Automated Feature Selection for Anomaly Detection in Network Traffic Data

MAKIYA NAKASHIMA, Texas A&M University-Commerce
ALEX SIM, Lawrence Berkeley National Laboratory
YOUNGSOO KIM, ETRI
JONGHYUN KIM, ETRI
JINOH KIM*, Texas A&M University-Commerce and Lawrence Berkeley National Laboratory

Variable selection (also known as *feature selection*) is essential to optimize the learning complexity by prioritizing features, particularly for a massive, high dimensional dataset like network traffic data. In reality, however, it is not an easy task to effectively perform the feature selection despite the availability of the existing selection techniques. From our initial experiments, we observed that the existing selection techniques produce different sets of features even under the same condition (e.g., a static size for the resulted set). In addition, individual selection techniques perform inconsistently, sometimes showing better performance but sometimes worse than others, thereby simply relying on one of them would be risky for building models using the selected features. More critically, it is demanding to automate the selection process since it requires laborious efforts with intensive analysis by a group of experts otherwise. In this paper, we explore challenges in the automated feature selection with the application of network anomaly detection. We first present our ensemble approach that benefits from the existing feature selection techniques by incorporating them, and one of the proposed ensemble techniques based on greedy search works highly consistently showing comparable results to the existing techniques. We also address the problem of when to stop to finalize the feature elimination process and present a set of methods designed to determine the number of features for the reduced feature set. Our experimental results conducted with two recent network datasets show that the identified feature sets by the presented ensemble and stopping methods consistently yield comparable performance with a smaller number of features to conventional selection techniques.

Additional Key Words and Phrases: feature selection, ensemble approach, network anomaly detection, cybersecurity analytics

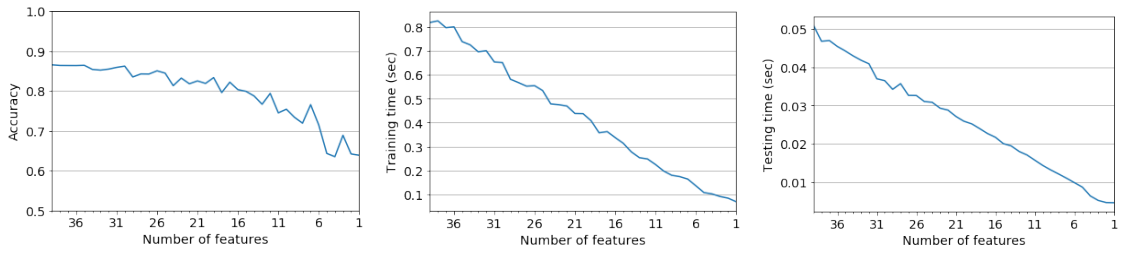
ACM Reference Format:
Makiya Nakashima, Alex Sim, Youngsoo Kim, Jonghyun Kim, and Jinoh Kim. 2018. Automated Feature Selection for Anomaly Detection in Network Traffic Data . 1, 1 (November 2018), 27 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The rapid advance in the machine learning industry has led to the active adoption of the learning techniques for a diverse range of applications. The network security area is not an exception, and machine learning is widely employed to analyze network traffic. For example, clustering has been utilized to detect network anomalies [1], to understand temporal

*Corresponding author

Authors' addresses: Makiya Nakashima, mnakashima@leomail.tamuc.edu, Texas A&M University-Commerce, Commerce, TX, 75428; Alex Sim, asim@lbl.gov, Lawrence Berkeley National Laboratory, Berkeley, CA, 94720; Youngsoo Kim, blitzkrieg@etri.re.kr, ETRI, Daejeon, Korea, 34129; Jonghyun Kim, jhk@etri.re.kr, ETRI, Daejeon, Korea, 34129; Jinoh Kim, jinoh.kim@tamuc.edu, Texas A&M University-Commerce and Lawrence Berkeley National Laboratory , Commerce, TX, 75428.



(a) Accuracy (b) Training time (c) Testing time

Fig. 1. Accuracy and timing over the number of features (classifier=LR, dataset=UNSW-NB15): Our preliminary experimental result shows that training and testing times decrease almost linearly over the number of features, while accuracy shows a flat pattern showing better performance with 7–20 features than the use of the entire features.

variation patterns [2], and to infer the class of individual connections for the purpose of labeling [3]. Various classification techniques such as Gradient Boosting (GB), Logistic Regression (LR), and Deep Neural Networks (DNNs) have also been employed mainly for detecting intrusive and/or anomalous events by analyzing network traffic [4–6].

Network datasets are often high-dimensional with multiple features (e.g., 49 features for UNSW-NB15 [7] and 85 features for IDS-2017 [8], which will be described in Section in 3.1.1). Many of the previous studies simply utilized the entire features provisioned in the data files. However, relying on all the features does *not* always lead to the best performance (e.g. detection rate); that is, a certain subset of features may yield an approximated or even better performance [9]. In addition, an invariant is that relying on a higher number of features should impose an increasing complexity, thus requiring a greater amount of time and resources for analyzing. **Fig. 1 shows an example of detection accuracy, training time, and testing time over different numbers of features with an LR classifier against UNSW-NB15[1]. From the figure, training and testing times decrease almost *linearly* with a smaller number of features used in the analysis. Accuracy also decreases over iterations but it shows more a gradual pattern than training and testing times. In this experiment, one feature was randomly selected and removed at each iteration until only a single feature is left. The results were averaged over multiple runs. We also calculated 95% confidence intervals and observed the intervals are quite negligible over iterations; the max value is 0.0025 (accuracy), 0.0016 (training time), and 0.00015 (testing time).**

The feature selection has been an active research topic over the past decade [10–16], and identifies relatively more important features by eliminating less essential ones. Alternatively, dimensionality reduction tools such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) project the given features into a lower-dimensional space. In contrast, feature selection preserves the definitions of the individual features but prioritizes them based on their importance, which may be beneficial for certain applications since it does not require any transformation in the analysis process. After identifying a subset of features, feature selection does not need any batch processing unlike PCA and t-SNE, which helps analyze streaming data such as network traffic traces. However, the actual use of a selection technique would *not* be straightforward, requiring laborious efforts by experts to determine a set of core features. In this work, we investigate a way to automate the feature selection process for network anomaly detection as an application use case. **To the best of our knowledge, this work is the first comprehensive exploration**

[1]We utilize the popular UNSW-NB15 dataset since it has been collected in 2015. The description of the dataset employed in this paper will be provided in Section 3.1.1.

and evaluation of automated feature selection methods for network traffic data. which enables the search of a subset set of features that maximizes the scalability of analysis by reducing the learning complexity but with a negligible performance loss (i.e., detection rate for anomaly detection).

While there can be several technical challenges, we consider the following two challenges to realize the automation:

Identifying core features. The most crucial question for feature selection would be how to discover a subset of features that approximates to the performance obtained with the entire features. **From our initial experiments with a set of existing selection techniques, we observed that each technique works consistently producing the same feature set as the result for a given dataset. However, individual techniques produce different sets of features even under the same condition (e.g., a static size for the resulted set), as will be discussed in Section 3.2 in detail.** In addition, an existing technique sometimes shows a better performance but sometimes it is worse than others, thereby simply relying on one of them would result in an inconsistent result. In this paper, we propose an *ensemble* approach that benefits from the existing feature selection techniques by incorporating them. We present two ensemble techniques: *heuristic selection* based on set theory and *greedy selection* based on greedy search. Our experimental results show that the greedy method works highly consistently, showing comparable results to the existing techniques, while the set-based heuristics largely works well but slightly less consistent sometimes.

Setting up the stopping condition of the feature elimination process. To enable the automation, it is essential to define a "good" termination condition to stop the elimination of features in the iterative process for feature selection. Many feature selection methods rely on a threshold or the final number of features to reduce. For instance, Sequential Feature Selection (SFS) eliminates less important features one by one at each iteration until it meets the final number of features specified. A critical problem here is how to determine the final number of features or the value for the threshold to stop. In this work, we set up and examine a set of methods that determine the number of features for the reduced feature set: (1) *MaxDelta* stops at iteration i that results in the largest decrease in terms of cross-validation score over the iterative process, (2) *MinPerfReq* stops as soon as the cross-validation score does not meet the specified requirement, and (3) *MaxScore* works based on a score function we developed, which penalizes a subset with a greater number of features. By combining those methods with the ensemble techniques, we show that our automated feature selection method can identify the core features, and it is possible to approximate the performance to the one with the entire features (i.e., without performing feature selection).

As mentioned, our primary interest lies in the feature selection for network anomaly detection. For developing relevant methods to address the challenges above, there would be a design choice whether or not utilizing domain-specific knowledge. In this work, we take the same approach as conventional feature selection methods that do not require any prior knowledge on the details of the dataset in analysis. In fact, even public network datasets have different formats and semantics. Our presented methods do not require in-depth understanding of the dataset without any laborious efforts by skilled experts. With the focus on the network anomaly detection, however, we would like to examine our proposed methods with different network datasets, including UNSW-NB15 [7], IDS-2017 [8], and NSL-KDD [17]. As reported in the later section of the paper, our automated methods address the above critical challenges effectively, yielding comparable performance with a much smaller number of identified features to traditional selection techniques.

This paper is organized as follows. Section 2 provides a summary of feature selection in terms of its classes and well-known techniques. Section 3 presents the experimental setting used in this study and discusses the preliminary

observations that motivate us to work on this topic. In section 4, we introduce our ensemble approach that incorporates multiple selection methods used in practice, while Section 5 discusses the question of when to stop the elimination process with three methods we propose in this study. We basically utilize the latest datasets (UNSW-NB15 and IDS-2017) for the experimentation; additionally, we will examine our proposed methods with NSL-KDD in Section 6, which has been widely used in the network anomaly detection research although quite old. We provide a description of closely related studies in Section 7 and finally conclude our presentation in Section 8 with a summary and future directions.

2 FEATURE SELECTION CLASSES AND METHODS

In this section, we introduce traditional feature selection classes and methods commonly applied in many application domains. Feature selection is often used for improving accuracy and reducing computational costs. There are three categories for classifying feature selection methods: filter method, wrapper method, and embedded method. **The basic definition and function of these methods can be found in [18] with their characteristics and limitations.**

The filter method selects features based on the intrinsic properties of features [19]. Using this method, the number of features can be reduced by based on distance, consistency, similarity, and information gain. A threshold value is used to discard the features below the predefined threshold. There are two types of methods belonging to this class: the univariate method handles each feature independently, while the multivariate method examines whole groups of features together. While straightforward based on statistical tests, one of the weaknesses of this method is the tendency of choosing a relatively large number of features. However, the filter method is faster than the wrapper and embedded methods without the use of an internal machine learning algorithm, and it is often used for pre-processing rather than relying solely on the wrapper (or embedded) method.

While the filter method relies on the statistical analysis, the wrapper method utilizes a machine learning classifier to measure the performance contribution provided by each feature [20]. The wrapper method requires determining the following three settings: (1) a search method to consider a wide range of feature subsets available (e.g., exhaustive search, random search, and heuristic search); (2) the explicit number of features targeted for the reduction; and (3) a machine learning classifier type (e.g., random forest and gradient boosting). The wrapper method is capable to find a subset of features with a good quality by considering a wide range of potential subsets. However, the wrapper method is expensive with respect to computational complexity due to the iterative learning steps and cross validation. Additionally, the selected features could be biased depending on the classifier being used, and one possible option to reduce the risk of the potential bias is the use of a different learning method for validating, which may introduce another level of complexity.

The embedded method also relies on a machine learning classifier as the wrapper method does. Unlike the wrapper method, however, the embedded method uses a built-in classification algorithm for the feature search process; that is, the feature selection algorithm is integrated as part of the machine learning algorithm. Hence, the learning process and feature selection process are tightly coupled (rather than independent each other). For instance, a decision tree algorithm can be used for feature selection, by determining more informative features along the tree construction process. Without repeated learning and cross validation, the embedded method is much lighter than wrapper-class methods with respect to computational complexity.

Under the three classes of feature selection methods, many different techniques have been introduced, and we consider the following five selection methods popularly utilized: Correlation-Based Feature Selection (CFS) [19] (filter), Univariate [21] (filter), Sequential Backward Selection (SBS) [22] (wrapper), Recursive Feature Eliminations (RFE) [23] (wrapper), and Importance [24] (embedded). A summary of the widely used feature selection methods and applications can be found from [20].

CFS is a filter method that gives features ranking based on a heuristic evaluation [19]. CFS seeks highly correlated and un-correlated features to discard redundant data based on the metric of Pearson correlation coefficients. Irrelevant features are disregarded since those features have a low correlation. Features highly correlated to one or more remaining features are considered as redundant features, and those features are screened out in this selection scheme. Univariate feature selection also belongs to the filter class, and reduces the dimensionality of features relatively quickly, which makes this scheme to be broadly applied [21]. This technique evaluates each feature by performing a statistical test, such as Pearson Correlation, mutual information and maximal information coefficient (MIC), and distance correlation, and then removes the features having a relatively weak relationship with the output feature (i.e., the feature for the classification label information).

SBS belongs to the wrapper class [22]. SBS uses a sequential selection algorithm, which is one of the greedy search algorithms, to reduce the number of features. Based on the greedy search, this technique discards features one by one until it meets the given number of features provided by the user. At first, a criterion function should be defined to tell which feature is discarded. Then, the criterion function calculates the performance of after and before the elimination of each feature, and the feature showing the least performance is discarded. RFE is also a wrapper method which relies on a machine learning classifier to discover the features less essential. Like SBS, this technique iteratively eliminates features until it gets to the targeted number of features configured by the user. RFE drops one feature at each iteration and re-ranks the remaining features to remove redundant and weak features. This selection technique has been widely used for a cancer classification application where the number of features is relatively large (greater than a thousand) with a small number of training samples around 100 [23].

Importance is an embedded method based on the random forest algorithm [24]. This scheme relies on a measure known as “Importance” value, and a feature with a too small value less than the specified threshold is removed from the feature set. The importance value is estimated when the internal random forest classifier is being constructed. For each node i in the constructed tree, the Gini index is calculated through $G_i = 2p(1-p)$, where p is the fraction of associating pairs assigned to node i while $(1-p)$ is the fraction of non-associating pairs. The importance value for a feature can then be derived from the sum of all decreases in the forest due to that feature, based on which it is determined whether the feature in question needs to be discarded or not.

3 EXPERIMENTAL SETTING AND PRELIMINARY RESULTS

In this section, we describe the experimental setting used across the paper including the description of network datasets, normalization techniques, machine learning classifiers, and evaluation metrics. Then, we discuss some of the key observations made from our preliminary experiments showing the limitation of conventional selection techniques for network anomaly detection.

3.1 Experimental Setting

We first provide a description of two public network datasets employed in this study. We then describe the importance of the data normalization process with our observations, and introduce our normalization technique used in this work. Additionally, the configuration of the classifiers used for network anomaly detection will be discussed.

3.1.1 Description of Datasets. A body of previous studies in the network anomaly detection research relied on some of the old network datasets such as KDDCup 1999 [25] that may not reflect the today’s network traffic adequately, mainly due to the limited availability of public network datasets. In this study, we mainly utilize two recently collected datasets

of UNSW-NB15 [7] and IDS-2017 [8] described below, which can better represent latest traffic patterns and attacks. Nonetheless, we also apply NSL-KDD [17], a variant of KDDCup 1999, for the evaluation in Section 6, due to its popular use in the network anomaly detection research.

UNSW-NB15 dataset [7]. This dataset is collected by the cybersecurity research group at the Australian Centre for Cyber Security (ACCS) in 2015 from a simulated network setting, which consists of servers for traffic generations and routers capturing the packets to create `pcap` files. The `tcpdump` command was used to capture 100 GB of the raw network traffic, which includes modern normal behaviors and attack activities. The attacks include Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, and Worms. The label information was constructed by using a bro-IDS tool[2]. In the dataset, there are 49 features (39 numeric) along with the label information for 2.5M records stored in four CSV files. The dataset provides a pair of training and testing files, which are smaller than the raw labeled data with 82,332 records for training and 175,341 records in the testing data file.

IDS-2017 dataset [8]. The IDS-2017 dataset was published by the Canadian Institute for Cybersecurity in 2018. This dataset provides the labeled data for network intrusion detection research, which resembles the true real-world data. Two networks were configured in the simulated setting: one for attacks and the other for victims. The captured dataset contains different types of attacks including Brute Force FTP, Brute Force SSH, DoS, Heartbleed, Web Attack, Infiltration, as well as background normal traffic that is generated by using the abstract behavior of 25 users based on the HTTP/HTTPS, FTP, SSH, and email protocols. The captured data span from 9 AM Monday, July 3, 2017 to 5 PM Friday July 7, 2017 (i.e., five days in total). The first day contains the normal traffic only, while the other days have malicious activities. The number of records is over 2.8M with 85 features (78 numeric) including the label information. We created 10 CSV files from IDS-2017 dataset for testing and training files, each of which contains 100,000 records randomly chosen from the entire dataset. Thus, individual data files contain both normal and attack instances. In our experiments, we used five for training and the other five for testing.

3.1.2 Normalization of Data. Normalization of data is essential for machine learning to minimize the possibility of being biased to a certain set of features. Without this pre-processing step, greater numeric feature values can overwhelm smaller feature values, which may weaken the effectiveness of statistical learning significantly [26]. In general, there exist two types of features: a *numeric* attribute contains an integer or floating point number, while a *categorical* attribute has a discrete value belonging to a specific finite set of classes. For example, "height" would be numerical, while "grade" := {A, B, C, D, F} is a categorical feature. The majority of features in network datasets are numeric, and we focus on prioritizing numeric features in this study.

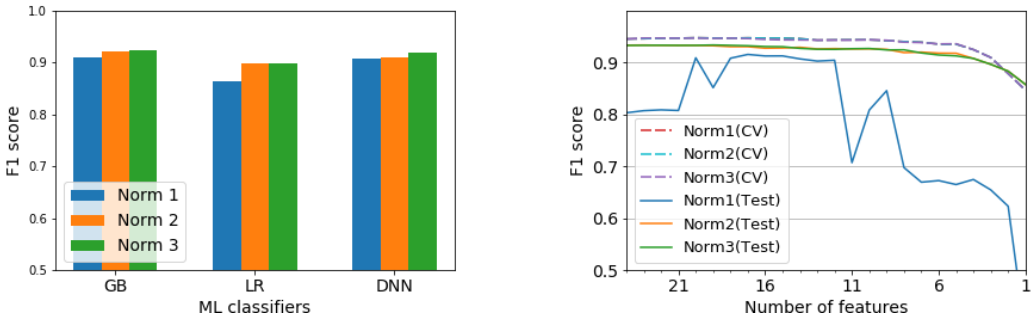
For numerical features, we rely on the *Min-Max* normalization defined as follows:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Here, x is the value given, x_{min} and x_{max} are the minimum and maximum values for that feature in the dataset, respectively, and x' is the normalized one. With this normalization, the value space for any feature becomes $[0, 1]$.

For a classification problem (like network anomaly detection), a set of data samples are provided in the training phase, and the output learning model from training is being used for future classification in the testing phase. It is quite straightforward to determine x_{min} and x_{max} from the training data but impossible to have them for testing. Thus, three different options would be considered for normalization:

[2]The Zeek Network Security Monitor, https://www.zeek.org



(a) F1-score (b) Cross-validation and testing performance

Fig. 2. Comparison of different normalization options (Classifier=GB, Dataset=UNSW-NB15): (a) F1-score with the entire features: the result shows that *Norm3* (normalizing independently but with the identical x_{min} and x_{max} obtained from the training data) works at least comparable to *Norm2* (based on the batch normalization); (b) CV and testing performance in F1-score over the number of features: *Norm1* may work poorly in testing although it worked well in the training phase from the cross validation (CV) result.

- *Norm1*: Do normalization independently for training and testing data
- *Norm2*: Do normalization together for training and testing data
- *Norm3*: Do normalization independently but with the identical x_{min} and x_{max} obtained from the training data

One problem here would be that testing data could be collected over time, and it may not be possible to perform *Norm1* and *Norm2*. In contrast, *Norm3* can perform its task even for such a streaming data analysis. However, it could observe an out-of-range value from a testing sample. It is possible to keep track of a set of x_{min} and x_{max} for individual features to update them over time. To minimize the overhead, we apply a simple approximation that assigns either 0 (if $x' < 0$) or 1 (if $x' > 1$).

Fig. 2(a) shows the results using four different classifiers (see Section 3.1.3 for the details about the classifiers). We can see that *Norm1* sometimes shows a relatively low accuracy (up to 12% lower than the others), while *Norm3* works quite well, yielding at least comparable performance to *Norm2*. Fig. 2(b) compares cross-validation (CV) score and the testing result when using different normalization options over the number of features. The figure shows that using *Norm3* produces a much better result than *Norm1* that works poorly in testing, although it worked well in the training phase from the CV result. Despite not shown in the figure due to space limitation, we made a similar observation from an experiment against the IDS-2017 dataset, showing that *Norm3* works well consistently while *Norm1* produces the relatively low accuracy. For this reason, we utilize *Norm3* for normalization in this paper.

3.1.3 Machine Learning Classifiers. The primary focus of this study is on developing a methodology for enabling automated feature selection that performs well across a different set of ML techniques (rather than any specific ML method to classify). For this reason, we chose three classification methods: Gradient Boosting (GB) from the tree-based approach, Logistic Regression (LR) from a regression-based method, and Deep Neural Network (DNN) from the neural network-based model. We used the Python scikit-learn library[3] for GB (Gradient Boosting), and LR (Logistic Regression) and

[3] scikit-learn Machine Learning in Python, https://scikit-learn.org/stable/

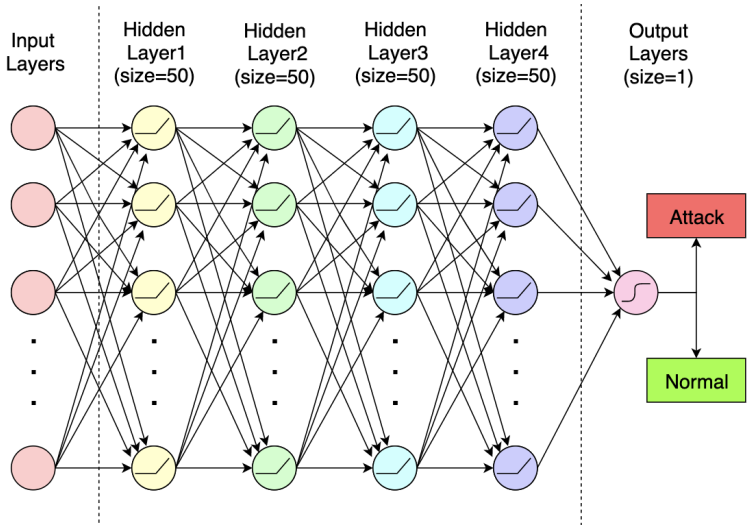


Fig. 3. DNN structure used in this study for network anomaly detection, with four layers configured with a `ReLU` activation function and dropout 0.2 in each layer. The output layer contains 1 unit with a `Sigmoid` activation function.

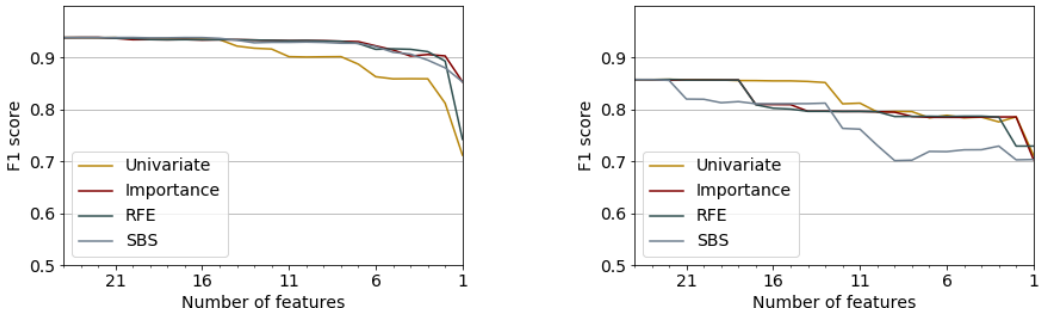
Keras library[4] for DNN (Deep Neural Network). For the classifiers, we basically utilize the default setting. We configured 200 trees for GB, and an inverse of regularization strength sets 100 and the Library for Large Linear (LIBLINEAR) classification solver for LR. For DNN, we configured four hidden layers, each of which contains 50 hidden units with a rectified linear unit (`ReLU`) activation function and the dropout of 0.2 in each layer. `ReLU` is one of the widely used activation functions for DNN with the benefit of the reduction of the vanishing gradient problem [27]. The output layer contains 1 unit with a `Sigmoid` activation function. The number of epochs and batch size along with the optimizer and loss function of our model is as follows: epochs=15, batch size=100, optimizer=Adam optimizer, and loss function=Binary Cross Entropy (BCE). Fig. 3 demonstrates the structure of the DNN model implemented for this study.

3.1.4 Evaluation Metrics. We basically examine the classification performance by utilizing multiple classifiers. Since the metric of *Accuracy* may lead to a biased result if the population of the minority class is too small, F1-score is widely accepted to minimize this concern. As an example, suppose 1% of cancer cells in the sample dataset. If a classifier confirms that the entire samples are benign, the accuracy becomes 99% that can give a misconception to the user, while the harmonic mean (known as F1-score) results in 0%. For this reason, we use the measure of F1-score to evaluate the performance of classifiers. The metrics are defined by using the elements in the confusion matrix, which are TP (True Positive), FP (False Positive), FN (False Negative) and TN (True Negative), as follows:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$\text{F1-score} = \frac{2TP}{2TP + FP + FN} \tag{2}$$

[4]Keras: The Python Deep Learning library, https://keras.io



(a) Cross-validation score (GB) (b) Cross-validation score (LR)

Fig. 4. Feature selection by four different selection techniques (RFE, SBS, Univariate, and Importance): (a) With GB, Univariate works quite differently from the others, showing some degradation with 15 or smaller number of features; (b) With LR, in contrast, Univariate works better for identifying core features over iterations, while SBS works somewhat poorly.

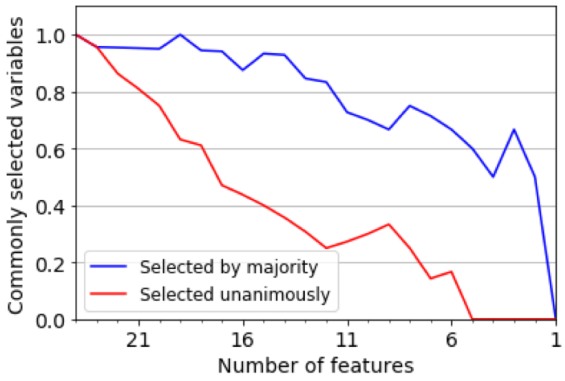


Fig. 5. Fraction of chosen features: the number of commonly chosen features by the entire selection methods drops significantly, whereas higher than 50% of the features are chosen by 3+ selection methods.

We use F1-score to report both of cross validation in training and testing results. In general, reporting CV score often assumes the metric of accuracy, but our implementation measures F1-score for CV to consider potential biases in the dataset. Hence, CV scores indicate training results, while F1-score is to report testing results throughout the paper.

3.2 Preliminary Results

As a preliminary study, we examined four existing feature selection techniques of RFE, SBS, Univariate, and Importance, to see how they work against a network traffic dataset (UNSW-NB15). The reason why we chose these four methods is that these are common feature selection methods, and we would like to examine different classes of selection methods (i.e., filter, wrapper, and embedded) without a bias.

The first observation from our preliminary experiment is that a feature selection tool shows quite different results under different settings. Fig. 4 shows an example. The figure reports the cross-validation results conducted with two different classifiers (GB in Fig. 4(a) and LR in Fig. 4(b)). The k-fold cross-validation was performed with $k = 5$ in our experiments.

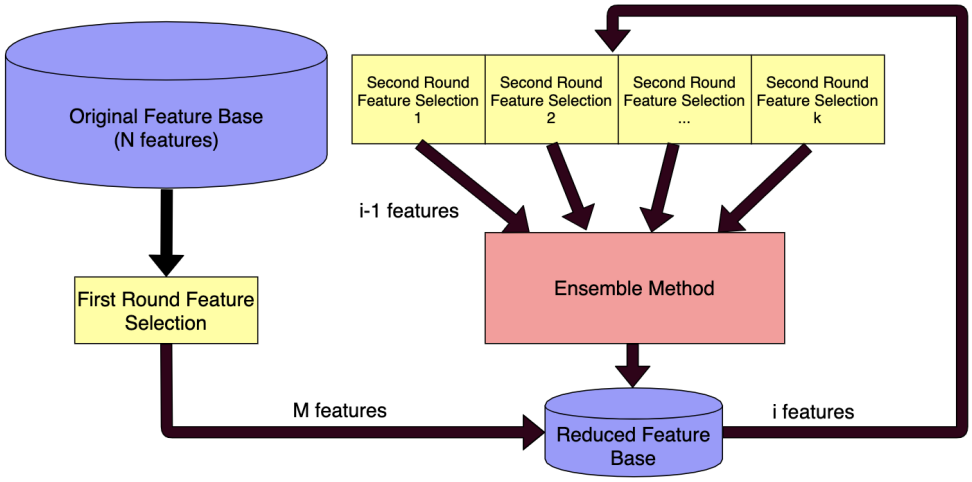


Fig. 6. **Process model with two independent rounds: The first-round process eliminates highly redundant features based on pair-wise correlation factors, and then the second-round process employs a set of reduction methods with the ensemble function (e.g., *Union*, *Intersection*, and *Quorum*). Each reduction method eliminates one feature at each iteration, while the ensemble function combines the result produced by each reduction tool. Under the assumption of the N features in the original dataset, it is reduced to M $(M \leq N)$ in the first round. At each iteration in the second round, each reduction tool eliminates one feature from i to $(i-1)$ features, and the ensemble function combines them, the result of which is fed back to the reduction tool unless $i \leq 1$.**

In the iterative process, each selection technique reduces the number of features one by one until it gets to a single feature. In Fig. 4(a), Univariate works quite differently from the others, showing some degradation with 15 or smaller number of features. In contrast, when using a different classifier in Fig. 4(b), we see that Univariate works better for identifying core features over iterations, while SBS works somewhat poorly.

Fig. 5 shows how many features were selected by the entire selection techniques ("Selected unanimously") or by a majority of the techniques ("Selected by majority"), across the number of features in the x-axis. From the figure, we can see that the fraction of the commonly chosen features by the entire selection methods drops significantly. For the majority selection, we can see that higher than 50% of the features are chosen by 3+ selection methods when the number of features is greater than 1.

We observed that the individual selectors produce different subsets as the reduction results. The results here imply that there would be *no* sole winner that outperforms other selection methods on all occasions. This motivates us to investigate a way to reliably identify core features, and we incorporate the multiple selection tools to attain this goal, as will be discussed in the following section.

4 AN ENSEMBLE APPROACH FOR IDENTIFYING CORE FEATURES

This section presents our *ensemble* approach that incorporates multiple feature selection methods in the process of feature reduction. We propose two techniques for ensemble: *heuristic-based* selection and *greedy-based* selection, as described next.

4.1 Ensemble Heuristics based on Set Theory

As discussed in the previous section, the existing selection techniques result in different sets as the outcome, and some techniques work better than the others when using a particular classification algorithm, but sometimes the others work better for another classifier. In this work, we embrace a different set of selection techniques to identify a set of features that could be more important than the others with respect to the classification performance in the testing phase.

Our first approach to the ensemble method is based on a set theory. Fig. 6 shows the overall procedure of how we utilize a different set of (existing) selection techniques. The procedure consists of two stages. The first-round task eliminates highly redundant features using a filter method. As shown from the figure, the N number of features provisioned in the original dataset is reduced to M ($M \leq N$) by the first-round reduction process. If $M == N$, it indicates that the reduction relies only on the second-round process without the use of the first-round reduction; although this would be a possible option, there can be a scalability issue because the complexity imposed by the second-round process is much higher than the first-round task, particularly when there exist a greater number of features. Given that the number of features for network connection datasets is high-dimensional (e.g., over 70 features for IDS-2017), we utilize a conventional filter method (CFS [28]) working based on pair-wise correlation factors for the first-round task in this work. The correlation threshold was set to 0.8, based on the observation of the maximum improvement in run-time with little cost to prediction accuracy with this threshold setting in the CFS work.

The second-round process takes M features as input. As shown from Fig. 6, "Reduced Feature Base" keeping the selected features contains M features initially. At each iteration, the current feature set with size i is disseminated to individual reduction tools, which eliminates one feature based on their own scheme and $(i-1)$ features are combined by the ensemble function (e.g., *Union*, *Intersection*, and *Quorum*). The resulted feature set is fed back to the reduction tool again if $i > 1$. We use four selection methods discussed in the previous section (RFE, SBS, Univariate, and Importance) for the second-round reduction in this study.

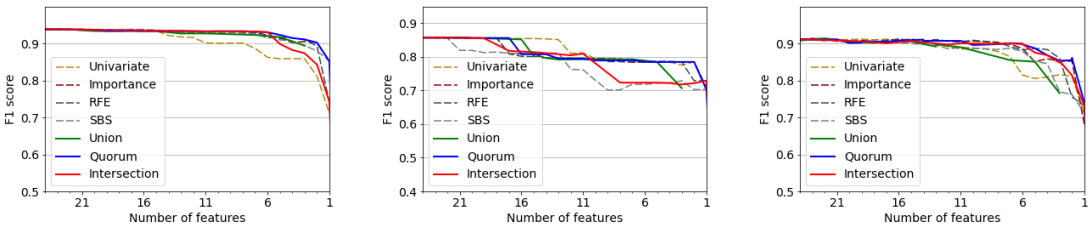
At every iteration in the second-round stage, there should be k different feature sets resulted by individual selection techniques. Hence, a question here would be *how to incorporate the feature sets produced by different selection functions*. For this purpose, we define three heuristic methods based on set theory, as follows:

- *Union*, which takes all the features that any of the selection methods suggests;
- *Intersection*, which takes the features that all selection methods agree upon;
- *Quorum*, which selectively takes the features agreed by the majority of the selection methods.

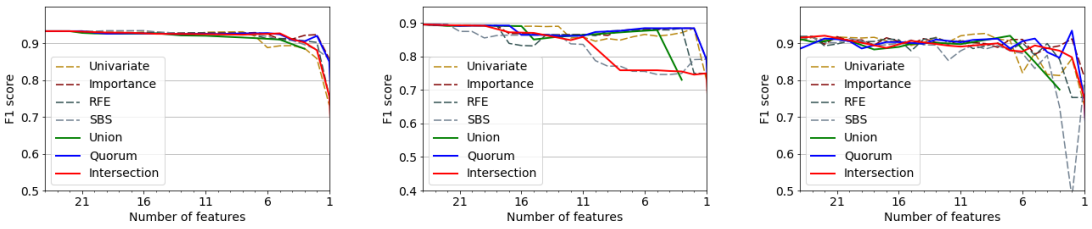
Based on set theory, $Union$ will produce a larger set, while $Intersection$ results in the smallest set size in general. Regardless of the size, the resulted set is disseminated to the reduction tools for eliminating the feature the least interesting using their own selection scheme. Finally, if the resulted set size becomes less than or equal to one (i.e., $i \leq 1$), the second-round process is finished.

Fig. 7 shows CV scores and testing performance in F1-score for the selection techniques including our set heuristics conducted using the dataset UNSW-NB15. From the figure, we can see that CV score would be a good indicator to expect the result in actual testing for this dataset, with similar patterns for individual classifiers. For the heuristics, *Quorum* shows quite consistent results for both CV and testing over the classifiers. In contrast, *Intersection* does not work well with LR, while it works well for the other two classifiers. *Union* largely works well but *Quorum* shows slightly better results.

We conducted the same experiment using the dataset IDS-2017, and Fig. 8 shows the experimental result with that data. Like the UNSW-NB15 result, the CV and testing results show similar patterns for the identical classifier. The heuristics also work in a similar way, and *Quorum* largely works better than the other heuristics and the existing methods.

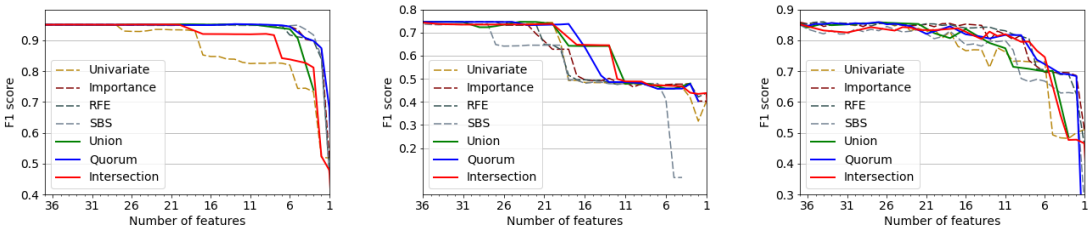


(a) CV F1-score (GB) (b) CV F1-score (LR) (c) CV F1-score (DNN)

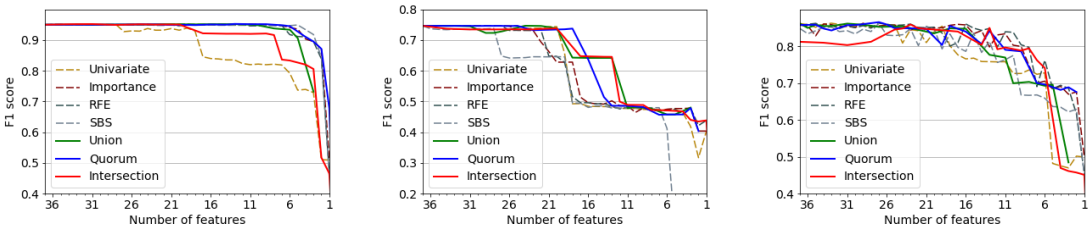


(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 7. CV score and testing performance (Dataset=UNSW-NB15): Overall, CV score would be a good indicator to expect the result in actual testing showing similar patterns for individual classifiers. *Quorum* shows more consistent results for both CV and testing over the classifiers than *Intersection* and *Union*.



(a) CV F1-score (GB) (b) CV F1-score (LR) (c) CV F1-score (DNN)



(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 8. CV score and testing performance (Dataset=IDS-2017): *Quorum* largely works better than the other heuristics and the existing methods.

Algorithm 1 Greedy reduction algorithm

1: **Input**: a feature set V; a set of selection techniques S
2: **Output**: a reduced feature set V' ($|V| - |V'| = 1$)
3: **for** (i=0; i<$|S|$; i++) **do**
4: Select $x \in V$ by selector $S[i]$
5: Calculate CV score y using a feature set $V - \{x\}$
6: $X[i] = x$
7: $Y[i] = y$
8: **end for**
9: $k = \arg\max_i(Y[i])$
10: $V' = V - X[k]$

4.2 A Greedy Approach for Ensemble

Another ensemble technique we present in this paper is a greedy-based search to maximize CV score. **The basic idea of this technique is to eliminate a feature, such that the remaining model without that feature has the best performance in cross validation, as Alg. 1 demonstrates.** Here is the detailed explanation of the algorithm. The algorithm takes a set of features as input (`line 1`) and eliminates the least interesting feature from the input feature set to return (`line 2`). Since we incorporate the existing selection methods, we let each selection method ($s \in S$) choose one feature to eliminate (`line 4`). The algorithm calculates a new CV score by eliminating the chosen feature (x) from the current feature set (V), which is stored in y (`line 5`). For each selection method, the chosen feature (x) and the new CV score (y) are stored temporarily (`line 6-7`). **Finally, the feature with which the performance is the least is chosen (`line 9`) and eliminated from the feature set (`line 10`).**

Here is an example. Let us suppose there exist three selection techniques (s_1, s_2, s_3) to incorporate. At a certain iteration, each selection technique selects one feature to eliminate. In the greedy algorithm, it executes a cross validation process with the eliminated feature v_i for each selection technique s_i. Assuming the resulted CV scores are 0.90 (for s_1), 0.88 (for s_2), and 0.92 (for s_3). Then, we know that eliminating v_3 would be the best option with respect to the performance, and the greedy algorithm selects v_3 to eliminate for that iteration. The same process continues at each iteration.

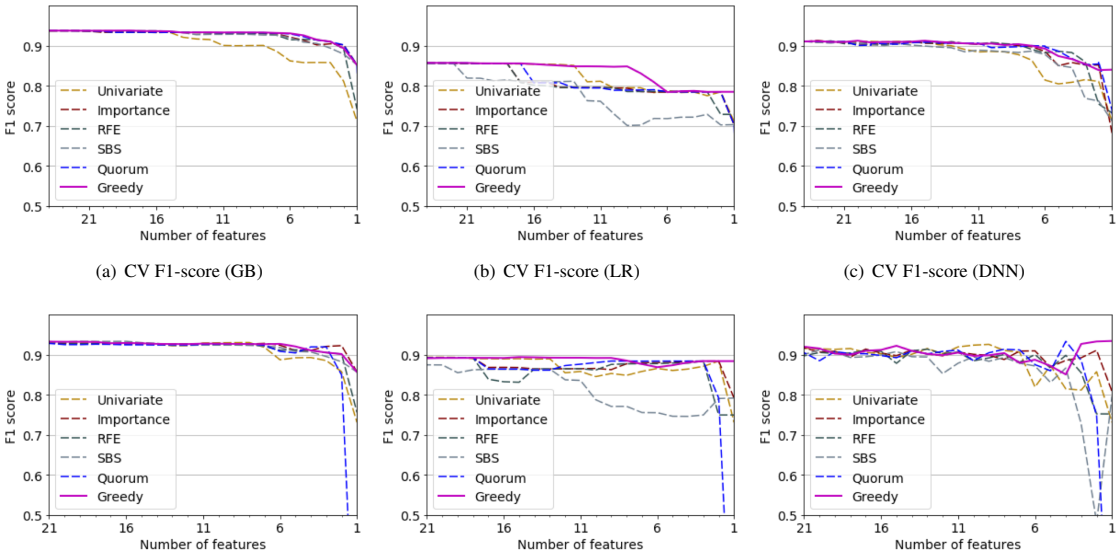
As described in Fig. 6, we assume that the first-round task filters out redundant features based on correlation factors, and the greedy-based reduction takes place in the second-round stage. **Fig. 9 shows how greedy selection works when using the dataset UNSW-NB15. As shown in the figure, the greedy method outperforms the other techniques across the classifiers. Fig. 10 compares the performance using the IDS-2017 dataset and shows the consistent results for the greedy method. From the figures, we can see that *Quorum* is slightly behind the greedy method, although *Quorum* overall works well and better than the traditional ones.**

5 DETERMINING STOPPING POINTS

The next question to enable the automated feature selection process is to determine a stopping condition, which indicates when to stop the second round reduction. An optimal stopping condition will result in the smallest feature set with the minimum performance loss approximating to one with the entire features. In this section, we present three methods to determine stopping points with the sensitivity and timing analysis results.

5.1 Methods for Determining Stopping Points

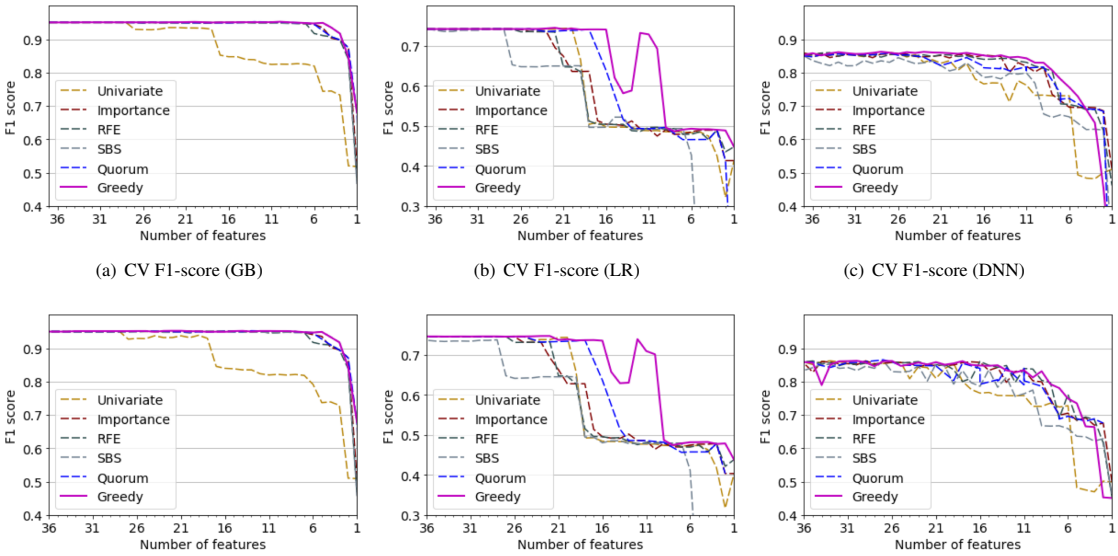
We set up the following three methods to determine the stopping point:



(a) CV F1-score (GB) (b) CV F1-score (LR) (c) CV F1-score (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 9. Greedy method CV score and testing performance (Dataset=UNSW-NB15): Overall, the greedy method outperforms the other techniques across the classifiers.



(a) CV F1-score (GB) (b) CV F1-score (LR) (c) CV F1-score (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 10. Greedy algorithm CV score and testing performance (Dataset=IDS-2017): The greedy methods works in a consistent manner, yielding better performance.

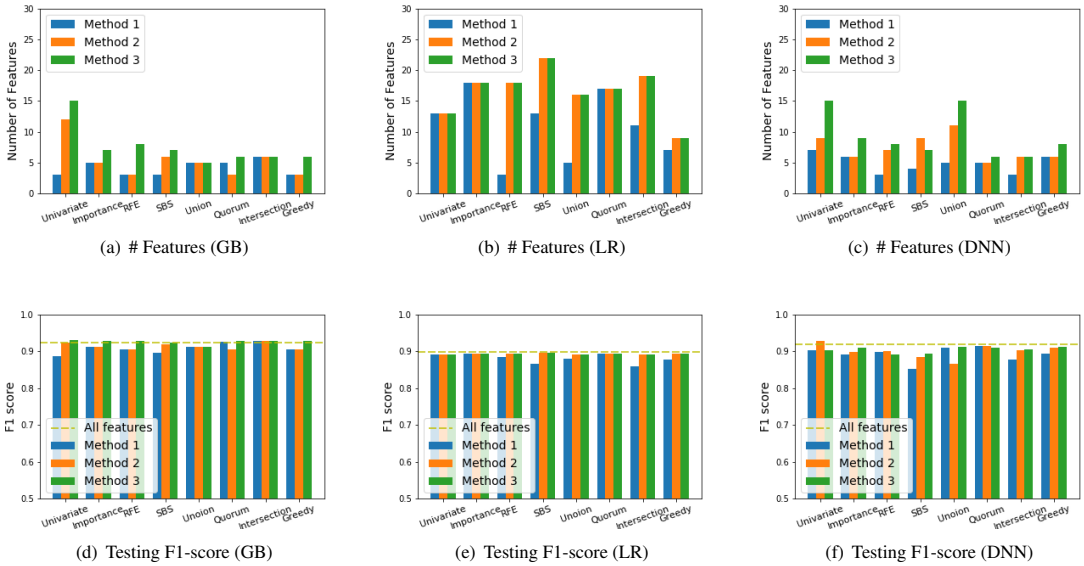
Method 1 *(MaxDelta).* **This method stops at iteration i that results in the largest decrease in terms of CV score between iteration k and $(k+1)$ where $k \geq 0$. Here, $k = 0$ indicates the initial stage in the second round before staring the reduction process. Suppose CV score at iteration k is defined as cv_k and $\Delta_k = cv_{k-1} - cv_k$ (for $k > 0$). The stopping point η is defined as $\eta = \arg\max_k \Delta_k$.**

Method 2 *(MinPerfReq).* **This method takes the minimum acceptable performance requirement as input. To set it up, we define ξ as the maximum tolerable CV bound compared to the initial CV score (cv_0) in the second round before the reduction process starts. Here is an example. If $\xi = 3\%$, this indicates that the second round stops immediately once $cv_0 - cv_k > \xi$, where k is the current iteration. We set $\xi = 3\%$ as the default configuration.**

Method 3 *(MaxScore).* **This method calculates a score at each iteration (ω_k for iteration k) to determine the termination point. We define a score function as follows: $\omega_k = cv_k - (\rho \times |V_k|)$, where V_k is the feature set at iteration k and ρ is a penalty factor to penalize an iteration having a higher number of features. Then, this method stops at the iteration with the greatest score; that is, it stops at η where $\eta = \arg\max_k \omega_k$. The intuition behind this score function is that we would like to penalize an iteration with a larger number of features compared to an iteration with a smaller number of features but showing the same performance. As an example, let us suppose that the CV score with 20 features is 0.973 and one with 10 features is 0.959. In this example, we can expect quite comparable performance only with 10 features. If we assume $\rho = 0.003$, the calculated score with 20 features is $0.973 - (20 \times 0.003) = 0.913$ while the score for 10 features becomes $0.959 - (10 \times 0.003) = 0.929$, and the method chooses the iteration for 10 features. A greater value for the penalty factor (ρ) results in a higher impact to an iteration with a larger number of features. By default, we set $\rho = 0.003$ in our evaluation, but discuss the sensitivity analysis of this parameter in Section 5.2.**

Using these methods, a subset of features is identified as the result of our automated selection process. In fact, Method3 shares the basic idea with Bayesian Information Criterion (BIC), which measures model strength based on the defined cost function consisting of complexity and performance [29]. BIC and other similar Bayesian methods (e.g., Akaike Information Criterion) depend on the prior distribution and work on maximizing the posterior predictive capability with the penalty on the model complexity. Similarly, Method3 attempts to balance two terms of the number of features (complexity) and CV score (performance) which are readily available at the time for determining when to stop, without additionally analyzing Maximum Likelihood Estimation (MLE) employed in BIC and its variants to measure the performance term.

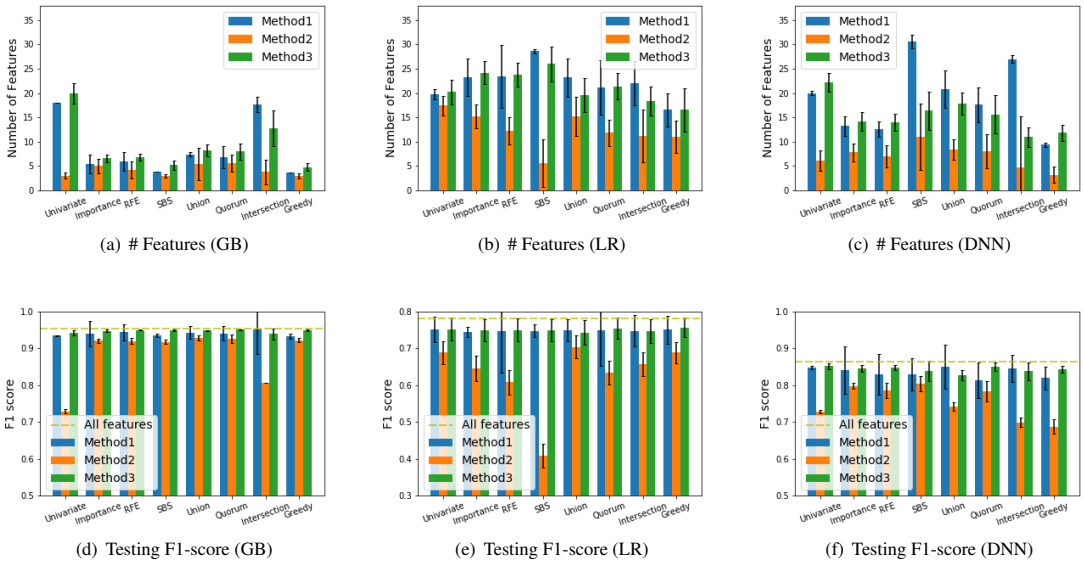
Fig. 11 shows the performance using the stopping methods against UNSW-NB15. We report the number of features in the chosen subset and F1-score measured with the subset for each selection method. It should be better if we can identify a smaller number of features that approximate to the performance with the entire features, since the computational overhead should be smaller in training and testing with a less number of features. From the figure, we can see that $Method3$ (*MaxScore*) yields slightly better performance with one or two more features overall. $Method1$ (*MaxDelta*) and $Method2$ (*MinPerfReq*) generally result in a smaller number of features, but sometimes they suffer from performance penalty. For example, SBS shows a degraded performance for LR with $Method1$ since it resulted in a too small number of features. When focusing on $Method3$, Univariate results in bigger sets with GB and DNN, while SBS has a bigger set with LR and Union resulted in a big one with DNN. The selection heuristics of *Quorum* and *Intersection* work well and are at least comparable to Importance and RFE. Overall, the figure shows *Greedy* works consistently well with a relatively small number of features.



(a) # Features (GB) (b) # Features (LR) (c) # Features (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 11. Automated selection result (Dataset=UNSW-NB15): $Method3$ yields slightly better performance with one or two more features overall. $Method1$ and $Method2$ generally result in a smaller number of features, but sometimes they suffer from performance penalty.



(a) # Features (GB) (b) # Features (LR) (c) # Features (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 12. Automated selection result (Dataset=IDS-2017): Overall, $Method3$ works in stable with smaller variances in F1-score than $Method1$ and $Method2$.

Table 1. Comparison of *Greedy* performance against the other techniques (using 2-sample t-test) for UNSW-NB15

Classifier	Measure	Univariate	Importance	RFE	SBS	Union	Quorum	Intersection
GB	F1-score diff.	1.20%	1.59%	1.74%	0.81%	2.41%	2.20%	2.44%
	p-value	0.24	0.12	0.091	0.425	0.023	0.035	0.022
LR	F1-score diff.	2.36%	4.30%	3.71%	4.14%	2.58%	3.44%	3.83%
	p-value	0.025	1.68E-04	0.001	2.57E-04	0.016	0.002	0.001
DNN	F1-score diff.	-1.78%	0.56%	0.99%	2.46%	1.64%	1.06%	1.22%
	p-value	0.085	0.58	0.33	0.020	0.11	0.30	0.24

Table 2. Comparison of *Greedy* performance against the other techniques (using 2-sample t-test) for IDS-2017

Classifier	Measure	Univariate	Importance	RFE	SBS	Union	Quorum	Intersection
GB	F1-score diff.	5.21%	2.40%	2.06%	3.71%	2.16%	2.12%	3.95%
	p-value	2.13E-06	0.019	0.043	0.0004	0.036	0.039	2.47E-04
LR	F1-score diff.	2.53%	2.98%	3.02%	4.47%	0.93%	1.21%	2.03%
	p-value	0.015	0.004	0.004	4.53E-05	0.36	0.23	0.050
DNN	F1-score diff.	2.74%	0.33%	0.63%	3.82%	0.56%	0.65%	3.65%
	p-value	0.008	0.74	0.53	3.74E-04	0.58	0.52	0.001

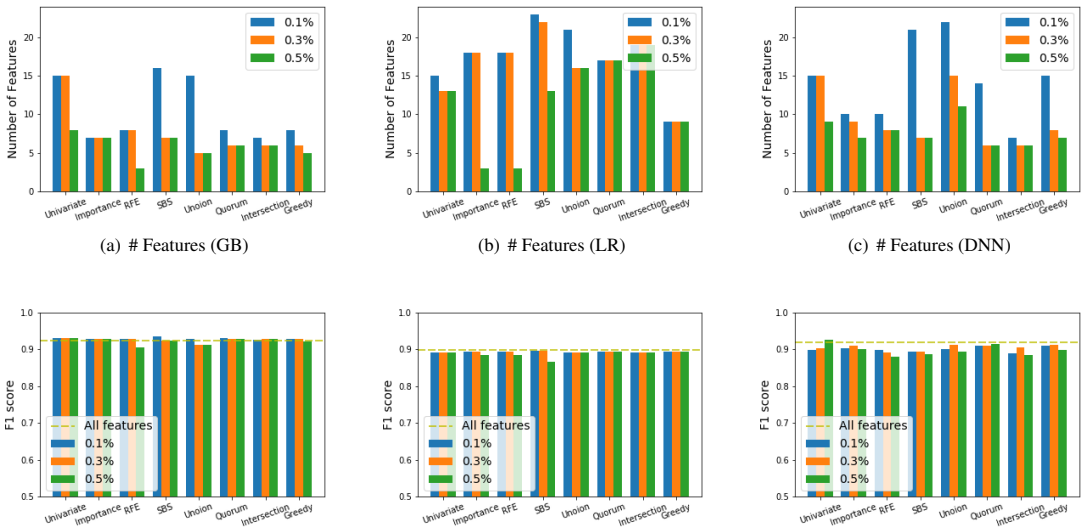
Fig. 12 shows the experimental results against IDS-2017. The experiment was repeated five times and we report the average and standard deviation in the figure. As in Fig. 11, *Method*3 works in stably without any significant variance in F1-score with the IDS-2017 dataset. The figure shows that *Greedy* works better than the others for identifying core features, while *Quorum* works better than the other selection heuristics like the result with UNSW-NB15. **In the figure, *Method*2 performs quite poorly due to too small number of features although the same parameter value ($\xi = 3\%$) were applied for both datasets, suggesting that *Method*2 is somewhat sensitive to the parameter configuration.**

In sum, we can see that *Greedy* with *Method*3 yields at least comparable performance (in F1-score) with a smaller number of features to the conventional techniques from Fig. 11 and Fig. 12. In regard to *Method*3, it outperforms the other two methods and works well not only with *Greedy* but also with the conventional selection techniques to help determine when to stop. **The details about the selected features by *Greedy* and *Method*3 for each classifier can be found in Table 5 and Table 6 in Appendix.**

As a further analysis of Fig. 11 and Fig. 12, Table 1 and Table 2 provide two-sample t-test results comparing the performance of *Greedy* with other techniques for UNSW-NB15 and IDS-2017, respectively. Along the iteration in the feature reduction to the stopping point (determined based on *Method*3), we analyze F1-score for each reduction technique in a statistical manner. **In this t-test analysis, the difference is the relative, not absolute, difference in performance between *Greedy* F1-score and the F1-score of the method in comparison, and hence, any positive number indicates the performance of *Greedy* is better than the other with the p-value chance (in percentage), and vice versa for negative values.** The comparison results in the tables show *Greedy* statistically outperforms the other selection methods with positive values; the only exception is that DNN shows a slightly lower F1-score (-1.78%) than *Univariate*.

5.2 Sensitivity Analysis

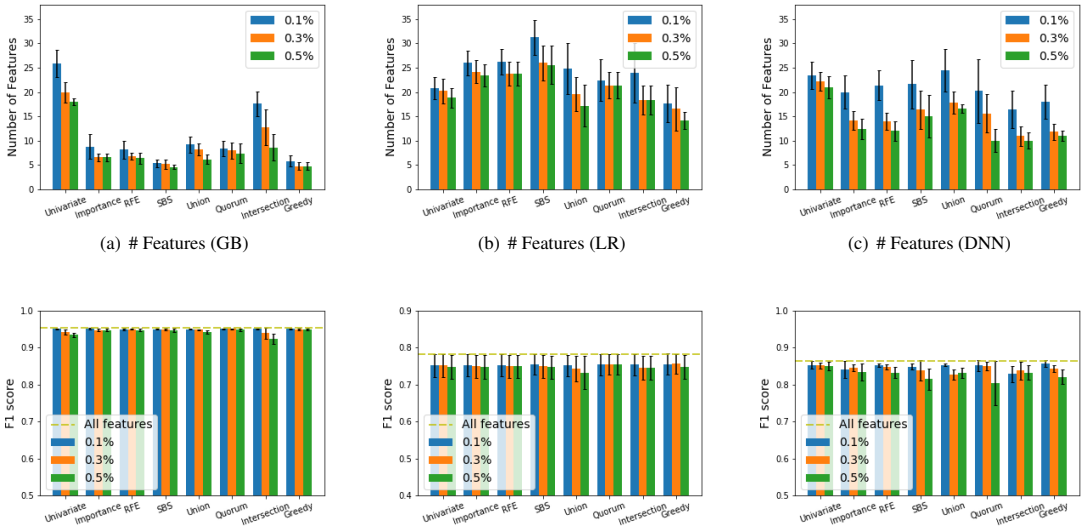
From the experimental results, we observed that *Method*3 works more consistently than the other two methods. Again, *Method*3 penalizes an iteration with a large number of features if the resulted performance is the same. We



(a) # Features (GB) (b) # Features (LR) (c) # Features (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

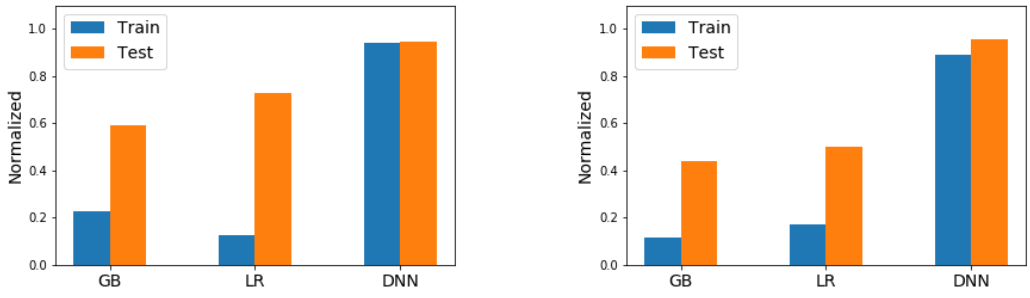
Fig. 13. 3 sensitivity analysis (Dataset=UNSW-NB15): A smaller ρ results in a larger number of features as outcome, while it shows slightly better performance in testing.



(a) # Features (GB) (b) # Features (LR) (c) # Features (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 14. $Method3$ sensitivity analysis (Dataset=IDS-2017): The overall trend is similar with the result in Fig. 13.



(a) UNSW-NB15 (b) IDS-2017

Fig. 15. Timing analysis: GB and LR show greater improvement with the selected features, and training shows greater gains than testing.

performed a sensitivity study to see the impact of the penalty factor (ρ) with a set of values empirically chosen: $\rho = \{0.001\ (0.1\%), 0.003\ (0.3\%), 0.005\ (0.5\%)\}$. Note that, by definition, a smaller penalty factor (ρ) will lead to a larger number of features as outcome, and vice versa. Fig. 13 and Fig. 14 show the result. **The experimental result shows a trade-off between the number of features and classification performance, as expected, though the differences in performance were not statistically significant.** Choosing a value for this penalty factor may depend on the user's requirement.

5.3 Timing Analysis

We measured the time taken for training and testing on a single dedicated computing machine, configured with Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, 16 GB memory, and Ubuntu 16.04.6 LTS. In this experiment, we compare the execution time for training and testing with a subset of features selected by $Method3$ to the execution results with the entire features. To consider variations, we report the average of the results with the 10 repeated measurements.

Fig. 15 shows the normalized results comparing to the use of the entire features (i.e., time taken with the entire features = 100%). From the figure, we can see that the gain for training is substantial; GB shows 4.4X-8.6X faster, while LR shows 5.8X-7.9X faster than the training time with the entire features. For testing, GB and LR show 1.4X-2.3X improvements. The gain for DNN is the least, but it still shows up to 11% for training and 6% for testing. Table 3 summarizes the per-connection training and testing time. From the table, training is more expensive almost by three orders of magnitude. We presume that this is why training shows much higher gains than testing.

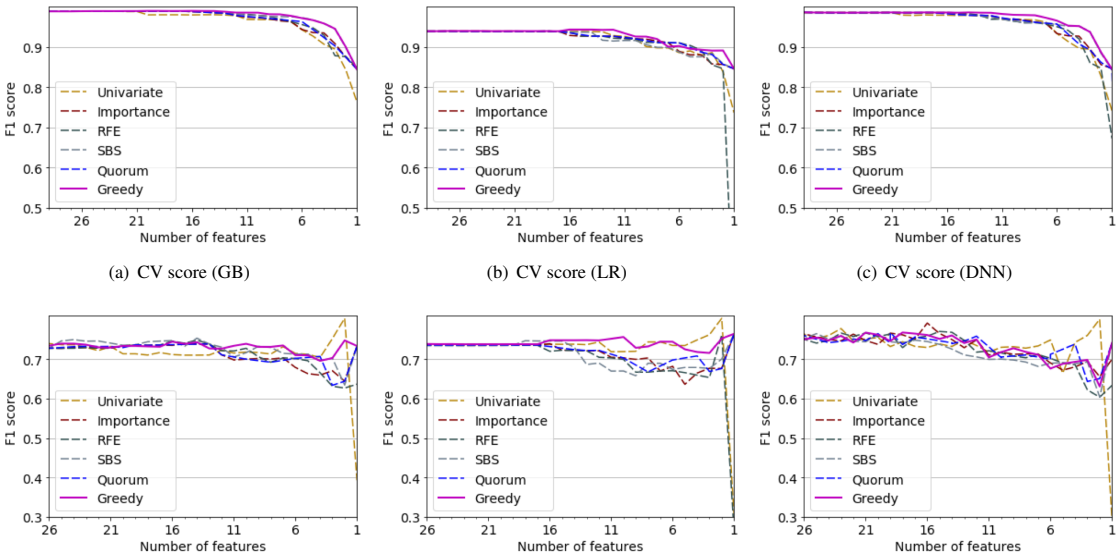
The complexity of our ensemble techniques can be analyzed in a straightforward manner since our approach loosely connects existing selection techniques and the aggregated sum will be the complexity of our approach. As they can be easily parallelized, the complexity will be approximately similar to the maximum complexity of the existing selection techniques.

6 EVALUATION WITH NSL-KDD

Thus far, we mainly utilized the recent datasets of UNSW-NB15 and IDS-2017 for the purpose of evaluation. The dataset of KDDCup 1999 [25, 30] was collected longer than two decades ago (in 1998) but has been widely employed for the

Table 3. Per-connection training and testing cost

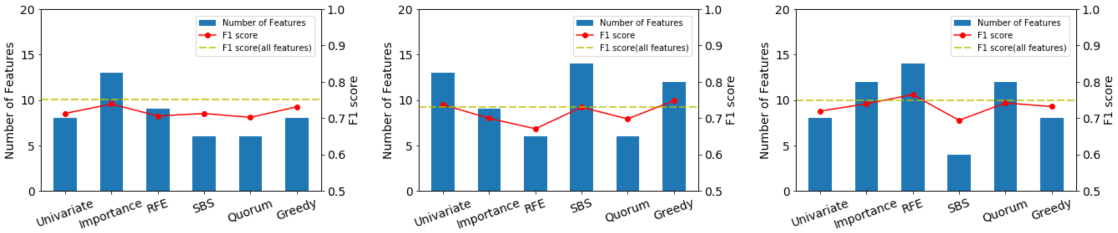
Dataset	Task	Entire features			Selected features		
		GB	LR	DNN	GB	LR	DNN
UNSW-NB15	Training	0.29ms	0.06ms	0.18ms	0.07ms	0.01ms	0.17ms
	Testing	37us	0.4us	32us	22us	0.3us	30us
IDS-2017	Training	0.50ms	0.09ms	0.19ms	0.06ms	0.02ms	0.17ms
	Testing	53us	0.8us	33us	23us	0.4us	32us



(a) CV score (GB) (b) CV score (LR) (c) CV score (DNN)

(d) Testing F1-score (GB) (e) Testing F1-score (LR) (f) Testing F1-score (DNN)

Fig. 16. CV score and testing performance (Dataset=NSL-KDD):



(a) GB (b) LR (c) DNN

Fig. 17. Automated selection using $Method3$ (Dataset=NSL-KDD):

network anomaly detection research. Due to its popular use, we evaluate our presented methods with NSL-KDD [17], a modified version of KDDCup 1999, in this section.

Table 4. Comparison of *Greedy* performance against the other techniques (using 2-sample t-test) for NSL-KDD

Classifier	Measure	Univariate	Importance	RFE	SBS	Union	Quorum	Intersection
GB	F1-score diff.	5.08%	2.64%	1.70%	-0.82%	0.51%	2.19%	2.96%
	p-value	8.21E-06	0.012	0.097	0.42	0.61	0.035	0.005
LR	F1-score diff.	2.14%	4.54%	4.41%	2.10%	3.98%	4.11%	0.31%
	p-value	0.040	6.64E-05	9.92E-05	0.044	3.81E-04	2.69E-04	0.76
DNN	F1-score diff.	1.04%	0.08%	0.51%	1.75%	0.02%	0.67%	1.62%
	p-value	0.30	0.94	0.62	0.087	0.99	0.51	0.11

Since KDDCup 1999 contains a large number of redundant instances that may cause biases in the performance evaluation, NSL-KDD significantly reduces repeated data records to improve the overall quality. In NSL-KDD (and KDDCup 1999), each record consists of 41 features with the associated label that can be classified into five classes: DOS (denial-of-service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to local root privileges), Probing (surveillance and other probing), and Normal (non-attack). There are four files in the NSL-KDD dataset, and we use the full training file ("KDDTrain+") and the full testing file ("KDDTest+") for our experiments. We considered numeric features (38 out of 41 features) for the reduction and classification.

We next report the experimental results for NSL-KDD. Fig. 16 shows the CV score and testing performance. Since the greedy method consistently outperforms the set-based heuristics, we mainly compare *Greedy* with the conventional selection methods (*Univariate*, *Importance*, *RFE*, and *SBS*); however, the extended comparison result will also be discussed shortly with Table 4. The CV results in the figure clearly show that *Greedy* outperform the other selection techniques. For testing F1-score, *Greedy* overall works better than the others when using GB and LR, while it shows the comparable performance when using DNN.

Fig 17 shows automated selection results when using $Method3$ (ρ=0.3%) to determine the stopping point. We can see that *Greedy* chooses a moderate number of features (i.e., not too small or large). The overall trend shows that choosing too small number of features may degrade the performance (e.g., *RFE* when using LR, and *SBS* when using DNN). *Greedy* yields the approximate performance to the one with the use of the entire features. Table 4 provides two-sample t-test results to compare the performance of *Greedy* with other techniques (including the set-based heuristics). As described, a positive number indicates the performance of *Greedy* is better than the other with the p-value chance (in percentage), and vice versa. The result in the table supports the effectiveness of *Greedy*, largely showing comparable or better performance.

7 RELATED WORK

In this section, we summarize the previous studies in feature selection for network anomaly detection. In the work of [31], the authors provide an analysis of network traffic features for anomaly detection. Their proposed selection method relies on feature weighting and ranking using filters and wrappers to obtain a tentative baseline of features with strong contributions. The selected features are then applied to a refinement process based on a brute force search analysis to decrease the number of features. The brute force search is composed of two stepwise regressions: backward elimination (BS) whose stop condition is based on maximum performance, and forward selection (FS) which adds a feature until the accuracy improvement becomes less than the threshold. Then, the final features of subsets are evaluated in different classifiers. The brute force nature of the proposed technique imposes a too expensive computational cost. In addition, the KDDCup 1999 dataset [30] (and its variant NSL-KDD [17]) was collected two decades ago, and it does not represent today's network traffic with biased, no longer close to contemporary attack vectors [25, 32].

In [33], the authors used the concept of feature ranking by combining information gain and correlation to reduce the number of features for network intrusion detection. They assumed the features with higher entropy values are more critical, while the features with smaller correlation scores would be more helpful to distinguish different classes. Using a neural network classifier with the reduced number of features (25 out of 41 features), the authors claimed the very high accuracy over 94%, but the confusion matrix shown in the paper does not support the reported result by the authors. Additionally, the metric of accuracy can be easily biased if the dataset is unbalanced in terms of classes like KDDCup 1999 used in this previous study. In our evaluation, we mainly utilize F1-score (a harmonic mean) as is reliable against the class imbalance problem.

Another work in [34] applied feature transformation on KDDCup 1999. Then this work utilizes the transformed features for training and testing. The authors considered a subset of the features in the basic and content groups (i.e., 21 features out of 41 features in total) but excluded the traffic feature group for their selection process. This previous study first identifies top-10 features for individual attacks based on a statistical correlation test (Chi-square); the top-10 features are then transformed into two dimensional coordinate space by aggregating five features for each dimension. Using the transformed features, the authors compared a set of classifiers, including CVM (Core Vector Machine), Support Vector Machines (SVM), Naive Bayesian classifier (NB), Decision trees (DT), Random Forest (RF) and AdaBoost, and reported that CMV works well compared to the other classifiers.

While a majority of previous studies relied on KDDCup 1999 or its variant (NSL-KDD), there are some studies utilized UNSW-NB15 and IDS-2017. The work in [35] simply utilized a set of embedded tools provided in Weka [36], to search a subset of features for anomaly detection. After examining using a set of machine learning techniques, the authors reported that using random forest with five selected features performs better than other settings (82% accuracy). Note that our proposed method yields 91% accuracy with the selected six features on the same dataset.

Khammassi et al. [37] introduced a wrapper approach that incorporates GA (Genetic Algorithm) for feature search with LR (Logistic regression) for feature selection. This approach consists of three steps: pre-processing, feature selection, and classification. In the pre-processing stage, the dataset is transformed into the representation that machine learning classifiers can deal with. After that, the GA-LR wrapper chooses subsets based on the accuracy of classification and the number of features in the feature selection stage. The classification stage evaluates the subset of features chosen in the feature selection stage. The authors evaluated their proposed scheme with the UNSW-NB15 dataset, and reported their GA-LR method yields comparable classification accuracy with 20 features compared to the one with the entire features for the multi-class intrusion detection.

The work in [38] investigated feature selection against IDS-2017. The authors first obtained each feature's performance using JRip (a rule-based classification tool), and then chose 18 top-ranked features as a result of feature selection. The authors evaluated the selected features using a k-nearest neighbors algorithm and a rule-based method building C4.5 decision tree, for classification. This previous study focused only on detecting DDoS attacks from the normal traffic, and hence, this study is quite limited despite the high accuracy reported (99% accuracy with and without feature selection). We consider all the attacks presented in the dataset for evaluating the selection process.

Although not placed in the area of network anomaly detection, the work in [28] investigates the parallelism for feature selection to reduce the training complexity. This previous study proposed a selection method based on Correlation-based Feature Selection (CFS) and Sequential Backward Selection (SBS) to find a reduced feature set. CFS remove highly correlated data while SBS eliminates the worst performance feature in each iteration. To utilize parallel computing resources for feature selection, the authors used Apache Spark as the back-end to execute SBS with concurrent processes. The authors reported that the runtime of the feature selection is 20 times faster compared to a lineal feature selection

method. Since the focus of this past work is the parallelization of the selection scheme to minimize the time to build up a training model, it does not answer when to stop to obtain a feature set, which is one of the main contributions of our paper.

8 CONCLUSION

Feature selection is beneficial by reducing the training and testing complexity, but it generally requires a lot of effort with intensive analysis to identify relatively more important features in advance. Automating the selection process is thus essential for the practical use of feature selection, but it is challenging with the existing methods since it is hard to determine the termination condition to stop searching. In addition, we observed that individual selection methods result in different feature sets as the output. In this paper, we first presented an ensemble approach that combines the independent results by individual selection methods. Two ensemble methods were introduced, one is based on set theory and the other based on greedy search, and we observed the greedy-based method consistently outperforms other techniques including the existing techniques and set-based heuristics. We also presented a set of methods to determine stopping points, which is an essential function to implement the automation of the selection process. Through the extensive experiments, we reported our score-based stopping method with greedy search yields comparable detection rates to the use of the entire features, only with 13% (UNSW-NB15) and 6% (IDS-2017) of the features in the datasets.

In this study, we evaluated the proposed method with two latest datasets widely used for the network anomaly detection research. An interesting direction for future tasks is to apply our method for other binary classification problems in network security (e.g., botnet detection using the CTU-13 dataset: https://www.stratosphereips.org/datasets-ctu13). We also plan to extend our method for attack identification, which is a multi-class classification problem that identifies attack classes to prepare the associated security measure. The identified core features will be beneficial for future data collection to improve the detection accuracy with the managed complexity.

ACKNOWLEDGMENT

This effort was supported in part by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231, and in part by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.2016-0-00078, Cloud-based Security Intelligence Technology Development for the Customized Security Service Provisioning). This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility.

REFERENCES

[1] Evangelos E Papalexakis, Alex Beutel, and Peter Steenkiste. Network anomaly detection using co-clustering. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 403–410. IEEE, 2012.

[2] Jinoh Kim, Alex Sim, Brian Tierney, Sang Suh, and Ikkyun Kim. Multivariate network traffic analysis using clustered patterns. *Computing*, 101(4):339–361, 2019.

[3] Sunhee Baek, Donghwoon Kwon, Jinoh Kim, Sang C Suh, Hyunjoo Kim, and Ikkyun Kim. Unsupervised labeling for supervised anomaly detection in enterprise and cloud networks. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 205–210. IEEE, 2017.

[4] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016.

[5] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C. Suh, Ikkyun Kim, and Kuinam J. Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22(Suppl 1):949–961, 2019.

[6] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2015.

[7] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[8] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018*, pages 108–116, 2018.

[9] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):1–45, 2017.

[10] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6):94, 2018.

[11] Salem Alelyani, Jiliang Tang, and Huan Liu. Feature selection for clustering: A review. In *Data Clustering*, pages 29–60. Chapman and Hall/CRC, 2018.

[12] Qi-Hai Zhu and Yu-Bin Yang. Discriminative embedded unsupervised feature selection. *Pattern Recognition Letters*, 112:219–225, 2018.

[13] Ahmed A Ewees, Mohamed Abd El Aziz, and Aboul Ella Hassanien. Chaotic multi-verse optimizer-based feature selection. *Neural Computing and Applications*, 31(4):991–1006, 2019.

[14] Fernando Jiménez, Carlos Martínez, Enrico Marzano, Jose Tomas Palma, Gracia Sánchez, and Guido Sciavicco. Multiobjective evolutionary feature selection for fuzzy classification. *IEEE Transactions on Fuzzy Systems*, 27(5):1085–1099, 2019.

[15] Yonghua Zhu, Xuejun Zhang, Rongyao Hu, and Guoqiu Wen. Adaptive structure learning for low-rank supervised feature selection. *Pattern Recognition Letters*, 109:89–96, 2018.

[16] Esma Nur Cinicioglu and Taylan Yenilmez. Determination of variables for a bayesian network and the most precious one. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 313–325. Springer, 2016.

[17] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, pages 53–58, 2009.

[18] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[19] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection–a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 178–187. Springer, 2007.

[20] K. Brkić N. Bogunović A. Jović. A review of feature selection methods with applications.

[21] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[22] Sebastian Raschuka. *Python Machine Learning*. Packet Publishing Ltd, 2015.

[23] Xue-wen Chen and Jong Cheol Jeong. Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 429–435. IEEE, 2007.

[24] Oznur Tastan, Yanjun Qi, Jaime G Carbonell, and Judith Klein-Seetharaman. Prediction of interactions between hiv-1 and human proteins by information integration. In *Biocomputing 2009*, pages 516–527. World Scientific, 2009.

[25] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6. IEEE, 2009.

[26] Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, page 105524, 2019.

[27] Devansh Arpit and Yoshua Bengio. The benefits of over-parameterization at initialization in deep relu networks. *arXiv preprint arXiv:1901.03611*, 2019.

[28] Jonathan Wang, Wucherl Yoo, Alex Sim, Peter Nugent, and Kesheng Wu. Parallel variable selection for effective performance prediction. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 208–217. IEEE, 2017.

[29] Alexandr Katrutsa and Vadim Strijov. Comprehensive study of feature selection methods to solve multicollinearity problem according to evaluation criteria. *Expert Systems with Applications*, 76:1–11, 2017.

[30] KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[31] Félix Iglesias and Tanja Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 101(1-3):59–84, 2015.

[32] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[33] Ishfaq Manzoor, Neeraj Kumar, et al. A feature reduced intrusion detection system using ann classifier. *Expert Systems with Applications*, 88:249–257, 2017.

[34] TH Divyasree and KK Sherly. A network intrusion detection system based on ensemble cvm using efficient feature selection approach. *Procedia computer science*, 143:442–449, 2018.

[35] Tharmini Janarthanan and Shahrzad Zargari. Feature selection in unsw-nb15 and kddcup'99 datasets. In *2017 IEEE 26th international symposium on industrial electronics (ISIE)*, pages 1881–1886. IEEE, 2017.

[36] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[37] Chaouki Khammassi and Saoussen Krichen. A ga-lr wrapper approach for feature selection in network intrusion detection. *computers & security*, 70:255–277, 2017.

[38] Gayatri V Patil, K Vinod Pachghare, and Deepak D Kshirsagar. Feature reduction in flow based intrusion detection system. In *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, pages 1356–1362. IEEE, 2018.

APPENDIX: DATASET STRUCTURE AND SELECTED FEATURES

Table 5. UNSW-NB15 numeric features and the selected features by *Greedy* and $Method3$ for each classifier

ID	Feature	Description	GB	LR	DNN
1	dur	Total duration			
2	sbytes	Number of bytes from source to destination			
3	dbytes	Number of bytes from destination to source			
4	sttl	TTL value from source to destination	✓	✓	✓
5	dttl	TTL value from destination to source		✓	
6	sloss	Source packets retransmitted/dropped			
7	dloss	Destination packets retransmitted/dropped			
8	sload	Source bits/s			
9	dload	Destination bits/s		✓	✓
10	spkts	Source to destination packet count			
11	dpkts	Destination to source packet count			
12	swin	Source TCP window advertisement value		✓	
13	dwin	Destination TCP window advertisement value			
14	stcpb	Source TCP base sequence number			
15	dtcpb	Destination TCP base sequence number			
16	smeansz	Mean of the flow packet size transmitted by source	✓		✓
17	dmeansz	Mean of the flow packet size transmitted by destination	✓	✓	✓
18	trans_depth	Pipelined depth into the connection of http request/response transaction			
19	res_bdy_len	Content size of the data transferred from the server's http service.			
20	sjit	Source jitter (msec)			
21	djit	Destination jitter (msec)			
22	sintpkt	Source interpacket arrival time (mSec)			
23	dintpkt	Destination interpacket arrival time (mSec)			
24	tcprtt	TCP connection setup round-trip time, the sum of synack and ackdat	✓	✓	✓
25	rate	Description not provided in the dataset	✓	✓	✓
26	synack	TCP connection setup time, the time between the SYN and the SYN_ACK packets			
27	ackdat	TCP connection setup time, the time between the SYN_ACK and the ACK packets			
28	is_sm_ips_ports	If source and destination IP address and port pairs are equal			
29	ct_state_ttl	No. for each state according to specific range of values for source/destination TTL		✓	✓
30	ct_flw_http_mthd	No. of flows that has methods such as Get and Post in http service.			
31	is_ftp_login	If ftp session is accessed by user and password			
32	ct_ftp_cmd	No of flows that has a command in ftp session.			
33	ct_srv_src	No. of conn. containing same service and src address in past 100 connections	✓	✓	✓
34	ct_srv_dst	No. of conn. containing same service and dest address in past 100 connections			
35	ct_dst_ltm	No. of conn. containing same dest address in past 100 connections			
36	ct_src_ ltm	No. of conn. containing same src address in past 100 connections			
37	ct_src_dport_ltm	No. of conn. containing same src address and dest port in past 100 connections			
38	ct_dst_sport_ltm	No. of conn. containing same dest address and source port in past 100 connections			
39	ct_dst_src_ltm	No. of conn. containing same src and dest address in past 100 connections			

Table 6. IDS-2017 numeric features and the selected features by *Greedy* and $Method3$ for each classifier

ID	Feature	GB	LR	DNN
1	Protocol		✓	✓
2	Flow Duration	✓	✓	✓
3	Total Fwd Packets			
4	Total Backward Packets			
5	Total Length of Fwd Packets	✓	✓	
6	Total Length of Bwd Packets			
7	Fwd Packet Length Max		✓	✓
8	Fwd Packet Length Min		✓	✓
9	Fwd Packet Length Mean			
10	Fwd Packet Length Std			
11	Bwd Packet Length Max		✓	✓
12	Bwd Packet Length Min		✓	✓
13	Bwd Packet Length Mean			
14	Bwd Packet Length Std			
15	Flow Bytes/s			
16	Flow Packets/s			
17	Flow IAT Mean		✓	
18	Flow IAT Std		✓	✓
19	Flow IAT Max			
20	Flow IAT Min		✓	
21	Fwd IAT Total			
22	Fwd IAT Mean			
23	Fwd IAT Std			
24	Fwd IAT Max			
25	Fwd IAT Min			
26	Bwd IAT Total			
27	Bwd IAT Mean			
28	Bwd IAT Std		✓	✓
29	Bwd IAT Max			
30	Bwd IAT Min			
31	Fwd PSH Flags		✓	
32	Bwd PSH Flags			
33	Fwd URG Flags			
34	Bwd URG Flags			
35	Fwd Header Length			
36	Bwd Header Length			
37	Fwd Packets/s			
38	Bwd Packets/s		✓	
39	Min Packet Length			

ID	Feature	GB	LR	DNN
40	Max Packet Length			
41	Packet Length Mean			
42	Packet Length Std			
43	Packet Length Variance			
44	FIN Flag Count		✓	
45	SYN Flag Count			
46	RST Flag Count		✓	
47	PSH Flag Count		✓	✓
48	ACK Flag Count		✓	✓
49	URG Flag Count		✓	✓
50	CWE Flag Count			
51	ECE Flag Count			
52	Down/Up Ratio		✓	
53	Average Packet Size			
54	Avg Fwd Segment Size			
55	Avg Bwd Segment Size			
56	Fwd Header Length			
57	Fwd Avg Bytes/Bulk			
58	Fwd Avg Packets/Bulk			
59	Fwd Avg Bulk Rate			
60	Bwd Avg Bytes/Bulk			
61	Bwd Avg Packets/Bulk			
62	Bwd Avg Bulk Rate			
63	Subflow Fwd Packets			
64	Subflow Fwd Bytes			
65	Subflow Bwd Packets			
66	Subflow Bwd Bytes			
67	Init_Win_bytes_forward	✓	✓	✓
68	Init_Win_bytes_backward	✓	✓	✓
69	act_data_pkt_fwd		✓	
70	min_seg_size_forward		✓	✓
71	Active Mean			
72	Active Std		✓	
73	Active Max			
74	Active Min			
75	Idle Mean			
76	Idle Std		✓	
77	Idle Max			
78	Idle Min			