time series in the future is normal or abnormal.

Consider the data pattern in **Figure 1** below. The shaded area was produced because of such statistical analysis. We could, therefore, apply statistical tests such that any data point outside of the shaded area is defined as abnormal and anything within it is normal.



GENERAL SCHEME

STEP 1 > **STEP 2** > **STEP 3**

STEP 1: Model the normal behavior of the metric(s) using a statistical model.

STEP 2: Devise a statistical test to determine if samples are explainded by the model.

STEP 3: Apply the test for each sample. Flag as anomaly if it does not pass the test.

Figure 1. *A general scheme for outlier detection.*

The graph below is a normal distribution represented by an average standard deviation. Given a large number of data points, 99.7% of the data points submitted should fall within the average, plus or minus three times the standard deviation. This model is illustrated with the formula in **Figure 2**.

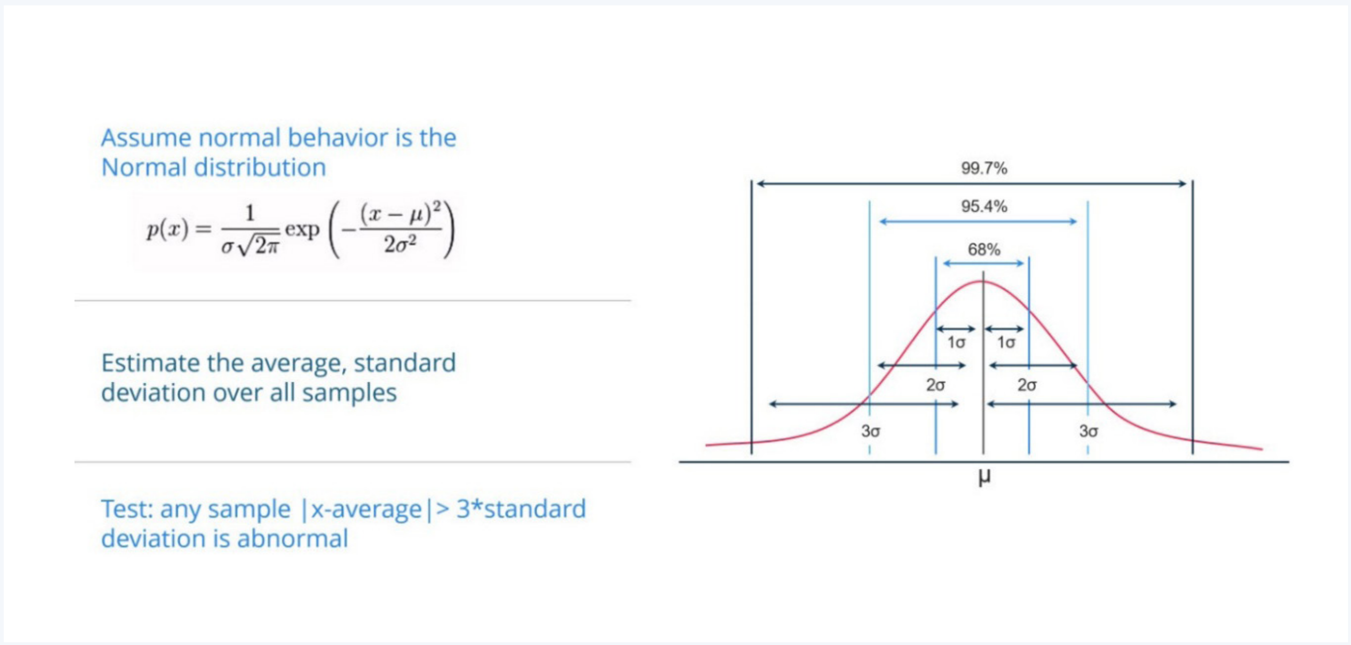
Making this assumption means that if the data comes from a known distribution, then 99.7% of the data points should fall within these bounds. If a data point is outside these bounds, it can be called an outlier because the probability of it happening normally is very small.

This is a very simple model to use and to estimate. It is well known and taught in basic statistics classes, requiring only computation of the average and the standard deviation. However, assuming any type of data will behave like the normal distribution is naïve; most data does not behave this way. This model is, therefore, simple to apply, but usually much less accurate than other models.

There are many different distributions that can be assumed on data; however, given a very large dataset, there are most likely many different types of behavior in the data. This very fact has been thoroughly researched for hundreds of years, and even more so in the last 50 years as data science has become important in the computing world. But the question is, given a huge amount of literature, techniques and models to choose from, how can someone choose only one model?

The answer is that it is not possible to choose just one.

At Anodot, we look at a vast number of time series data and see a wide variety of data behaviors, many kinds of patterns, and diverse distributions that are inherent to that data. There is not one type of distribution that fits all possible metrics. There has to be some way to classify each signal to decide which should be modeled with a normal distribution, and which should be modeled with a different type of distribution and technique.



Assume normal behavior is the Normal distribution

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Estimate the average, standard deviation over all samples

Test: any sample |x-average|> 3*standard deviation is abnormal

Figure 2. *The mathematical formula for average standard deviation.*

Choosing just one model does not work, and we have seen it even within a single company when they measure many different metrics. Each metric behaves differently. In Part I of this document series, we used the example of a person's vital signs operating as a complete system. Continuing with that example, the technique for modeling the normal behavior of a person's heart rate may be very different from that which models his or her temperature reading.

The body's temperature is a reading that is very close to a normal distribution. If we were to graph it for a single person over time, we would see that it fits the normal distribution very well, but a lot of other vital signs are not like that at all.

If we look at a person's heart rate, it changes constantly throughout the day depending on whether the person is awake or not, or whether he or she is doing something strenuous like exercising. It is multimodal and seasonal— multimodal because the person's heart rate changes to different states based on activities, and seasonal because it is likely to follow the daily human cycle (sleep/awake). The heart has a different rate while the person is running compared to when he or she is relaxing; even while running, the rate changes to different modes of operation during a steady run or strenuous sprints.

A SINGLE MODEL DOES NOT FIT ALL METRICS

In the Anodot system, every dataset that comes in goes through a classification phase where we categorize it according to what type of model it fits best. We have created a bank of model types that each fits one of these signal types, some of which are illustrated in **Figure 3**.

For companies that choose to build their own outlier detection system, this is often where the first part of the complexity comes into play. Most open source techniques deal with "smooth metrics." The metrics are not normal distribution, but they tend to be very regularly sampled and have stationary behavior. They tend to have behaviors that don't change rapidly, and they don't exhibit other behaviors. Applying open source techniques only covers a fraction of what is measured and if they are applied on metrics that are not smooth, the result will either be a lot of false-positives or there will be many outliers that are not detected (i.e. false-negatives), because the metric is fitted with the wrong model.

Not everything is smooth and stationary, and those models only work on a fraction of the metrics. Worse, it is difficult to know which metrics are like this. Those datasets would somehow have to be identified.

Consider the pattern in the signal shown in **Figure 4**. If the smooth techniques are applied on this data, the little spikes that seem completely normal would be considered anomalous and would generate alerts every minute. The smooth model would not work here.

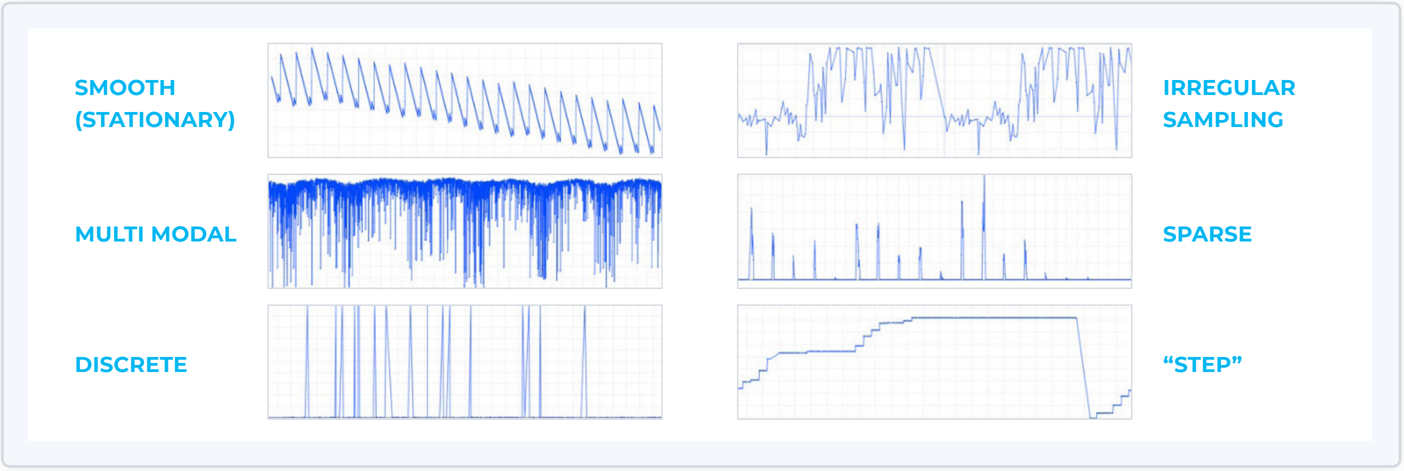


Figure 3. *A sampling of some of the data models Anodot uses.*

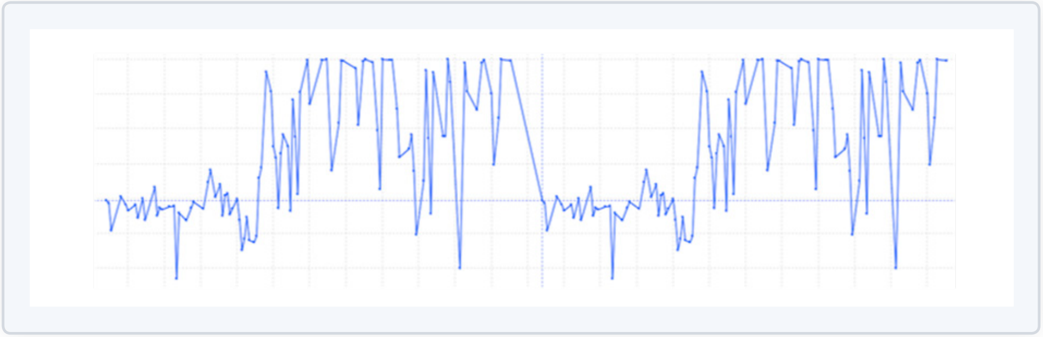


Figure 4. *A metric with an unusual pattern doesn't fit a smooth model.*

Knowing what a data pattern looks like in order to apply an appropriate model is a very complex task.

If a company has 10 metrics, it is possible to graph the data points with a statistician. With only 10 metrics, this is feasible to do manually; however, with many thousands or millions of metrics, there is no practical way to do this manually. The company would have to design an algorithm that would determine the proper data model to use for each metric.

There is another aspect we have observed quite often with the data we see from our customers: the model that is right today, may not be right tomorrow. In **Figure 5**, we see how a metric's behavior can change overnight.

We have seen this happen many times, and each time, it was totally unexpected; the data starts out one way and then goes into a totally different mode. It may start kind of smooth and then change to steep peaks and valleys—and stay there. That pattern becomes the new normal. It is acceptable to say at the beginning of the new pattern, that the behavior is anomalous, but if it persists, we must call it the new normal.

Let us consider how this affects the company building its own detection system. The company's data scientist will spend several weeks classifying the data for the company's 1,000 metric measurements and make a determination for a metric model. It could be that a week from now, what the data scientist did in classifying the model is irrelevant for some of them—but it may not be clear for which ones.

What is needed, then, is an automated process that constantly looks at the changing nature of data signals and decides what the right model is for the moment. It is not static.

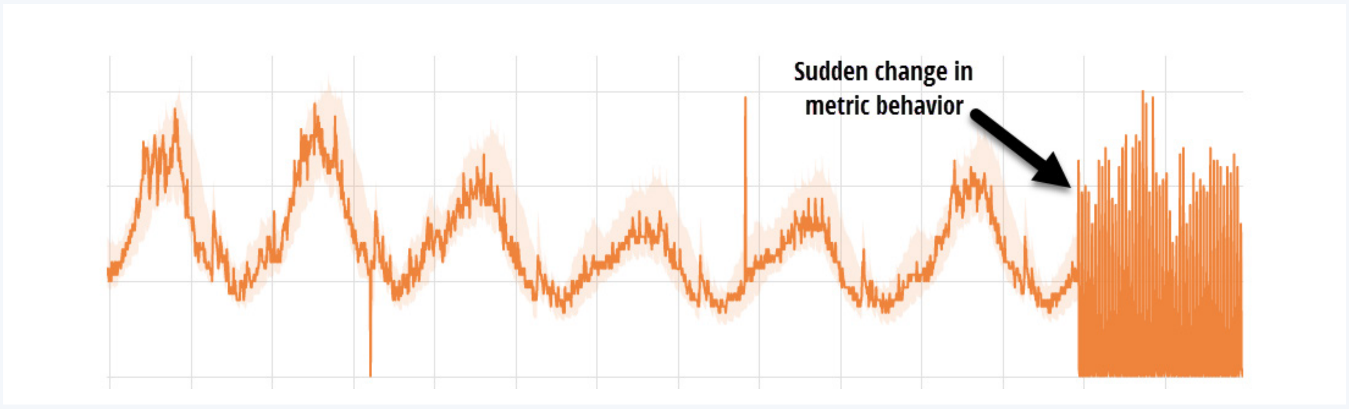


Figure 5. *Sudden change in metric behavior.*

THE IMPORTANCE OF MODELING SEASONALITY

Other important aspects that should be included in the algorithms and the model is whether the data has seasonal patterns and what the seasonal periods are. A seasonal pattern exists when a series is influenced by seasonal factors (e.g., the quarter of the year, the month, the hour of the day or the day of the week). Seasonality is frequently, but not always, of a fixed and known period. This is illustrated in **Figure 6** below.

We know that many different metrics that are measured have seasonal patterns, but the pattern might be unknown. Nevertheless, it is important to take the seasonal pattern into consideration for the model. Why? If the model of what is normal knows to account for a metric's seasonal pattern, then it is possible to detect the outliers in samples that vary from the seasonal pattern. Without considering the seasonal pattern, too many samples might be falsely identified as outliers.

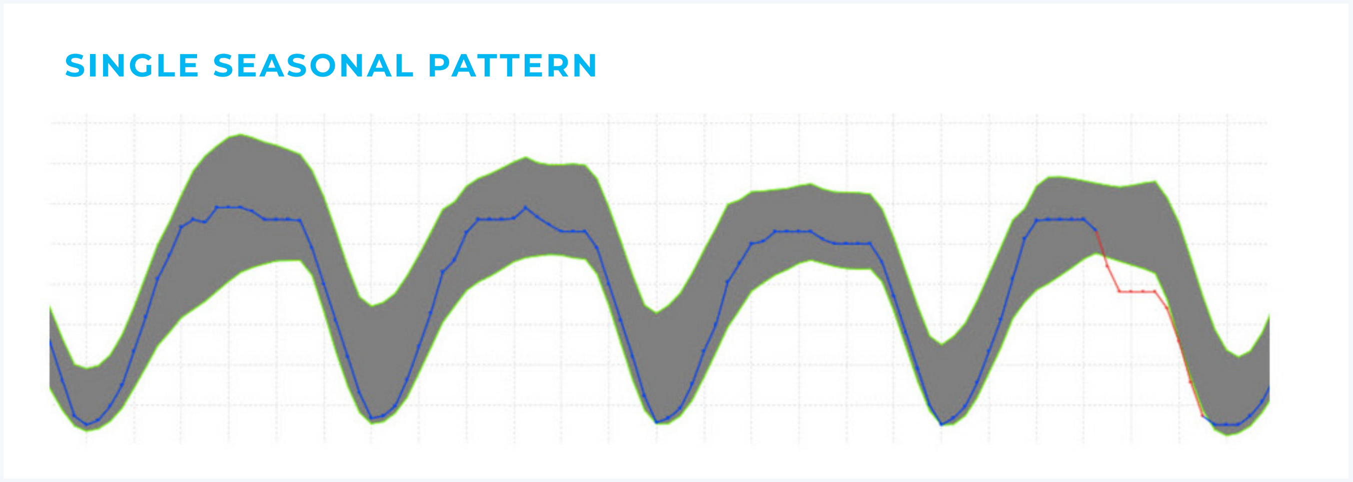


Figure 6. *Illustration of a single seasonal pattern.*

Often we see, not just a single seasonal pattern, but multiple seasonal patterns and even different types of multiple seasonal patterns, like the two examples shown in **Figure 7**.

Figure 7 shows an example of a real metric with two seasonal patterns working together at the same time. In this case, they are weekly and daily seasonal patterns. The image shows that Fridays and weekends tend to be lower, while the other days of the week are higher. There is a pattern that repeats itself week after week, so this is the weekly seasonal pattern. There is also a daily seasonal pattern that illustrates the daytime hours and nighttime hours; the pattern tends to be higher during the day and lower during the night.

These two patterns are intertwined in a complicated way. There is almost a sine wave for the weekly pattern, and another faster wave for the daily pattern. In signal processing, this is called amplitude modulation, and it is normal for this metric. If we do not account for the fact that these patterns co-exist, then we do not know what normal is. If we know how to detect it and take it into account, we can detect very fine outliers like the ones shown in orange in Figure 7 above. The values in orange indicate a drop in activity which may be normal on a weekend but not on a weekday. If we do not know to distinguish between these patterns, we will not understand the outlier, so we either miss it or we create false-positives.

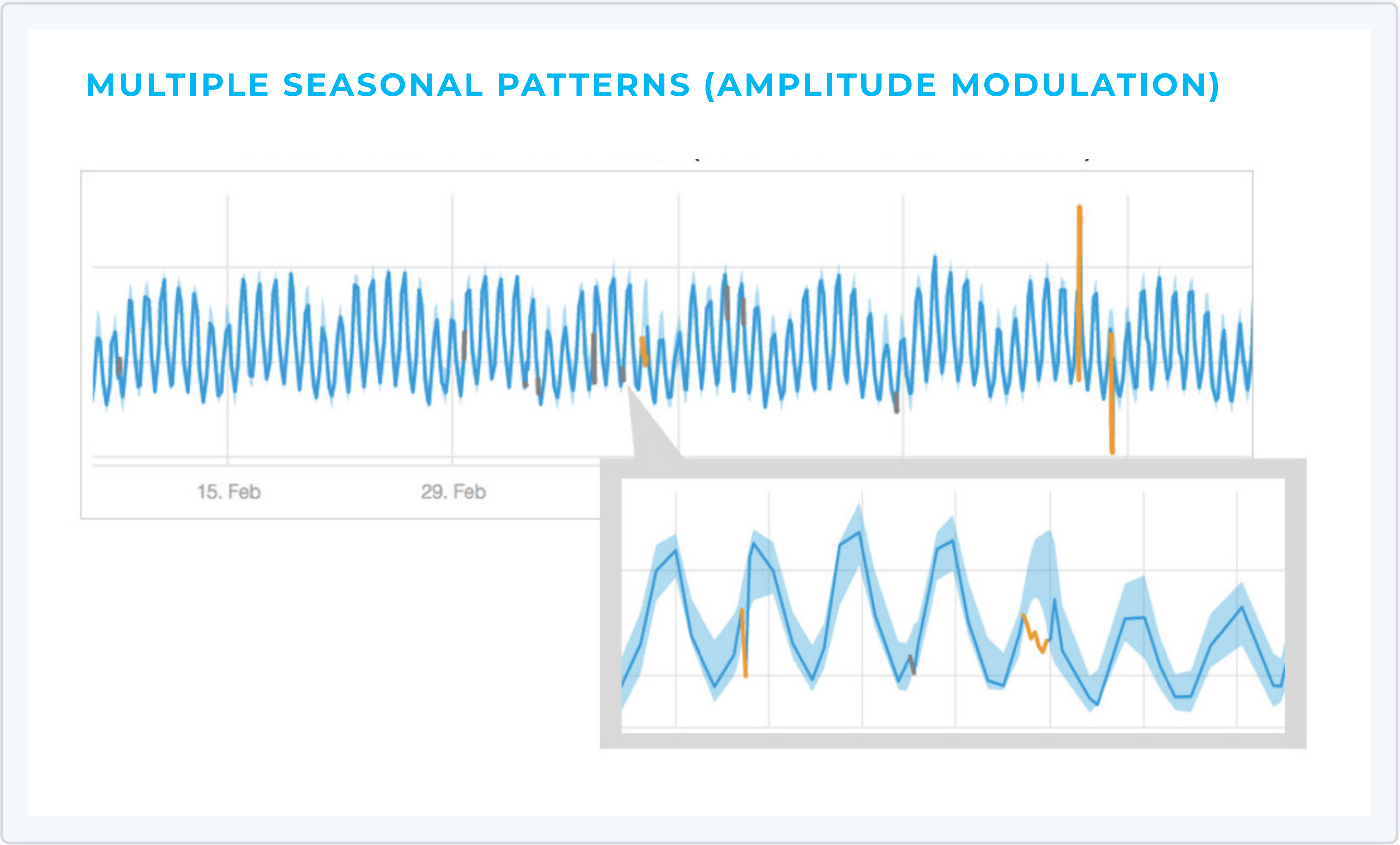


Figure 7. *Illustration of multiple seasonal patterns.*

Figure 8 below shows an example of another type of multiple seasonal patterns—one with additive signals.

In the example above, we see a clear daily pattern. In addition, we see an event that occurs every four hours which causes a spike that lasts for an hour and then comes down. The spikes are normal because of a process or something that happens regularly. The orange line shows an outlier that would be very hard to detect if we did not take into account that there is both the daily pattern and the spikes every four hours. We call this pattern "additive" because the spikes are added to what normally happens during the day; the pattern shows a consistent spike every four hours on top of the daily pattern.

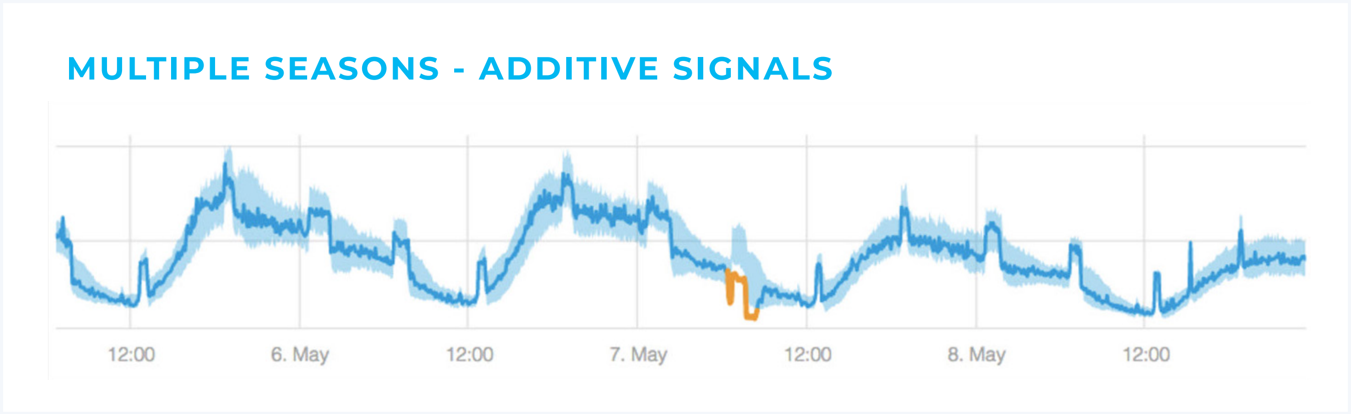


Figure 8. *An Illustration of multiple seasonal patterns with additive signals.*

CAN A SEASONAL PATTERN BE ASSUMED?

At Anodot, we have observed millions of metrics and built algorithms that detect the seasonal patterns – if any – that exist in them. Some of them – in fact, most of them – do not have a seasonal pattern. Out of millions of metrics that Anodot has seen, about 14% of them have a season to them, meaning 86% of the metrics have no season at all. Out of the metrics with a seasonal pattern, we have observed that 70% had a 24-hour pattern to them, and 26% had weekly patterns. The remainder of the metrics with a seasonal pattern had other types of patterns—four hours, six hours, and so on.

If we assume there are no seasonal patterns in any of the metrics and we apply standard techniques, we are either going to be very insensitive to outliers or too sensitive to them, depending on what technique we use. However, making assumptions about the existence of a seasonal pattern has its issues as well.

There are two problems with assuming a seasonal pattern (e.g., daily or weekly). First, it may require too many data points to obtain a reasonable baseline (in case there is no seasonal pattern in the metric), or it would produce a poor normal model all together (if there is a different seasonal pattern in the metric).
If we assume a weekly seasonal pattern for all our metrics, it would require many more data points to converge to a metric baseline. Not only does this take time, but the process might not converge to the right distribution if it is a variable metric.

Two problems with assuming a seasonal pattern:

- **May require too many data points to obtain a reasonable baseline**
- **May produce a poor normal model**

Second, if the wrong seasonal pattern is assumed, the resulting normal model may be completely off. For example, if the data point is assumed to be a daily seasonal pattern, but it is actually a 7-hour pattern, then comparing 8 AM one day to 8 AM another day is not relevant. We would need to compare 8 AM one day to 3 PM that same day. Improperly defining the seasons will lead to many false-positives due to the poor initial baseline.

For this reason, some tools require the user to define the season in order for the tool to estimate the baseline. Of course, this is not scalable for more than a few dozen metrics. What is needed is a system that will automatically and accurately detect seasonality (if it exists). If this capability is not built into the system, assumptions will have to be made that are going to cause an issue, either from the statistics side in needing more data, or from the accuracy side in identifying outliers.

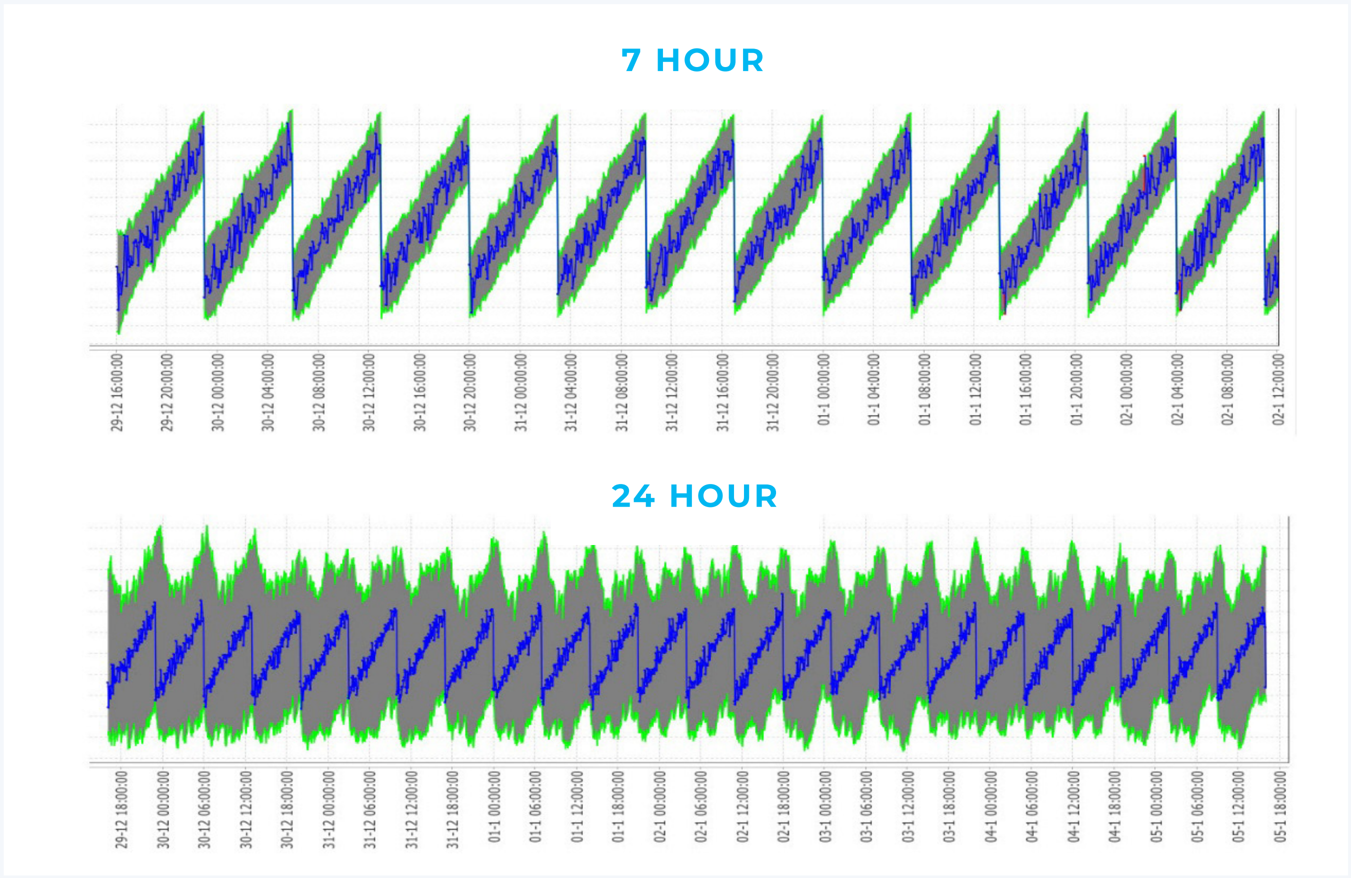


Figure 9. *Comparing a 7-hour seasonal pattern with an assumed 24-hour seasonal pattern.*

EXAMPLE METHODS TO DETECT SEASONALITY

Now that we have established the importance of determining if seasonality is present in the data, we will briefly discuss a few common methods to detect it.

One method uses Fourier transform of signals, a technique in mathematics that takes a signal, transforms it to the frequency domain and finds frequencies that are local maximums (peaks) in the power of the Fourier transform. Those peaks tend to occur where there are seasonal patterns. This technique is fast and efficient, but it does not work well when there are multiple seasonal patterns. Additionally, it is difficult to detect low-frequency signal patterns like weekly, monthly or yearly, and this technique is very sensitive to any missing data. Also, issues like aliasing in the Fourier transform can cause multiple peaks to be present, some of which are not the actual seasonal frequency, but rather artifacts of the Fourier transform computation.

Another technique is autocorrelation of signals, also known as serial correlation or autocorrelogram (ACF), the correlation of a signal with itself at different points in time. Informally, it is the similarity between observations as a function of the time lag between them. It is a mathematical tool for finding repeating patterns. Compared to the Fourier transform method, it is more accurate and less sensitive to missing data, but it is computationally expensive.

Anodot developed a proprietary algorithm which we call Vivaldi (patent pending). At a high level, Vivaldi implements detection using the ACF method, but overcomes its shortcomings by applying a smart subsampling technique, computing only a small subset of ACF coefficients, thus reducing computational complexity. In addition, to accurately identify multiple seasonal patterns, the method is applied on multiple filtered versions of the time series. The method has been proven to be accurate both theoretically and empirically, while very fast to compute.

FINDING MAXIMUM(S) IN FOURIER TRANSFORM OF SIGNAL	FINDING MAXIMUMS IN AUTOCORRELATION OF SIGNAL	ANODOT VIVALDI METHOD
• Challenging to detect low frequency seasons • Challenging to discover multiple seasons • Sensitive to missing data	• Computationally expensive • More robust with regard to gaps	• Based on autocorrelation • Uses smart subsampling to reduce computational complexity • Provably accurate

Table 1. *Techniques to detect seasonality.*

REAL-TIME DETECTION AT SCALE REQUIRES ONLINE[1] ADAPTIVE LEARNING ALGORITHMS

Companies that want immediate insight to changes in their business operations need prompt notification of outliers in their data. This means that the algorithms used to detect outliers must have all the properties that we have discussed above – i.e., detecting seasonality, automatically determining the proper model, etc. – but they also must adapt to changing conditions in the datasets they are processing. Unexpected changes in the data can be outliers, at least initially, but they can also be indicative of a change in the data pattern. This can happen if, for example, e-commerce sales or the number of visitors to a website suddenly surges due to a successful marketing campaign.

This underscores the need for online adaptive learning algorithms which learn the model with every new data point that comes in. This type of learning algorithm does not wait to receive a batch of data points to learn from; rather, it updates what has been learned so far with every new data point that arrives. These so-called online learning algorithms do not have to be adaptive, but by nature they usually are, which means that every new data point changes what has been learned up to that time.

1. What do we mean by "online" machine learning? This is not a reference to the Internet or the World Wide Web. Rather, "online" is a data science term that means the learning algorithm takes every data point, uses each one to update the model and then does not concern itself with that data point ever again. The algorithm never looks back at the history of all the data points, but rather goes through them sequentially.

It is not necessarily real-time because time is not a factor here. In an e-commerce example, time can be a factor, but in the general sense of an online learning algorithm, it just means that if there are 1,000 data points to learn from, the algorithm goes through them one by one to learn from them, throws them away and then moves on to the next one. It is more of a sequential learning algorithm. The word "online" is widely used in the world of data science but it has nothing to do with the Internet; this is simply the term used in literature. For more information, see the Wikipedia entry about online machine learning.

We can contrast an online learning model to a model that uses data in batch mode. For example, a video surveillance system that needs to recognize human images will learn to recognize faces by starting with a dataset of a million pictures that includes faces and non-faces. It learns what a face is and what a non-face is in batch mode before it starts receiving any real data points.

In the online learning paradigm, the machine never iterates over the data. It gets a single data point, learns what it can from it, and then throws it away. It gets another data point, learns what it can from it, throws it away, and so on. The machine never goes back to previously used data to relearn things; this is similar to how our brains learn. When we encounter something, we learn what we can from it and move on, rather than storing it for later use.

An online adaptive learning algorithm works by initializing a model of what is normal. It takes a new data point in the next second, minute, hour or whatever timeframe is appropriate. First, the machine tests if the current data point is an outlier or not, based on what it already knows. If it marks the data point as not being an outlier, then it updates the current model about what is normal based on that data point. And then it repeats the process as individual new data points come in sequentially.

The machine never goes back to previously viewed data points to put them into a current context. The machine cannot say, “Based on what I see now, I know that this data point from five days ago is actually not an outlier.” It cannot consider, “Maybe I should have done something different.” The luxury of going back in time and reviewing the data points again does not exist, which is one of the shortcomings of this paradigm. The advantage of this approach is that it is fast and adaptive; it can produce a result now and there is no need to wait to collect a lot of data before results can be produced. In cases where a rapid analysis is needed, the advantages of this approach far outweigh its disadvantages.

There are various examples of online adaptive learning models that learn the normal behavior of time series data that can be found in data science, statistics and signal processing literature. Among them are Simple Moving Average, Double/Triple Exponential (Holt-Winters) and Kalman Filters + ARIMA and variations.

The following is an example of how a simple moving average is calculated and how it is applied to outlier detection. We want to compute the average over a time series, but we do not want the average from the beginning of time until present. Instead, we want the average during a window of time because we know we need to be adaptive and things could change over time. In this case, we have a moving average with a window size of seven days, and we measure the metric every day. For example, we look at the stock price at the end of every trading day. The simple moving average would compute the average of the stock price over the last seven days. Then we compare tomorrow’s value to that average and see if it deviates significantly. If it does deviate significantly from the average value, it is an outlier and if not, then it is not an outlier. Using a simple moving average is a straightforward way of considering whether we have an outlier or not.

The other models listed above are (much) more complex versions of that but, if one can understand a simple moving average, then the other models can be understood as well.

THE IMPACT OF LEARNING RATE – AVOIDING PITFALLS

All of these adaptive online algorithms have some notion of learning rate. In the stock price example, we looked at the average value over the last seven days of the stock price and then compared the next day to that value. In this example, the seven-day period is a parameter known as the "learning rate." Why not 30 days? Why not 180 days? The shorter we make the learning rate, the more of an effect each daily data point has on the moving average. If we make it a moving average of the last three days, it will learn any changes that happen faster. If we make it 365 days, then it will learn very slowly because every day will have a very small effect on that average.

If our learning rate is too slow, meaning our moving average window is very large, then we would adapt very slowly to any changes in that stock price. If there are big changes in the stock price, then the baseline – the confidence interval of where the average should be – will be very large, and we will be very insensitive to changes.

If we make the rate too fast – i.e., the window is very small – then we will adapt too quickly and we might miss things. We might think that outliers are not outliers because we are adapting to them too quickly.

These scenarios are depicted in **Figure 10** below.

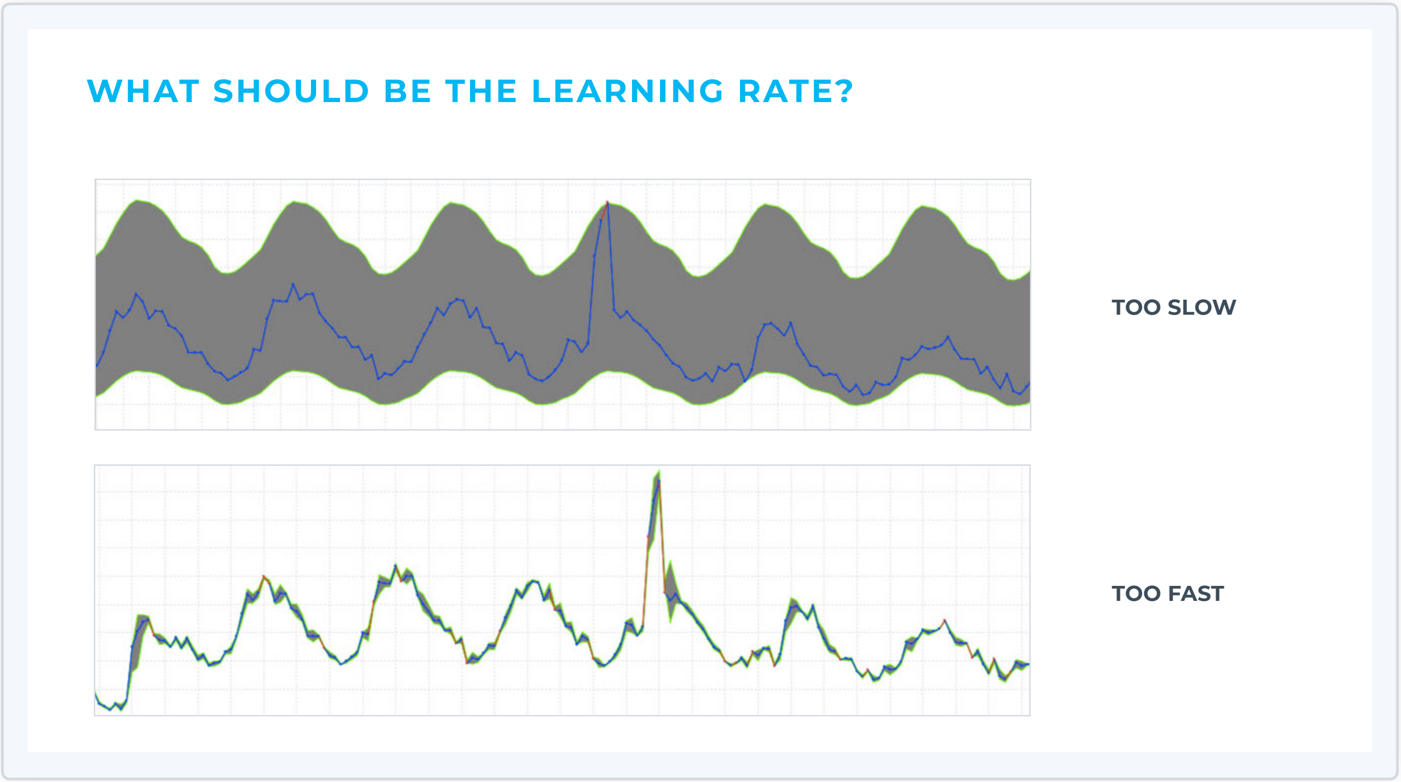


Figure 10. *The effect of learning rate on detecting outliers.*

How do we know what the learning rate should be? If we have a small number of time series – 100 or fewer – we could inspect and change the parameters as needed. However, a manual method will not work when we have a large number of time series, so the algorithms need to automatically tune themselves.

There are many different metrics and each one has its own behavior. The rate at which these metrics change could be fast or slow depending on what they are; there is no one set of parameters that fits them all well. Auto-tuning these parameters is necessary to provide an accurate baseline for millions of metrics. This is something that is often overlooked by companies building outlier detection systems (incidentally, auto-tuning is built into the Anodot system). Auto-tuning is not an easy task, but it is an important one for achieving more accurate results.

There is another pitfall to be aware of. If we have a metric and we tune the learning rate to fit it well when it behaves normally, what happens when there is an outlier?

Consider a scenario where we have a data point that is an outlier. Recall that the three steps of online learning are to read the sample, update the model using the data point, and move on to the next data point. What happens if a data point is an outlier? Do we update the model with the new data point or not?

Continuing the example of learning the stock price model using the moving average method, if we include an anomalous data point in the learning process, the stock price now becomes anomalous as well. If we use it the next day to compute the next moving average, then we completely shift the average toward that outlier. Is that okay or not okay? Good or bad? What happens in reality is, if we allow it to shift the average or shift the parameters of the model as usual, then if the outlier persists beyond that single data point, we will start shifting the normal behavior towards that outlier. If the outlier lasts for a while, then at some point we will say this is the new normal, and we might even miss other outliers that come along later. Or, whenever it goes back to normal, we will say that is an outlier as well.

Updating the model with every data point (including abnormal ones), is one strategy, but it is not a very good one.

ADAPTING THE LEARNING RATE

A better strategy is to adapt the learning rate by assigning weight to the validity of the data points, with an outlier carrying a lower weight than a normal value. This is a tactic that Anodot uses. Whenever Anodot sees that a data point is an outlier, the system assigns that value a very low weight when adapting the model parameters.

Going back to the moving average example, if we have a seven-day moving average and we get the next data point and see it is outside the expected range, we categorize it as anomalous compared to the previous average. The Anodot system will use the anomalous data point to update the moving average; but instead of using it as is, Anodot gives it a weight as though it is one of 1,000 data points. That data point will affect the average, but only in a very small way.

Why not just ignore the outlier in terms of learning and not include it in the model? Quite often outliers happen because something really has changed—and it is okay that it changed. We want to pick up on an outlier when it changes, but we eventually want to adapt to it and go back to that new state. A metric could spike and stay high for a very long time. Perhaps somebody made a change in the measured item and it is perfectly fine. We want to know about it in the beginning, but after a while we want the system to adapt to that new state. If we do not let it affect what has been learned, then the system will never adapt to the new state, and will be stuck in the previous state.

An example of this would be a company that does a stock split. All of a sudden, the stock price is cut in half; instead of it being $100 a share, it suddenly drops to $50. It will stay around $50 for a while and the outlier detection system must adapt to that new state. Identifying that the drop is an outlier is not a bad thing, especially if we are unaware there was a split, but eventually we want our normal value to go down to around that $50 state.

Another example would be a merger. One company acquires another company, the stock price goes up or down and it may stay at that new value for a long time. The valuation of the company has changed quite suddenly, and the system eventually needs to adapt to that new valuation.

In the online world, these types of changes happen a lot. For example, a company has a Web application and after a large marketing campaign, the number of users quickly increases 25 percent. If the campaign was good, the number of users may stay elevated for the long term. When a SaaS company adds a new customer, its application metrics will jump, and that is normal. They might want to know about that outlier in the beginning, but then they will want the outlier detection system to learn the new normal.

These kinds of events happen frequently; we must not ignore them by not allowing those data points to affect anything from now until eternity. On the other hand, we do not want the system to learn too quickly, otherwise all outliers will be very short, and if it goes back to the previous normal state, measurements will be off. There is a fine balance here: how fast we learn versus how adaptive we are.

In the Anodot system, when we see outliers, we adapt the learning rate in the model by giving the anomalous data points a lower weight. If the outlier persists for a long enough time, we begin to apply higher and higher weights until the anomalous data points have a normal weight like any other data point, and then we model to that new state. If it goes back to normal, then nothing happens; it just goes back to the previous state and everything is okay.

These two approaches to updating the learning rate are shown below. In **Figure 11**, the model is updated without weighting the outliers. In this instance, most of the outlier is actually missed by the model being created.

In **Figure 12**, outliers are weighted differently to minimize their impact on normal, unless it becomes apparent that the outliers are the new normal. This method allows for the outlier to be fully captured.

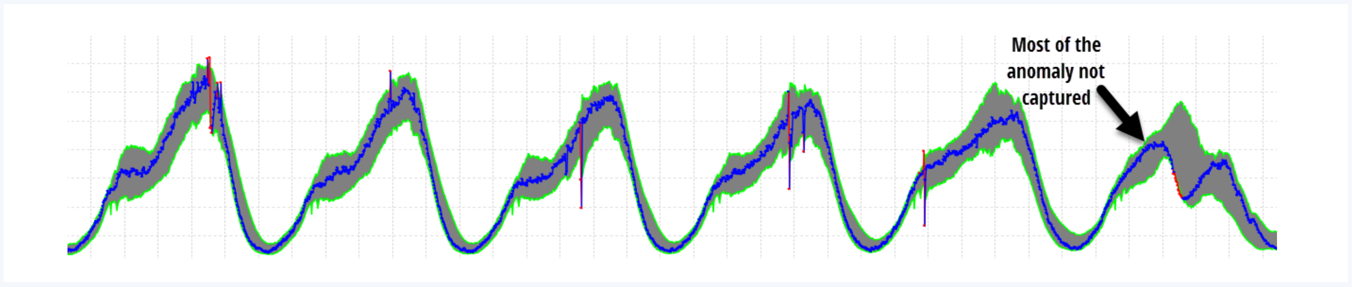


Figure 11. *Updating a model using an outlier as a fully-weighted data point.*

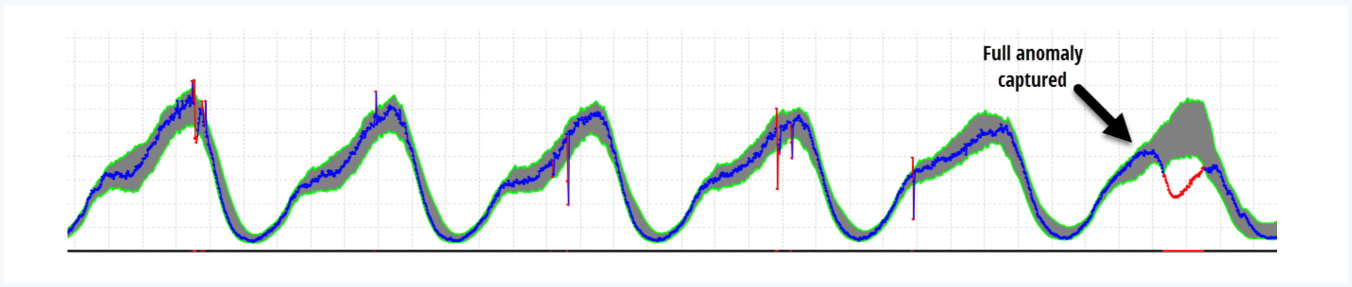


Figure 12. *Updating a model by assigning a lower weight to outliers.*

OTHER METHODS FOR LEARNING NORMAL BEHAVIORAL PATTERNS

This paper covers online adaptive learning methods that fit with the design principles laid out in Part I of this document series. These are the methods that Anodot has selected for its solution; however, there are other methods for learning normal behaviors in data patterns. We summarize them in the table below according to the design criteria defined in Part I.

NORMAL BEHAVIORAL LEARNING METHODS

Name	Adaptive?	Real-time?	Scalable?	Uni/Multi Variate
Holt-Winters	Yes	Yes	Yes	Univariate
ARIMA + Kalman	Yes	Yes	Yes	Both
HMM	No	Yes	No	Multivariate
GMM	No	No	No	Both
DBScan	No	No	No	Multivariate
K-Means	No	No	No	Multivariate

Table 2. *Other Normal Behavioral Learning Methods*

SUMMARY

This document outlines a general framework for learning normal behavior in a time series of data. This is important because any outlier detection needs a model of normal behavior to determine whether a new data point is normal or abnormal.

There are many patterns and distributions that are inherent to data. An outlier detection system must model the data, but a single model does not fit all metrics. It is especially important to consider whether seasonality is present in the data pattern when selecting a model.

Real-time detection of outliers at scale requires online adaptive learning algorithms, and there are various learning models that can be found in data science, statistics and signal processing literature. Anodot has chosen a model that adapts its learning rate to give outliers their due treatment without over-emphasizing their impact on the model going forward.

In Part III of this series, we will look at the processes of identifying and correlating abnormal behavior, which help to distill the list of outliers down to the most significant ones that warrant investigation. Without these important processes, a system could identify too many outliers to investigate in a reasonable amount of time.

In Part III of this series, learn more about correlating abnormal behavior in time series behavior.

Download Part III

For more information, please contact Anodot:

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

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

anodot

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/.