Intrusion Detection with Unsupervised Heterogeneous Ensembles using Cluster-based Normalization

Scott Ruoti, Scott Heidbrink, Mark O'Neill, Eric Gustafson, Yung Ryn Choe
Sandia National Laboratories*
scott@ruoti.org, {sheidbr,moneil,edgusta,yrchoe}@sandia.gov

ABSTRACT

Outlier detection has been shown to be a promising machine learning technique for a diverse array of fields and problem areas. However, traditional, supervised outlier detection is not well suited for problems such as network intrusion detection, where proper labelled data is scarce. This has created a focus on extending these approaches to be unsupervised, removing the need for explicit labels, but at a cost of poorer performance compared to their supervised counterparts. Recent work has explored ways of making up for this, such as creating ensembles of diverse models, or even diverse learning algorithms, to jointly classify data. While using unsupervised, heterogeneous ensembles of learning algorithms has been proposed as a viable next step for research, the implications of how these ensembles are built and used has not been explored.

1. INTRODUCTION

There are many open questions about the best ways to aggregate the responses of ensemble learners [1; 4]. To examine these and similar questions, we ran thousands of experiments on the NSL-KDD dataset [2], comparing different combinations of algorithms, settings for those algorithms, normalization methods, and aggregation approaches. The results from these experiments demonstrate best practices for heterogeneous, ensembled unsupervised outlier detection as well as giving insight to several open questions in the area.

The contributions of our research are as follows:

- **Novel cluster-based normalization method.** We have developed a novel approach for normalization of cluster-based outlier detection algorithms.
- **Evidence that higher numbers of clusters produce more accurate outlier detection.** Across all of our tests, the most determinant factor in performance was the number of clusters that were built by each clustering algorithm. In all cases, algorithms performed better when they built a larger number of clusters. This suggests that when learning the behavior of a network, it is best for each individual type of behavior to be classified into its own cluster, and not aggregated with other clusters.
- **Comparison of aggregation methods.** We compare various methods for aggregating the responses of algorithms within an ensemble. Our results show that the using the mean value is the top performing aggregation method, followed by using the maximum value and then by having a voting scheme requiring agreement of any two algorithms.
- **Analysis of algorithm performance for intrusion detection.** In our testing, we found that the best performing systems were always ensembles of algorithms, and not a single algorithm by itself. We found that SimpleKMeans was in all of the top performing ensembles. We also found that adding multiple algorithms from the same family (e.g., SimpleKMeans and XMeans) can also be more beneficial than adding a diversity of algorithms.

2. SYSTEM

Our system was built using standard and third-party unsupervised algorithms implemented in Weka which were empirically verified to be able to run at near line speed, shown in Table 1.

We modified the algorithms to output a single score which represents the *"outlierness"* of an instance. For clustering algorithms we did this in the following way: (1) We retrieved the cluster that the passed instance should be classified into. (2) We then retrieved the metric that was used by the cluster to determine the cluster assignment, and returned that metric as the "outlier" score. In several cases, we modified the value so that it scaled linearly.

We then normalized these scores with two different approaches so that scores from different algorithms can be directly compared. The first approach scales the data based outputs across the entire training set. We analyze the scores produced during training, and calculate the minimum score (min_score), the maximum score (max_score), and the range of scores ($range = max\_score - min\_score$). These values are stored for each algorithm. During analysis, scores are normalized using the following formula: $normalized\_score = \frac{(score - min\_score)}{range}$. To address limitations in this approach, we propose our novel

*Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energys National Nuclear Security Administration under contract DE-NA0003525.

978-1-5386-0752-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICWS.2017.107

Algorithm	Source	Type
InterquartileRange	Native	Filter
SimpleKMeans	Native	Clusterer
XMeans	Native	Clusterer
IsolationForest	Plugin	Classifier
Expected Maximization (EM)	Native	Clusterer
MTree	Plugin	Clusterer
Learning Vector Quantization (LVQ)	Plugin	Clusterer
CascadeSimpleKMeans	Native	Clusterer
SelfOrganizingMap (SOM)	Plugin	Clusterer
SequentialInformationBottleneck (sIB)	Native	Clusterer

Table 1: Weka algorithms used in the system

normalization technique, where normalization models are tracked *per cluster*. In other words, instead of having a single *min_score*, *max_score*, and *range* for each algorithm, we generated a *min_score*, *max_score*, and *range* for each cluster created by the algorithm. Normalization for an instance is then applied by which cluster it was closest to.

Next, we aggregate the normalized scores to calculate a single score for the ensemble. We allow for four common approaches: minimum value, maximum value, mean value, or a voting scheme.

Finally, this score is compared against a threshold to determine whether the given instance is an outlier. The score from the aggregation step is compared against a predefined threshold to determine whether or not the instance is an outlier.

3. EXPERIMENT METHODOLOGY

Using our system, we ran millions of experiments using the NSL-KDD dataset [2], a labeled dataset that attempts to mimic real network traffic. We then compared the results from different configurations and identified trends which provided insight into how to best perform intrusion detection with unsupervised outlier detection ensembles.

3.1 Configurations and Experiments

For our experiments, we evaluated the effect of the following factors:

1. **Algorithm selection:** We used ten total algorithms in this work, and ran experiments on all possible combinations of these.

2. **Cluster count:** We use two values for this factor: low number of clusters and high number of clusters. Our target sizes for each of these is five and twenty, respectively. We say "target" here because some algorithms lack an explicit parameter to specify the number of clusters during model generation.

3. **Normalization method:** We tested both traditional normalization and our novel cluster-based normalization method.

4. **Aggregation method:** We tested four styles of aggregation. This included three numeric integration methods (min, max, mean) as well as voting-based aggregation. For voting-based aggregation, we ignored VOTE-1 and VOTE-N schemes.

We conducted experiments using all possible combinations of these factors. We call the set of values for each combination a *configuration*. In total, this gave us $40 + 3 * 4,052 + (\sum_{n=2}^{9} \sum_{i=n}^{10} \binom{10}{i}) * 2 * 2 = 28,580$ configurations.

For each of these configurations we evaluated its performance against a range of threshold values, ranging from [0, 2] in .01 increments. This allowed us to determine optimal threshold levels and generate ROC curves for each configuration. In total, $5,744,580$ experiments were conducted for generating the data for each configuration.

3.2 Metrics

For each experiment we calculated the true-positive rate, false-positive rate, and true-negative rate. Using this information, we generated a ROC curve for each configuration. Additionally, for each configuration we also calculate several single score metrics: the area under the curve (AUC), partial AUC from [0, 1] with .1 increments, the F_1 score, the $F_{0.5}$ score, and Youden's J-statistic [3]. These metrics are helpful in obtaining a quick sense of the performance of a given configuration.

Because of the large number of configurations, it was impossible to manually perform exhaustive pairwise ROC curve comparisons for all them. To filter our result set, we remove from consideration all configurations whose ROC curve is always strictly below another configuration's ROC curve. After removal of these dominated configurations, we then ordered the remaining configurations by our various single-score statistics. This allowed us to quickly establish the top performing configurations, which were then examined manually. In total there were $13,553$ dominated configurations, and their removal reduced the analysis space by 47.4%.

4. RESULTS

In this section we describe the results of our experimentation. While we analyzed all of our results, in this section we discuss a sample of the data that clearly shows what was learned from our results, but we emphasize that the trends discussed in this section held over all our results.

4.1 Normalization

As shown in figure 1, cluster-based normalization outperforms algorithm-based normalization. For the top-performing 163 configurations, cluster-based normalization has 100% representation and algorithm-based normalization has almost none. Out of the non-dominated set of configurations (N = $15,027$), 44% used cluster-based normalization while 56% used algorithm-based normalization. When directly comparing algorithm-based normalization against cluster-based normalization, i.e. where configurations match in aggregation method and algorithms used, 70% of the configurations using cluster-based normalization obtain higher J-Statistics than their algorithm-based normalization counterparts. This gives strong evidence that our novel normalization method improves upon the standard normalization approach, at least for outlier detection tasks on datasets similar to NSL-KDD.

4.2 Cluster Size

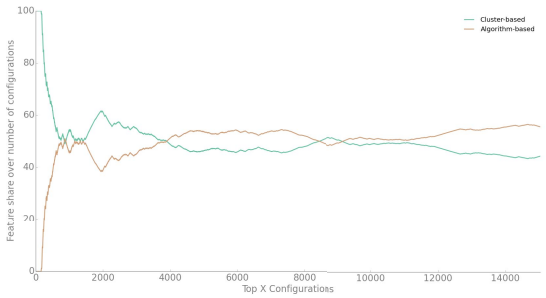

Figure 1: Percent share of the two normalization methods with respect to configuration rank (ordered by descending J-Statistic). Only non-dominated configurations are considered. Cluster-based normalization commands 100% share for the top 163 configurations.

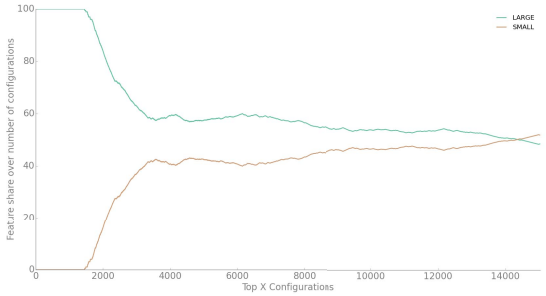

Figure 2: Percent share of the two cluster sizes with respect to configuration rank (ordered by descending J-Statistic). Only non-dominated configurations are considered. Large clusters command 100% share for the top 1453 configurations.

Figure 2 shows the share of our two cluster values across top-performing configurations by J-Statistic. Large clusters command 100% share for the top 1453 configurations, further demonstrating the utility of higher number of clusters for this learning task. Small clusters are not represented significantly until the top 2000 configurations are considered. When directly comparing cluster sizes, keeping other factors constant, 67% of large cluster configurations outperform their small-cluster counterparts.

4.3 Aggregation

Figure 3 shows the share of aggregation methods with respect to top-performing configurations. Mean, Max, and Vote-2 are the most relevant for the highest performing configurations, and Vote-2 replaces Mean as the most represented method after considering configurations beyond the top 200. It is interesting to note that Vote-2 is represented over Max (i.e., Vote-1) after the top 16 configurations, but that Vote-3 or any other voting method never overtakes Vote-2. This suggests that voting as a concept is not well-suited for this task, but that the usage of "second opinion" shows promise.

4.4 Algorithms

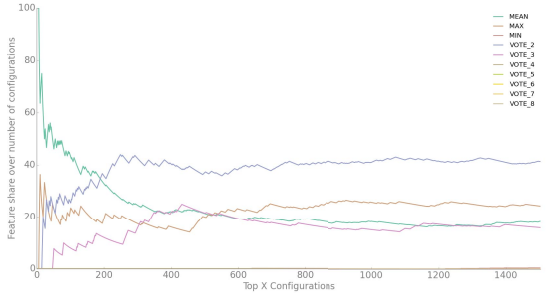

Figure 3: Percent share of the various aggregation methods with respect to configuration rank (ordered by descending J-Statistic). Only the top 10% of non-dominated configurations are shown. The Mean aggregation method dominates the top performing configurations but is replaced by Vote-2 after the configurations beyond the top 200 are considered.

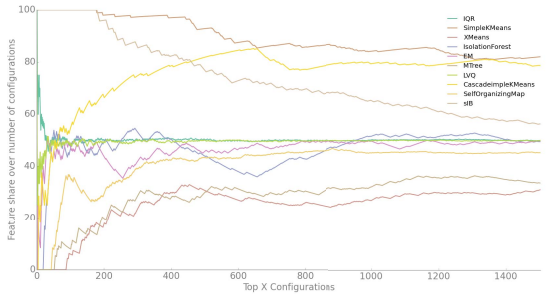

Figure 4: Percent presence of the various algorithms in a configuration with respect to configuration rank (ordered by descending J-Statistic). Only the top 10% of non-dominated configurations are shown. IQR, SimpleKMeans, sIB have the greatest presence in the set of highest ranked configurations.

None of the top performing configurations used only a single algorithm. Our results are actually rather definitive in asserting that ensembles far outperform single algorithm solutions.

We also note that SimpleKMeans and sIB are in all of the top-performing configurations. The next most common algorithm is interquartile range (IQR). This same pattern continues through the rest of the data, with the top configurations always including SimpleKMeans, and most including IQR. This can be seen at a large-scale with Figure 4. We did find the presence of all algorithms in at least some of the top 100 configurations. This suggests that there is value to each approach.

Past research has suggested that it is preferable to include two algorithms from different families than the same family[4]. While our data shows that this is true for the first two or three algorithms, it doesn't hold for greater numbers of algorithms. CascadeSimpleKMeans and SimpleKMeans are often both used in the top performing

configurations, even though they are in the same algorithm family. Even more interesting, when we ran a set of experiments where we simultaneously included multiple versions of the KMeans algorithms—differing only in the number of clusters (5, 10, 15, 20)—we found that the top configurations often contained multiple copies of SimpleKmeans. While this result is not definitive, it indicates that more research needs to be done on algorithm selection, specifically with respect to algorithm diversity.

4.5 ROC Comparison of Top Configurations

The differences in the ROC curves between the top performing configurations becomes almost indistinguishable after about a false positive rate of 0.1. However, when the considering normalization method, we see that the naive normalization method has a long tail before reaching a true positive rate of 1, but until the false positive rate of roughly 0.2 they perform nearly as well as the top 20 performing configurations.

For different clusterings we find that small-cluster configurations perform worse at lower false positive rates, but don't have the long tail until they reach a value of 1 for true positive rate. Summarily, large-cluster configurations perform better at a lower value for acceptable false positives and normalization-by-cluster configurations perform better at larger acceptable false positive rates.

5. DISCUSSION

We seek to provide further incite to three important aspects of outlier ensembles as discussed by Zimek et. al.: "assessment of diversity, normalization of scores, and combination procedures." Most importantly we provide, to the best of our knowledge, a new normalization method to explore when creating ensembles of clustering algorithms.

While Zimek et. al. and Aggarwal both highlight numerous issues to consider when assessing the diversity of an ensemble we focused on two: different cluster sizes and different algorithm families. While combining different families was essential in our top performing ensembles, including another algorithm from the same family outperformed further increasing diversity. An initial consideration is that perhaps there is a different normalization method that would allow a better "translation" between the two families scores so that each's score could be compared correctly. We attempted to resolve this with a normalization method based on each cluster's actual behavior rather than on the algorithm as a whole. While this did perform better than a naive normalization, it did not resolve the issue. We also verified that Aggarwal was correct in suggesting that the same algorithm used with different parameters, is a good form of ensembling. This is promising in that parameters perhaps don't need to be "tuned" to each application, but rather an ensemble of various parameters can be used instead.

We also believe that combining aggregation methods could produce more accurate results. Many algorithms, such as XMeans, only performed best in certain aggregation schemes. This suggests that these algorithms may "pull down" the outlier score in non-optimal aggregation methods. Perhaps then combining aggregation methods based on the alogirhtms can produce a better result, such as taking the max within algorithm families and then average across different families.

5.1 Future Work

There are four specific future work directions we believe should be taken. First, an analysis of normalization methods in regards to "translating" between algorithm families. Second, combining aggregation methods in ensembles. For example averaging the results between the same family of algorithms while taking the max value between different algorithm families. Third, analyzing different distance metrics could help resolve issues in clustering algorithms where some have features with large ranges, and others with much smaller ranges. Finally, analyzing how these algorithms perform given a sequential ordering, rather than an end aggregation method.

6. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *ACM Sigmod Record*, volume 30, pages 37–46. ACM, 2001.

[2] M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.

[3] W. J. Youden et al. Statistical methods for chemists. *Wiley publications in statistics*, 1951.

[4] A. Zimek, R. J. Campello, and J. Sander. Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):11–22, 2014.