ONLINE DATASET DRIFT IN $O(\log N)$

Sven Cattell
Senior Data Scientist at Elastic
sven.cattell@elastic.co

ABSTRACT

Dataset drift is a major issue for security models. We present an online method for tracking drift that takes $O(\log(n))$ per sample. This is accomplished by using the geometric properties of covertrees to divide and conquer the problem. We also present a library that implements the method, and some future directions that are now open due to the extreme speedup over the current methods of measuring dataset drift like Wasserstein.

1 INTRODUCTION

Adversarial environments have a naturally high rate of drift. Defenders try to detect and mitigate attacks against their networks. Adversaries try to bypass the detections and mitigations. This produces a highly innovative environment, and the efficacy of machine learning models deployed by defenders degrade over time. The models we use in security are trained on data collected up until that point, and may have an acceptable efficacy at that point. The daily error rate tends to climb, until the model no longer meets quality standards. The easy solution is to monitor the error rate over the last hour, day, or week, and to retrain and redeploy once it exceeds a pre-defined threshold. However, labels are often expensive and the error rate is often unknown until it is too late. The simple solution is to retrain and redeploy on a fixed schedule. This carries costs, and there could be a new popular attack mid-way though a model's lifecycle that it does not detect. Our method uses a covertree to create a discrete bayesian distribution that approximates the true distribution. We can compute the posterior in an online manner in $O(\log N)$ time, and can compute the Kullback–Leibler divergence between the prior and posterior in $O(k \log N)$ time, where k is the number of post-deployment observations, and N is the number of points in the the training set. We use this to monitor a production model trained on approximately 60,000,000 malware samples. While our main figures use a proprietary dataset and are thus not publicly reproducible, we provide code to reproduce our results using the EMBER open malware dataset in the Goko repository at github.com/elastic/goko Anderson & Roth (2018)Cattell (2021). The proprietary dataset is similar to EMBER, but is much larger and thus a better demonstration of the scalibility of this method.

1.1 CURRENT SOLUTIONS

Current methods to measure dataset drift are computationally prohibitive. Traditional optimal transport methods take $O(n^3 \log(n))$ Villani (2008). Sinkhorn distances and other clever tricks, like clustering, reduce the cost of the core linear program Cuturi (2013). These can make it possible to calculate an approximation of the Wasserstein distance in high dimensions, but it is still cost prohibitive on our large, complex dataset. The other popular measure is the Kullback–Leibler divergence, which requires that we fit a distribution to the data. Current methods of appropriating the distribution, like a variational auto-encoder or a gaussian mixture model, both do not fit our malware data set well and are still too computationally expensive to compute the KL-divergence at our scale Jean-Louis Durrieu (2012), Yong et al. (2020).

Another solution is to deploy models that can be trained online. This does not work for models at the edge and introduces a new attack surface. Microsoft's Twitter bot Tay was trained online and famously exploited in 2016 Lee (2016). Partial retraining using transfer learning can help reduce costs for a redeployment, but this is still reactive.

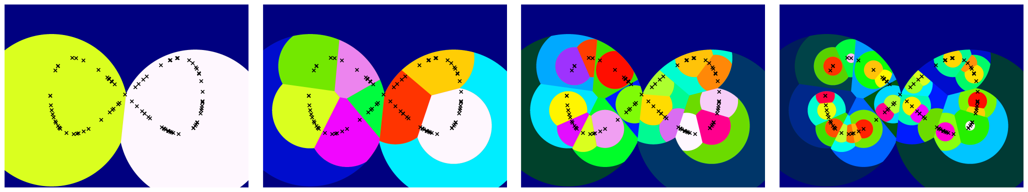

Figure 1: The partition of the covertree, artificially stopping the tree at depths 1,0,-1, and -2. The colors show the partition as if we had stopped at that layer, each color in each image denoting a particular component of the partition of $\mathbb{R}^2$ that this cover tree produces. The components from the previous layer are not discarded, we just carve smaller balls from those. Note how the refinement conforms to the dataset over the layers.

2 COVERTREES

A *covertree* over a dataset $X = \{x_1, \dots x_n\}$ is a filtration of a dataset into m*-layers*, with a scale base of S

$$\{x_r\} = C_k \subset C_{k-1} \subset \cdots \subset C_{k-m} = X,$$

which satisfies the following properties:

1. *Covering Layer*: For each $x_j \in X$ and $i \in \{k, \dots k-m\}$, there exists $p \in C_i$ such that $d(x_i, p) < s^i$.
2. *Covering Tree*: For each $p \in C_{i-1}$ there exists $q \in C_i$ such that $d(p, q) < s^i$.
3. *Separation*: For all $p, q \in C_i$, $d(p, q) > s^i$.

We associate each $p \in C_{i-1}$ to a parent $q \in C_i$ which satisfies (2), to form a tree. See the appendix section A.1 for details of how we improved on the originial definition and construction.

We use the notation $N_{s,i}$ to denote a node on the sth level, with center x_i, $B_{s,i}$ to denote the ball of radius S^s centered at x_i, and we say the coverage, $C_{s,i}$, to mean all points covered by the node under our partitioning. $C_{s,i} \cap X$ is the set of points who are associated to, or are the centers of, the decedents of the node. Due to the use of singletons we also have the set $S_{s,i}$, which are those points uniquely associated to $N_{s,i}$. We abuse notation, and use this notation to mean to be both subsets of $\mathbb{R}^D$, or X. One can think of the partition either globally or locally. Globally it is the collection of singletons sets, $\{S_{s,i}\}$. Locally at each node $N_{s,i}$ with children $\{N_{t,j_k}\}$ the partition consists of $\{S_{s,i}, C_{t,j_1}, C_{t,j_2}, \dots\}$. However "local" in this case may be a misnomer as a node can cover the entire tree, and thus have a very coarse partition. However, every node's partition still contains some geometric information that contributes to the global partition, see figure 1 for a diagram.

3 THE DISTRIBUTIONS

The core component of the distribution is a Dirichlet distribution attached to each node in the covertree. The "categories" for this distribution are each of the children of the node, and the singletons (as a single category). We use the coverage of each child and the number of singletons for the concentration parameters. These coverage counts are all based on the training data and represent the prior distribution. So, to node $N_{s,i}$ with children $\{N_{t,j_k}\}$, the prior distribution is,

$$\text{Dir}_{s,i} = \text{Dir}(|S_{s,i}|, |C_{t,j_1}|, |C_{t,j_2}|, \dots).$$

This distribution represents a prior for the true distribution based on the training data locally at a particular node. For the global picture of the entire tree we use the Dirichlet distribution representing the probability of landing in each single partition,

$$\text{Dir}_T = \text{Dir}(\{|S_{s,i}| \text{ s.t. } N_{s,i} \text{ is a node in } T\}).$$

The KL divergence for these discrete distributions is easy to compute, they are just finite sums. This is much simpler than the high dimensional intergrals that a continous distribution would require, and does not require any linear programing. As we are concerned with computing the divergence

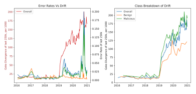

Figure 2: For the graph on the right we partition the feed into the two classes and graph the divergence of each class, as well as the overall divergence. For the graph on the left we compare the error rate to the overall divergence.

between the posterior and the prior, we only need to consider terms which differ in the sum, so the computation is linear in the number of observations.

If we observe a new point, like a query from a user, then we can determine which component of the partition that query belongs to. This can be done in $O(\log N)$ time by recursively checking which component of each node's partition the query point belongs to until we end up in a singleton component. We call the sequence the *path*, $(N_{s_1,i_1}, \ldots, N_{s_k,i_k})$. We can update the global partition, Dir_T by incrementing the count for S_{s_k,i_k}, and each of the local partitions in a similar fashion. We can also forget this point just as easily by decrementing the respective counts. In practice we create a double headed queue for storing the last k paths from queries, and a small hashmap with the count of each parent-child pair in the aformentioned queue. Adding a few other statistics to this datastructure allows us to compute the KL-divergence between the prior and posterior for the global distribution Dir_T in $O(\log N)$, and in $O(k \log N)$ for the local distribution.

The expected memory consumption for each of these evidence data structure is $O(k \log n)$ where k is much smaller than n. For our experiments with Ember 2018, which has 900,000 points in the training set we usually use a history of 1,000-5,000 points for our evidence set. This also enables us to track many users, each with their own interests simultaneously.

4 Results

We have a proprietary dataset with a large amount of both benign and malicious software samples paritally collected from VirusTotal LLC (2020). We chose a subset of samples collected from VirusTotal from before the first of January 2019 for our training set. We use the first seen timestamp field from virus total, so that this is reproducible for other researchers. There are about 60,000,000 samples in both the training and test sets. We trained both a normal production model on this subset, a covertree using Goko, and built a distribution using the plugin. A seperate test set was chosen, which was not chronologically constrained. We fed the test set chronologically to a to a posterior tracker with different window sizes.

We can see in figure 2 that the KL-divergence of our distributions starts to increase dramatically the moment they start seeing data from their future. We compare the overall divergence to both the relevant error rates and the divergence for the benign and malicious components. As we can see, unsurprisingly, in the right figure the malicious samples drift faster than the benign ones. We can see in the left figure that the false negative rate (FNR) is how many malicious samples we miss, which may be extremely damaging ransomware, also increases at a dramatically higher rate than the false positive rate (FPR).

The false positive rate is very important to security analysts, the relative abundance of benign software on an enterprise network means even a relatively mild rate can cause alert fatigue. As you can see from the left figure 2, the FPR stays relatively constant even with an extremely old model, and the FNR eventually spikes and is chaotic afterwards. This is likely due to an unpredictable event which is what we're trying to detect. This spike is long after our scheduled redeployment, so while the current strategy worked in this case and we might be tempted to reduce our costs by relaxing our schedule from this figure, the event may come earlier next time. The initial spike in FNR is also a few months after the chronological drift is detected, indicating that the model we're monitoring is able to generalize to the new data, but breaks down at some point. The major spike in both drift and error happens concurrently in late 2020.

The local distributions might be the key to understanding the differences between the error rate and the drift. Figure 3 is a rough aggregate of the local divergences. We are currently researching

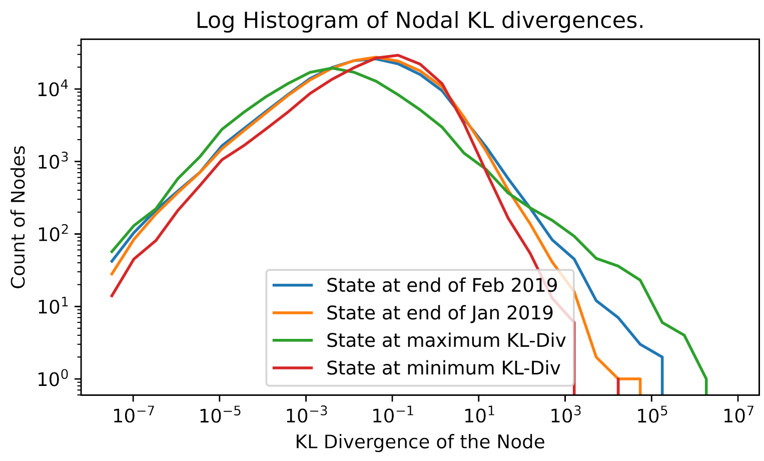

Figure 3: The log-log histogram of the local distribution at each node. The histogram's bins are logarithmically sized to make these curves more sensitive to the very small divergences that most nodes have. The Y axis is also logarithmic to make the tail on the right visible, though this does suppress the major differences in the smaller nodal divergences.

methods for visualizing these at a more granular level. We can see that when the drift is at its maximum for the entire tree, it is at its minimum for most of the nodes in the tree, but has a much longer tail. The log-histogram curve for the end of February is still very close to the log-histogram curve for the state at minimum global divergence, compared to the log-histogram curve for the state at maximum global divergence. There might be a more complex measure of drift that is more closely correlated to the efficacy we want to monitor, which is another area for future work.

5 Conclusion

With a dataset with over 60,000,000 points on a single AWS EC2 r5a.16xlarge node we were able to process between 60,000 and 80,000 points per second. The dataset is much larger than any other method can handle, and the computation of the drift allows for it to be placed in-line with an inference node. A security data scientist can set up an alert for when the global divergence exceeds a desired quantity, then either investigate or retrain. However, as we show, this simple alert trigger may not completely capture the conditions a security enterprise cares about. A more complex trigger based on some aggregate local nodal divergences may be able to address this shortfall. This is all in addition to a standard KNN algoritm which can be used in other parts of the pipeline. We believe than any organization using ML in an adversarial setting could make use of this technique especially if they also need a KNN solution.

References

Hyrum S. Anderson and Phil Roth. Ember: An open dataset for training static pe malware machine learning models, 2018.

Sven Cattell. Goko. `https://github.com/elastic/goko`, 2021.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Mike Izbicki and Christian Shelton. Faster cover trees. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1162–1170, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/izbicki15.html`.

Finnian Paul Kelly Jean-Louis Durrieu, Jean-Philippe Thiran. Lower and upper bounds for approximation of the kullback-leibler divergence between gaussian mixture models. *2012 Ieee International Conference On Acoustics, Speech And Signal Processing (Icassp)*, pp. 4833–4836, 2012.

Peter Lee. Learning from tay's introduction, 2016. URL `https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/`.

Wenjing Liao and Mauro Maggioni. Adaptive geometric multiscale approximations for intrinsically low-dimensional data, 2016.

Chronicle LLC. Virustotal, 2020. URL `www.virustotal.com`.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Bang Xiang Yong, Yasmin Fathy, and Alexandra Brintrup. Bayesian autoencoders for drift detection in industrial environments. 06 2020. doi: 10.1109/MetroInd4.0IoT48571.2020.9138306.

A Appendix

A.1 Covertree Improvements and Memory Reduction

The original paper calls the filtration requirement a *nesting* condition, and does not require that the *covering layer* condition be satisfied. Their algorithm guarantees this condition too, and it is helpful for motivating the partition so we include it. Several other papers that use covertrees also fix the scale base at 2 in their definition, but not the code, which causes excessive per-node branching in many datasets Liao & Maggioni (2016). The separation condition is prohibitive in practice as it induces a syncronization bottleneck in the creation of a covertree, which is solved by weakening the condition. Izbicki and Shelton introduced the *weak separation* where we only require children of a parent to be separated, Izbicki & Shelton (2015). Izbicki et all also remove the nesting requirement in an effort to reduce the number of distance computations, but this is unnecessary.

We build a *weak cover tree* where we use the per-node separability of Izbicki, but all other conditions from the standard definition. The other distinction we make is to treat children of a node that only cover a single point of the dataset differently from the other children. These points are not relevant to the structure of the tree and lead to the following definition:

A *weak partial covertree* is a covertree of a subset of $R \subset X$ that satisfies the weak separability condition over layers C_i, together with an association of each point in $X - R$ to a point in a layer, that is for each $x \in X - R$ there is an $i \in \{m, \dots m - k\}$ and a $p \in C_i$ such that $d(x, p) < s^i$. Additionally, if q is a child of p, then $d(x, q) > s^{i-1}$.

The points in R are called the *routing points*, and the points in *X-R* are called *singleton points*, and if their associated routing point is a leaf of the covertree, we call it a *leaf point*. We split the points into these two categories to enable a cold storage of the singletons (on a network attached storage for instance), and hot storage of the more relevant routing nodes. We've modified the KNN algorithm to minimize references to the singletons, and they are completely ignored by the drift calculation. We also expose a *Leaf cutoff*, L_c, parameter to control the size of the routing set. If a node covers less than L_c points other than itself, then we associate all covered points to singleton points associated to this node. The final new parameter is the *Minimum resolution index* R_m to reduce sensitivity to local noise and allow us to use fast but potentially numerically unstable norms. We stop the tree at layer R_m, associating all remaining unassociated points that to the nodes that cover them.

Figure 4 is the motivation behind singletons. They form the bulk of the ember dataset, representing 48.3% of the data. It is important to node that they are not concentrated in the lower levels of the dataset. The density of the data varies wildly. A good analogy is that most of the data is at the density of the vacuum of space, with some concentrated parts reaching the density found at the centers of stars. When a node splits, the population of each child is distributed like an exponential distribution. When we enable singletons, and set the L_c to a more reasonable 50, we see that the node coverage histogram is still concentrated near 0, but the concentration is drastically reduced. This also reduces the size of the routing set in the partial covertree to 16.5% of the training dataset, leading to a much lower memory usage for the hot routing points in practice.

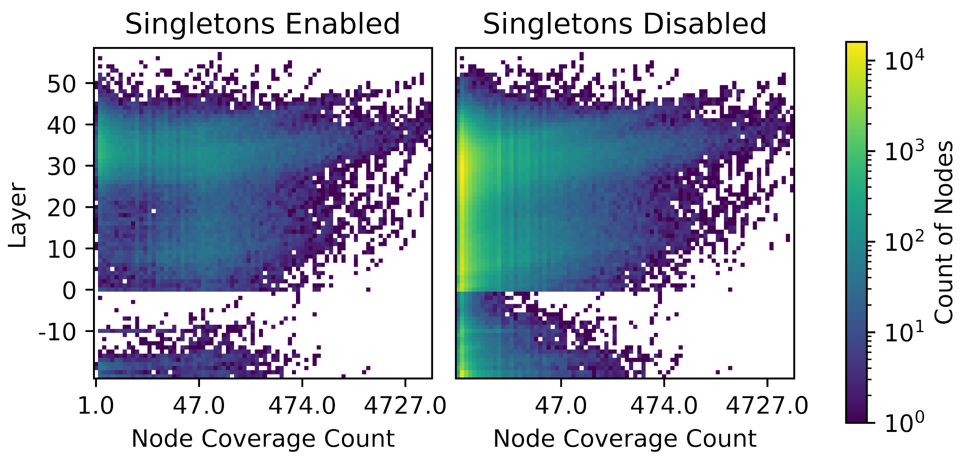

Figure 4: A heatmap histogram of the number of nodes with a given coverage count of the nodes of a covertree, stratified by layer. The histogram bins for the node coverage count axis are also logarithmic. The covertree was built on the Ember dataset, with scale base 1.5, no use of singletons and a leaf cutoff of 0. The transition at the layer of scale index 0 is because to a large portion of the features are integers, so the layers below this scale are relying a on small number of features.

A.2 Sampling from the Distribution

The main use of the distribution in this paper is a drift calculation, but it can also be used as a model of the underlying distribution. Sampling from this is simple, it's equivilant to sampling from a categorical distribution over the relative concentrations, and can be done recursively from the root until you sample the singleton partition for a node. When you do, you can sample from an gaussian built off of the singletons associated to that node. See figure 5 for some examples generated from MNIST. The probability of sampling from a particular node's singleton gaussian is:

$$P(S_{s_k,i_k}) = P(S_{s_k,i_k}|C_{s_k,i_k})P(C_{s_k,i_k}|C_{s_{k+1},i_{k+1}})\dots P(C_{s_2,i_2}|C_{s_1,i_1}),$$

where N_{s_j,i_j} is the parent of $N_{s_{j+1},i_{j+1}}$, and N_{s_1,i_1} is the root. Each of these conditional probabilities can be computed locally at the node.

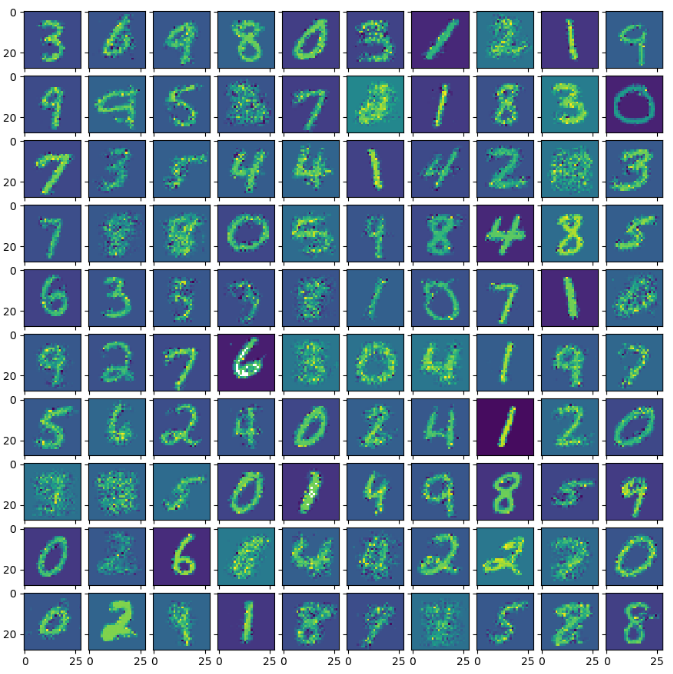

Figure 5: Samples from the distribution described in 3. The singletons