Comparing Machine Learning Techniques for Zeek Log Analysis

Daniel K. Andrews
U.S. Military Academy
West Point, New York 10996
daniel.k.andrews.mil@mail.mil

Rajeev K. Agrawal
U.S. Army E.R.D.C
Vicksburg, MS 39180
rajeev.k.agrawal@erdc.dren.mil

Suzanne J. Matthews, Alexander S. Mentis
U.S. Military Academy
West Point, New York 10996
{suzanne.matthews,alexander.mentis}@westpoint.edu

2019 IEEE MIT Undergraduate Research Technology Conference (URTC) | 978-1-7281-5818-1/19/$31.00 ©2019 IEEE | DOI: 10.1109/URTC49097.2019.9660501

***Abstract*—Network logs from intrusion detection and prevention systems such as Zeek provide a plethora of information to help network analysts identify malicious activity. However, the volume of data collected necessitates an automated way to filter it. Traditional signature-based "misuse detection" is unable to detect previously unseen malicious activity. In contrast, machine learning methods can suggest classifying network activity as normal or malicious by learning hard-to-define patterns that discriminate between the two classes. Previous work has applied a variety of machine learning techniques to this problem with some success, but the proprietary nature of real-world data often makes accurately comparing the performance of different techniques impossible. In this paper, we compare the performance of eight machine learning models on the same real-world dataset comprised of HTTP log data gathered over six months from an enterprise network. Our experiments show that, when trained and tested on the same data, k Nearest Neighbors results in 90.3% accuracy and outperforms the others in several ways.**

I. Introduction

Almost every major organization in the world relies on the Internet to stay connected with people inside and outside of their institutions. Consequently, large amounts of Internet traffic travels through these organizations daily, including malicious connections that can be very harmful. In an effort to improve the quality of protection, network analysts typically employ "misuse detection" approaches [1], [2], which use signatures to scan networks for known attacks [1]. For example, researchers recently used Apache Spark to create a parallel misuse detection approach for Bro logs [3]. However, misuse detection approaches cannot detect new types of attacks, and must be constantly updated with new rules and signatures [1].

Machine learning enables network operators to identify activity that is *similar* to previously seen traffic, without requiring a precise signature [2]. However, applying machine learning to network activity is difficult due to the lack of availability of real-world, labeled training data and the high cost of errors [2]. As a result, most prior work uses publicly-available (but synthetic) trace-route data such as NSL-KDD [4] or the 1999 DARPA IDS dataset [5]. For example, researchers proposed an enhanced Support-vector Machine (SVM) approach for network intrusion detection and tested their work on the DARPA IDS dataset [6]. Adentunmbi et al. [7] compared rough set and k Nearest Neighbors (kNN), Osareh et al. [8] compared neural network and SVM, and Meng [9] compared three different machine learning techniques on the KDD dataset. However, the use of these datasets have been criticized [2] due to their age and synthetic attack data. When researchers have used real-world data in their testing, they only tested a limited number of machine learning techniques on proprietary data [10], [11].

Network log data derived from monitoring tools such as Zeek (formerly Bro) [12] contain useful information that can help network analysts glean patterns in network activity. Zeek logs a variety of protocols (i.e. HTTP, SSH, DNS, Kerberos, SMTP, etc.). While some research has looked at using log data correlation to identify attacks [13], [14], our research uses only HTTP logs. Due to the difficulty in accessing real data, however, there is little prior work on applying machine learning techniques to HTTP log data; BBAC [15], [16] is a notable exception.

This paper compares the performance of eight unsupervised and supervised machine learning techniques on Zeek HTTP logs gathered from an enterprise network over a six-month time span. Our work is novel in several ways. First, it benchmarks performance of eight standard machine learning approaches on a common set of data and evaluates each method by runtime as well as accuracy. In contrast, most prior work evaluates a single or a very small number of machine learning techniques. Second, we run our analysis on real HTTP log data gathered from an enterprise network. Our results show that kNN yields the highest accuracy at 90.3%, and is the most resilient to the removal of the "Bot" user-agent string from the list of features.

The rest of the paper is as follows. Section II describes the classification methods studied and related work. Section III describes our experimental setup and results. Finally, we present our conclusions in Section IV.

II. Methods

A. Machine Learning Techniques

The eight model types included in our study are k-Means, kNN, Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes, Bernoulli Naive Bayes, Non-linear Support-vector Machine (NL-SVM), and Linear Support-vector Machine (L-SVM). All of the machine learning models used in this study were implemented with the Scikit-learn [17] Python library. The techniques can be divided into three groups: similarity-based algorithms, Naive Bayes algorithms, and Support-vector Machines. A description of each method and how it was implemented is given below.

1) Similarity-Based Algorithms: The k-Means machine learning technique is an unsupervised clustering algorithm in which k random points are used as starting points for the "centroids" of the k clusters. The features of the entries in the data set are used to plot each entry and associate them with the nearest centroid. Then the centroids are moved to the center of their assigned entries and the process repeats. When complete, all of the entries associated with each centroid constitute a cluster. We used Scikit-learn's built in `MiniBatchKMeans` to generate clusters. Since this technique is sensitive to the choice of value for the hyperparameter k, we used an elbow method analysis to determine that a k value of 10 worked best for our data. After finding the ten clusters, we assigned each cluster a classification of "normal" or "malicious" by determining the majority class in the cluster, according to their labels.

The kNN algorithm is similar to k-Means because it also determines similarity between events based on the proximity of the plotted data. Instead of using a pre-determined number of centroids, each unlabeled point is assigned a classification based on the majority class of its k nearest neighbors. Because we had two classes of interest, we chose $k = 3$ for our experiments to prevent ties in determining the majority class. We used Scikit-learn's `KNeighborsClassifier` for these trials.

Visualizing the similarity of entries can be informative when using similarity-based algorithms; however, because our data set has more than two or three features, it is difficult to plot our results in a usable way. To solve this problem we use principal component analysis to make our data two-dimensional for plotting. We used Scikit-learn's built in function `PCA` for this.

2) Naive Bayes Algorithms: Naive Bayes algorithms are a class of probabilistic classifiers that calculate the probability an entry belongings to a class, given its features, under the assumption that the features are conditionally independent. Each of the Naive Bayes algorithms that we used are very similar in the way that they calculate probability; however, each algorithm has certain strengths and weaknesses. The Gaussian Naive Bayes algorithm is better for classifying continuous data; multinomial, complement, and Bernoulli Naive Bayes algorithms are better for categorical data. The data set we used for this study has both categorical and continuous data in it which made each type of model a potential candidate. We used Scikit-learn's `GaussianNB`, `MultinomialNB`, `ComplementNB`, and `BernoulliNB` functions for these tests.

3) Support-vector Machines: Linear Support-vector Machines find the equation for a line that separates the entries in their training set into two classes. Given the choice between several such lines, SVMs select the line that maximizes the distance between the line and the data points on either side. Linear SVMs can be transformed into non-linear SVMs through the use of a kernel function that projects the feature space into higher dimensions. In this usage, the kernel function acts as a measurement of similarity between inputs. To create these models we used Scikit-learn's `SVC` and `LinearSVC` functions. We used the default radial basis function (RBF) kernel for the non-linear SVM.

B. Model Features

One of the most important parts of building any machine learning model is selecting the proper features to use. The data set we used for training and testing started with 58 features. We were able to remove several features by looking at how much variance there was within the feature-set and using our own domain knowledge about how important each feature was likely to be. After this initial selection process, we were left with 11 features. We did an additional round of analysis using a heat map based on the correlation between a given feature and the classification. This allowed us to identify and remove four more features that did not influence the classification. We were left with seven features: whether or not the entry had a cookie associated with it, the destination port of the entry, the request body length, the response body length, the URI length, the HTTP verb associated with the entry, and whether or not the user was a bot (referred to as "Bot" in this paper) based on its user agent string.

III. Experimental Setup & Results

The data set used for this experiment was derived from $1,461,238$ Zeek HTTP log events gathered from from an enterprise network over a six-month time span using the HACSAW [18] API. Of these, $312,511$ events were labeled malicious. To create a data set consisting of a $50-50$ split of normal and malicious data, an additional $312,511$ normal entries were selected at random to generate a final data set of $625,022$ entries sorted by time-stamp. Since log data is temporal, we processed data under the assumption that entries closer together in time were more similar to one another. As a result, we decided to train on events occurring earlier in time, and test on events that were more recent, as opposed to training/testing against random entries throughout the set. Thus, the the first 80% ($500,017$ events) of the data was used to train our models while the last 20% ($125,005$ events) of the data was used for testing.

The metrics that we used to evaluate our models were accuracy, recall, and precision:

$$accuracy = \frac{correctPredictions}{attemptedPredictions}$$

$$recall = \frac{correctlyPredicted_{Malicious}}{actualMalicious}$$

$$precision = \frac{correctlyPredicted_{malicious}}{totalPredicted_{malicious}}$$

Experimentation was done on an Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz and 32 GB of RAM. As an initial test, we measured the training times of each of the eight approaches. Results are shown in Table I. Times represent

TABLE I
RESULTS SORTED BY FASTEST TRAINING TIME (SECONDS)

Model	Training Time (seconds)
Gaussian NB	0.17
Multinomial NB	0.17
Bernoulli NB	0.19
Complement NB	0.19
k-Means	0.27
L-SVM	78.6
kNN	547
NL-SVM	13,978.6

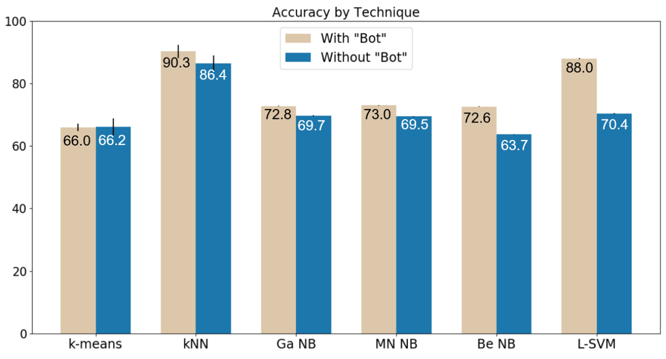

Fig. 1. Accuracy of each Technique

an average of 30 runs, except for the non-linear SVM (NL-SVM), due to its exceptionally long training time. Even without considering NL-SVM, we observed a large disparity in training time between the models. k-Means and the Naive-Bayes approaches took less than a second to train. The linear SVM (L-SVM) took a little over a minute. k-NN took under ten minutes to train. In contrast, the NL-SVM took 3.88 hours to train—180 times longer than the L-SVM. Preliminary testing showed that the accuracy obtained by the NL-SVM was 88.1%, compared to 88% achieved by the L-SVM. For this reason, we excluded NL-SVM from further analyses.

In our next set of experiments, we measured the accuracy, precision, and recall of the different approaches using the testing dataset against the models trained for each ML technique. Results are shown in Figures 1-3. Complement Naive Bayes test results are not shown because they yielded the exact same accuracy, recall, and precision results as the multinomial Naive Bayes results. We also trained an additional model for each technique without the "Bot" feature in order to determine the impact of that feature on performance.

Our results show that kNN produces models with the best accuracy and recall. The kNN model with the "Bot" feature obtained a recall of 89.0% and an accuracy of 90.3%. In other words, when a kNN model sees a malicious entry, it is able to predict that it is malicious 89.0% of the time. When trained without the "Bot" feature, every technique (save for k-Means) experienced a drop in accuracy of 3-18%. On the Without "Bot" set, kNN's recall is 85.6%, only a 4% drop from the testing with the "Bot" feature. In these tests, kNN appears to be highly resistant to the absence of the user-agent string.

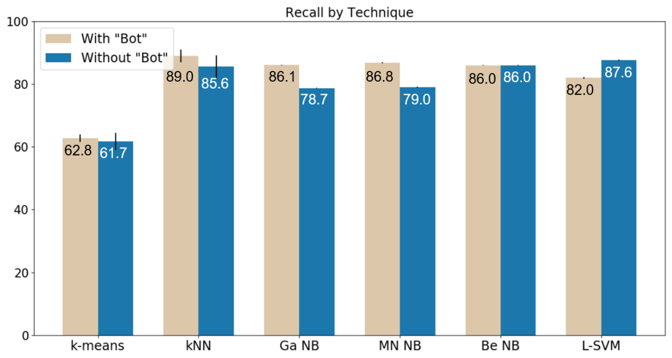

Fig. 2. Recall of each Technique

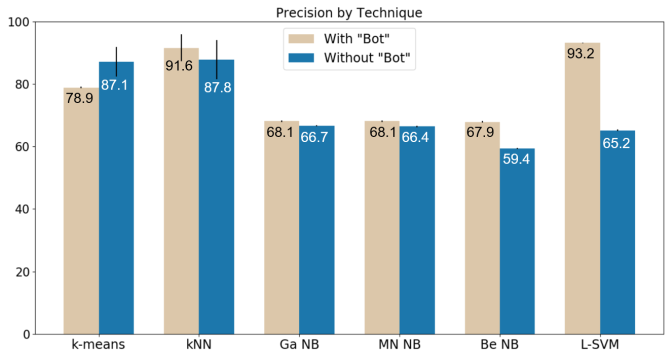

Fig. 3. Precision of each Technique

While k-Means models also appear to be highly resistant to the absence of the "Bot" feature, they are the worst performing. This is interesting because, while both kNN and k-Means classify examples based on similarity to proximate neighbors, they had a difference of about 24% in accuracy. This implies a high degree of intermixing between the classes that is indistinguishable when classifying an example according to a large cluster's centroid label rather than a smaller number of local neighbors. Figure 4 and Figure 5 show a PCA plot of how k-Means and kNN classify the training data. Green represents entries classified as 'normal' and red represents "malicious" data. The k-Means plot shows three perceivable clusters. In Figure 6 we can see where k-Means classified our test set incorrectly. Blue represents entries classified correctly and red represents incorrectly classified entries. We can see why k-Means performed so poorly as it struggled in areas where there was overlap.

IV. CONCLUSION

The most successful model from our data was the kNN model. The model trained with the "Bot" feature had the highest accuracy and recall. While this is good for our results, it may not translate well into real-world performance since user agent strings can be manipulated very easily by attackers; however, kNN also showed itself to be highly resistant to the removal of the "Bot" feature, while other ML techniques were not. Finally, the kNN model takes minutes to train. Speed

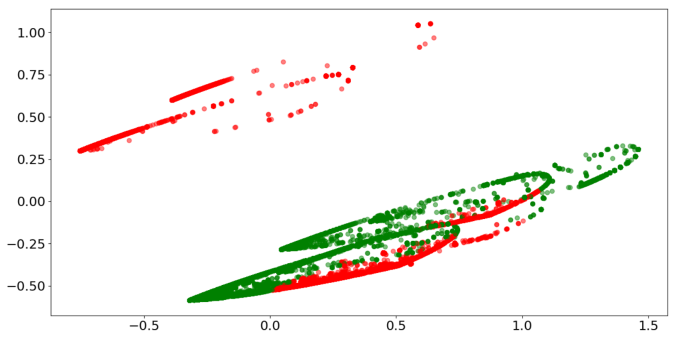

Fig. 4. PCA plot colored by k-Means classification (training data)

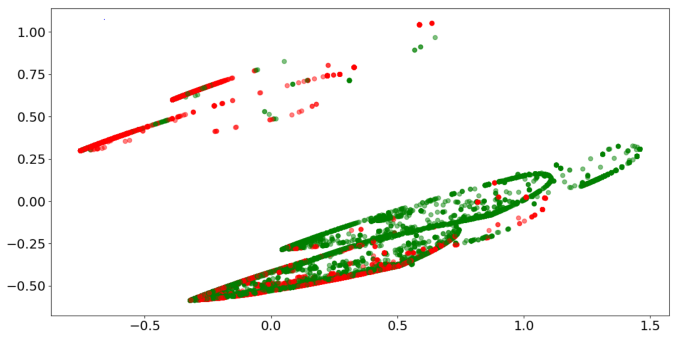

Fig. 5. PCA plot colored by kNN classification (training data)

of training is important to any practical application of these techniques, since deployment models are likely to be trained on much larger datasets.

If an organization intends to implement machine learning in their cybersecurity infrastructure, it needs to gather a large amount of labeled data and ensure that it is being labeled properly. In our research we found that there was high variability in the input data and that training data can be difficult to gather; however, our research shows that there is potential for machine learning to be implemented in network security. Future work could include more experimentation with features that might produce even better models.

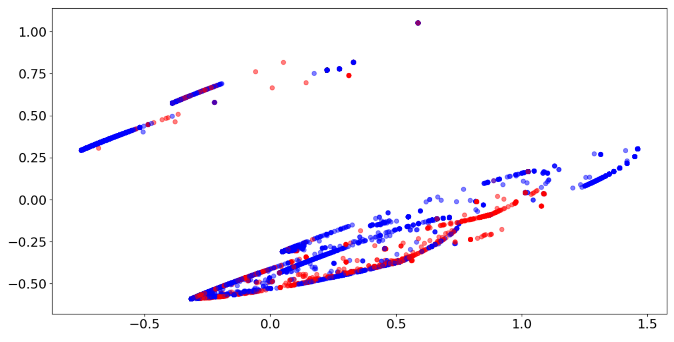

Fig. 6. PCA plot of k-means colored by correctness of classification (test data)

Acknowledgment

This work was supported in part by high performance computer time and resources from the DoD High Performance Computing Modernization Program. The opinions expressed in this work are solely of the authors and do not reflect those of the U.S. Military Academy, the U.S. Army, or the DoD.

References

[1] A. L. Buczak and E. Guven, “A survey of data mining and machine learning methods for cyber security intrusion detection,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[2] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 305–316.

[3] S. Deaton, D. Brownfield, L. Kosta, Z. Zhu, and S. J. Matthews, “Real-time regex matching with apache spark,” in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2017, pp. 1–6.

[4] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. IEEE, 2009, pp. 1–6.

[5] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, “The 1999 DARPA off-line intrusion detection evaluation,” *Computer networks*, vol. 34, no. 4, pp. 579–595, 2000.

[6] T. Shon and J. Moon, “A hybrid machine learning approach to network anomaly detection,” *Inf. Sci.*, vol. 177, no. 18, pp. 3799–3821, Sep. 2007. [Online]. Available: https://doi.org/10.1016/j.ins.2007.03.025

[7] A. O. Adetunmbi, S. O. Falaki, O. S. Adewale, and B. K. Alese, “Network intrusion detection based on rough set and k-nearest neighbour,” *International Journal of Computing and ICT Research*, vol. 2, 2008.

[8] A. Osareh and B. Shadgar, “Intrusion detection in computer networks based on machine learning algorithms,” *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, 2008.

[9] Y. Meng, “The practice on using machine learning for network anomaly intrusion detection,” in *2011 International Conference on Machine Learning and Cybernetics*, vol. 2, July 2011, pp. 576–581.

[10] T. Pietraszek and A. Tanner, “Data mining and machine learning-towards reducing false positives in intrusion detection,” *Inf. Secur. Tech. Rep.*, vol. 10, no. 3, pp. 169–183, Jan. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.istr.2005.07.001

[11] J. Stokes, J. Platt, J. Kravis, and M. Shilman, “ALADIN: Active learning of anomalies to detect intrusions,” Microsoft Reserarch, Tech. Rep. MSR-TR-2008-24, March 2008.

[12] V. Paxson. (2014) The Zeek network security monitor. [Online]. Available: https://www.zeek.org/

[13] B. Deokar and A. Hazarnis, “Intrusion detection system using log files and reinforcement learning,” *International Journal of Computer Applications*, vol. 45, no. 19, 2012.

[14] C. Abad, J. Taylor, C. Sengul, W. Yurcik, Y. Zhou, and K. Rowe, “Log correlation for intrusion detection: a proof of concept,” in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, Dec 2003, pp. 255–264.

[15] M. Mayhew, M. Atighetchi, A. Adler, and R. Greenstadt, “Use of machine learning in big data analytics for insider threat detection,” in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, Oct 2015, pp. 915–922.

[16] A. Adler, M. J. Mayhew, J. Cleveland, M. Atighetchi, and R. Greenstadt, “Using machine learning for behavior-based access control: Scalable anomaly detection on tcp connections and http requests,” in *MILCOM 2013 - 2013 IEEE Military Communications Conference*, Nov 2013, pp. 1880–1887.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] L. Leonard and W. Glodek, “HACSAW: A trusted framework for cyber situational awareness,” in *Proceedings of the 5th Annual Symposium and Bootcamp on Hot Topics in the Science of Security*, ser. HoTSoS ’18. New York, NY, USA: ACM, 2018, pp. 12:1–12:1. [Online]. Available: http://doi.acm.org/10.1145/3190619.3190641