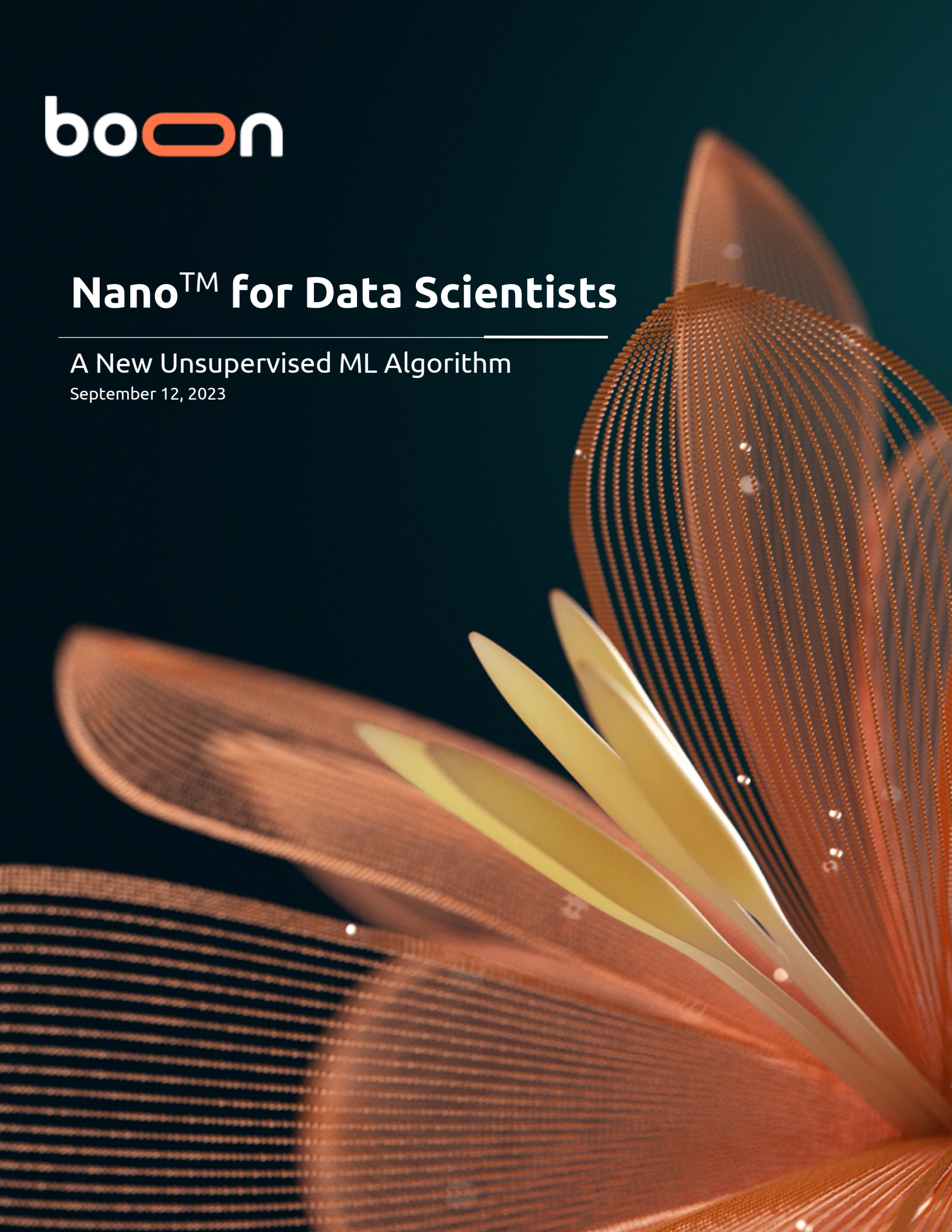

Nano™ for Data Scientists

A New Unsupervised ML Algorithm

September 12, 2023

Summary

The Nano

- An unsupervised machine learning (UML) technology.
- Performs the core UML activities of *n*-dimensional segmentation and anomaly detection.
- Produces accurate clustering results comparable to or better than K-means.
- Incredibly fast and computationally lightweight, performing segmentation 1000x faster than K-means, head-to-head, on the same hardware and data sets.
- Supports thousands of clusters in a single model.
- Not based on neural networks.
- Does not use GPUs.
- Self-configures its hyperparameters in seconds.
- Trains and runs entirely at the edge on Intel SGX-enabled processors.
- Proprietary algorithm, not based on K-means, DBSCAN, graphs, forests, self-organizing maps, or autoencoders.

Table of Contents

Introduction

This document is intended for data scientists who wish to place the Nano within the context of other common approaches to data science including both traditional and machine learning-based methods. The Nano is an extremely high-speed, high-accuracy, computationally lightweight, *n*-dimensional segmentation technology that has extensions ideally suited for anomaly detection. It is not based on neural networks. It supports self-configuration and can be trained and deployed entirely at the edge on commodity CPUs without any GPU or cloud support. The core algorithm of the Nano is proprietary, but what it does can be compared to other well-known techniques, and this document will present several applications along with speed and accuracy benchmarks to help data scientists understand how it may be used as a tool for creating solutions not possible with other approaches.

Segmentation

Simplistic Segmentation versus Real-World Segmentation

A foundational task of data science is the segmentation of semi-structured, *n*-dimensional data into clusters of similar vectors. Segmentation has the benefit of

- Reducing what may be millions or billions of vectors of data into a small set of representatives
- Measuring the relative frequency of occurrence of vectors in the data set
- Identifying anomalies in a data set, either based on frequency of occurrence or distance from other clusters of vectors

Textbooks and internet sources use simplistic plots like those in Figure 1 to illustrate *n*-dimensional segmentation.

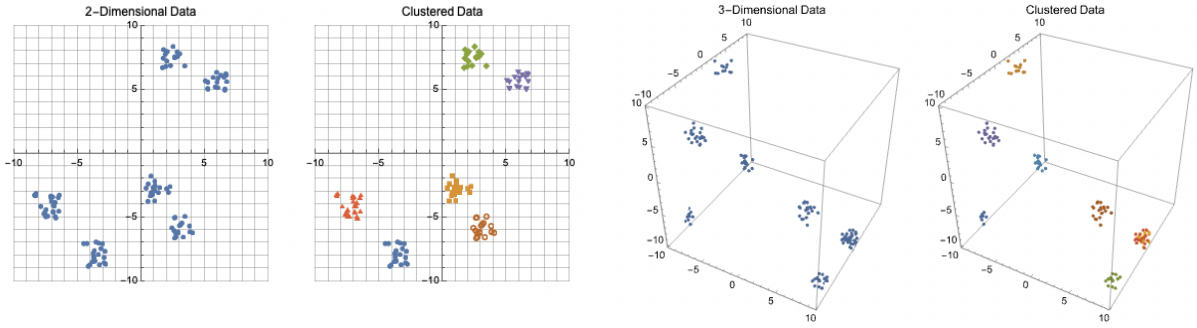

Figure 1: Simplistic examples of segmentation: very few data points, very small clusters, very low cluster count, very low dimensionality, well-separated clusters.

While useful for illustrating the goal of segmentation, plots like those in Figure 1 misrepresent the difficulty of segmentation in important, real-world domains such as the following (Figure 2).

Domain	Source	Sample Count	Input Vector Dimensionality	Cluster Count	Cluster Separation
Medical Biomarkers	Multi-electrode EEG and EKG	Thousands per second per electrode	12 dimensions	Hundreds	Poor (e.g., at seizure onset)
Medical Biomarkers	Volumetric imaging (MRI, CT, Ultrasound)	Tens of millions per scan	Up to 1024 dimensions	Hundreds to thousands	Poor (e.g., at tissue gradients)
Manufacturing	Asset sensor telemetry (vibration, temperature, etc.)	Up to thousands per second per sensor	5 to 100 dimensions	Hundreds	Fair depending on the asset operational modes
Network Monitoring and Security	Ethernet packets or network flow measurements	Millions per second	10 to 50	Hundreds	Fair
Image Processing	*N* x *M* images either B&W or Color	Millions per image	Up to thousands depending on segmentation technique	Thousands for video data	Poor due to natural lighting gradients

Figure 2: Some common, real-world data sources for which segmentation is especially challenging.

In practice, any real-world, semi-structured data that a data scientist might encounter (and the examples are endless), will have characteristics more like Figure 2 than Figure 1.

Here are two more complications for the simplistic conception of segmentation illustrated by the domains in Figure 2

- Many data sources are streaming rather than existing as a batch.
- Many data sources are idiosyncratic rather than universally representative.

Streaming data is collected continuously over many hours (as with biomarkers) or even 24/7 (as with manufacturing data or network monitoring data). As such it creates temporal relationships between vectors in addition to the *n*-dimensional relationships they have. This time factor is not simply another feature in the dimensional stack of the vectors in the data set. Time is what transforms segmentation from pattern matching into multivariate time series analysis. This streaming characteristic can create massive data stores in a short amount of time. Many manufacturing sites are collecting petabytes of data each year but lack the data scientists to analyze it and derive value from it.

Another reality is that data is mostly idiosyncratic based on the individual data source and based on time. Here are some examples.

- Manufacturing data is idiosyncratic: A company may have thousands of gas turbine generators in its operational fleet across many sites, each one worth millions of dollars. Each generator will have its own unique maintenance history extending over decades of near continuous operation. Many rotational assets are used differently in different stages of the overall operation and at different sites. This creates, for each asset, unique relationships between the hundreds of sensors measuring the asset's operational health. Data science models derived from a single complex asset do not generally transfer to the other assets, even assets of the same make and model.

- Biological data is often idiosyncratic: EEG and EKG data is individualized to the person. What appears normal in one person could indicate a pathology in someone else or vice versa. Furthermore, normal EEG data for each person changes during their life, especially during the continuous transitions from childhood through adolescence to adulthood and aging. It is also highly dependent on the ongoing activity of the person.
- Network packet traffic is idiosyncratic: Tapping and monitoring network packet traffic at a chosen point within one LAN of an organization will create a very different segmentation model from another LAN within the same organization. The segmentation will also be affected by time of day (working hours versus off hours) and the day of the week (Sunday versus Monday) and the month of the year (August versus September).

The examples above show that it is uncommon to find a discrete, pre-recorded batch of data that can be used in a transferable and deployable data science effort. The Nano is uniquely suited among the dominant other segmentation tools and approaches to address these complex data source challenges.

Coping with Complexity: The Nano versus other Common Algos and Approaches

The Nano can be applied to all the data sources shown in the examples in Figure 2. Among the other common segmentation approaches and algorithms, the Nano is uniquely suited for

- High-volume data with billions of vectors
- High-dimensional data with up to hundreds of features describing the data source
- High-velocity streaming data flowing into the Nano as a multivariate time series or as an image stream of video data
- Relationally complex data sets requiring hundreds or thousands of distinct clusters for an accurate model
- Idiosyncratic data where an individualized model must be built for each data corpus or data stream

The next section will discuss next some of the common approaches to dealing with data complexity and how the Nano compares to them.

Approaches to High-Volume Data

Random sampling (or fixed-interval downsampling) is one of the most common reduction techniques for building a model based on a data corpus having millions, billions, or even trillions of data points. This reduction approach becomes especially important for a data scientist using an iterative algorithm that creates the segmentation model incrementally during *multiple passes* through the data corpus. This would include for example, K-means, self-organizing maps, and agglomerative methods like DBSCAN. Without reduction, a data set with billions or trillions of vectors will bring these algorithms to a near stand-still, even on multi-core compute resources.

In contrast, the Nano is a *single*-pass training algorithm and is ideal for massive data sets. As a rule-of-thumb, the average model training time of the Nano is 1 microsecond per vector on a single CPU core. For example, on the KDD Cup 1999 Cyber intrusion data set having 4.9 million vectors of network flow data, the Nano trains its model in about 5 seconds on a single core of an Intel NUC. In contrast, for the same cluster count, K-means will take more than an hour to build its segmentation model, which is why multicore and GPU-accelerated approaches are used in practice with K-means. The models created by the Nano and K-means will be very similar with some important differences that will be demonstrated in the benchmarks below. Agglomerative and neural network segmentation methods are orders of magnitudes slower than K-means and cannot really be considered deployable for real-world applications on compute platforms with less than hundreds or thousands of cores (depending on the model complexity).

Approaches to High-Dimensional Data

High dimensionality is problematic for *n*-dimensional segmentation algorithms such as K-means, density-based methods, and self-organizing neural networks, since it means that the time to compute distance between vectors grows linearly as the number of dimensions increases. A common approach to coping with high dimensionality is *dimensionality reduction*. For instance, features of the vector that have dependency on other features can be removed thereby reducing the length (i.e., the dimension) of the vector over which *n*-dimensional distance must be calculated. For example, using a covariance matrix, a data scientist will identify dimensional dependencies and strategically eliminate some features. Principal Component Analysis (PCA) uses the covariance matrix to compute a vector space transformation where the transformed features are sorted in order of the relative contribution so that one might take the first 3 features of 50 in the transformed vector space as they might account for 95% of the covariance between vectors. This introduces the need during deployment to calculate this vector transformation for each raw vector prior to applying whatever segmentation algorithm one may choose.

The primary risk of dimensionality reduction is that some clusters may only be separable in the full *n*-dimensional space and not in the reduced-dimension space. This is almost guaranteed to happen for high-dimensional data with enough complexity to require hundreds of clusters. The dimensionality reduction agglomerates vectors in the reduced-dimension model that would have been separated in the full *n*-dimensional space. This effect is especially problematic if the reduced-dimension model is being used for anomaly detection since anomalies often show deviation from the training data only in the eliminated features (or hyperplanes). As anomalies in training sets are often absent or only very sparsely represented, they will not be accounted for in the reduced-dimension vector space and anomalous vectors processed in a deployment will likely be assigned to the remaining dominant clusters. Autoencoders are a neural network-based dimensionality reduction technique specialized for anomaly detection (rather than segmentation) and will be discussed later.

The Nano is very tolerant of high-dimensional data, showing only very slight linear growth in inference time as dimensionality increases. This has several benefits. Data scientists using the Nano:

- Do not need to spend time applying dimensionality reduction techniques to their source data. No covariance matrices, no PCA, no SVD, etc.
- Gain the full descriptive power of the segmentation model without wondering if eliminating features will reduce accuracy
- Can also use the resulting segmentation model for anomaly detection

Generally, data scientists using the Nano should include all the features available provided those features are (by common sense) relevant to the segmentation. Obviously, features showing random behavior should be eliminated, although the Nano is tolerant of some randomness among many other non-random features.

Feature significance is an optional output of the Nano that indicates for each cluster the relative significance of each feature in forming that cluster (Figure 3). Since all features are included in the model this can be useful for data scientists who want to understand something more about the structure of the model.

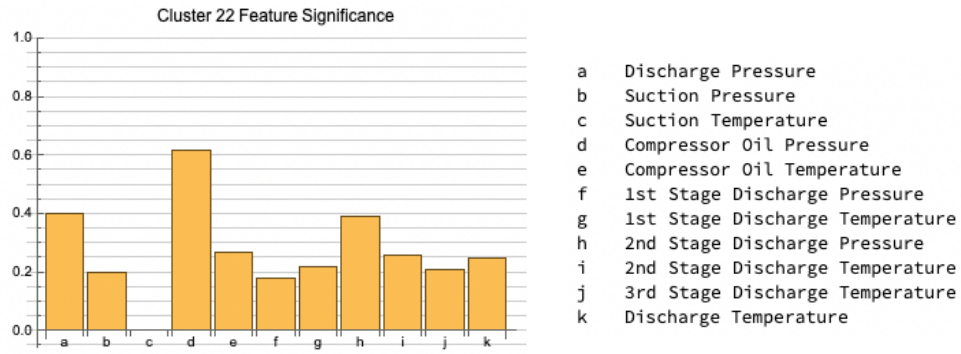

Figure 3: Nano's feature significance vector for cluster 22 of a 78-cluster model trained from 23,038 sensor fusion vectors of an industrial compressor. This shows that "Compressor Oil Pressure" was most significant in differentiating this cluster from the rest of the model and that "Discharge Pressure" and "2nd Stage Discharge Pressure" were of about equal secondary importance. The feature "Suction Pressure" (c) shows a feature significance of 0.0 (which is unusual) and the customer found it is 0.0 for all 78 clusters. This led the customer to investigate the "Suction Pressure" sensor and they found that it was always sending the value of 65,535, indicating that the sensor was malfunctioning. See also Figure 4.

Approaches to High-Velocity Streaming Data

Figure 2 gives examples of continuously streaming data sources such as network packet traffic, biomarker signals such as EEG and EKG, and manufacturing asset sensor telemetry. Figure 4 shows one thousand values from each of the 11 sensors of the asset in Figure 3. These were sampled at the (very slow) rate of one per minute, but many streaming data sampling rates run into the millions of samples per second, for example, audio data and vibration data.

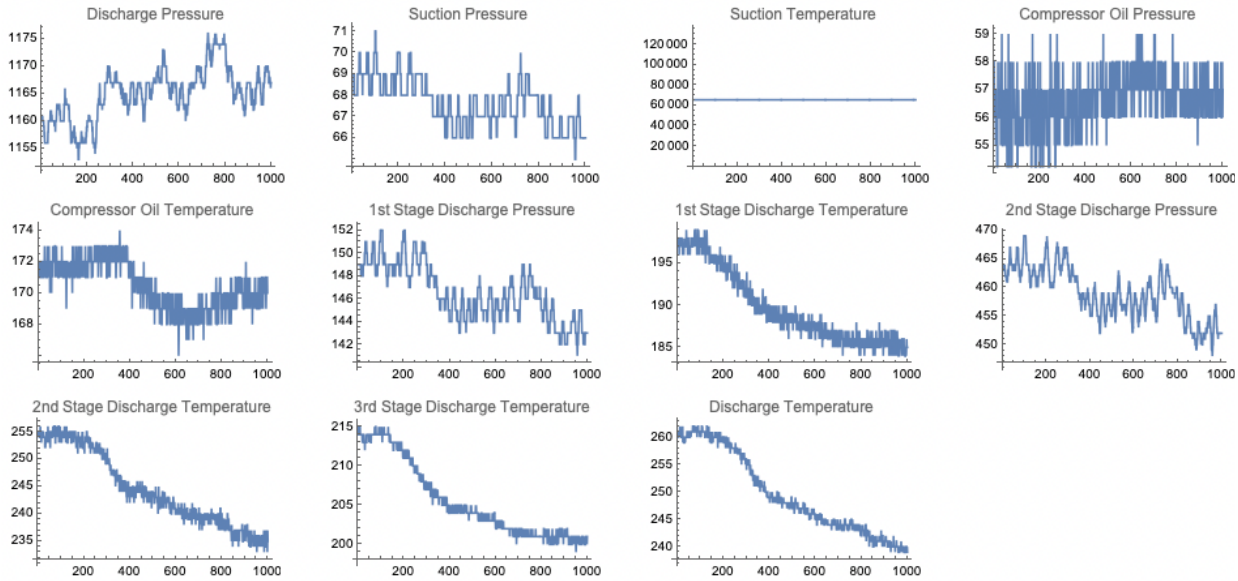

Figure 4: One thousand temporally correlated sensor telemetry readings from a high-value industrial compressor.

The difficulty of segmentation of streaming data is mostly in creating a performant pipeline that can assign cluster IDs to incoming vectors at a rate equal to the data flow. Here is another situation where the speed of the Nano is crucial. For K-means on a single core, a common inferencing throughput rate is one thousand samples per second. For the Nano on a single core, it is one million samples per second.

Nano Theory of Operation

Definition of a Pattern

A pattern is an *n*-dimensional vector of numeric values. The domains from which these vectors may be drawn is far too large to enumerate here, but include all of the following where the Nano has been applied

- multivariate time series including manufacturing sensor telemetry, medical biomarkers, flight recorder data
- vectorized 2D image data including hi-def video for quality control and single image streams
- vectorized 3D medical imaging scans
- vectorized network packet data
- power spectra from acoustic signals
- edge-triggered impulsive data

Training

The Nano clusters its input data by assigning to each pattern an integer as its cluster ID. Patterns assigned the same cluster ID are similar in the sense of having a small L1-distance from each other. The similarity required for patterns assigned to the same cluster is determined by the percent variation hyperparameter and the configured feature ranges. Sometimes a pattern is processed by the Nano that is not sufficiently near any of the existing clusters. In this case, one of two actions is taken. If learning mode is on and if the maximum allowed cluster count has not been reached, the pattern becomes the first member of a new cluster and the number of clusters in the model increases by one. If learning mode is off or the maximum number of clusters has been reached, the pattern is assigned the special cluster ID 0 or a negative ID if root cause analysis is enabled. There is no assumption that can be made about the similarity of patterns assigned a negative or zero cluster ID, but they are all known to be significantly different from the clusters with positive IDs in the model. This dynamic learning process of assigning patterns to existing clusters or, if needed, creating new clusters is called training a model.

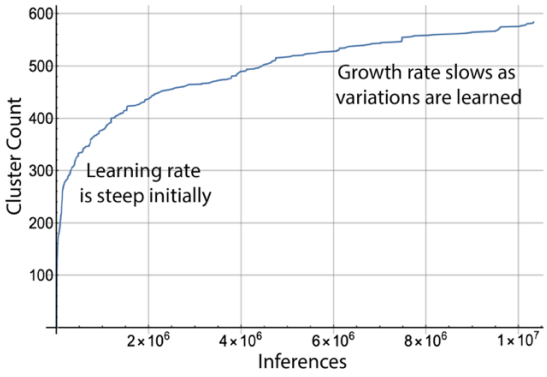

Figure 5: The number of clusters grows quickly as the first patterns from the input data are processed. The slope of the learning curve levels off as the model matures and as nearly all incoming patterns already have a cluster to which they can be assigned.

This process of assigning to existing clusters or adding new clusters to the model creates a learning curve such as shown in Figure 5. The learning curve is a key indicator of the relative maturity of the model relative to the data corpus being used to train it. As the learning curve levels off, the Nano model already has clusters for nearly every new vector passed into it.

Configuration

Pattern Feature Ranges: Each feature in a pattern has a range that represents the expected range of values that feature will take on within the input data. This range need not represent the entire range of the input data. For example, if the range of a feature is set to -5 to 5, then values greater than 5 will be treated as if they were 5 and values less than -5 will be treated as -5. This truncation (or outlier filtering) can be useful in some data sets.

Percent Variation: The percent variation setting ranges from 1% to 20% and determines the degree of similarity the Nano will require between patterns assigned to the same cluster. Reducing the percent variation settings creates more clusters, and each cluster will have more similarity between the patterns assigned to it. Increasing the percent variation produces coarser clustering with fewer clusters.

Streaming Window Size: This indicates the number of successive samples to use in creating overlapping patterns. For instance, it is typical in single-sensor streaming mode applications to have only one feature and a streaming window size greater than 1. If there is one feature, and the streaming window size is 25, then each pattern clustered by the Nano is composed of the most recent 25 values from the sensor. In this case, each pattern overlaps 24 samples with its predecessor. In sensor fusion applications, there are multiple features fused together in each sample pushed into the Nano. This is also called multivariate time series. In this case, it is typical to set the streaming windows size to 1 although values greater than 1 may be useful in some applications.

Autotuning

As it pre-scans the training set, The Nano automatically calculates the optimal percent variation and the range for each feature, a process called autotuning. The range for each feature can be autotuned either individually or a single range can be autotuned to apply to all features.

One of the most difficult parameters to configure in unsupervised machine learning is the desired number of clusters needed to produce the best results (as with K-means) or (in the case of the Nano) the desired percent variation to use. This is because one would not generally know *a priori* the underlying proximity structure of the input vectors to be segmented. The same principle applies to choosing the number of nodes in the output layer of a self-organizing neural network.

To address this, the Nano can automatically tune its percent variation to create a balanced combination of coherence within clusters and separation between clusters. In nearly all cases, autotuning produces the best value for the percent variation setting. However, if more granularity is desired you can lower the percent variation manually. Similarly, if the autotuned percent variation is creating too much granularity (and too many clusters) then you can choose to manually increase the percent variation above the autotuned value.

Nano Machine Learning Outputs

When a single pattern is assigned a cluster ID, this is called an inference. Besides its cluster ID, several other useful analytic outputs are generated. Here is a description of each of the ML-based outputs generated by the Nano in response to each input pattern.

Cluster ID

The Nano assigns a cluster ID to each input vector as they are processed. Following the configuration of the Nano, the first pattern is always assigned a cluster ID of 1. The next pattern, if it is within the defined percent variation of cluster 1, is also assigned to cluster 1. Otherwise, it is assigned to a new cluster 2. Continuing this way all patterns are assigned cluster IDs in such a way that each pattern in each cluster is within the desired percent

variation of that cluster's template. In some circumstances the cluster ID 0 may be assigned to a pattern. This happens, for example, if learning has been turned off or if the maximum cluster count has been reached. It should be noted that cluster IDs are assigned serially so having similar cluster IDs (for instance, 17 and 18) says nothing about the similarity of those clusters. However, several other measurements described below can be used to understand the relation of the pattern to the model.

In some conditions, a negative cluster ID may be assigned to a pattern. A negative ID indicates that the pattern is not part of the learned model, but the Nano can still give information about the pattern as it relates to the learned model. This can be especially useful for anomaly detection where a pattern falls outside the model. In this case, Root Cause Analysis using the negative cluster ID will provide additional information about the reason the pattern is outside the model. (Figure 3, Figure 6.)

Anomaly Indexes

The Nano can measure anomalies along two orthogonal axes: probability and distance. Together these two axes capture the full scope of what may be considered as an "anomaly" within a set of input patterns. These two distinct concepts of an anomaly are described in more detail in the section below on Anomaly Detection. Each pattern processed by the Nano is assigned a Probability Anomaly Index and a Distance Anomaly Index.

Root Cause Analysis

The Nano associates each input pattern with an associated root cause vector. This vector has the same number of values as the length of the patterns of the model. Each value in the vector is a measurement of that feature's significance in the cluster to which it was assigned. Significance values range from 0 to 1 where relatively high values indicate features that were more influential in the assignment of the pattern to the cluster. Values close to 0 lack statistical significance and no conclusion can be drawn from them. The root cause vector is especially important for anomaly detection where the RCA vector can be used to determine which features are implicated in a detected anomaly (Figure 6).

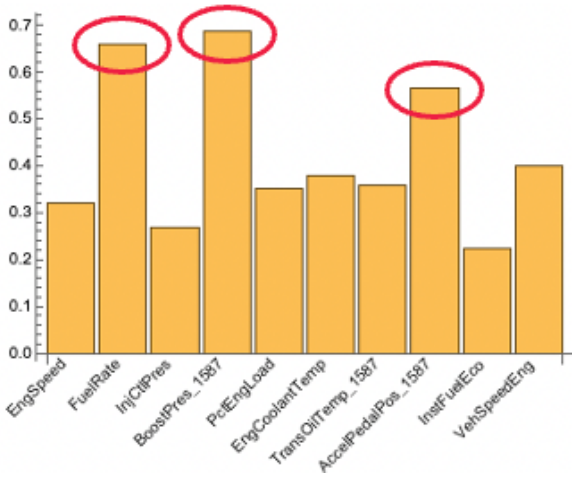

Figure 6: Root Cause Analysis (RCA) vector for an anomaly detected by the Nano from sensor telemetry of a military vehicle engine. The three highlighted features told the reliability team that the truck's fuel system was implicated.

Model Status

The measurements in the previous section give back ML-based information about each individual vector passed through the Nano. There is also information about the overall model that may be of interest in understanding the data corpus from which the model was trained.

Cluster Count

The cluster count tells how many distinct clusters were discovered in the training data relative to the percent variation hyperparameter used during training. We rarely see fewer than 20 clusters in real-world data sets and commonly see cluster counts in the hundreds or even thousands. For example, models built from hi-def video streams commonly create more than 1000 clusters prior to the model maturing and stabilizing.

Learning Curve

The learning curve (Figure 5) is a model status measurement indicating the sample indexes from the training data at which new clusters were created. This can be useful in understanding whether the model reached a mature state during the consumption of the training data. If not, it is typically an indication that more training data is needed or that the percent variation is too low. A linearly increasing learning curve is usually an indication that there is a problem with the input data. This may indicate that the input data is scrambled or random. Another common reason is that the pattern length was misconfigured in Nano and doesn't match the pattern length in the training data.

Cluster Locale

Although the Nano does not work via dimensionality reduction, for visualization purposes, there is a PCA-type measurement available which provides for each cluster a three-tuple (x, y, z) coordinate that can be used, for example, to create a 3D plot of the model. These 3D plots are generally not very enlightening. However, using the three-tuple to scale an RGB XR-overlay for image data can be quite interesting as it shows gradients between image regions and highlights similarities and differences not visible to the unaided eye (Figure 7).

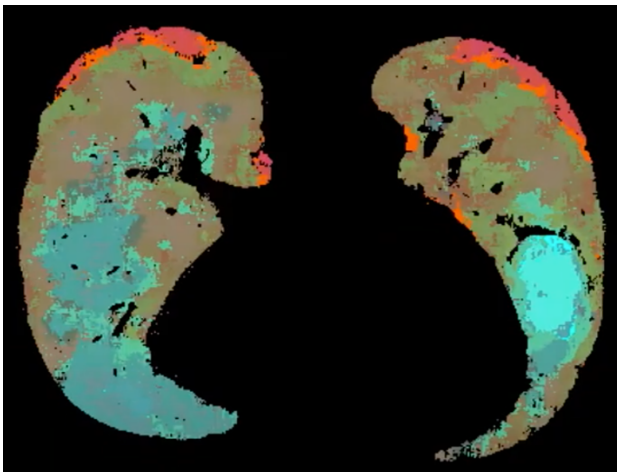

Figure 7: The cluster locale output of the Nano can be used to create an XR overlay for CT imagery or any other planar or volumetric data source.

Speed and Accuracy Benchmarks

KDD Cup 1999

The KDD Cup 1999 is a benchmark for network intrusion detection that has become, over the past two decades, a standard dataset to show both anomaly detection accuracy and segmentation speed. Each row of the 4.9 million row dataset has 32 features representing a network connection, and the challenge is to differentiate benign from malicious connections. Figure 8 compares the average time in microseconds required to assign a cluster ID to each row of the dataset for an increasing number of clusters. The number of clusters in the model (the horizontal axis) is important for the accuracy of a segmentation model as there must be sufficient separation of clusters to distinguish benign from malicious traffic. The computational complexity of K-means and the Nano are both linear in the number of clusters, as can be seen from the figure. However, the complexity growth constant of K-means is 1000x times larger than the Nano. This means that in a deployment, the Nano algorithm can process 1000 rows in the time it takes K-means to process one row. The accuracy of the Nano was 99.9%, which is comparable to the best performers on this dataset.

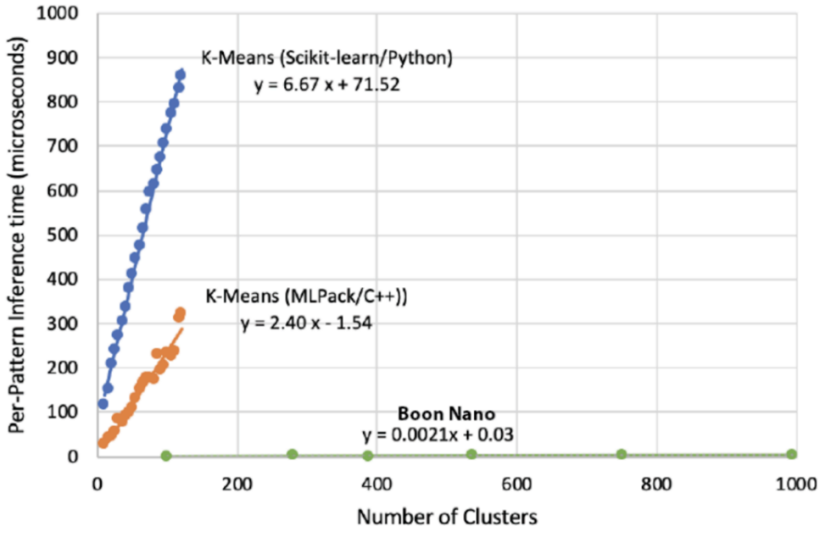

Figure 8: Benchmark comparisons on a single x86 core. Regression plots show sharp linear growth in per-pattern inference time of popular implementations of K-means (Scikit-learn/Python and MLPack/C++) as they cluster the 4.9 million rows in the KDD Cup data set. The growth in inference time for the Nano is essentially flat in comparison to K-means. The 1000x inference time advantage of the Nano over K-means can be seen in the growth constants for the regression plots (2.4 for K-means vs 0.0021 for the Nano).

Well-Separated Clusters

The following benchmark uses a data set containing 100,000 25-dimensional vectors that are separated by ground truth into 100 clusters of sizes ranging from 1 to 3899. The ground truth clusters are convex and well-separated in distance. It is essentially like Figure 1, but with

larger clusters, more clusters, and in 25-dimensional space. Sets like this can easily be generated using Python, MATLAB, etc. or this benchmark dataset is available upon request.

This type of idealized data set provides a way to compare the clustering accuracy of K-means to the Nano when there is no ambiguity about cluster boundaries. The Nano's clustering results are deterministic, whereas most K-means implementations have some built-in randomness from one run to the next based on where the initial centroids are chosen.

Configuration

The Nano hyperparameters were autoconfigured as described above and produced an optimal Percent Variation hyperparameter of 10.2%. With this automatically chosen setting, the one pass segmentation of the data set produced 100 clusters, the correct number of clusters. The K-means used for this benchmark was the standard K-means implementation in Wolfram *Mathematica* (https://reference.wolfram.com/language/ref/method/KMeans.html). The target cluster count "K" was manually set to 100 indicating that K-means should initialize its model to have 100 centroids. This produced 100 clusters for the K-means segmentation.

Results

The time required for the segmentation is shown in Figure 9. It is important to note that while the Nano autoconfigured itself, K-means was manually configured to build a model with 100 clusters since this was already known to be the correct number of differentiated clusters in the ground truth. In general, a data scientist would not know the optimal value of the K hyperparameter, so an "elbow" procedure is typically used to find a value for K. This creates a decidedly "supervised" and computationally slow initial step prior to running the segmentation.

	Nano	K-means
Autoconfiguration duration	3.7 seconds	manual configuration required
Total clustering time	0.12 seconds	55.7 seconds
Per vector inference time	1.2 microseconds	557 microseconds

Figure 9: Comparison of configuration and segmentation time for 100,000 25-dimensional vectors across 100 clusters.

Figure 10 shows the accuracy comparison between the Nano and K-means. With this setting the Nano automatically generated 100 clusters.

Nano Clustering Results

GT ID	GT Size	Recall %	Precision %	Accuracy %
All	100 000	99.8	100.	99.9
1	2	100.	100.	100.
2	2	100.	100.	100.
3	1	100.	100.	100.
4	1012	100.	100.	100.
5	694	100.	100.	100.
6	893	100.	100.	100.
7	644	100.	100.	100.
8	670	100.	100.	100.
9	825	100.	100.	100.
10	733	100.	100.	100.
11	898	100.	100.	100.
12	959	100.	100.	100.
13	1018	100.	100.	100.
14	757	100.	100.	100.
15	664	100.	100.	100.

Clusters 16 – 100 not shown

K-means Clustering Results

GT ID	GT Size	Recall %	Precision %	Accuracy %
All	100 000	88.2	82.8	85.4
1	2	100.	0.1	0.3
2	2	50.	0.1	0.2
3	1	100.	0.1	0.1
4	1012	100.	100.	100.
5	694	100.	39.6	56.7
6	893	100.	100.	100.
7	644	100.	46.7	63.7
8	670	100.	52.6	69.
9	825	100.	52.	68.4
10	733	100.	53.2	69.4
11	898	100.	100.	100.
12	959	52.6	100.	68.9
13	1018	100.	100.	100.
14	757	51.8	100.	68.2
15	664	100.	100.	100.

Clusters 16 – 100 not shown

Figure 10: Comparison of accuracy of the Nano versus K-means for 100 well-separated clusters in 25-dimensional space. (Clusters 16 – 100 not shown). “GT ID” is the ground truth cluster ID and “GT Size” is the number of vectors in each ground truth cluster.

As an example, the ground truth cluster ID 7 shows 100% recall in K-means and 46.7% precision. The 100% recall indicates that all the ground truth vectors from cluster 7 were found together in a single cluster in the K-means segmentation. The precision of 46.7% means the same K-means cluster also contained many more vectors not from ground truth cluster 7 (more than half of them). Ground truth cluster 12 shows 52.6% recall and 100% precision, meaning that the K-means cluster with the largest overlap with ground truth cluster 12 only contained 52.6% of those vectors and the others were assigned elsewhere. The 100% precision indicates that the overlap of ground truth cluster 12 and the K-means matching cluster contained only vectors from ground truth cluster 12 (no intermixing). Accuracy was computed as the harmonic mean of the recall and precision.

The overall accuracy of K-means was 85.4% which is significantly lower than the 99.9% overall accuracy of the Nano. The K-means overall precision of 82.8% indicates that K-means is producing significant intermixing between clusters. The 100% recall of K-means for most clusters shows that K-means cluster purity is generally good with an overall recall of 88.2%. The Nano on the other hand shows nearly perfect recall (99.8%), perfect precision (100%) and overall accuracy of 99.9%. These results for both K-means and the Nano are typical in any dimension and with any number of clusters.

Of special importance is the results for the anomaly clusters with IDs 1, 2, and 3. In the next section, an anomaly will be defined along two axes as a cluster whose size is relatively small within the model (probability anomaly) or whose distance from other clusters is relatively large (distance anomaly). As can be seen in Figure 10, the ability of K-means to differentiate anomalies is very poor as those anomalous elements are grouped into other larger clusters as indicated by the extremely low precision results. This poor performance of K-means in isolating small, distant clusters stems from the way it works, whereby centroids are created on multiple passes via centroid-based weighting. A very small cluster, distant from other much larger clusters, will not be able to retain its cluster purity and will eventually be

combined into a nearby large cluster. The Nano clearly differentiates even very small anomalous clusters among the many other typically large clusters.

Anomaly Detection

Definition of an Anomaly

Anomalies may be measured along two distinct axes: probability and distance (Figure 11).

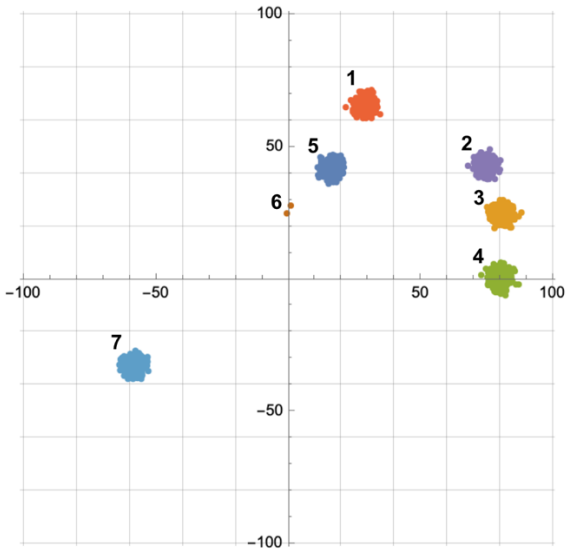

Figure 11: Cluster 6 is an example of a probability anomaly: unusually small relative to the other clusters. Cluster 7 is an example of a distance anomaly: unusually distant from the rest of the model.

Probability Anomalies: This is a common meaning people assign to the word "anomaly" that means a pattern that occurs very rarely among all patterns in the input set. It could be called an infrequency anomaly, as it is the kind of anomaly that indicates a pattern that occurs only very infrequently. Considering the overall probability density function represented by any trained model (with each ID as an event mapped to its probability of occurrence), the probability anomalies are the cluster IDs that are the most improbable.

Distance Anomalies: This is a pattern that is very different from the model in terms of *n*-dimensional distance without any reference to whether it is occurring frequently or infrequently.

Often these two axes agree where an anomaly of one type also corresponds to an anomaly of the other type (Figure 12 (a)). However, Figure 12 also illustrates that an identical distance from the central value of a model may correspond to very different probabilities of occurrence.

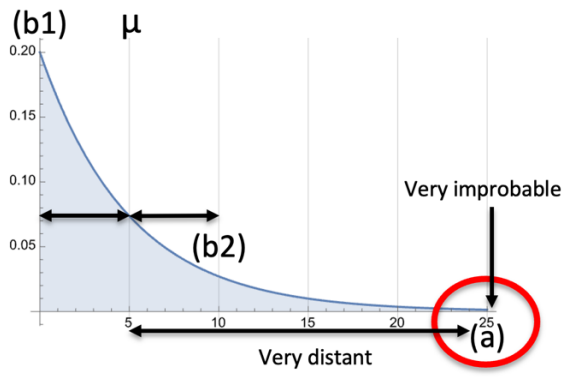

Figure 12: The plot shows the probability density function for an exponential distribution. Values near (a) show both anomaly types: they are both very improbable and very distant from the central value μ. However, note that values near (b1) and (b2) have equal distance to the central value, but values near (b2) are much less probable.

The Nano provides separate measurements of each of these independent axes. Very often we have seen in practice that the Nano's distance anomaly index is sufficient to detect most anomalies, however, we have seen that the probability anomaly index will sometimes show a very rarely occurring cluster ID that is very close in distance to the trained model.

Other anomaly detection approaches use at best only one of these axes. They do not differentiate them and rarely give a normed anomaly measurement for either axis as the Nano does for both axes. This limitation in other methods includes hierarchical methods, isolation forests, autoencoders, and statistical anomaly detection methods. K-means is often used as an anomaly detection method, but in practice is not well suited to the task as illustrated in the benchmarks above.

Other Common Approaches to Anomaly Detection

Autoencoders

Autoencoders are a common neural network-based anomaly detection approach whereby a three-layer network is constructed with its input layer and output layer having the same number of nodes as the dimensionality of the input data. Each input vector is encoded as it passes from the input layer to the inner layer and then decoded as it passes from the inner layer to the output layer. The network is trained to reduce the error between each input vector and the network's encoded-decoded output. Once trained, input vectors having anomalies will not be well reconstructed by the network as they were not included in the training set, thus creating the opportunity to define a sensitivity threshold for anomaly detection. Autoencoders create a dimensionality reduction that allows the normal variation in the training to be represented in the interconnection layers of the network. As an autoencoder is inherently a dimensionality reduction technique, it will not easily detect anomalies that are close in distance to the trained model or indicate the relative probability of occurrence of the anomaly. A head-to-head benchmark between the Nano as an anomaly detector and autoencoders is in process and will be published in the future.

K-means and Gaussian Mixture Models

As shown in the benchmark above, K-means at its algorithmic core is not well-suited to anomaly detection, being much better suited to segmentation of the dominant clusters in the input data corpus. Like K-means, Gaussian Mixture Models (GMMs) create K clusters. However, each cluster of a GMM is more nuanced, with its own *n*-dimensional Normal distribution

around each centroid and each dimension with its own mean and standard deviation. This has the advantage of giving a normed distance measurement for anomalies. However, the same problems as K-means exist if there are anomalies in the training set as they will get consumed by larger neighboring clusters. In addition, GMMs are very slow to train in practice and, like K-means, are orders of magnitudes slower than the Nano per core both in training and during inferencing.

Isolation Forests

Isolation forests are designed specifically for anomaly detection. For each vector in the training set, a random feature index is chosen and a random splitting value in the range is chosen. If that choice isolates the vector from the rest of the training set, then stop. Otherwise, split again until the vector is isolated. The fewer splits needed to isolate the vector the more anomalous it is. Like the Nano, isolation forests are very performant on general purpose processors and use relatively little memory (unlike autoencoders, K-means, and GMMs). However, they do not create a segmentation model and do not provide an *n*-dimensional distance or probability measurement as the Nano does. As with the K-means benchmark above, they have difficulty distinguishing a probability anomaly when it is close in distance to a large cluster since the splitting process does not easily separate the anomaly from the nearby larger cluster. Isolation forests also become less accurate as the dimensionality of the vector space increases.

Other Common Questions

What about Categorical Data?

In general, *n*-dimensional segmentation is complicated when one or more features is categorical. Categorical features such as blood type in medical databases or job title in personnel databases, do not have a natural linear ordering based on magnitude. For example, vectorizing a network packet will require identifying its protocol (TCP, UDP, DNS, etc.) There are more than a dozen internet protocols possible and there is no natural magnitude ordering of these protocols. One-hot encoding has been used with K-means and other segmentation algorithms to expand categorical features into multiple additional features in the vectorization scheme for the input data. This approach has many problems, not the least of which is the weighting bias it creates in the *n*-dimensional segmentation based on the number of categories in the encoding.

The Nano can be used with one-hot encoding of categorical features, but an even better approach leverages the extreme speed and small memory footprint of the Nano. If a feature in the data vector is categorical, a separate segmentation model is trained for each category in that feature. Continuing the network traffic example, one Nano model would be trained for DNS packets, another for TCP packets, another for UDP packets, and so on. As each packet is consumed during training (and during inferencing), it is routed to the appropriate model for

its category. This retains the distance integrity within each model both for the segmentation and for anomaly detection.

Is it possible to weight features in a pattern?

The Nano supports the ability to weight features to increase or decrease their influence in the segmentation. For instance, if one feature has a weight of 3 and all other features have a weight of 1, then the triple-weighted feature will act like three identical columns each weighted with a 1. This allows hand tuning when it is believed that one feature should be given more weight. In practice, differential weighting is not common, but it is available and does not increase the training or inference time. A weight of 0 applied to a feature causes the segmentation to behave as if the feature were not there.

What about Parallel Architectures?

The speed benchmarks above compared three different single-core K-means implementations to a single-core Nano implementation. When multiple cores are available, the Nano supports perfect linear speed up in inference mode and near-perfect linear speed up during training. This means that if the Nano is processing one pattern per microsecond on a single core, then making 10 cores available to the Nano will result in a throughput of 10 patterns per microsecond. As an example, one recent application used 40-cores to process 100 gigabit ethernet traffic in real time. During training a global synchronization block of the parallel threads only occurs when a new cluster is added to the model, a rare occurrence in a large data set.

Returning to K-means, there are numerous articles showing how it can be accelerated with multi-core implementations, including implementations on GPUs. As a vector is processed through K-means, its distance to each centroid in the model must be calculated to assign it to the nearest cluster. During inference mode on a general-purpose processor, this K-sized computational task can be accelerated in several ways across many cores. Parallel K-means training is more challenging since nearly every iteration creates a change to one of the centroids, and that change must be communicated to all the operational cores. For GPUs this is especially problematic as the device memory is not large enough to store the entire training set. Despite the thousands of cores available on GPUs, the Nano segmentation is faster on *a single core* than a GPU running K-means with 3500 cores on the same data set. Thus, running the Nano with a modest 16-core general purpose processor creates a segmentation capability more than an order of magnitude faster than GPU and with a much lower compute cost.