12-9-2019

Intelligent Feature Engineering for Cybersecurity

Paul Maxwell
Army Cyber Institute

Elie Alhajjar
Army Cyber Institute

Nathaniel D. Bastian
Army Cyber Institute

Recommended Citation

P. Maxwell, E. Alhajjar and N. D. Bastian, "Intelligent Feature Engineering for Cybersecurity," 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 5005-5011, doi: 10.1109/BigData47090.2019.9006122

Intelligent Feature Engineering for Cybersecurity

Paul Maxwell
Army Cyber Institute
United States Military Academy
West Point, NY
paul.maxwell@westpoint.edu

Elie Alhajjar
Army Cyber Institute
United States Military Academy
West Point, NY
elie.alhajjar@westpoint.edu

Nathaniel D. Bastian
Army Cyber Institute
United States Military Academy
West Point, NY
nathaniel.bastian@westpoint.edu

***Abstract*— Feature engineering and selection is a critical step in the implementation of any machine learning system. In application areas such as intrusion detection for cybersecurity, this task is made more complicated by the diverse data types and ranges presented in both raw data packets and derived data fields. Additionally, the time and context specific nature of the data requires domain expertise to properly engineer the features while minimizing any potential information loss. Many previous efforts in this area naively apply techniques for feature engineering that are successful in image recognition applications. In this work, we use network packet dataflows from the Defense Research and Engineering Network (DREN) and the Engineer Research and Development Center's (ERDC) high performance computing systems to experimentally analyze various methods of feature engineering. The results of this research provide insight on the suitability of the features for machine learning based cybersecurity applications.**

Keywords—artificial intelligence, machine learning, feature engineering, cybersecurity

I. INTRODUCTION

A key function of network security includes continually monitoring the computer network to detect when unauthorized and unauthenticated access to secure information has occurred. The monitoring of network traffic and host logs may find evidence of malicious activity (i.e., an attack) that can then be raised as a network security incident. Malicious actors pose considerable threats against these computer networks by their continued attempts to exploit potential vulnerabilities. To detect malicious activity in near real-time, intrusion detection systems (IDS) are commonly used to alert cybersecurity analysts who then take appropriate action against the alerts. Two broad classes of intrusion detection methods include knowledge-based detection (e.g., use of signatures to match traffic to a list of known-malicious byte patterns) and behavior-based detection (e.g., flagging traffic that deviates from "normal" as anomalous, with the assumption it is then malicious) [1]. Regardless of the method used, analysts are frequently overwhelmed by the deluge of alerts (both false and true positives) and suffer from alert fatigue. As a result, machine learning is applied to the alert stream in hopes of reducing analyst workload through removal of false positives in the data stream.

More recently, machine learning (ML) is a data-driven artificial intelligence paradigm that has been used to improve upon the limitations of these intrusion detection methods, as well as in other cybersecurity applications. One of the main difficulties in applying ML to the intrusion detection problem is that network traffic cannot be used directly as ML algorithm input. Feature vectors constructed/engineered from the network traffic must be used. Thus, feature engineering (extraction/encoding) and selection is a critical task to use ML for intrusion detection. This task is complicated by the diverse data types and ranges presented in both raw data packets and derived data packets [2]. Additionally, the time and context specific nature of the data requires domain expertise to properly engineer the features while minimizing any potential information loss.

The aim of this research is to identify experimentally which feature engineering techniques work best for cybersecurity related data, particularly network flow data. Due to the high-dimensional feature space of feature engineered network traffic data, high performance computing systems are employed for the experimental design and execution. The current literature relies on anecdotal, best practices-based methods for this domain, whereas quantitative results over a large test framework will improve the state-of-the-art for the field. This work does not attempt to solve the challenge of feature selection which uses the product of our work as its input. The remainder of the paper is organized as follows: a review of related works is examined in section II, whereas section III details feature engineering experimentation. Results are presented in section IV, and conclusions, limitations, and future work are described in section V.

II. RELATED WORKS

To effectively apply Machine Learning for intrusion detection or other network security applications, feature engineering and selection serves as an essential data preprocessing step to transform network traffic data into useful feature vectors that optimize detection performance. Typically, a combination of domain knowledge and automated methods is used to clean, engineer, reduce, and select the most useful features.

In fields other than intrusion detection, some work has been done to compare engineering techniques, though most work focuses on feature selection. Work done in [3] examines the effects of various transformation techniques on numerical features. The authors define a feature representation named the *Quantile Sketch Array* that enables their system to predict the ability of a feature transformation to improve classification accuracy. The work performed in [4] examines different engineering techniques to improve fraud detection in credit card environments. Here they experiment with a variety of methods that incorporate user behavior using variable time windows and other methods. Both studies differ from our work in the domain and the types of data that are engineered.

In terms of data cleaning and engineering, several studies have performed normalization, encoding and other transformation techniques (indicator variables, conditional probabilities, etc.) to convert network traffic attributes into data that is useable in machine learning algorithms [5]–[10].

In [5], [7], [8] the authors use Z-score standardization on numeric values in the KDD'99 dataset. For discrete and categorical data, [5] and [2] use one-hot encoding. The challenge with this method is feature explosion resulting in large input matrices. The authors in [5] and [6] use frequency information about the data to encode these values. This results in more compact feature sets but relies heavily on the quality of the underlying samples in comparison to the population. Davis [2] extracts embedded features from string data transforming it into three values: string length, information entropy, and unigrams (frequency vector of ASCII values). In all of these works, no other engineering techniques are considered and thus, the appropriateness of them is not evaluated.

The authors in [9] and [11] do a limited experiment on feature engineering methods for specific fields using the KDD'99 dataset. The work in [9] compares three engineering techniques to an 'arbitrary' technique that is equivalent to label encoding. They use indicator variables (one-hot encoding), conditional probabilities (N-dimensional probability vector), and a separability split value (tree-based method) for comparisons. They show that all three of the techniques are more accurate than the one-hot technique but do not show the superiority of any of the three. Their work is limited in comparison to ours in that they only experiment on three categorical fields: *protocol*, *service*, and *flag*.

The authors in [11] perform a comparison of engineering techniques on IPv4 addresses. Here, they compare a 32-bit vector representation, a 4-octet representation, and an extended-octet representation (method where cross-octet information is incorporated) in a machine learning application. They use a self-created dataset of benign and malicious websites to test their methods. They do not compare their methods against standard techniques such as one-hot encoding nor do they incorporate other features into their malicious website detection system.

Experiments were done in [12] to determine how ten numerical engineering techniques (e.g., counts, logarithms, square roots, and polynomials) effect the accuracy of four classifiers. The results indicated that neural networks and SVM classifiers benefited from certain techniques and the gradient boosting algorithms and Random Forest algorithms benefited from similar techniques. Our work differs from this research through the expansion to categorical and textual features and our use of a novel dataset.

Work beyond feature engineering has been done by many to perform feature selection. Principal component analysis has been commonly used for data reduction to alleviate the curse of dimensionality [6], [10], [13], [14]. In terms of feature selection, for example, Li et al. [15] used a modified random mutation hill climbing algorithm, whereas Chebrolu et al. [16] used a Markov blanket model. Feature selection is an important component of machine learning systems; however, this work relies on well-engineered features to produce reliable results.

III. Feature engineering experiments

The data used in this work was collected from real network traffic on the U.S. Department of Defense (DoD) Defense Research and Engineering Network (DREN) using the compute resources available at the U.S. Army Corps of Engineers' Engineer Research and Development Center (ERDC). ERDC's HACSAW API allows researchers to query the Cybersecurity Environment for Detection, Analysis, and Reporting (CEDAR) [17] database containing processed dataflows of DREN traffic to include live traffic.

The database was queried to obtain 250,000 unique entries for our training set and 25,000 unique entries for our test set from *http* alert data. *HTTP* traffic was chosen due to the prevalence of this type of traffic in the database. The alert data entries are those marked as entries requiring expert review by the network's Zeek (formerly Bro) intrusion detection system [18]. Each entry contains 68 fields reported by the Zeek tool and includes a label (*normal* versus *bad*) created by the subject matter expert who reviewed the alert. This raw data was processed to drop 52 of the fields (administrative in nature such as *notes*, *owner*) leaving 16 fields per entry. Table I contains the fields and descriptions remaining in the data sets.

Using CEDAR's JupyterLab notebook interface, the data set was feature engineered to create a baseline solution to use for comparison with other engineering techniques. The methods chosen to engineer the baseline solution were chosen to model commonly used techniques such as those found in [17]. For the fields *connection*, *country codes*, *severity*, and *method*, scikit-learn's [19] label encoding was used to create unique code identifiers for each of the entries in those fields. Integer fields (request and response body lengths) were scaled using a MinMaxScaler to the range [0,1]. The *origin host*, *response host* and *port* fields were converted to bit vectors similar to the technique in [11] to keep the number of feature columns small, maintain information contained in the associated data structure, and handle values not seen in the training set. The *port* fields were converted to a 16-bit value to contain all possible values 0-65535. The *origin host* and *response host* fields were converted to 128-bit fields to handle the IPv4 and IPv6 addresses found in the data set. The *origin org* and *response org* fields were binarized to indicate either *internal* or *external* traffic. The *user agent* field was parsed to extract the operating system (os) and browser. These two new fields replaced the *user agent* string in the data set after label encoding the resulting data. The *host* field was binned into 11 frequently used top-level domains (e.g., .mil, .com, .edu) and an *other* category for low frequency domains (e.g., .bit, .info, .ua). These bins were then label encoded in the final data set.

Next, we considered different methods of feature engineering the fields of the data set. In each experimental variation, only one field was altered to isolate the effects of the change compared to the baseline solution. The variations were not compared exhaustively due to the large number of perturbations and the resulting computing resources that would require. For the *user agent* field, we tested three alternate engineering methods. We examined two alternate engineering techniques for the *connection*, *country code*, *origin/response host*, and *port* fields. For the *http method*, *body length*, *severity*, and *origin/response org* fields we used one alternate method. The number of variations examined depended on the nature of the data and how it could be manipulated without losing information.

Table I. HTTP Alert Data Set Fields

Field	Example Data
connection	open, close, none
host	my.connection.edu
origin host, response host	192.168.0.1
origin host country code, response host country code	US
origin host org	external, internal
origin port, response port	12345
response host org	ca.hostconn-0031.host.edu
method	GET, POST
request body length, response body length	10
user_agent	Mozilla/5.0[en](X11,U;OpenVAS-VT 8.0.9)
severity	H, M, L
status (i.e., label)	normal, bad

The *user agent* field had the most opportunity to experiment with feature engineering techniques given the information contained in its text strings. The first technique used was to binarize the field into entries with the term ‘bot’ and those without. The resulting values were then one-hot encoded resulting in two output features. The premise is that http traffic generated by a bot may be more likely to be malicious. The next technique was to binarize entries based upon the browser used in the traffic. Traffic using a commercial browser in the set [Chrome, IE, Firefox, Safari, Edge, Android, Mobile Safari] were grouped while all other browsers were placed an ‘other’ group. The resulting field was then made one-hot to create two features. Finally, the last technique used one-hot encoding on the extracted operating system field (e.g., Windows, Mac OS X, iOS) that resulted in 12 new feature columns.

For the *connection* field, the experimental techniques used were one-hot encoding and binarization. The increase in the number of data set columns with one-hot encoding is limited due to the fixed size of the data in that field and is thus not onerous (columns increased by 15). The other technique binarized the connections into *NONE* versus all others. This presumed that connections with a *NONE* status were malformed and therefore indicative of malicious behavior.

The *origin* and *response country code* field used one-hot encoding and binning as alternative engineering methods. The one-hot encoding resulted in 178 new feature columns. This is an increase of 59% in the number of features. Certainly, the number of countries is relatively stable and fixed so feature growth is not a huge concern. The other technique used first binned the entries into one of the 17 most frequently used codes based on the dataset or into an ‘*other’* bin. The results were then encoded using one-hot encoding to create 18 new feature columns. This technique was a compromise between information contained in the country code and the number of columns introduced in the data set.

For the *origin* and *response host* fields, experiments were done using one-hot encoding and creating sixteen 8-bit integers. One-hot encoding of the IP addresses is a common technique but suffers from feature explosion and a resulting high-dimensionality problem. IPv4 addresses have over 4 billion possible values and thus the ability to model all of them is limited. Additionally, addresses not seen during model training cause problems with predictions by the trained model. IPv6 addresses complicate this further given their 2^{28} possible values. An alternative method proposed in [11] treats each ‘quad’ or octet of the address as an integer or in the case of IPv6 addresses, two integers. Using this method, each IP address is parsed into sixteen 8-bit integers [0,255]. This method uses only 32 features compared to the 256 required in the baseline encoding while maintaining some network information contained in the IP address structure.

The *port* field was altered using one-hot encoding and binning. Similar to the IP data, one-hot encoding can expand the number of columns dramatically. In this case, the number of features grew to 48,306 from 300. This is problematic as it caused one of the classifier algorithms to not converge and predictions based on unseen ports would be problematic. The other technique used binned the *origin port* into four bins. The bin cutoffs (frequency based) were: 0.0, 0.30, 0.50, 0.75, and 1.0. This resulted in port numbers binned with the following cutoffs: 0, 38534, 48201, 55940, and 65535. The output feature indicated in which bin the port was located. The *response port* was binarized to indicate either port 80 or not. Port 80 is the expected response port as the data set is comprised of http traffic.

The *http method*, *severity*, and *origin/response org* fields all used one-hot encoding for their alternate engineering method. The *http method* and *severity* fields are limited in their allowed values and therefore the growth in feature columns is small. The *origin org* field is limited to two values (*internal*, *external*) and thus converts easily to one-hot encoding. The *response org* field is more variable and thus results in a large growth of output feature columns once one-hot encoding is applied based upon the data set used for training. For our experiments, the number of columns expanded by 228 after the encoding.

The last alternate engineering technique applied was to the *body length* field. To determine if large outliers in the data were affecting the results, the scikit-learn [19] RobustScaler was applied to the data without centering. This method removes the median from the data and scales the data according to the quantile range.

IV. RESULTS

The project baseline and all the alternate engineering techniques were evaluated against multiple classifiers. All of the classifiers used are from scikit-learn’s API. Table II shows the classifiers and any non-default settings utilized. The Voting Ensemble classifier uses the Random Forest, Multi-layer

Perceptron, Complement Naïve Bayes, Gradient Boosting, Logistic Regression, and Linear Discriminant Analysis classifiers for its sub-models.

Table II. Scikit-Learn Classifiers Used for Evaluation

Classifier	Non-default Settings
Random Forrest (RF)	n_estimators=300, criterion=entropy
Multi-layer Perceptron (MLP)	
Complement Naïve Bayes – (CNB)	
Gradient Boosting (GB)	
Logistic Regression (LR)	
Linear Discriminant Analysis (LDA)	solver='lsqr', shrinkage='auto'
Quadratic Discriminant Analysis (QDA)	reg_param=0.7
Gaussian Bayes (GaB)	
Bagging (Bag)	
Decision Tree (DT)	
Voting Ensemble (VE)	voting='soft'

As discussed in the previous section, the choice of feature engineering technique can affect the number of output features in the data set. Table III shows the relationship between the technique and the number of features. Of note, some of the classifiers did not converge during training due to the large number of features (ex. *port* one-hot encoding).

Given these results, we used the Voting Ensemble Classifier accuracy results to select the 'best' technique for each of the features. The techniques used were: connection binarize/one-hot; host top-level domain; http method one-hot; port binning; severity one-hot; user-agent 'bot' detection binary/one-hot; origin/response host sixteen x 8-bit ints; and org one-hot. For the remaining features, the techniques in the project baseline were used (country code label encoding; length MinMaxScaler). The results of this '*Best Technique*' method is shown by the line plot in Figure 1. Though the '*Best Technique*" method generally resulted in better accuracy than average, it is not supreme. Of note though, this method dramatically improved the accuracy of the QDA, Bagging, and DT classifiers.

Figure 1 shows the classification accuracy results of the experimental feature engineering techniques for the eleven classifiers examined. The distribution of the data reveals that many of the classifiers perform similarly across the range of feature engineering techniques evaluated. No one technique or classifier resulted in supremacy. The best engineering method varied by classifier.

From these results, we identified that certain classifiers perform poorly for this type of data. Except for the '*Best Technique*' method, the average accuracy of the Quadratic Discriminant Analysis, Gaussian Bayes, Bagging, and Decision Tree classifiers is 23.1% lower than the remaining classifiers. As a result, it is recommended that IDS developers do not use these classifiers for their products.

Figure 2 shows how each feature engineering technique performed against the eleven classifiers in terms of accuracy. As shown, the performance is similar with the best accuracies ranging between 72% and 83%. The exception is the '*Best Technique*' whose distribution across the classifiers is tighter and has a higher average accuracy.

The execution times for each of the classifiers except for the MLP classifier is fairly uniform except when the number of feature columns grows rapidly. As expected, the predictors run much slower when the large number of features encountered in the one-hot encoding of the port and origin/response host fields. The execution times of the classifiers are shown in Figure 3.

Table III. Number of Features by Engineering Technique

Feature	Engineering Technique	Number of Features
	Baseline	300
Connection	binarize	300
	one-hot	315
Country Code	binning	334
	one-hot	478
Host	top-level domain	300
	one-hot	4288
Http method	one-hot	306
Body length	robust scaler	300
Port	one-hot	48306
	binning	270
Severity	one-hot	303
User Agent	os one-hot	312
	browser binarize/one-hot	301
	'bot' detection binary/one-hot	300
origin/response host (IP addr)	Sixteen x 8-bit integers	108
	one-hot	8318
Org	one-hot	528

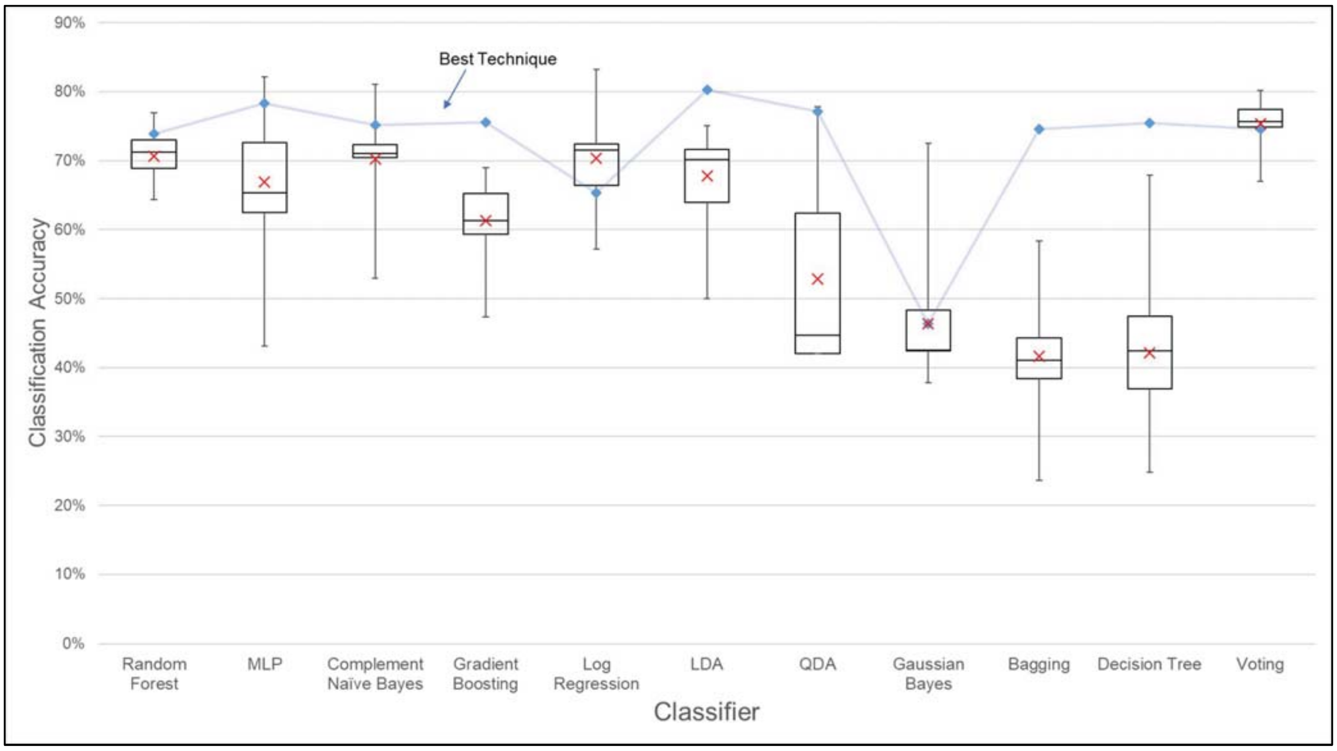

Figure 1. Classifier Average Accuracy Across All Engineering Techniques.

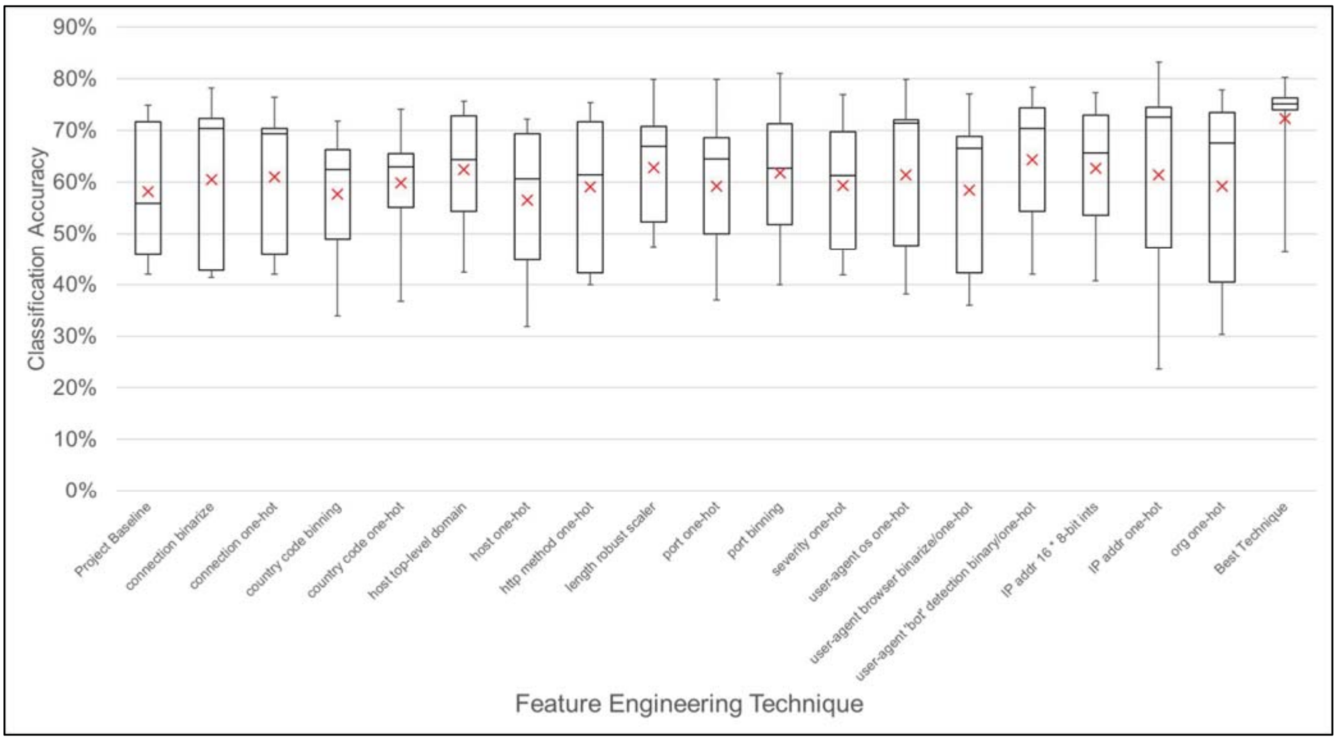

Figure 2. Feature Engineering Techniques versus the Classification Accuracy of Experimental Classifiers.

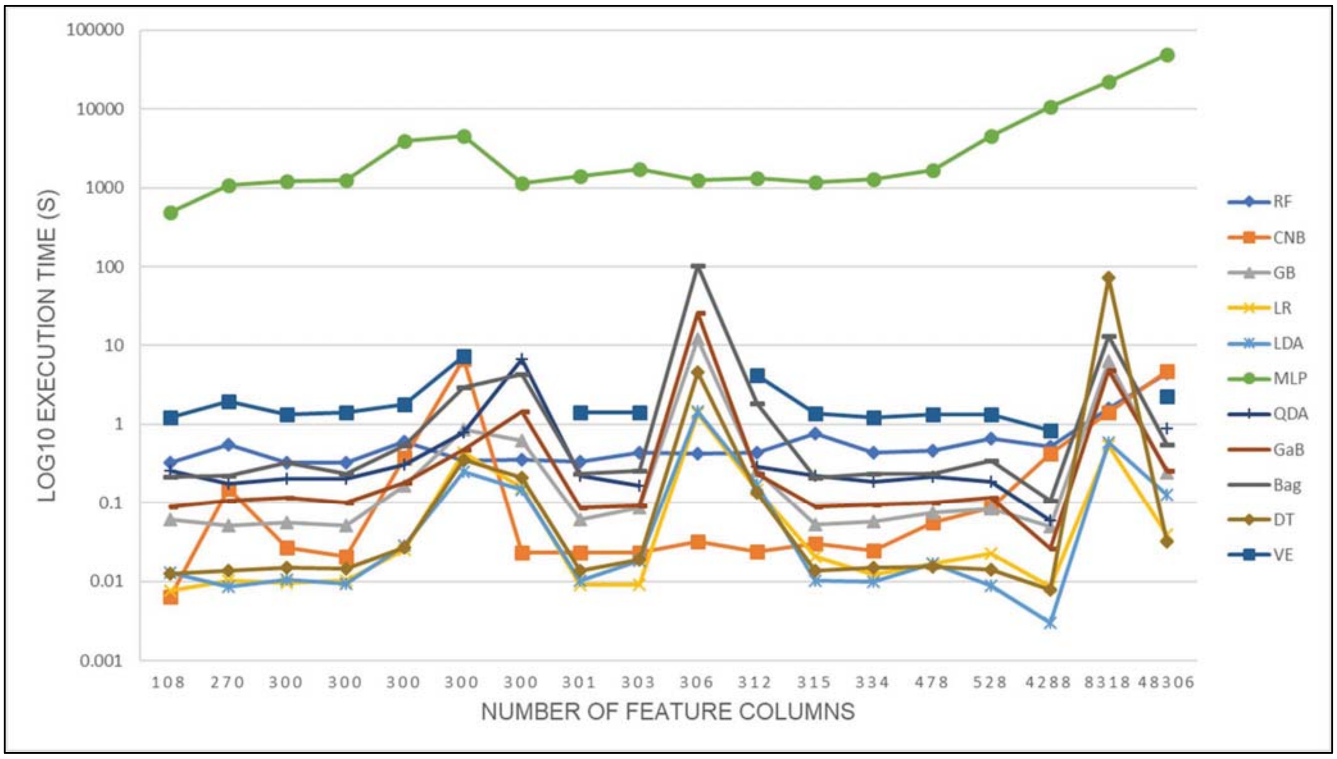

Figure 3. Classifier Prediction Execution Times versus Number of Features

V. CONCLUSIONS

Feature engineering network datasets is challenging. Methods used in the image recognition and other common domains do not readily transfer to this domain. In this work, we presented experimental results for a variety of feature engineering techniques against a variety of ML-based classifiers. Though a combination of 'best techniques' resulted in good performance against the experimental techniques, no black box type feature engineering solution was discovered that improves the detection rate of malicious traffic.

Future work in this area includes testing these techniques against other datasets and other traffic flows in this dataset (e.g., dns, connection). Additionally, it would be valuable to explore feature engineering techniques against the raw network data to evaluate its effectiveness without the reliance on the underlying IDS and netflow algorithm. Finally, comparative work using unsupervised learning techniques on this data would be valuable to understand how to engineer features that have large ranges such as IP addresses.

ACKNOWLEDGMENT

The authors would like to thank R. Agrawal and C. Lorenzen for their assistance and feedback on this work.

REFERENCES

[1] H. Debar, M. Dacier, and A. Wespi, "Towards a Taxonomy of Intrusion Detection Systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[2] J. Davis, "Machine Learning and Feature Engineering for Computer Network Security," Queensland University of Technology, 2017.

[3] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga, "Learning Feature Engineering for Classification," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia, 2017, pp. 2529–2535.

[4] A. Correa Bahnsen, D. Aouada, A. Stojanovic, and B. Ottersten, "Feature engineering strategies for credit card fraud detection," *Expert Systems with Applications*, vol. 51, pp. 134–142, Jun. 2016.

[5] P. Laskov, P. Dussel, C. Schafer, and K. Rieck, "Learning intrusion detection: supervised or unsupervised?," *Image Analysis and Processing (ICIAP 2005)*, pp. 50–57, 2005.

[6] Y. Bouzida, F. Cuppens, N. Cuppens-Boulahia, and S. Gombault, "Efficient intrusion detection using principal component analysis.," *Proceedings of the Third Conference on Security and Architectures Research*, 2004.

[7] Y. Li and L. Guo, "An active learning based tcm-knn algorithm for supervised network intrusion detection.," *Computers & Security*, vol. 26, no. 7–8, pp. 459–467, 2007.

[8] Y. Li, B. Fang, and Y. Chen, "Network anomaly detection based on tcm-knn algorithm.," in *Proceedings of the 2nd ACM symposium on Information, Computer and Communications Security*, 2007.

[9] E. Hernandez-Pereira, J. A. Suarez-Romero, O. Fontenla-Romero, and A. Alonso-Betanzos, “Conversion methods for symbolic features: A comparison applied to an intrusion detection problem,” *Expert Systems with Applications*, vol. 36, no. 7, pp. 10612–10617, 2009.

[10] X. Xu, “Adaptive intrusion detection based on machine learning: feature extraction, classifier construction and sequential pattern prediction.,” *International Journal of Web Services Practices*, vol. 2, no. 1–2, pp. 49–58, 2006.

[11] D. Chiba, K. Tobe, T. Mori, and S. Goto, “Detecting Malicious Websites by Learning IP Address Features,” in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, Izmir, Turkey, 2012, pp. 29–39.

[12] J. Heaton, “An Empirical Analysis of Feature Engineering for Predictive Modeling,” in *Proceedings of the IEEE SoutheastCon 2016*, Norfolk, VA, 2016, p. 6.

[13] W. Wang and R. Battiti, “Identifying intrusions in computer networks with principal component analysis.,” in *The First International Conference on Availability, Reliability and Security (ARES)*, 2006.

[14] M. L. Shyu, S. C. Chen, K. Sarinnapakorn, and L. W. Chang, “A novel anomaly detection scheme based on principal component classier.,” in *Proceedings of the IEEE Foundations and New Directions of Data Mining Workshop*, 2003.

[15] Y. Li, J. Wang, Z. Tian, T. Lu, and C. Young, “Building lightweight intrusion detection system using wrapper-based feature selection mechanisms.,” *Computers & Security*, vol. 28, no. 6, pp. 466–475, 2009.

[16] S. Chebrolu, A. Abraham, and J. P. Thomas, “Feature deduction and ensemble design of intrusion detection systems.,” *Computers & Security*, vol. 24, no. 4, pp. 295–307, 2005.

[17] C. Lorenzen, R. Agrawal, and J. King, “Determining Viability of Deep Learning on Cybersecurity Log Analytics,” in *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, 2018, pp. 4806–4811.

[18] “The Zeek Network Security Monitor.” [Online]. Available: https://www.zeek.org/. [Accessed: 26-Aug-2019].

[19] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *Machine Learning in Python*, p. 6.