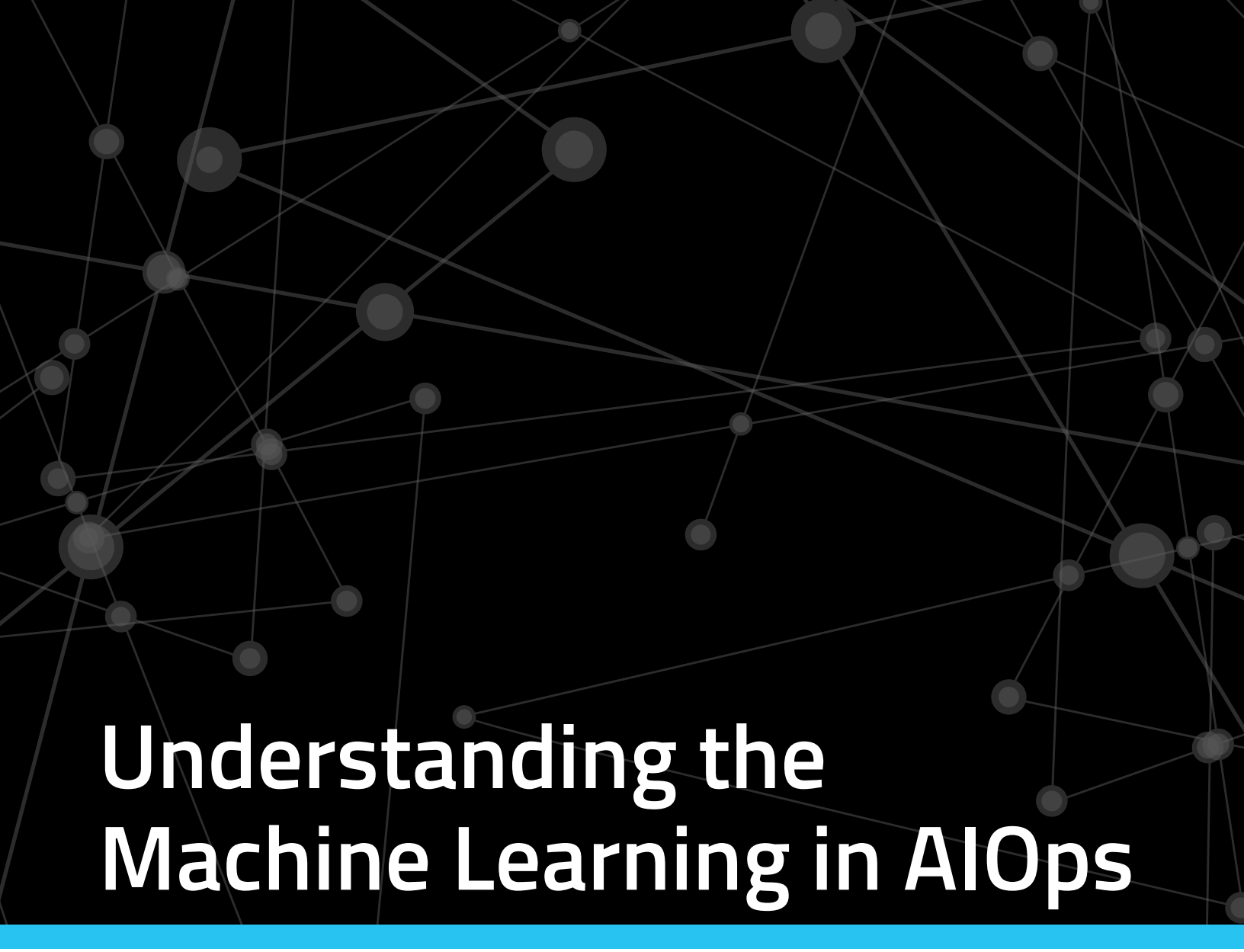

Understanding the Machine Learning in AIOps

Moving beyond the buzzwords: The meaning of AI, machine learning, and deep learning, and understanding their relationship.

by Robert Harper

May 23, 2018

WHITE PAPER

For Information about Moogsoft visit www.moogsoft.com.

Moogsoft®

AIOps is a new category of IT operations tools, created primarily to deal with the challenges associated with operating the next generation of IT infrastructure. Enterprises are taking notice, with Gartner estimating that half of all global enterprises will be actively using AIOps by 2020.

The core appeal of AIOps is its use of algorithms and machine learning to automate tasks and processes that have traditionally required human intervention. Machine learning for IT incident management is available today; however, it does not necessarily exist in every vendor solution that claims AIOps.

Part 1: Beyond the Buzzwords

Two of the biggest buzzwords to cross from the world of computer science and technology startups are "machine learning" and "artificial intelligence." Throw in "deep learning," and we've got the start of a great game of buzzword bingo. These terms are closely linked and are often used interchangeably, but they aren't quite the same thing.

AI covers the broadest range of technologies, machine learning is a set of technologies within AI, and deep learning is a specialization within machine learning.

AI: More Artificial or More Intelligent?

One of the most general definitions of AI, taken from the Merriam-Webster dictionary, is "The capability of a machine to imitate intelligent human behavior." The term "machine" is important, because AI does not have to be restricted to computers.

True artificial intelligence would require multiple technologies from a wide range of subjects, including areas such as speech recognition and natural language processing, computer vision, robotics, sensor technologies, and one of our other buzzwords, "machine learning." In many cases, machine learning is a tool used by these other technologies.

In its very earliest days, AI relied upon prescriptive expert systems to work out what actions to take, an "if this happens, then do that" approach. And while prescriptive expert systems still have a place in some sectors, their influence is much diminished, and that function has largely been replaced by machine learning.

A prime example of modern AI is in virtual and voice assistants such as Siri, Cortana, or Alexa, all of which employ technologies that allow them to "hear" a human voice, understand which sounds correspond to which words and phrases, infer meaning from the series of words that were identified, and formulate an answer. These are all systems that require multiple technologies, including machine learning.

What is Machine Learning, Then?

Machine learning is a field within computer science that has applications under the wider umbrella of AI. A preferred definition is one quoted in Stanford University's excellent machine learning course: "Machine learning is the science of getting computers to act without being explicitly programmed." So rather than programming a system using an "if this, then that" approach, in the world of machine learning, the decisions that the system makes are derived from the data that have been presented to it. It's like a "learn by example" approach, but with more sophistication.

Machine learning is now so common in the world around us that there are countless applications where we may not even realize it plays a part. Automatic mail sorting and speed limit enforcement systems rely upon incredibly accurate implementations of Optical Character Recognition (OCR), which is basically identifying text in images. It's a technology that allows us to identify addresses on envelopes and parcels, or the license plates on a vehicle as it passes through a red light or travels too fast outside a school. OCR would not exist without machine learning, though unfortunately, speeding tickets still would.

Supervised and Unsupervised: Learning by Example

Machine learning falls into two categories, supervised and unsupervised, with differences in their underlying algorithms and their

applications. Unsupervised techniques are generally simpler, and try to find patterns within a set of given observations. Recommender systems rely heavily on these techniques.

In contrast, supervised learning is the "learn by example" approach. Supervised learning systems need to be given examples of what is "good" and what is "bad" — this email is spam, this email isn't, for example.

In the field of OCR, the system would be provided with multiple images of different letters and told which letter that image represents. As a system is provided with more examples, it learns how to distinguish between a spam email and one that isn't, and it learns the different arrangements of pixels that can represent the same letters and numbers. When a new example is presented to the system, specifically an example it hasn't seen before, it can then identify correctly whether or not the email is spam, or which address the letter needs to go to, or the licence plate of the speeding car.

Neural Networks, a Part of Supervised Learning

Within the field of supervised learning there are numerous techniques, one of which is called "neural networks." Neural networks are software systems that try to mimic, often crudely, the way a human brain works. The concept of the neural network has been around for decades but it is only relatively recently that its true power has been realized. A neural network is made up of artificial neurons, with each neuron connected to other neurons. As different training examples are presented to the network (for instance, an image or an email) along with the expected output of the system (the letter in the image, or whether or not the email is spam), the network works out which neurons it needs to activate in order to achieve the desired output.

Here is how it works: The neural network is able to configure itself so that the neurons that get activated when a spam email is presented to it will be different from those triggered by a non-spam email. As a result, the rest of the system can then make a decision on how to handle that email.

One Last thing: Deep Learning

We now get to our final buzzword, "deep learning." It's a very specific and phenomenally exciting field within neural networks. In the same way that machine learning enables artificial intelligence, deep learning enables machine learning.

Think of a deep network as a larger and more complex network, with more complex and sophisticated interactions between the individual nodes. Deep learning employs multiple "layers" with complex interactions within each layer and between layers to identify patterns and solve problems.

Deep learning is at the leading edge of machine learning research, and some of the advances in it have resulted in technologies

such as automatic translation, automatic caption generation for images, automatic text generation, and even creating plays in the style of Shakespeare. And in the same way that machine learning is the main enabler of AI, deep learning, right now, is the main enabler of advances in machine learning.

Part 2: A Deeper Look at Machine Learning

Machine learning systems try to predict a value for something using three things:

1. A way of describing the subject of our prediction
2. A question that we want to answer
3. An algorithm that can take the description and provide an answer to our question

In machine learning terminology, the way that we tell our system about the subject of our question is by using something called a "feature vector." That may sound a bit abstract, but you have no doubt heard the phrase "If it looks like a duck, walks like a duck and sounds like a duck, then it's a duck."

These attributes — how it walks, how it sounds, how it looks — are examples of different features, and the value of each feature will help the machine learning system decide whether the object is or isn't a duck.

Every type of object has its own set of features, and different instances of each type of object will have different values for those features. All ducks may swim and quack, but some ducks are bigger than others and have different colored plumage.

A common example used in machine learning courses is that of predicting a house price. Houses have countless different attributes: How old is it? How many bedrooms does it have? What is the total size? Does it have a garden? What color is the front door? What are the local schools like? Is it well maintained?

By aggregating the values of each of these attributes or features into a list or vector, we have a way of telling the algorithm in the machine learning system about the characteristics of the house, and other houses that may spark our interest.

Features Affect Values

Let's follow through with this housing example. The size of a house likely will have the biggest impact on value, whereas the quality of the local schools may be important for those buyers with families of school age. The color of the front door will have no impact.

So while a subject may have many features, not all features are relevant to a given problem. As a result, it's important that your machine learning system uses features that discriminate between the different states that you're trying to identify, which depends upon the question you are asking of your machine learning system.

The process of selecting the most appropriate features for any given problem is called "feature selection" or "feature engineering."

Input Question, Output Answer

Feature engineering gives us a way of describing our subjects to an algorithm, but what are we actually trying to do? What is the question we are asking of our system?

There are two types of questions that machine learning systems attempt to answer: "Is it a duck?" or "What is it?" and questions like "How big is it?" or "How much is a house worth?"

Questions about size produce answers that have what is called a "continuous distribution," where the value can be anywhere between some practical constraints. This class of problem is called a "regression" problem. Trying to predict the price of a house, a stock, the size of a crop yield, or the capacity of a new data center are all examples of regression problems.

Regression problems are solved using supervised machine learning techniques, because a set of values or labels are required upon which to base the prediction. Let's return to the house price example. We have a set of different houses and a set of features for each house. We know the size of the house and garden, where it's located, and maybe even the color of the front door. We also have a label, knowing how much each house is worth.

Now at some point in all of our experiences, whether it was in a math class or at work, there is a good chance that you will have plotted a graph of some data and then fitted a line to that

data. So, if we have a graph that shows how the value of a house changes with its size, and there is a relationship between those two attributes, then a simple curve-fitting exercise will allow us to estimate the price of another house based only on its size.

For many people, that doesn't feel much like machine learning. If you do an Internet search, there is debate about whether it is or isn't, as it certainly doesn't have a wow factor like, say, automatic translation between languages or automatic captioning of an image. But recall the definition of machine learning from earlier: machine learning is a technique that allows a machine to make a decision on data which it has not seen before. Whether there is a wow factor or not is irrelevant; the techniques, such as linear regression used in curve-fitting, even though they are very simple, form the basis of numerous algorithms in machine learning. These undramatic but useful techniques are a fundamental component of data scientists' and machine learning engineers' toolkits.

In contrast to regression, the answer to a "What is it?" type of question will come from a set of categories rather than a continuous range. In the machine learning world, the these kinds of questions can be handled in a number of ways, depending upon what we want to achieve, and the available data. Questions of this type can be answered by both supervised and unsupervised techniques, but the best approach depends upon the specifics of your question.

In supervised learning, the "What is it?" question is called a "classification" problem, and the system that is used to answer these questions is called a classifier. In the world of unsupervised learning, the "What is it?" question is a "clustering" problem, and the system used to answer these questions is typically called a clustering engine.

Classification vs Clustering

Let's explore that distinction in more detail. Classification aims to define the best category that an object fits into given a predefined set of possible options. Does the image contain a face? Is that animal a duck? These are examples of what is called "Binary Classification," because there are only two categories to choose from: "duck" and "not duck," or "there is a face" and "there is not a face."

There are also classification problems in which there are multiple categories, systems that can handle these questions are called "multi-class" classifiers. For example, "Is that animal a duck, a dog, or a horse?"

Earlier we briefly described OCR, a machine learning technique that tries to read text in images. If we use English text (and for simplicity ignore upper/lower case characters and punctuation marks), then each character will be one of either 26 letters or 10 digits — and so OCR becomes a 36-class classification problem.

The fundamental difference between classifiers and clustering engines is that in clustering, the groups into which something is assigned are unknown in advance and are determined entirely by the patterns in the data. Clustering algorithms take a set of objects and split them into groups, or clusters, where everything in each cluster, is similar to everything else in that cluster but different from items in other clusters.

Let's say we are trying to create a system that can recognize different animals, and I have two systems, a supervised machine learning approach (a classifier) and an unsupervised approach (a clustering engine). We will also assume that the classifier has been well trained and produces accurate results. Now, if my collection contains multiple different animals such as ducks, dogs, and horses, and equal numbers of each, then if I present that collection to my classifier, it would correctly recognize each one and assign it to the appropriate category.

Similarly, if I presented that collection to a clustering engine and I had chosen my features well, I may also expect it to split the collection into three clusters, one for each animal. Importantly, though, the unsupervised system would be unable to label those clusters, as no one has told it what each cluster represents. It just knows that each cluster contains similar things.

But if I change my data so the collection of animals contains only ducks, then the two systems start to behave differently. The supervised system (the classifier) will still be able to say that each animal is a duck. It doesn't care that every example has the same label. It just compares the features of the animal it has been presented with against the features of everything it has previously been told is a duck, and tries to determine if there is a good enough match.

However, the unsupervised system, the clustering engine, which is looking for patterns within the data it has been presented with, is now looking for patterns only within that set of ducks. Many of the features of a duck which will help to distinguish it from other animals, (Does it have feathers? Does it have webbed feet? Does it quack?) will have the same value for every duck and so the clustering engine will ignore those features. The clustering engine tries to find patterns in all the other features it has been given. So if those other features include the animal's color and size, the clustering engine may well split the collection into different colored ducks or ducks of different sizes.

These differences in behavior highlight the strengths and weaknesses of both approaches. Supervised systems need to know up-front what they are looking for and they need to be trained to look for those categories.Those activities take time, but the advantage is that

a duck will always be a duck. Unsupervised techniques will look for hidden patterns in your data — the "unknown unknowns" — and if your data changes, then your patterns will change, too.

So, if you know that you are trying to identify whether an animal is a duck, an unsupervised system probably won't give you the answer you expect. But if you want to find groups of similar ducks, groups of big ducks and little ducks, or white ducks and yellow ducks, then clustering techniques are the best approach.

Clustering, regression, and classification can be used to answer a vast array of questions or solve a multitude of problems. Problems exist everywhere, including in the world of IT Operations.

Part 3: Fishing in a Sea of Data

Before ITOps teams can utilize machine learning and AI to analyze data, they need to define what exactly they're trying to achieve. We have already looked at terminology used in machine learning, and explored machine learning techniques, including clustering, classification, and regression, and the problems that they are best suited to address. Here, we will start to investigate how machine learning can solve many of the problems that are faced everyday in IT Operations, and specifically how ML helps with the process of data ingestion and the reduction of alert fatigue on operators.

What are IT Ops Teams trying to achieve?

It is stating the obvious to say that the ongoing objective of IT Operations teams is to minimize resolutions times, reduce costs, and eliminate customer impacting outages.

Breakages are a fact of life in any system, regardless of the underlying architecture. It is how an operations team deals with those failures, and the quality of the tools at their disposal, that allows them to achieve their goals and meet business needs.

No one sets out to design a system that is hard to manage or prone to failure, but some architectures can increase the demands on an operations team. Often, the system architectures that a business requires — such as cloud computing, micro-services, and continuous deployment — are the ones that can add significant management complexity and increase the number of points of failure in that system, making the tools that are available to an operations team even more important.

The Pain Points

The pain points that show up as long resolution times and customer-facing outages stem from things such as:

- Alert fatigue
- Difficulty in identifying the cause of a problem
- Inefficient communication
- Poor collaboration
- Poor remediation processes

Adopt an approach and toolset that solve these issues, and your team is no longer fighting fires, but has the time to improve. You're using time now to save time in the future, while meeting the commitments made to your customers.

These are the problems that AIOps was designed to address.

Fitting Machine Learning into IT Operations

Machine learning and artificial intelligence are used everywhere, and there is no doubt that these technologies can produce extraordinary solutions. Still, throwing machine learning at a problem isn't ideal.

The variety of techniques is huge, and the right ones need to be adopted for the specific problem. Machine learning has its shortcomings, too. There are circumstances that are better suited for a logic-based algorithm approach. Algorithms need to be coupled with a clean and efficient user experience, and sometimes it's the UX innovations that are key.

So let's dig into some of the pain points around event ingestion, and see where machine learning techniques can provide some or all of the solution.

Alert Fatigue

Examples of alert fatigue exist everywhere. It is exemplified by those things that happen around us everyday that we ignore because they are so commonplace. When was the last time that you really noticed a fire drill?

Alert fatigue comes about through the avalanche of data that modern systems generate. In even a modest-sized enterprise, an

IT infrastructure can generate millions of events a day. Add raw time-series data to that mass of data, and the volume can increase significantly. Buried somewhere in all of those application heartbeats and "authentication failed' messages will be the handful of alarms that pinpoint a customer-impacting failure and its root cause.

Minimizing alert fatigue isn't simply about reducing the volume of events that need to be processed, though — that's easy, and the wrong approach. Filtering an event stream to ignore certain sources and thresholding to only process critical alarms are examples of techniques that will reduce the volume of data, but at the same time discard the possible cause. They are also techniques that require a human to maintain.

One of the most enduring techniques for volume reduction is event deduplication, the act of grouping repeating events to a single alert. On its own, this approach can no longer produce the required impact. The volume of data, even after deduplication, is still huge. But to its advantage, it doesn't remove data from the system. Your team is presented with a more manageable amount of data.

The real solution to alert fatigue needs a different approach, and it's an approach made up of several stages.

Yes, it is about discarding those alerts that are meaningless, but it is also about processing what's left in a way that allows your ITOps team

> Machine learning is the science of getting computers to act without being explicitly programmed.

to get to the cause of the problem quickly, and displaying the information that gets them there in an easily consumable way.

It's about knowing what normal is, and what it isn't, and this is where machine learning and data science techniques are needed — using past data to provide a benchmark of what is normal for your infrastructure.

Normal Is Different for Everyone

For the initial part of the process, AIOps uses a concept of entropy as a way of achieving noise reduction. In this context, entropy means that we are looking for events with lower probability, which means that they carry more information than higher-probability events. This value encapsulates the what, when, and where of an event: What is the event? When does it happen? And where in the infrastructure is it coming from? And we use these questions to build a picture of what is normal, based on the analysis of past events. So we can then evaluate whether events entering the system require an action.

First, let's take a look at time series data.

Time Series Data: Only Forward the Anomalies, Please

Time series data is the periodic reporting of status data from a server, or application reported by a monitoring solution. If all is well, your server may report that it has "25% CPU utilization" and "45% free disk capacity," and it will keep doing that every five minutes, until you tell it to stop. Another 288 events per metric per server per day. Almost every one of which will be reporting, "I'm fine, there's nothing to see here!"

An ops team doesn't need to see this sort of data; they only need to know when something has gone wrong, when something is out of the ordinary, and this is where we step into the world of outlier detection and anomaly detection.

Outlier detection and anomaly detection are terms that are sometimes used interchangeably. At Moogsoft we favor the following definitions.

An outlier is a value of a metric that is different from other values of that same metric when you would expect them all to be similar. For example, the CPU load on the servers behind a load balancer may be expected to lie within a very specific range. Let's say CPU utilization fluctuates between 40% and 50% but at one specific time of day there is a single server running at 70% CPU. That specific measurement may be classified as an outlier: it is different to all the other servers' CPU usage.

The presence of an outlier may also indicate an error in your monitoring, a value that is so far from expected that perhaps it's not the systems being monitored, but the method itself. However, just because a value is an outlier in one context, it doesn't necessarily mean that it is anomalous. An anomaly is where a

measurement doesn't follow historical trends. So our hot-running server, whilst an outlier in the context of similar servers at a specific point in time, may always run at 70% because, for the sake of argument, it is the master server in an HA group. If its utilization spiked to say 95%, that would be an anomaly because it is not following its historical behavior.

But what has this got to do with machine learning and alert fatigue? Simply ignoring time series data is not an option, despite the volume of data, so instead of forwarding every piece of time series data to your operators, you should only forward your anomalies. But how do we do that?

A simplistic approach is to use a simple threshold: If CPU is greater than 80% it's an anomaly. But what is right for one set of servers won't be for another set. Maybe CPU spikes are expected at certain times of day but running at 95% in the middle of the night isn't. Accounting for these scenarios with manually created rules quickly becomes too complex for these thresholding techniques.

The more complex your criteria become, the more complex the underlying algorithms tend to be. Some very effective (and non-machine learning) algorithms do exist to capture these use cases, techniques such as dynamic thresholding and "seasonality and trend decomposition." But to capture the full array of scenarios, you need to add machine learning techniques into your algorithmic toolbox.

Unsupervised clustering techniques such as k-means, or nearest-neighbor clustering are often used for outlier detection. But when business needs require us to identify whether a metric is following historical behaviors and trends, we soon get into the state-of-the-art deep-learning-based solutions using recurrent neural networks — solutions involving techniques such as Hierarchical Temporal Memory or Long Short-Term Memory.

Enrichment

At this point in the life cycle of an alert we have de-duplicated our event stream, removed the noise, and are reporting only anomalies from our time-series monitoring solution. But the impact of machine learning on data ingestion doesn't end there.

The more AIOps knows about an alert, the higher the accuracy with which that alert can be processed. But the richness of the data in an alert is highly dependent upon its source. Events forwarded from an APM platform will contain highly relevant data about an application and the services that it provides. The SNMP Traps generated on your network hardware comply with strict protocols, and generally contain well structured, explicitly labeled data.

Contrast that with the events from a data aggregator or a raw application log file, and the situation is very different. Your systems need to be able to extract the relevant parts of the message to create a coherent alert.

As always there is a solution that relies upon manually created and maintained rules: regular expressions to match tokens such as IP addresses and dates and time, keyword matching such as "LinkDown" and "LinkUp" to match fail/clear pairs, or "login fail" and "invalid password" to indicate the alert is related to a security issue. While this approach has utility and can be highly effective, it is now outdated. The complexity and quantity of the different look-ups soon becomes overwhelming: the maintenance issue is obvious, and it is surprisingly resource intensive when applied in real time against thousands of alerts per second.

It will come as no surprise by now that machine learning techniques can help us out. Named Entity Recognition techniques borrowed from the field of Natural Language Processing provide more efficient ways of extracting different types of tokens from the event text. Supervised learning techniques such as classification can help us identify the class of an alert: is it related to "audit" or "security," or is it from an application or a piece of network hardware, even if it is a state-transition event as part of a fail/clear pair?

So far we've looked at some of the ways machine learning can be used to help ingest data and reduce the volume of data presented to your operations team whilst retaining the important stuff. But that's not where we end our quest to reduce alert fatigue.

Part 4: Applying Machine Learning Algorithms Within AIOps

As we continue the journey of an alert through the management process, we begin to see how machine learning can be used to reduce alert fatigue. Previously, we concentrated specifically on reducing alert fatigue during event ingestion. Prior to that we looked at the background to machine learning, examining some of the terminology and buzzwords, along with an overview of the different types of problems that machine learning can be applied to, and their solution techniques — techniques such as clustering, supervised and unsupervised learning, and problem types such as classification and regression.

Alert Fatigue

During the event ingestion process, deduplication and unsupervised machine learning techniques such as entropy are well known approaches that AIOps exploits to reduce event noise. Add in anomaly and outlier detection for processing time-series data, and we have an effective way of reducing thousands of events to little more than a handful of alerts. But reducing alert fatigue doesn't stop there.

The whole concept of situational awareness, part of the founding principles behind AIOps, brings about the next layer of relief from the pain of alert fatigue. Add a sprinkling of Probable Root Cause into the mix, and we start to attack

our other pain points too, specifically: identifying the cause of an incident, and poor remediation processes.

Once an alert has been ingested, assigned an entropy, deduplicated, and enriched with external data, we have everything needed to inform the creation of actionable incidents, or "situations," in the form of operationally significant groups of alerts. Sometimes, the required information may be incomplete. Sometimes the enrichment data may have been retrieved from multiple sources leading to conflicts in what should be canonical data — not ideal, but the sort of real-world problem that needs to be handled by IT management systems

The process of grouping alerts is generally referred to as "correlation." The dictionary definition of correlation is "a mutual relationship or connection between two or more things which tend to occur together in a way not expected on the basis of chance alone."

Correlation

In the world of IT Operations, correlation is often interpreted as the ability to make deep connections between seemingly disparate data, and while that is certainly part of the challenge, it isn't the entire challenge. What constitutes a relationship or a connection in the first place? Well, it's anything the managing enterprise wants it to be, and what is meaningful in one organization may mean nothing in another.

> A prime example of modern AI is in virtual and voice assistants such as Siri, Cortana, or Alexa, all of which employ technologies that allow them to "hear" a human voice, understand which sounds correspond to which words and phrases, infer meaning from the series of words that were identified, and formulate an answer.

There are certainly failure scenarios in which the correlation need is universal across all organizations — correlating link-up and link-down pairs to identify a flapping interface, for example. But there are many use cases in which the approach to correlation chosen by one enterprise may not align with how another needs to manage their infrastructure.

Consider an ISP that chooses to manage their street-level access infrastructure based on network topology. What about the retail bank that wants to manage its branches and Automated Teller Machine network based on street address? Or the Web Service Provider that finds the most efficient way to manage their infrastructure is to group alerts in a way that mirrors the remedial steps that its operators need to take, even though the failures may be on disparate parts of its infrastructure and share no form of topological or geographical proximity?

The core events across all of these use cases will be very similar, maybe even identical. But there isn't, yet, a one-size-fits-all algorithm that can understand that these otherwise identical alerts need to be handled in a certain way in one organization, and in a completely different way in another.

Consequently, and in order to address the wide variety of operational methodologies across different enterprises, AIOps use multiple different criteria to correlate alerts, criteria such as event arrival times, network topological-

proximity, and contextual similarity between combinations of alert attributes.

In AIOps we call the processes responsible for finding the connections between alerts and for creating situations "sigalizers." A single instance of AIOps can run multiple types of sigalizers and multiple instances of each sigalizer concurrently.

Where necessary, events can be routed along different processing paths. This allows different sigalizers to process different events depending upon each event's characteristics. For example, "availability" events from the core of a network may be processed independently of other events by the time-based or topology-based sigalizers, while "application" and "security" events may need processing together by a sigalizer based on contextual similarity.

What About the Machine Learning?

All of our sigalizers use machine learning in some form, whether via an unsupervised clustering technique alongside a fuzzy matching algorithm, or algorithms that learn from user interaction.

"Tempus," "Nexus," and "Speedbird" are all examples of sigalizers that rely exclusively upon unsupervised machine learning.

Tempus

Tempus correlates alerts based on time, grouping alerts with similar event arrival patterns. At its core are community-detection algorithms borrowed from the world of graph theory. Tempus requires only a single piece of data for its operation: the event arrival time. It takes no account of, and has no need for, any other event attributes. The sweet spot for Tempus is availability-related failure scenarios in which all the different failure events are likely to be coincident in time.

Nexus

For topology-based use cases, the sigalizer of choice is Nexus, and so perhaps unsurprisingly, it requires access to a topology database. Nexus clusters alerts based on where they are in the network, and can only cluster events from entities within that topology.

Speedbird

Speedbird uses contextual similarity as its correlation criteria, grouping events based on the similarity of one or more event attributes such as description, or severity, or any other data enriched into it.

Both Speedbird and Nexus utilize a proprietary, unsupervised, clustering engine based upon the wellknown 'k-means' algorithm. One of the perennial challenges with k-means clustering is the need to supply a value for 'k', the number of clusters the algorithm looks for. AIOps uses a patented way of determining that information, so it can automatically adapt to the inbound event data.

Feedback

At the opposite end of the machine learning spectrum is our "Feedback" sigalizer. While Tempus, Nexus, and Speedbird use unsupervised learning, Feedback uses supervised learning. It looks at the actions performed by an operator during the incident resolution process: Did the operator need to remove outlier alerts? Were there some related alerts that needed to be added to the situation? Was the situation given a 5-star rating, or marked as being of low-quality? These are the training triggers that AIOps can take and use to inform the creation of future situations.

Pure unsupervised and supervised techniques provide great utility, but there are circumstances in which they may not be the optimal choice. As we've seen, the power of unsupervised techniques is to find hidden structure within a dataset. But what if that hidden structure doesn't align with your management strategy all of the time? Tempus and Speedbird are highly capable at generating seed situations that Feedback can refine, but as we learned in earlier posts, these are statistical methods that rely upon the quality of the training data they are provided with. "Garbage In, Garbage Out" as the saying goes.

ACE

This is where ACE, our Algorithmic Clustering Engine, comes in. One perspective on ACE is that it is, as a hybrid, method somewhere between supervised and unsupervised learning.

The thing is, by the letter of the definitions provided in data science textbooks, ACE isn't really supervised learning — we don't provide the system with labelled data — but at the same time it is so much more than a pure unsupervised technique. We can give ACE hints on how we want it to behave, so "guided learning" or "guided intelligence" are perhaps better descriptions of its capabilities.

ACE uses unsupervised clustering algorithms bespoke to AIOps. ACE is a novel streaming-based clustering algorithm, but the criteria used to assign an event to a cluster or situation are controlled by simple, similarity-based directives that use fuzzy matching techniques to find the most appropriate group of events to match it with. Something that it is impossible to do with a standard unsupervised clustering algorithm.

And One More Thing...

We have tracked our alerts through the management process, they have been ingested, de-duplicated, and entropy-based noise reduction has allowed the meaningless alerts to be discarded. And alerts meeting the requisite correlation criteria have been grouped into situations. But there's one more application of AI to consider in the fight against alert fatigue — a feature called "Probable Root Cause," or PRC.

A situation may contain only a handful of alerts, it may contain many tens or even hundreds. And of course, while AIOps has removed the

meaningless events and collapsed potentially thousands of events into a single actionable incident, operators still need to know which alert to fix first, and that's where Probable Root Cause comes in. PRC is the call to action, the way to inform operators of the alert they need to focus their attention on.

Probable Root Cause is an application of supervised machine learning, specifically a classification problem solved with a neural network.

By providing feedback during the incident resolution process, AIOps learns those alerts that are root causes, and the circumstances in which they happen. It learns the circumstances under which a humble ping fail is the root cause, but also, when different alerts have been triggered, that the same ping fail is now only a symptom — a symptom of a power supply failure, for example.

By utilizing supervised learning, PRC isn't dependent upon the static behavioral models that legacy root cause analysis systems rely upon — behavioral models that are slow to adapt to rapidly changing virtualized infrastructures, models that can't handle common infrastructure implementations such as overlay networks. PRC is not constrained to a predetermined model of an infrastructure or the device types therein, and importantly, it can learn the way an organization chooses to manage its infrastructure.

Significantly, PRC also has a capability that other approaches to root cause analysis simply cannot hope to emulate. It can tap into the tribal knowledge buried in the minds of your operators, the knowledge built up over years of managing your infrastructure, knowledge than remains in your organization even when your key personnel move onto new pastures. Not only does PRC, AIOps' machine-learning approach to root cause analysis, contribute to the fight against alert fatigue, but it also informs the remediation process.

The power of machine learning is unquestionable, but it isn't a silver bullet — different algorithms have different sweet-spots. That applies across the board, whether you are using machine learning or an algorithm based on pure logic. And that is where the art lies, and what AIOps achieves — knowing the optimal approach to solving a problem so you don't need to.

Author Bio

Rob Harper is Chief Scientist at Moogsoft. Previously, Rob was founder and CTO of Jabbit and has held development and architect positions at RiverSoft and Njini.

Moogsoft

Moogsoft builds AIOps solutions that help IT teams work faster and smarter to provide better customer experiences. With patented algorithms analysing billions of events daily across the world's most complex IT environments, Moogsoft's unique technology helps enterprise companies such as SAP® SuccessFactors®, Intuit®, GoDaddy™, and HCL Technologies avoid outages and increase their operational agility. To learn more, visit www.moogsoft.com.