Zuech *et al. Journal of Big Data* (2022) 9:119
https://doi.org/10.1186/s40537-022-00661-9

Journal of Big Data

RESEARCH Open Access

A new feature popularity framework for detecting cyberattacks using popular features

Richard Zuech*, John Hancock and Taghi M. Khoshgoftaar

*Correspondence:
rich@richardzuech.com

Florida Atlantic University, 777 Glades Road, Boca Raton, FL, USA

Abstract

We propose a novel feature popularity framework, and introduce this new framework to the cybersecurity domain. Feature popularity has not yet been used in machine learning or data mining, and we implement it with three web attacks from the CSE-CIC-IDS2018 dataset: Brute Force, SQL Injection, and XSS web attacks. Feature popularity is based upon ensemble Feature Selection Techniques (FSTs) and allows us to more easily understand common and important features between different cyberattacks. Three filter-based and four supervised learning-based FSTs are used to generate feature subsets for each of our three different web attack datasets, and then our feature popularity frameworks are applied. Classification performance for feature popularity is mostly similar as compared to when "all features" are evaluated (with feature popularity subsets having better performance in 5 out of 15 experiments). Our feature popularity technique effectively builds an ensemble of ensembles by first building an ensemble of FSTs for each dataset, and then building another ensemble across a dataset agreement dimension. The Jaccard similarity is also employed with our feature popularity framework in order to better identify which attack classes should (or should not) be grouped together when applying feature popularity. The four most popular features across all three web attacks from this experiment are: Flow_Bytes_s, Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total. When only using these four features as input to our models, classification performance is not seriously degraded. This feature popularity framework granted us new and previously unseen insights into the web attack detection process with CSE-CIC-IDS2018 big data, even though we had intensely studied it previously. We realized these four particular features cannot properly identify our three web attacks, as they operate mainly from the time dimension and NetFlow features from layers 3 and 4 of the OSI model. Conversely, our three web attacks operate in the application layer (7) of the OSI model and should not leave signatures in these four features. Feature popularity produces easier to explain models which provide domain experts better visibility into the problem, and can also reduce the complexity of implementing models in real-world systems.

Keywords: Feature popularity, Feature similarity, Feature selection, Intrusion detection, Web attacks

Introduction

With consumers spending over $600 billion on e-commerce in the United States during 2019 [1], cybersecurity is becoming increasingly important to help defend against attackers. Machine learning can be employed to help in detecting cyberattacks. Feature selection is a common technique used by machine learning practitioners. Benefits of feature selection include improving classification efficiency by training models with fewer features which requires less computing resources, and feature selection can sometimes even improve classification performance.

Another benefit of feature selection, in the context of cybersecurity, is it can help practitioners better understand the attack detection process. This can be accomplished because feature selection can identify the most important features during the model building process. Gaining a better understanding of the most important features not only helps during the machine learning model building process, but it can be even more helpful when machine learning models are deployed and implemented into real-world products and systems.

Various Feature Selection Techniques (FSTs) can generate very different feature lists on the same dataset (which we identify throughout this study). However, finding "common features" between different FSTs can sometimes help us find an even better feature subset through the diversity of different FSTs acting in an ensemble [2, 3]. While ensemble FSTs can sometimes improve classification performance, they might still be desirable even with minor degradations in classification performance. Reasons for using ensemble FSTs with minor decreases in performance may include: feature stability [4], reducing model complexity for real-world implementations, and simply providing models which are easier to understand and are more explainable.

Feature similarity is a concept which is implicit in the ensemble FST process. For example, an ensemble FST can find similar (or "common") features between the different FSTs by identifying features which appear in common among the "Top N" Feature Importance Lists from different FSTs. However, we extend this concept and introduce the notion of "feature popularity" [5] by also finding common features across different datasets. For example, in cybersecurity there are different types of cyberattacks and we can generate different datasets which are based upon those different attacks.

To explore feature popularity, we utilize the CSE-CIC-IDS2018 dataset which was created by Sharafaldin et al. [6]. CSE-CIC-IDS2018 is a more recent version of the popular CIC-IDS2017 dataset [7], which was also created by Sharafaldin et al. The CSE-CIC-IDS2018 dataset includes over 16 million instances which includes normal instances, as well as the following family of attacks: web attack, Denial of Service (DoS), Distributed Denial of Service (DDoS), brute force, infiltration, and botnet.

The CSE-CIC-IDS2018 dataset is big data, as it contains over 16 million instances. While big data has not been formally defined in terms of the number of instances, one study [8] considers only 100,000 instances to be big data. Other studies [9, 10] have considered 1,000,000 instances to be big data. Since CSE-CIC-IDS2018 is more than 1,000,000 instances, we consider it to be big data as well.

Given its richness in containing many different attack labels, CSE-CIC-IDS2018 is a good dataset for investigating feature popularity. To do so, we only evaluate the following

three different web attacks from CSE-CIC-IDS2018: Brute Force (BF), Cross-Site Scripting (XSS), and SQL Injection (which we commonly refer to only as "SQL" throughout this document). Basically, we generate three new datasets by combining each of these three attack labels with all of the normal traffic from the full CSE-CIC-IDS2018 dataset. We then compare feature popularity results between these three different web attack datasets.

Brute Force web attacks correspond to brute force login attacks targeting web pages. Next, the XSS web attack refers to where attackers inject malicious client-side scripts into susceptible web pages targeting web users which view those pages. Finally, the SQL Injection web attack represents a code injection technique where attackers craft special sequences of characters and submit them to web page forms in an attempt to directly query the back-end database of that website. The feature popularity techniques which we introduce to CSE-CIC-IDS2018 in this study allows us to visually explain and quantify common features across these three different web attacks.

We selected web attacks to implement feature popularity because they are important to cybersecurity practitioners and they still commonly appear in the Open Web Application Security Project (OWASP) "Top 10 Web Application Security Risks" [11]. Also, the web attacks from the CSE-CIC-IDS2018 have three different web attack labels, and this allows us to partition the datasets into three new datasets in order to implement feature popularity. In other words, with feature popularity we can find the most popular features across these three different web attacks by applying feature popularity to these three newly created datasets.

The remaining sections of this paper are organized as follows. The "Related work" section studies existing literature for feature popularity with CSE-CIC-IDS2018 data. Then, the "Methodologies" section describes the data preparation, classifiers, FSTs, performance metrics, and feature popularity techniques applied in our experiments. Next, we provide a walk-through example of feature popularity in the Creating Feature Popularity Lists with Web Attack Datasets section. The "Results and discussion" section provides our results and analysis. Finally, the "Conclusion" section concludes our work.

Related work

Sarhan et al. [12] focus on how to explain models with feature selection and the eXplainable Artificial Intelligence (XAI) method using the CSE-CIC-IDS2018 dataset. Their motivation in using this XAI method is to be able to more easily explain the attack detection process through a better understanding of what the most important features are after applying the feature selection process. To score the most important features, they assign a Shapley value to each of the features. The Shapley value "is the weighted average of the respective contribution of a feature value" (which is essentially the amount a feature contributes towards making a prediction).

Random Forest and Deep Feed Forward classifiers are employed by Sarhan et al. to score the top 20 features of CSE-CIC-IDS2018 with Shapley values. These two different classifiers produce two very different lists of top 20 features for each of the classifiers. For example, the top ranked feature from the Deep Feed Forward classifier is Bwd_Packets_s, while the Random Forest classifier ranks this same feature as the 16th

best feature overall. It is difficult to ascertain how similar the two feature subsets are between the two different classifiers. Their research does not compare the feature similarity between these two different feature subsets like our research does. They do not benchmark the classification performance of only using the top 20 features versus all of the features, but their main motivation is to better explain and interpret the classification models by understanding the most important features used to generate those models.

Leevy et al. [13] apply an ensemble feature selection technique to the full CSE-CIC-IDS2018 dataset, and employ binary classification by merging the multiple attacks to one attack label. The ensemble feature selection technique considers seven different FSTs, of which three are filter-based FSTs and four are supervised FSTs. Different feature subsets are generated by finding common features among the seven different FSTs where a certain number of the FSTs agree. This ensemble FST concept is similar to this current study, but the main difference is that study only considers one attack dataset while this current study extends the approach by also finding common features across multiple attack datasets. In other words, this current study is different as it not only finds common features for a single attack dataset, but it also finds common features across multiple attacks. Also, the current study introduces the Jaccard similarity for quantifying feature subset similarity between different attacks.

Fitni et al. [14] compare two different feature selection techniques with the full CSE-CIC-IDS2018 dataset and map the multiple attacks to a binary classification problem with only attack and normal labels. Their two feature selection techniques are Chi-Squared (top 22 features) and Spearman's rank correlation coefficient (top 23 features), and they compare these two FSTs with Logistic Regression and Decision Tree classifiers. The Spearman's rank correlation coefficient technique performed better with F1 scores of 0.983 and 0.974, as compared to the Chi-Squared results of F1 scores of 0.791 and 0.974. Also, the full feature set performed best with the Decision Tree yielding an Area Under the Receiver Operating Character-istic Curve (AUC) score of 0.975.

Based on the results of these two classifiers (Logistic Regression and Decision Tree), Fitni et al. performed further experimentation using seven classifiers with only the Spearman's rank correlation coefficient FST and the full feature set. They do a nice job displaying their feature subsets which provides good insight into the attack detection process (similar to the XAI motivations of Sarhan et al. [12]). However, their research does not consider feature popularity concepts like our study does.

Beechey et al. [15] apply feature selection to the Goldeneye and Slowloris Denial of Services attacks from CSE-CIC-IDS2018. Their dataset only considers one day out of the ten days of available network traffic from CSE-CIC-IDS2018 with 1,048,575 instances, while our experiment considers all ten days of normal traffic encompassing over 13 million instances. Eight feature selection techniques were employed with at least six classifiers, and their Table 6 indicates perfect AUC scores for several combinations of FSTs and classifiers (sometimes overfitting can be associated with perfect classification). While Beechey et al. and others [16] apply feature ranking techniques to CSE-CIC-IDS2018,

they do not explore the notion of feature popularity between different attacks or common features between different FSTs.

We thoroughly surveyed Google Scholar to find related works to CSE-CIC-IDS2018 and our feature popularity research, and we searched for terms like "feature popularity", "CSE-CIC-IDS2018 feature similarity", and "CSE-CIC-IDS2018 feature selection". First, Google Scholar did not provide any results significantly related to our "feature popularity" focus from any application domain, and so we are the first to conceive the feature popularity concept. Second, after reviewing more than 261 CSE-CIC-IDS2018 works at the time of this writing, only the works by Sarhan et al. [12] and Leevy et al. [13] had aspects which were remotely similar to our feature popularity research. While the rest of the CSE-CIC-IDS2018 corpus did contain some feature selection aspects, we only included [14, 15] as those had the most compelling details of feature ranking with CSE-CIC-IDS2018.

The XAI method highlighted by Sarhan et al. provides good insights, and we agree that a better understanding and explanation of feature subsets is important to the attack detection process when implementing machine learning models in the real world. Moreover, this is especially important when considering different attacks like we have done, as different attacks can generate very different feature subsets. To the best of our knowledge, ours is the first study to define the concept of feature popularity, especially in the context of network intrusion detection. In addition, it is the first study to utilize the Jaccard similarity metric for examining feature similarity between different attack types.

Methodologies

Data preparation

In this section, we describe how we prepared and cleaned the dataset files used in our experiments. Properly documenting these steps is important in being able to reproduce experiments.

We dropped the "Protocol" and "Timestamp" fields from CSE-CIC-IDS2018 during our preprocessing steps. The "Protocol" field is somewhat redundant as the "Dst Port" (Destination_Port) field mostly contains equivalent "Protocol" values for each Destination_Port value. Additionally, we dropped the "Timestamp" field as we wanted the learners not to discriminate attack predictions based on time, especially with more stealthy attacks in mind. In other words, the learners should be able to discriminate attacks regardless of whether the attacks are high volume or slow and stealthy. Dropping the "Timestamp" field also allows us the convenience of combining or dividing the datasets into ways more compatible with our experimental frameworks. Additionally, a total of 59 records were dropped from CSE-CIC-IDS2018 due to header rows being repeated in certain days of the datasets. These duplicates were easily found and removed by filtering records based on a white list of valid label values.

The fourth downloaded file named "Thuesday-20-02-2018_TrafficForML_CICFlowMeter.csv" was different than the other nine files from CSE-CIC-IDS2018. This file contained four extra columns: "Flow ID", "Src IP", "Src Port", and "Dst IP". We dropped these four additional fields. Also of note is that this one particular file contained nearly half of

all the records for CSE-CIC-IDS2018. This fourth file contained 7,948,748 records of the dataset's total 16,232,943 records.

Certain fields contained negative values which are invalid values and so we dropped those instances with negative values for the "Fwd_Header_Length", "Flow_Duration", and "Flow_IAT_Min" fields (with a total of 15 records dropped from CSE-CIC-IDS2018 for these fields containing negative values). Negative values in these fields were causing extreme values that can skew classifiers which are sensitive to outliers.

Eight fields contained constant values of zero for every instance. In other words, these fields did not contain any value other than zero. Before starting machine learning, we filtered out the following list of fields (which all had values of zero):

1 Bwd_PSH_Flags
2 Bwd_URG_Flags
3 Fwd_Avg_Bytes_Bulk
4 Fwd_Avg_Packets_Bulk
5 Fwd_Avg_Bulk_Rate
6 Bwd_Avg_Bytes_Bulk
7 Bwd_Avg_Packets_Bulk
8 Bwd_Avg_Bulk_Rate

We also excluded the "Init_Win_bytes_forward" and "Init_Win_bytes_backward" fields because they contained negative values. These fields were excluded since about half of the total instances contained negative values for these two fields (so we would have removed a very large portion of the dataset by filtering all these instances out). Similarly, we did not use the "Flow_Duration" field as some of those values were unreasonably low with zero values.

The "Flow Bytes/s" and "Flow Packets/s" fields contained some "Infinity" and "NaN" values (with less than 0.6% of the records containing these values). We dropped these instances where either "Flow Bytes/s" or "Flow Packets/s" contained "Infinity" or "NaN" values. Upon carefully and manually inspecting the entire CSE-CIC-IDS2018 dataset for such values, there was too much uncertainty as to whether they were valid records or not. As sorted from minimum to maximum on these fields, neighboring records were very different where "Infinity" was found. Similar to Zhang et al. [17], we did attempt to impute values for these columns by taking the maximum value of the column and adding one. In the end, we abandoned this imputation approach and dropped 95,760 records from CSE-CIC-IDS2018 for records containing any "Infinity" or "NaN" values.

We also excluded the Destination_Port categorical feature which contains more than 64,000 distinct categorical values. Since Destination_Port has so many values, we determined that finding an optimal encoding technique was out of scope for this study. For each of the three web attacks in Table 2, we dropped all the other attack instances and kept all the normal instances from all ten days in Table 1 (except for those instances which we removed as indicated earlier in this section). Each of the three final datasets for our individual web attacks ended up having roughly 13 million instances as specified in Table 2. For full descriptions of all 79 features of the downloaded CSE-CIC-IDS2018 dataset, please refer to its website [18].

Table 1 Entire CSE-CIC-IDS2018 Dataset by files/days (only web attacks and normal traffic are used in our experiments)

Day	Normal Instances	Attack Instances
02/14 Wed - Brute Force	667,626	380,949
02/15 Thurs - DoS	996,077	52,498
02/16 Fri - DoS	446,772	601,802
02/20 Tues - DDoS	7,372,557	576,191
02/21 Wed - DDoS	360,833	687,742
02/22 Thu - Web	1,048,213	362
02/23 Fri - Web	1,048,009	566
02/28 Wed - Infiltration	544,200	688,871
03/01 Thurs - Infiltration	238,037	93,063
03/02 Fri - Bot	762,384	286,191
Total Records	13,484,708	2,748,235

Table 2 Web attacks used in this experiment from CSE-CIC-IDS2018

Attack type	Attack instances	Normal instances
Brute Force (BF) Web	611	13,390,234
Sql Injection (SQL) Web	87	13,390,234
XSS Web	230	13,390,234

Classifiers

Five different classifiers are utilized in this experiment: Decision Tree (DT), Ran-dom Forest (RF), CatBoost (CB), LightGBM (LGB), and XGBoost (XGB). These classifiers are used for two different purposes in our experiment. The first purpose these classifiers are employed is to implement feature selection, and the second purpose these classifiers are employed is to train and test our models. In other words, a classifier could first be used to apply feature selection and then later that same (or different) classifier could be used to train and test our model.

For all models trained and tested in this study, stratified 5-fold cross validation [19] is used. Stratified refers to evenly splitting each training and test fold so that each class is proportionately weighted across all folds equally. Splitting in a stratified manner is especially important when dealing with high levels of class imbalance, as randomness can inadvertently skew the results between the cross-validation folds [20].

To treat the severe class imbalances encountered throughout this experiment, we employ random undersampling (RUS). Every model trained throughout this experiment first applies RUS at a 1:1 sampling ratio during the model training process (sampling is not applied during the model testing process). Further details for how we apply RUS can be found in [21].

To account for randomness, each stratified 5-fold cross validation was repeated 10 times. Therefore, all of our AUC results are the mean values from 50 measurements (5 folds x 10 repeats). All classifiers from this experiment are implemented with Scikit-learn [22] and their respective Python modules. Next, our five classifiers are described.

Decision Tree is a learner which builds branches of a tree by splitting on features based on a cost [23]. The algorithm will attempt to select the most important features to split branches upon, and iterate through the feature space by building leaf nodes as the tree is built. Cost functions utilized to evaluate splits in the branches are Entropy and Gini impurity [24].

Random Forest is an ensemble of independent decision trees. Each instance is initially classified by every individual decision tree, and the instance is then finally classified by consensus among the individual trees (e.g., majority voting) [25]. Diversity among the individual decision trees can improve overall classification performance, and so bagging is introduced to each of the individual decision trees to promote diversity. Bagging (bootstrap aggregation) [26] is a technique to sample the dataset with replacement to accommodate randomness for each of the decision trees.

CatBoost [27] is based on gradient boosting, and is essentially another ensemble of tree-based learners. It utilizes an ordered boosting algorithm [28] to overcome prediction shifting difficulties which are common in gradient boosting. CatBoost has native built-in support for categorical features.

LightGBM, or Light Gradient Boosted Machine [29], is another learner based on Gradient Boosted Trees (GBTs) [30]. To optimize and avoid the need to scan every instance of a dataset when considering split points, LightGBM implements Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) algorithms [31]. LightGBM also offers native built-in support for categorical features.

XGBoost is another ensemble based on GBTs. To help determine splitting points, XGBoost utilizes a Weighted Quantile Sketch algorithm [32] to improve upon where split points should occur. Additionally, XGBoost employs a sparsity-aware algorithm to help with sparse data to determine default tree directions for missing values. Categorical features are not natively supported by XGBoost, and must be encoded outside of the learner with a technique such as One Hot Encoding (OHE) [33].

Unless specified otherwise here, default values were utilized for hyper-parameters with the classifiers. To prevent overfitting with Decision Tree, max_depth = 5 was assigned. For Random Forest, both n_estimators = 5 and max_depth = 6 were set to prevent overfitting. When using CatBoost both iterations = 4 and max_depth = 5 were assigned to prevent overfitting, while thread_count = 8 was set to take advantage of parallel processing functionality. Finally, with XGBoost both n_estimators = 4 and max_depth = 5 were set to prevent overfitting, and n_jobs = 8 was assigned to leverage parallel processing functionality. Also, objective = "binary logistic" was set to specify the objective function for binary classification with XGB.

Feature selection techniques and rankers

At its core, our new feature popularity framework utilizes underlying feature selection techniques to build our ensembles. Any FST can be utilized which produces ranking lists of the most important features, also known as Feature Importance Lists. In this study, all seven of the FSTs we employ utilize their respective Python libraries [34] to generate their respective Feature Importance Lists.

These Feature Importance Lists, are based upon feature importance [35] rankings. Effectively, each FST will produce lists of features which are sorted and ranked according to their most important features. While we selected only the top 20 features from the seven rankers in this study, future work could also experiment with different cutoff points for Feature Importance Lists and vary the number of features to be included. This experiment utilizes the following four supervised learning-based FST rankers and three filter-based FST rankers.

Supervised learning-based FST rankers

The four supervised learning-based FSTs rankers from this study are implemented from four of our classifiers: RF, CB, LGB, and XGB. These four classifiers are used for two purposes in this study. First, these four classifiers will be utilized to generate their respective Feature Importance Lists (rankings of the most important features). Second, these four classifiers are also used later in the experiment to train and test our machine learning models. The Decision Tree classifier is also used to train and test our machine learning models, but DT is not used in the FST process to build Feature Importance Lists. Future work can consider experimenting with different underlying FSTs and the feature popularity framework.

These supervised learning-based FST rankers offer a simple feature selection technique in the sense that they are implemented by the Feature Importance Lists from their respective RF, CB, LGB, and XGB Python libraries. For example, to generate the top 20 feature rankings with the RF supervised learning-based FST ranker, its Python library would generate a Feature Importance List and we would select the top 20 features from that list. Essentially, the Feature Importance Lists being generated from these supervised learning-based FST rankers are effectively serving as a feature selection technique. Please note that sometimes these supervised learning-based FST rankers might only produce less than 20 features, and in those cases their Feature Importance Lists will contain less than 20 features.

Filter-based FST rankers

Three filter-based FST rankers are applied in this study: Chi Squared, Information Gain, and Information Gain Ratio. Filter-based techniques utilize independent algorithms or statistical measurements when selecting features, and filter techniques are easily identified since they do not use learning algorithms in the feature selection process. Feature Importance Lists are generated from these three filter-based FST rankers in the same manner as the supervised learning-based FST rankers. These three FSTs are described below.

The Chi Squared technique tests which variables are most independent to the class, and the "chi2" function [36] from Python is used to calculate this list of ranked features. Information Gain, which is also known as Mutual Information evaluates features to determine which one maximizes the information gained [37]. Information Gain Ratio is similar to Information Gain, but Information Gain Ratio decreases its bias when the number of branching features is high [38]. In this experiment, both Information Gain and Information Gain Ratio are implemented with "info_gain" and "info_gain_ratio"

Python functions from [39]. Further details for how we implement these three filter-based FST rankers can be found in [13].

Classification performance metrics

AUC is a metric which measures the area under the Receiver Operator Charac- teristic (ROC) curve. The ROC curve is a plot of the True Positive Rate (TPR) along the y-axis versus the False Positive Rate (FPR) along the x-axis. The area under this ROC curve corresponds to a numeric value ranging between 0.0 to 1.0, where an AUC value equal to 1.0 would correspond to a perfect classification system. An AUC value of 0.5 would represent a classifier system performing as well as a random guess, similar to flipping a coin. The AUC metric scores how effective a classification system is in terms of comparing TPR to FPR across all classification thresholds [40].

Feature popularity metrics

The Jaccard similarity metric allows us to obtain quantitative scores for how similar feature sets are between each other. If we have a set of features from one dataset and another set of features from a different dataset, then we can score how similar these feature sets are with the Jaccard similarity. The Jaccard similarity of two sets is the ratio of the size of the intersection of the sets to the size of the union of the sets. That is, for two sets A and B, their Jaccard similarity is:

$$\text{Jaccard similarity} = \frac{|A \cap B|}{|A \cup B|}$$

We calculate the Jaccard similarity for pairs of different web attacks, and present those Jaccard similarity scores in tables in the "Results and discussion" section. These tables give the reader a quantitative score indicating the degree to which different FSTs agree between pairs of different web attacks. In other words, the Jaccard similarity gives us a quantitative sense for how similar feature subsets are for one web attack dataset as compared to a different web attack dataset after an ensemble of FSTs has been applied to each dataset.

New feature popularity framework

We present a new feature popularity framework which can identify popular features across different cyberattack datasets. For example, if we have three different web attacks we can identify the most popular features across these three different web attacks. Identifying the most popular features across different attack datasets enables application domain experts to gain new insights into the problem. By using fewer features (which are also the most important), models can be more explainable and less complex to implement and deploy. We introduce this new feature popularity framework to the cybersecurity domain, but it can be applied to any application domain which has multiple class labels in a dataset.

At a high level, our feature popularity framework is based upon an ensemble of ensembles with respect to underlying feature selection techniques and datasets. First, Feature

Importance Lists are generated from the rankings of various FSTs (such as filter-based and supervised FSTs) for each dataset. Second, an ensemble of FSTs is built for each dataset where a specified number of FSTs must agree for a feature to be included in a "FST Agreement List". This FST Agreement List includes features which are common across one particular dataset where a specified number of FSTs are in agreement for a particular feature to be included in this list. Third, a final ensemble across the different datasets constructs a "Feature Popularity List" where a feature can only be included into this list if it appears in a minimum number of "FST Agreement Lists" from the previous step. The following notations are defined to more easily provide examples of our feature popularity framework:

$$n_T = \text{Number of Top (cutoff) features per FST Ranker}$$
$$fst_T = \text{Total Number of FSTs}$$
$$fst_A = \text{Agreement criteria of FSTs}$$
$$ds_T = \text{Total Number of Datasets}$$
$$ds_A = \text{Agreement criteria of Datasets}$$

Figure 1 indicates the three main dimensions for our feature popularity framework. The "Feature Importance Dimension" is the first step of constructing the Feature Importance Lists across each dataset for each FST. Next, FST Agreement Lists are built in the "FST Agreement Dimension" for each dataset where a specified number of FSTs must agree (which is denoted by fst_A). Finally, Feature Popularity Lists are generated in the "Dataset Agreement Dimension" where a feature can only be included into a list if it appears in a minimum number of datasets (which is denoted by ds_A) for each level of fst_A. In the next section, we provide step by step examples to illustrate our new feature popularity framework.

Creating feature popularity lists with web attack datasets

In this section, we visually illustrate our feature popularity framework with tables of features being built at each step along with pseudocode listings to describe the process. For these examples, three datasets are constructed from CSE-CIC-IDS2018 as indicated in Table 2 for three different web attacks: Brute Force, SQL Injection, and XSS. For the

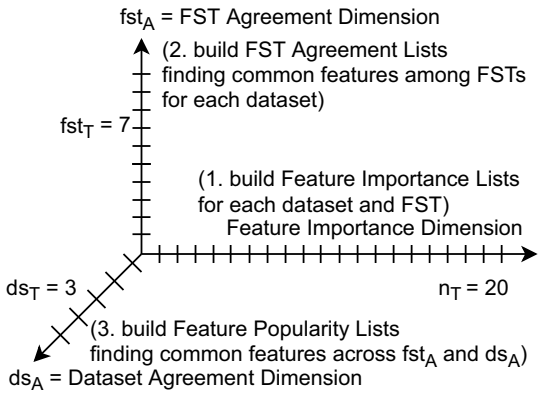

Fig. 1 Feature Popularity Dimensions

pseudocode listings, we initialize variables in Listing 1 where we define our variables for our earlier notations: n_T, fst_T, fst_A, ds_T, and ds_A.

Listing 1 Initialization for Feature Popularity Code

```
# computer pseudocode to demo feature popularity technique

datasets = ["web Brute Force", "web Sql Injection", "web XSS"]

feature_selection_techniques = ["Chi Squared", "Information Gain",
                                "Information Gain Ratio", "XGBoost",
                                "Random Forest", "CatBoost",
                                "LightGBM"]


fst_total_number = len(feature_selection_techniques) # fst_T = 7
datasets_total_number = len(datasets) # ds_T = 3

number_of_top_cutoff_features = 20 # n_T

fst_agreement_critera = [4,5,6,7] # fst_A
datasets_agreement_critera = [2, 3] # ds_A

feature_importance_lists = []
common_feature_lists = []
feature_popularity_lists = []
sublist_temp = []  # temporary list placeholder (code comprehension)
```

Create feature importance lists (Step 1)

Creating Feature Importance Lists is the first step of the process for building Feature Popularity Lists. There is nothing new in this step for machine learning practitioners who are familiar with feature selection. In this step, we simply generate Feature Importance Lists which are standard in many machine learning Python modules (such as the ones we include here for supervisor-based and filter-based FSTs).

For example, the idea is to use these FSTs as Rankers to only include the Top 20 features into our Feature Importance Lists (since n_T=20). We note that other experiments can easily consider using a different cutoff point for the number of top features to be included into these lists. Future work could also consider incorporating weighted numerical scores of feature importance into our feature popularity framework.

In our example, a total of 21 Feature Importance Lists will be constructed since we consider three datasets (ds_T=3) and seven FSTs (fst_T=7). The three web attack datasets are: BF, SQL, and XSS. Also as described in the "Methodologies" section, these seven FSTs are employed: Chi-Squared, Information Gain, Information Gain Ratio, Random Forest, XGBoost, Random Forest, CatBoost, and LightGBM. Listing 2 contains an example of the pseudocode to iterate over the three datasets and seven FSTs to create these 21 Feature Importance Lists. These 21 Feature Importance Lists are displayed in Tables 14, 15, 16, 17, 18, 19 of the Appendix.

Listing 2 Create Feature Importance Lists

```
# Generate 21 Feature Importance Lists (3 Datasets x 7 FSTs)

def GetFeatureImportanceList(ds, fst, number_of_top_cutoff_features):
    # ... return the Top 20 features for dataset and fst combination
    return ["feature1", "feature2", "feature3", "..."]

for i, ds in enumerate(datasets):
    for j, fst in enumerate(feature_selection_techniques):
        sublist_temp = GetFeatureImportanceList(ds, fst,
                        number_of_top_cutoff_features)
        feature_importance_lists.append([ds, fst, sublist_temp])
```

Create FST agreement lists (Step 2)

Generating FST Agreement Lists is the second step of the process for building Feature Popularity Lists. These FST Agreement Lists are built by identifying common features across the seven FSTs (denoted by fst_T) for each dataset, where a specified threshold of these FSTs must agree (denoted by fst_A). For example, if fst_A=4 then we build a FST Agreement List where at least four of the seven FSTs must agree in order for common features to appear in that particular list for that dataset.

Table 3 contains the four FST Agreement Lists for only the Brute Force web attack. There are four different FST Agreement Lists as we consider four different levels for fst_A: 4, 5, 6, and 7. Each numeric superscript in this table indicates the number of FSTs agreeing (fst_A) on that particular feature for that particular web attack. For example, the first listed feature Bwd_IAT_Mean of the Brute Force web attack is found to have exactly four of seven ("4/7") FSTs agree that it is a common feature between those FSTs for that web attack. In other words, Bwd_IAT_Mean occurs in exactly "4/7" Feature Importance Lists for the Brute Force web attack found in Tables 14 and 15 from the Appendix. Likewise, the Fwd_IAT_Total feature is found in all "7/7" FSTs for the Brute Force web attack. All of the other superscripts in Table 3 are found the same way by parsing their respective

Table 3 4 FST Agreement Lists for only Brute Force web attack (superscripts indicate fst_A, which is the number of FSTs agreeing on that feature)

BF 4/7 FSTs Agree	BF 5/7 FSTs Agree	BF 6/7 FSTs Agree	BF 7/7 FSTs Agree
Bwd_IAT_Mean [4]	Flow_Bytes_s [5]	TotLenFwdPkts [6]	Fwd_IAT_Total [7]
Flow_IAT_Mean [4]	Flow_IAT_Max [5]	Fwd_IAT_Total [7]	
Flow_IAT_Std [4]	Flow_Packets_s [5]		
FwdPktLenMean [4]	Fwd_IAT_Std [5]		
Subflow_Fwd_Bytes [4]	TotLenFwdPkts [6]		
Flow_Bytes_s [5]	Fwd_IAT_Total [7]		
Flow_IAT_Max [5]			
Flow_Packets_s [5]			
Fwd_IAT_Std [5]			
TotLenFwdPkts [6]			
Fwd_IAT_Total [7]			

Feature Importance Lists and finding the number of FSTs which agree upon a particular feature in a given web attack.

Next, Table 4 provides one consolidated listing for 12 different FST Agreement Lists. Each of the three web attack columns actually contains four different FST Agreement Lists, where these four lists simply differ by fst_A. We consolidated everything into one table so that parsing these lists is easier in the third (later) step. For example, the "4/7" FST Agreement List for BF contains all 11 features displayed in the BF column of Table 4 as it is actually inclusive of the FST Agreement Lists where 5, 6, and 7 FSTs agree as well. Table 3 is simply just a more detailed example of the four different FST Agreement Lists for the Brute Force column of Table 4. These tables are the simplest way to visually illustrate how FST Agreement Lists are created.

Listing 3 Create FST Agreement Lists

```
# Generate 12 FST Agreement Lists
# (4 FST Agreement Lists for each of the 3 Datasets)

def GetFstAgreementList(ds, fsta, feature_importance_lists):
    # ... return the FST Agreement List
    # for each dataset and fsta combination
    # (where fsta of fst_total_number "agree" for ds)
    return ["feature1", "feature2", "feature3", "..."]

for i, ds in enumerate(datasets):
    for fsta in fst_agreement_critera:
        sublist_temp = GetFstAgreementList(ds, fsta,
                                           feature_importance_lists)
        fst_agreement_lists.append([ds, fsta, sublist_temp])
```

Table 4 12 FST Agreement Lists for 3 Web Attacks with each having 4 levels of FST Agreement (numeric superscripts indicate fst_A, the number of FSTs agreeing on that feature for that attack)

BF	SQL	XSS
Bwd_IAT_Mean [4]	Bwd_IAT_Mean [4]	Bwd_IAT_Std [4]
Flow_IAT_Mean [4]	Bwd_Packets_s [4]	Bwd_Packets_s [4]
Flow_IAT_Std [4]	Flow_IAT_Std [4]	Flow_IAT_Min [4]
FwdPktLenMean [4]	FwdPktLenMax [4]	Flow_Packets_s [4]
Subflow_Fwd_Bytes [4]	FwdPktLenStd [4]	Fwd_IAT_Mean [4]
Flow_Bytes_s [5]	Packet_Length_Std [4]	Fwd_Packets_s [4]
Flow_IAT_Max [5]	BwdPktLenStd [5]	Max_Packet_Length [4]
Flow_Packets_s [5]	Flow_Bytes_s [5]	Packet_Length_Mean [4]
Fwd_IAT_Std [5]	Flow_IAT_Mean [5]	PktLenVar [4]
TotLenFwdPkts [6]	Flow_Packets_s [5]	Flow_Bytes_s [5]
Fwd_IAT_Total [7]	Fwd_IAT_Mean [5]	Fwd_IAT_Max [5]
	Fwd_IAT_Std [5]	Fwd_IAT_Total [5]
	Fwd_IAT_Total [5]	Packet_Length_Std [5]
	Max_Packet_Length [5]	TotLenFwdPkts [5]
	PktLenVar [5]	Flow_IAT_Max [6]
	Flow_IAT_Max [6]	Flow_IAT_Mean [6]
	Fwd_IAT_Max [6]	Fwd_IAT_Std [6]

Listing 3 illustrates the pseudocode for building FST Agreement Lists. First, we iterate over each dataset (ds_T=3) and build a specified number of FST Agreement Lists for each dataset. In our example, we have selected fst_A=4 to be the minimum agreement criteria for our ensemble FST otherwise too many features would have been included in our FST Agreement Lists. In other words, it would be undesirable to select fst_A=3 as many of those FST Agreement Lists would contain more than 20 features (n_T). We have also selected fst_A=7 to be our maximum agreement criteria. The inner loop of Listing 3 then iterates over each of our four values for fst_A={4,5,6,7}, effectively building four FST Agreement Lists for each of our three datasets.

Create feature popularity lists (Step 3)

Finally, creating the actual Feature Popularity Lists is the third and final step of the process for building Feature Popularity Lists. FST Agreement Lists from the prior step are used as the main input of this step. The best way to understand this process is through a specific example like the following. Table 5 indicates the two Feature Popularity Lists where four of seven datasets agree (fst_A=4 and fst_T=7). This table contains two different lists as we have one list where two of three web attack datasets agree (ds_A=2 and ds_T=3), and a second list where three of three datasets agree (ds_A=3 and ds_T=3).

The "2/3 Datasets & 4/7 FSTs Agree" Feature Popularity List in Table 5 is generated by parsing the three FST Agreement Lists from the prior step where fst_A=4. We include features into this new Feature Popularity Lists where at least two of the three datasets agree that it is a popular feature. For example, the Bwd_IAT_Mean feature is found in exactly two of the three FST Agreement Lists for web attacks where fst_A=4 and so it is included in the "2/3 Datasets & 4/7 FSTs Agree" list. However, Bwd_IAT_Mean is not included in the "3/3 Datasets & 4/7 FSTs Agree" list because it does not occur in all three FST Agreement Lists for web attacks where fst_A=4. Flow_Bytes_s is included in both the "2/3 Datasets & 4/7 FSTs Agree" and "3/3 Datasets & 4/7 FSTs Agree" Feature Popularity

Table 5 2 Feature Popularity Lists where 4/7 FSTs Agree (fst_A=4 and ds_A={2,3})

2/3 Datasets & 4/7 FSTs Agree	3/3 Datasets & 4/7 FSTs Agree
Bwd_IAT_Mean	Flow_Bytes_s
Bwd_Packets_s	Flow_IAT_Max
Flow_Bytes_s	Flow_IAT_Mean
Flow_IAT_Max	Flow_Packets_s
Flow_IAT_Mean	Fwd_IAT_Std
Flow_IAT_Std	Fwd_IAT_Total
Flow_Packets_s	
Fwd_IAT_Max	
Fwd_IAT_Mean	
Fwd_IAT_Std	
Fwd_IAT_Total	
Max_Packet_Length	
Packet_Length_Std	
PktLenVar	
TotLenFwdPkts	

Lists because it occurs in all three of the FST Agreement Lists where fst_A=4. In other words, the "3/3 Datasets" lists are a subset of the "2/3 Datasets" lists.

Next, in Table 6, Flow_Bytes_s also appears in both the '2/3 Datasets & 5/7 FSTs Agree" and "3/3 Datasets & 5/7 FSTs Agree" Feature Popularity Lists, and it is one of the most "popular features." The "5/7 FSTs Agree" lists are also a subset of the "4/7 FSTs Agree" lists. This is one of the nice characteristics of the feature popularity technique. We can keep adding more (or less) restrictive criteria for fst_A and ds_A until we have too few or too many features in our Feature Popularity Lists.

Feature Popularity Lists are mostly empty for fst_A=6 and fst_A=7 in Tables 7 and 8. The only feature which appears is Flow_IAT_Max, and it appears in the least restrictive of these four Feature Popularity Lists where fst_A=6 and ds_A=2. So, we can say that Flow_IAT_Max is the most popular feature overall. However, we did not consider running machine learning models with only one input feature as we thought that was too few for the purposes of this experiment. Overall, we construct eight Feature Popularity Lists with four levels of FST agreement criteria (fst_A={4,5,6,7}) and two levels of dataset Agreement criteria (ds_A={2,3}). However, we only conduct machine learning experiments with the four Feature Popularity Lists from Tables 5 and 6 as the other four Feature Popularity Lists were either empty or contained only one feature.

In Listing 4 we illustrate this last step with pseudocode, where the FST Agreement Lists from the prior step are used as the main input of building the Feature Popularity Lists. First, we iterate over all four values of our FST agreement criteria (fst_A={4,5,6,7}). Next in the inner loop, we iterate over our two values of dataset agreement criteria (ds_A={2,3}). For each value of ds_A, we are finding common (popular) features among the three datasets (ds_T=3) for the FST Agreement Lists for particular values of fst_A.

Table 6 2 Feature Popularity Lists where 5/7 FSTs Agree (fst_A=5 and ds_A={2,3})

2/3 Datasets & 5/7 FSTs Agree	3/3 Datasets & 5/7 FSTs Agree
Flow_Bytes_s	Flow_Bytes_s
Flow_IAT_Max	Flow_IAT_Max
Flow_IAT_Mean	Fwd_IAT_Std
Flow_Packets_s	Fwd_IAT_Total
Fwd_IAT_Max	
Fwd_IAT_Std	
Fwd_IAT_Total	
TotLenFwdPkts	

Table 7 2 Feature Popularity Lists where 6/7 FSTs Agree (fst_A=6 and ds_A={2,3})

2/3 Datasets & 6/7 FSTs Agree	3/3 Datasets & 6/7 FSTs Agree
Flow_IAT_Max	No Features (empty list)

Table 8 2 Feature Popularity Lists where 7/7 FSTs Agree (fst_A=7 and ds_A={2,3})

2/3 Datasets & 7/7 FSTs Agree	3/3 Datasets & 7/7 FSTs Agree
No Features (empty list)	No Features (empty list)

Table 9 Jaccard similarities by dataset for FST Agreement Lists where 4/7 FSTs Agree (fst_A=4)

	BF	SQL	XSS
BF	1.0	0.42105	0.38095
SQL	0.42105	1.0	0.61905
XSS	0.38095	0.61905	1.0

Table 10 Jaccard similarities by dataset for FST Agreement Lists where 5/7 FSTs Agree (fst_A=5)

	BF	SQL	XSS
BF	1.0	0.45455	0.6
SQL	0.45455	1.0	0.66667
XSS	0.6	0.66667	1.0

For example, with fst_A=4 and ds_A=2, we will find common (popular) features where 4 of 7 FSTs agree and 2 of 3 datasets agree. In this case, we will find common (popular) features among two (ds_A=2) of the three FST Agreement Lists (ds_T=3) having fst_A=4. From the second step of our example, only three FST Agreement Lists were generated where fst_A=4 (one for each dataset).

Listing 4 Create Feature Popularity Lists

```
# Generate 8 Feature Popularity Lists
# (4 fst_agreement_critera x 2 datasets_agreement_critera)

def GetFeaturePopularityList(fsta, dsa, fst_agreement_lists):
    # ... return the Feature Popularity List
    # for each fsta and dsa combination
    return ["feature1", "feature2", "feature3", "..."]

for fsta in fst_agreement_critera:
    for dsa in datasets_agreement_critera:
        sublist_temp = GetFeaturePopularityList(fsta, dsa,
                       fst_agreement_lists)
        feature_popularity_lists.append([fsta, dsa, sublist_temp])
```

Results and discussion

Dataset similarity

First, results are provided for Jaccard similarity scores between the FST Agreement Lists of the three different web attacks: Brute Force, SQL Injection, and XSS. Jaccard similarity scores are provided between these three web attacks for the following four different levels of FST Agreement criteria: fst_A={4,5,6,7}. Tables 9, 10, 11, 12 include the Jaccard similarity scores for the FST Agreement Lists of these three web attacks and varying levels of fst_A.

From Tables 9 and 10 where fst_A=4 and fst_A=5 respectively, we can easily observe that SQL and XSS have the most features in common between their respective subsets. However, we cannot easily determine which pairs of web attacks have the least amount of features in common for these two different values of FST agreement criteria. One pair

Table 11 Jaccard similarities by dataset for FST Agreement Lists where 6/7 FSTs Agree (fst_A=6)

	BF	SQL	XSS
BF	1.0	na	0.2
SQL	na	1.0	0.16667
XSS	0.2	0.16667	1.0

Table 12 Jaccard similarities by dataset for FST Agreement Lists where 7/7 FSTs Agree (fst_A=7); na values indicate there is no feature 7 out of 7 rankers agree on for at least one dataset in a pair

	BF	SQL	XSS
BF	1.0	na	na
SQL	na	1.0	na
XSS	na	na	1.0

(BF/XSS) has the lowest score for fst_A=4, while the other pair (BF/XSS) has the lowest score for fst_A=5. Regardless, these scores are close enough to each other for our purposes of generating Feature Popularity Lists as we were able to obtain a desirable amount of fewer and popular features in Tables 5 and 6 (where these lists had fewer than 20 features but more than 2-3 features). In the next section, we will employ machine learning to determine whether we have serious performance degradation with these lists of fewer features.

For the Jaccard similarity scores of Tables 11 and 12 where six or seven FSTs must agree for the FST Agreement Lists between the three different web attacks, we notice a sharp dropoff in Jaccard similarity scores. This is also evidenced in the Feature Popularity Lists for those FST agreement criteria in Tables 7 and 8 as well, which have mostly empty lists. In this regard, Jaccard similarity scores may help us understand more desirable FST agreement criteria thresholds to employ (by looking for steep dropoffs in scores).

Similarly, Jaccard similarity scores could also help us understand which different types of cyberattacks might be good candidates to generate Feature Popularity Lists with. Or, very low Jaccard similarity scores between certain cyberattacks could indicate they are not good candidates to group together within the same Feature Popularity Lists. And possibly, for different classes of attacks, they might be better suited to group them into separate Feature Popularity Lists. For example, if Denial of Service attack types as compared to web attack types obtained very divergent Jaccard similarity scores for their FST Agreement Lists, then maybe separate Feature Popularity Lists could be created for each different class of attack as appropriate.

This is an introductory study with our feature popularity framework and only three different attacks. But, employing these techniques to dozens or even hundreds of different types of cyberattacks might be even more helpful to properly group different cyberattacks into different Feature Popularity Lists by using Jaccard similarity scores of their respective FST Agreement Lists. Future work can extend these feature popularity frameworks towards many different types of cyberattacks, by grouping different types of cyberattacks into different groupings of Feature Popularity Lists. Collectively, additional

Feature Popularity List groupings for different cyberattacks might even improve classification performance. Although, at a minimum, it would provide better insights into the application domain problem with easier to explain models.

Feature popularity performance

In this section, classification performance results are provided for both before and after we apply our new feature popularity framework. Overall, we observe that classification performance is not degraded too much with our Feature Popularity Lists which have fewer features. In some cases, classification performance is even improved. Regardless of classification performance, employing Feature Popularity Lists is a powerful framework which enabled us to uncover previously unseen insights into the attack detection process with CSE-CIC-IDS2018 data.

Table 13 provides the classification performance results with four Feature Popularity Lists and "All Features" for five classifiers (CB, DT, LGB, RF, and XGB) with three different web attacks: BF, SQL, and XSS. For the FST column, "All Features" refers to the full feature set of 66 features (before any feature selection technique is applied), and the four Feature Popularity Lists comprise: "2/3 & 4/7 Agree", "2/3 & 5/7 Agree", "3/3 & 4/7 Agree", and "3/3 & 5/7 Agree". With these Feature Popularity Lists, the first fractional term refers to ds_A (specifying how many datasets agree) and the second fractional term refers to fst_A (specifying how many FSTs agree). Classification performance is presented in terms of AUC for these five different levels of Feature Popularity Lists in the FST column across the five classifiers for each of the three web attacks. The three different web attacks are represented as three columns in the table. "SD AUC" refers to the standard deviation for each AUC score. Top AUC scores are indicated in bold for each combination of: FST, classifier, and web attack.

Overall, when visually inspecting Table 13 we can see the classification performance for the Feature Popularity Lists is not seriously degraded. For 5 of the 15 classifier and web attack combinations, Feature Popularity Lists have higher scores as compared to "All Features". The best AUC scores of the Feature Popularity Lists are not more than 0.02 AUC lower than the "All Features" score. In other words, "All Features" AUC scores are not more than 0.02 AUC above the best score of the Feature Popularity Lists. In particular, the least restrictive "2/3 & 4/7 Agree" Feature Popularity List also does not perform worse than 0.02 AUC score of the "All Features" score. The other Feature Popularity Lists are just subsets of this "2/3 & 4/7 Agree" list (with fewer features due to more restrictive agreement criteria). Mostly throughout this experiment, classification performance is only mildly degraded by employing Feature Popularity Lists and performance is even improved in several cases.

Cybersecurity analysis and insights

A major benefit of feature popularity is providing domain experts with new insights from models which are more explainable. Our feature popularity framework led us to major discoveries into the web attack detection process within the CSE-CIC-IDS2018 dataset, even though we had intensely researched this dataset in prior work [21]. Based on our survey of other CSE-CIC-IDS2018 studies, none of them have identified these insights into the web attack detection process as of the date of this writing.

Table 13 Classification performance for 4 feature popularity lists (plus all features), 3 web attacks, and 5 classifiers

	Brute force (web)		SQL injection (web)		XSS (web)	
CatBoost						
FST	AUC	SD AUC	AUC	SD AUC	AUC	SD AUC
All Features	**0.93277**	0.00920	**0.90008**	0.02382	**0.93743**	0.01432
2/3 & 4/7 Agree	0.91121	0.01242	0.87876	0.03297	0.92574	0.01290
2/3 & 5/7 Agree	0.91121	0.01322	0.86604	0.02766	0.92791	0.01226
3/3 & 4/7 Agree	0.88749	0.01398	0.87531	0.03579	0.92627	0.01420
3/3 & 5/7 Agree	0.88858	0.01770	0.87057	0.03077	0.92167	0.01696
Decision Tree						
FST	AUC	SD AUC	AUC	SD AUC	AUC	SD AUC
All Features	**0.92252**	0.01355	0.90876	0.03032	0.93830	0.01244
2/3 & 4/7 Agree	0.90346	0.01235	0.91000	0.02972	0.93417	0.01456
2/3 & 5/7 Agree	0.91178	0.01391	0.91206	0.03209	**0.94020**	0.01389
3/3 & 4/7 Agree	0.87643	0.01820	**0.91527**	0.03260	0.93716	0.01701
3/3 & 5/7 Agree	0.87384	0.01813	0.91334	0.03429	0.93335	0.01301
LightGBM						
FST	AUC	SD AUC	AUC	SD AUC	AUC	SD AUC
All Features	**0.93863**	0.00975	**0.93499**	0.03401	**0.94622**	0.01416
2/3 & 4/7 Agree	0.93511	0.01052	0.91610	0.03595	0.94354	0.01320
2/3 & 5/7 Agree	0.93536	0.01078	0.90934	0.03810	0.94191	0.01491
3/3 & 4/7 Agree	0.93379	0.01159	0.90404	0.04282	0.94170	0.01360
3/3 & 5/7 Agree	0.93652	0.01084	0.91161	0.04016	0.94166	0.01347
Random Forest						
FST	AUC	SD AUC	AUC	SD AUC	AUC	SD AUC
All Features	**0.93945**	0.00910	**0.91773**	0.02602	0.94216	0.01316
2/3 & 4/7 Agree	0.93799	0.00944	0.91105	0.03448	**0.94455**	0.01214
2/3 & 5/7 Agree	0.93441	0.01089	0.90349	0.03102	0.94036	0.01592
3/3 & 4/7 Agree	0.92851	0.01070	0.89572	0.03404	0.94274	0.01418
3/3 & 5/7 Agree	0.92106	0.01235	0.90950	0.02971	0.93966	0.01546
XGBoost						
FST	AUC	SD AUC	AUC	SD AUC	AUC	SD AUC
All Features	**0.93668**	0.00891	0.89892	0.04456	0.93581	0.01363
2/3 & 4/7 Agree	0.92197	0.01063	0.89720	0.03518	0.92918	0.01379
2/3 & 5/7 Agree	0.92613	0.01357	**0.90861**	0.03699	**0.94058**	0.01425
3/3 & 4/7 Agree	0.91230	0.01552	0.90421	0.03625	0.93121	0.01521
3/3 & 5/7 Agree	0.91188	0.01677	0.90233	0.03867	0.92933	0.01460

Bold values indicate the best AUC score for each web attack and FST.

Our most restrictive "3/3 & 5/7 Agree" Feature Popularity List (ds_A=3 and fst_A=5) only includes the following four features from Table 6: Flow_Bytes_s (flow bytes per second), Flow_IAT_Max (maximum time between two flows), Fwd_IAT_Std (standard deviation time between two packets sent in the forward direction), and Fwd_IAT_Total (total time between two packets sent in the forward direction). Using only these four input features, our machine learning models from Table 13 achieved favorable classification performance which was nearly as good as the "All Features" dataset. All four of

these features are mainly based upon the time dimension. From a cybersecurity analyst's perspective, these four features do not truly signal SQL Injection or XSS web attacks. In other words, detection for these two web attacks is not primarily based upon temporal features. For the third and only other web attack label in CSE-CIC-IDS2018, it is not so clear whether Brute Force web attacks should be detected primarily on time-based features and so we will discuss this separately.

Attack characteristics of SQL Injection and XSS web attacks are mainly found in the application layer (7) of the OSI model [41], as the payloads for these two web attacks operate at protocols which are in layer 7 of the OSI model. The four features (Flow_Bytes_s, Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total) are features based on NetFlows [42, 43] and are operating at lower layers 3 and 4 of the OSI model. Overall, these four features are not indicating attack signatures for these web attacks, because their attack fingerprints occur in the application layer (7) of the OSI model.

For example, Flow_Bytes_s does not signal a SQL Injection or XSS web attack. The Flow_Bytes_s feature is merely indicating the number of bytes per second in a network flow. Normal web traffic can just as easily produce similar values for the Flow_Bytes_s feature, as compared to SQL Injection or XSS web attack traffic. In other words, the Flow_Bytes_s does not properly discriminate normal web traffic as compared to SQL Injection or XSS web attack traffic. Normal web traffic can just as easily have lower or higher values for Flow_Bytes_s as compared to SQL Injection or XSS web attack traffic.

One small and brief web request representing normal traffic could just as easily have similar values for Flow_Bytes_s as compared to a slow and stealthy web attack for either a SQL Injection or XSS web attack. This same logic applies towards moderate velocity normal traffic as compared to moderate velocity traffic for SQL Injection and XSS web attacks for the Flow_Bytes_s feature. While it could be argued that very high velocity traffic for the Flow_Bytes_s feature could be signalling web attacks such as for SQL Injection or XSS, this is simply not the case in the CSE-CIC-IDS2018 dataset as high velocity attack traffic does not exist for these two attack web labels. In CSE-CIC-IDS2018, the SQL Injection label only encompasses 87 instances and the XSS label only encompasses 230 labels. Plus, this approach would not detect slow and stealthy web attacks.

The other three features (Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total) have the identical problems as compared to Flow_Bytes_s in discriminating between normal web traffic and SQL Injection and XSS web attacks. These features are all signaling information from layers 3 and 4 of the OSI model, and not layer 7 of the OSI model. Plus, these four features are heavily focused on the time dimension. SQL Injection and XSS web attacks do not typically have characteristics which are based on temporal features (especially when executed in a slow and stealthy fashion by attackers seeking to avoid detection). Instead of detecting these classes of web attacks based on time, other attack characteristics could be used such as those found in the application layer. Better examples of attack characteristics for these classes of web attacks are parsing text payloads for malicious sequences of characters or monitoring error logs (both are in the application layer of the OSI model).

Then, the question arises of what could be signaling such good detection of SQL Injection and XSS web attacks within the CSE-CIC-IDS2018 dataset. We can only speculate

on this question, as this question deserves further research. One possibility could be unintentional contamination in the data collection process, where the machine learning models are detecting patterns that are discriminating between attack and normal traffic based on temporal patterns of the data collection and not the underlying signatures of the web attacks. Future work can further investigate this phenomenon.

With regards to the Brute Force web attacks, the same arguments are true as for the SQL Injection and XSS web attacks that these four (Flow_Bytes_s, Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total) features do not necessarily signal a web attack. It is true that these four features might signal a Brute Force web attack during a very extreme scenario of massive web traffic spikes. An example for this would be a Brute Force attack which is similar to a Denial of Service attack where the attacker is causing a massive flood of web traffic. However, this approach would not detect Brute Force web attacks which are more slow and stealthy in nature. Many attackers seek to evade detection, and only using these four features would effectively miss detecting one of the most important classes of attacker adversaries (those seeking to avoid detection).

Most importantly, the CSE-CIC-IDS2018 dataset only contains 611 labels for Brute Force web attacks as compared to over 2 million "Normal" labels for those two days of the data collection for web attacks. Given the small fraction of Brute Force web attack labels of 0.03% compared to the normal traffic for those two days, our machine learning models are not detecting some sort of "flood" type of Brute Force web attack. Instead, our machine learning models are likely detecting other patterns regarding the data collection which requires future research.

Even for Brute Force web attacks, the higher application layer (7) of the OSI model contains better attack characteristics as compared to the lower network layers of 3 and 4 (containing NetFlow features). The OWASP Top 10 [11] contains two items on how to handle Brute Force web attacks at the application layer. First, "OWASP A2:2017-Broken Authentication" [44] indicates that web applications should "limit or increasingly delay failed login attempts" and "log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected". Second, "OWASP A10:2017-Insufficient Logging & Monitoring" [45] highlights that "exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident". Essentially, properly designed web applications would remove "flood" types of Brute Force web attacks by increasingly delaying their logins. Sensors from application layer logs would still be best equipped to detect Brute Force web attacks which are more slow and stealthy.

Even though we have obtained respectable classification results for web attacks in this study, our newly conceived feature popularity framework allowed us to realize that the features we were detecting upon did not make sense from a cybersecurity analyst's perspective. When looking at all 79 independent features of the downloaded CSE-CIC-IDS2018 dataset, it can be difficult for a cybersecurity analyst to ascertain whether those NetFlow-based features might be good candidates in detecting web attacks. Even after generating a myriad of Feature Importance Lists in Tables 14, 15, 16, 17, 18, 19, it still was not clear as these lists of features were very divergent from each other. After employing feature popularity which enabled us to visualize more explainable models, we could

then ascertain that our top four features (Flow_Bytes_s, Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total) did not properly characterize the web attack signatures in question. Overall, future research can further answer the question of whether or not Netflow-based features are even good candidates for detecting web attacks from the application layer of the OSI model.

Conclusion

Feature popularity is a novel framework that we introduce in this study, and we implement it with CSE-CIC-IDS2018 big data and the following three web attacks: Brute Force, SQL Injection, and XSS. These three web attacks are partitioned into three separate datasets so that we can employ feature popularity. For our underlying feature selection techniques, we use three filter-based rankers and four supervised-based rankers: Chi Squared, Information Gain, Information Gain Ratio, CB, LGB, RF, and XGB.

First, we generate Feature Importance Lists where the top 20 features are generated for our three web attack datasets and our seven (FST) rankers. Second, we create FST Agreement Lists which find the common features across each dataset's Feature Importance Lists according to varying levels of FST agreement criteria (fst_A) among our seven FSTs. Third, we build Feature Popularity Lists from FST Agreement Lists according to varying levels of dataset agreement criteria ($ds_A=\{2,3\}$) and FST agreement criteria ($fst_A=\{4,5,6,7\}$). Our feature popularity technique effectively builds an ensemble of ensembles by first building an ensemble of FSTs for each dataset, and then building another ensemble across a dataset agreement dimension.

We also introduce the use of the Jaccard similarity score with our FST Agreement Lists to give a quantitative sense for how similar or not various pairs of FST Agreement Lists compare to each other across different FST agreement criteria (fst_A). Employing Jaccard similarity scores in this manner becomes more important as many more datasets are considered with the feature popularity framework. These Jaccard similarity scores can help decide which classes of attacks should be grouped together with feature popularity, versus those which should be broken apart into different feature popularity groupings.

Classification performance did not seriously degrade with Feature Popularity Lists (which contain fewer features), as compared to the "All Features" list. In 5 out of 15 cases, Feature Popularity Lists even fared better as compared to the "All Features" list. AUC scores did not degrade by more than 0.02 for the best of the Feature Popularity List groupings as compared to the "All Features" list. Classification performance was evaluated with the following five classifiers: CB, DT, LGB, RF, and XGB. Overall LightGBM performed the best, and classification performance held especially well for Feature Popularity Lists with LGB.

Not only does feature popularity produce models which are easier to understand and implement, but it can also provide new insights to application domain experts. Even though we had been working intensely with web attacks from CSE-CIC-IDS2018, we did not discover new realizations until working with models produced with feature popularity and results from its most popular features. With our feature popularity experiment

and underlying FSTs, the four most popular features for CSE-CIC-IDS2018 web attacks are: Flow_Bytes_s, Flow_IAT_Max, Fwd_IAT_Std, and Fwd_IAT_Total.

When using only these four most popular features as input to our classifiers, we still achieved nearly the same favorable classification performance as compared to the "All Features" list. We realized these four features do not signal attack characteristics for our three web attacks: Brute Force, SQL Injection, and XSS. These four features are mainly based upon the time dimension and Netflow-based attributes from layers 3 and 4 of the OSI model.

However, our three web attacks should not be leaving signatures in these four features as they operate at the application layer (7) of the OSI model. Instead, something other than attack signatures of these three web attacks is causing them to be correctly classified as attacks. Future work can evaluate whether unintentional contamination in the data collection patterns for CSE-CIC-IDS2018 is signalling these web attacks. Also, future work can consider whether NetFlow-based features can legitimately signal web attack payloads from the application layer (7) of the OSI model in terms of forensic evidence.

Feature popularity is a powerful new framework which can be applied to any application domain. Any multi-class classification problem (containing more than two classes), can be decomposed with feature popularity so that the most popular features for its multiple classes can be discovered through ensembles across a new dataset agreement dimension (ds_A) as well as a FST agreemention dimension (fst_A). The beauty of the feature popularity technique, is that its agreement criteria for ds_A and fst_A can be tuned by a practitioner until the desired classification performance is achieved with a more desirable number of fewer and more popular features.

Future work can consider additional application domains as well as additional types of cyberattacks, as feature popularity is a flexible framework that can accommodate any application domain or cyberattack dataset which have multiple class labels (more than binary classification). Other FSTs and classifiers can also be implemented with feature popularity. Additionally, different cutoff points for the "Top N" features of the Feature Importance Lists could be investigated. Feature stability could also be explored with feature popularity to determine its effectiveness with the same most popular features as datasets evolve with new data over time, and whether this ensemble approach provides better feature stability as compared to more simplistic feature selection techniques. Finally, future research could develop these feature popularity frameworks in an automated manner through open-source tools to allow easier exploration of popular features across different cyberattacks or datasets.

Appendix

See Tables 14, 15, 16, 17, 18, 19.

Table 14 Top 20 features for Brute Force Web Attacks ranked by Filter-based techniques

Chi Squared	Information Gain	Information Gain Ratio
Idle_Std	Fwd_Packets_s	Fwd_Packets_s
Fwd_IAT_Total	Flow_Packets_s	Flow_Packets_s
Flow_IAT_Std	Fwd_IAT_Mean	Fwd_IAT_Mean
Idle_Max	Flow_IAT_Mean	Flow_IAT_Mean
Flow_IAT_Max	Fwd_IAT_Total	Fwd_IAT_Total
Fwd_IAT_Std	Flow_IAT_Std	Flow_IAT_Std
Active_Min	Fwd_IAT_Std	Fwd_IAT_Std
Fwd_IAT_Max	Flow_IAT_Max	Flow_IAT_Max
Active_Mean	Flow_Bytes_s	Flow_Bytes_s
Bwd_IAT_Max	Fwd_IAT_Max	Fwd_IAT_Max
Bwd_IAT_Total	Packet_Length_Mean	Packet_Length_Mean
Flow_IAT_Mean	Average_Packet_Size	Average_Packet_Size
Fwd_IAT_Mean	FwdPktLenMean	FwdPktLenMean
Active_Max	AvgFwdSegSize	AvgFwdSegSize
Idle_Min	Max_Packet_Length	Max_Packet_Length
Bwd_IAT_Mean	Bwd_Packets_s	Bwd_Packets_s
Bwd_IAT_Min	TotLenFwdPkts	TotLenFwdPkts
TotLenFwdPkts	Subflow_Fwd_Bytes	Subflow_Fwd_Bytes
Subflow_Fwd_Bytes	Packet_Length_Std	Packet_Length_Std
Bwd_IAT_Std	PktLenVar	PktLenVar

Table 15 Top 20 features for Brute Force web attacks ranked by Supervised-based Feature Importance Lists

XGBoost	Random Forest	CatBoost	LightGBM
Fwd_IAT_Min	AvgFwdSegSize	Idle_Min	FwdPktLenMean
RST_Flag_Count	TotLenBwdPkts	FwdPktLenStd	Flow_IAT_Min
Fwd_IAT_Total	Max_Packet_Length	ECE_Flag_Count	Fwd_IAT_Min
Flow_Packets_s	FwdPktLenMean	Flow_Bytes_s	Bwd_IAT_Min
Flow_IAT_Max	RST_Flag_Count	Fwd_IAT_Total	Flow_IAT_Std
act_data_pkt_fwd	TotLenFwdPkts	RST_Flag_Count	FwdPktLenStd
Bwd_IAT_Mean	Flow_Bytes_s	Active_Max	Flow_Bytes_s
Bwd_IAT_Min	Bwd_IAT_Max	Fwd_Header_Length	Fwd_IAT_Total
	Subflow_Fwd_Bytes	BwdPktLenStd	Bwd_Packets_s
	FwdPktLenStd	Idle_Std	Bwd_IAT_Mean
	AvgBwdSegSize	Min_Packet_Length	Flow_Packets_s
	Bwd_IAT_Mean	Active_Min	Fwd_IAT_Std
	Fwd_IAT_Std	TotLenFwdPkts	Fwd_Packets_s
	Idle_Mean	act_data_pkt_fwd	Flow_IAT_Max
	Subflow_Bwd_Packets		TotLenFwdPkts
	Bwd_IAT_Std		Idle_Std
	Fwd_IAT_Total		Fwd_Header_Length
	BwdPktLenMean		Flow_IAT_Mean
	Fwd_IAT_Min		FwdPktLenMax
	Flow_Packets_s		Total_Backward_Packets

Table 16 Top 20 features for SQL Injection web attacks ranked by Filter-based techniques

Chi Squared	Information Gain	Information Gain Ratio
Fwd_IAT_Total	Fwd_Packets_s	Fwd_Packets_s
Idle_Max	Flow_Packets_s	Flow_Packets_s
Bwd_IAT_Total	Flow_IAT_Mean	Flow_IAT_Mean
Idle_Mean	Fwd_IAT_Mean	Fwd_IAT_Mean
Idle_Min	Fwd_IAT_Total	Fwd_IAT_Total
Fwd_IAT_Max	Bwd_Packets_s	Bwd_Packets_s
Fwd_IAT_Min	Flow_IAT_Max	Flow_IAT_Max
Fwd_IAT_Mean	Flow_IAT_Std	Flow_IAT_Std
Flow_IAT_Min	Fwd_IAT_Max	Fwd_IAT_Max
Flow_IAT_Mean	Fwd_IAT_Std	Fwd_IAT_Std
Flow_IAT_Max	Flow_Bytes_s	Flow_Bytes_s
PktLenVar	PktLenVar	PktLenVar
Bwd_IAT_Max	Packet_Length_Std	Packet_Length_Std
Bwd_IAT_Min	Bwd_IAT_Std	Bwd_IAT_Std
Active_Max	Bwd_IAT_Mean	Bwd_IAT_Mean
Flow_Bytes_s	FwdPktLenStd	FwdPktLenStd
Fwd_IAT_Std	Bwd_IAT_Total	Bwd_IAT_Total
Active_Mean	BwdPktLenStd	BwdPktLenStd
Idle_Std	Packet_Length_Mean	Packet_Length_Mean
Bwd_IAT_Mean	Max_Packet_Length	Max_Packet_Length

Table 17 Top 20 features for SQL Injection web attacks ranked by Supervised-based Feature Importance Lists

XGBoost	Random Forest	CatBoost	LightGBM
Max_Packet_Length	BwdPktLenMax	ECE_Flag_Count	Flow_IAT_Min
BwdPktLenMax	Flow_Packets_s	Fwd_IAT_Max	Flow_Bytes_s
Bwd_IAT_Min	Flow_IAT_Mean	PSH_Flag_Count	Fwd_IAT_Min
TotLenFwdPkts	RST_Flag_Count	Max_Packet_Length	BwdPktLenStd
RST_Flag_Count	FwdPktLenMax	Total_Fwd_Packets	Bwd_Packets_s
Packet_Length_Mean	AvgBwdSegSize	Bwd_IAT_Max	Bwd_IAT_Min
Flow_IAT_Max	Total_Backward_Packets	Subflow_Bwd_Bytes	FwdPktLenMean
Fwd_Header_Length	Max_Packet_Length	BwdPktLenStd	Fwd_IAT_Total
FwdPktLenStd	ECE_Flag_Count	AvgBwdSegSize	Flow_IAT_Mean
Bwd_Packets_s	BwdPktLenMean	FwdPktLenMax	FwdPktLenMax
Packet_Length_Std	Subflow_Bwd_Packets	Subflow_Bwd_Packets	Flow_Packets_s
Bwd_IAT_Max	Fwd_IAT_Total	Bwd_Header_Length	Flow_IAT_Max
FwdPktLenMax	BwdPktLenStd	TotLenBwdPkts	Fwd_IAT_Mean
Flow_Bytes_s	Fwd_IAT_Max	Down_Up_Ratio	TotLenFwdPkts
Bwd_IAT_Mean	FwdPktLenMean		FwdPktLenStd
Fwd_IAT_Std	Flow_IAT_Std		Fwd_Packets_s
Fwd_IAT_Min	min_seg_size_forward		Fwd_IAT_Std
FwdPktLenMean	PktLenVar		Flow_IAT_Std
Flow_Packets_s	Packet_Length_Std		Fwd_IAT_Max
Fwd_IAT_Mean	Flow_IAT_Max		PktLenVar

Table 18 Top 20 features for XSS web attacks ranked by Filter-based techniques

Chi Squared	Information Gain	Information Gain Ratio
Bwd_IAT_Total	Fwd_Packets_s	Fwd_Packets_s
Fwd_IAT_Total	Flow_Packets_s	Flow_Packets_s
Idle_Min	Fwd_IAT_Mean	Fwd_IAT_Mean
Idle_Mean	Fwd_IAT_Total	Fwd_IAT_Total
Flow_IAT_Max	Flow_IAT_Mean	Flow_IAT_Mean
Fwd_IAT_Max	Flow_IAT_Std	Flow_IAT_Std
Idle_Max	Fwd_IAT_Std	Fwd_IAT_Std
Fwd_IAT_Mean	Fwd_IAT_Max	Fwd_IAT_Max
Fwd_IAT_Min	Flow_IAT_Max	Flow_IAT_Max
Flow_IAT_Min	Flow_Bytes_s	Flow_Bytes_s
Flow_IAT_Mean	Bwd_Packets_s	Bwd_Packets_s
Bwd_IAT_Max	PktLenVar	PktLenVar
TotLenFwdPkts	Packet_Length_Std	Packet_Length_Std
Subflow_Fwd_Bytes	Packet_Length_Mean	Packet_Length_Mean
PktLenVar	Average_Packet_Size	Average_Packet_Size
Subflow_Bwd_Bytes	FwdPktLenMean	FwdPktLenMean
TotLenBwdPkts	AvgFwdSegSize	AvgFwdSegSize
Bwd_IAT_Std	TotLenFwdPkts	TotLenFwdPkts
Fwd_IAT_Std	Subflow_Fwd_Bytes	Subflow_Fwd_Bytes
Bwd_IAT_Mean	Max_Packet_Length	Max_Packet_Length

Table 19 Top 20 features for XSS web attacks ranked by Supervised-based Feature Importance Lists

XGBoost	Random Forest	CatBoost	LightGBM
BwdPktLenMean	BwdPktLenMax	Flow_Bytes_s	Flow_IAT_Min
Total_Backward_Packets	BwdPktLenMean	Max_Packet_Length	Fwd_IAT_Min
TotLenFwdPkts	Max_Packet_Length	Destination_Port	Flow_IAT_Mean
FIN_Flag_Count	Fwd_Packets_s	Fwd_IAT_Std	Fwd_Packets_s
Total_Fwd_Packets	Packet_Length_Std	Flow_IAT_Mean	Flow_Bytes_s
Flow_IAT_Min	ECE_Flag_Count	Bwd_IAT_Max	Flow_Packets_s
BwdPktLenMax	Bwd_IAT_Std	Bwd_IAT_Mean	Flow_IAT_Max
Bwd_IAT_Mean	Idle_Min	AvgBwdSegSize	Bwd_Packets_s
Flow_Packets_s	AvgBwdSegSize	TotLenFwdPkts	Fwd_IAT_Total
Down_Up_Ratio	Subflow_Bwd_Packets	Idle_Std	Fwd_IAT_Std
Packet_Length_Std	Fwd_IAT_Total	Bwd_IAT_Std	Fwd_IAT_Max
Flow_Bytes_s	Flow_IAT_Mean	Flow_IAT_Max	Flow_IAT_Std
ACK_Flag_Count	Flow_IAT_Max	Active_Min	Bwd_IAT_Std
RST_Flag_Count	Fwd_IAT_Min	act_data_pkt_fwd	Fwd_IAT_Mean
Bwd_IAT_Max	Bwd_Header_Length	min_seg_size_forward	BwdPktLenMin
Idle_Max	Flow_IAT_Min		Fwd_Header_Length
Fwd_IAT_Std	Active_Mean		Bwd_IAT_Min
Packet_Length_Mean	Idle_Max		Packet_Length_Std
Active_Min	Packet_Length_Mean		Bwd_IAT_Total
Bwd_Packets_s	Fwd_IAT_Max		PktLenVar

Abbreviations

XAI	eXplainable Artificial Intelligence
FST	Feature Selection Technique
RUS	Random undersampling
DT	Decision Tree
RF	Random Forest
CB	CatBoost
LGB	LightGBM
XGB	XGBoost
NB	Naive Bayes
LR	Logistic Regression
GBT	Gradient Boosted Tree
AUC	Area Under the Receiver Operating Characteristic Curve
ROC	Receiver Operator Characteristic
TPR	True Positive Rate
FPR	False Positive Rate
GOSS	Gradient-based One-Side Sampling
EFB	Exclusive Feature Bundling
BF	Brute Force
XSS	Cross-Site Scripting
FIL	Feature Importance List
FAL	FST Agreement List
FPL	Feature Popularity List
OWASP	Open Web Application Security Project
OHE	One Hot Encoding

Acknowledgements
We would like to thank the reviewers in the Data Mining and Machine Learning Laboratory at Florida Atlantic University.

Author Contributions
RZ prepared the manuscript and the primary literary review for this work. JH performed the statistical analyses. All authors provided feedback to TMK and helped shape the research. TMK introduced this topic to RZ, and helped to complete and finalize this work. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 21 March 2022 Accepted: 20 October 2022
Published online: 15 December 2022

References

1. Young J. US ecommerce sales grow 14.9% in 2019. 2020. https://www.digitalcommerce360.com/article/us-ecommerce-sales/. Accessed 28 Nov 2020
2. Saeys Y, Abeel T, Van de Peer Y Robust feature selection using ensemble feature selection techniques. In: Joint European conference on machine learning and knowledge discovery in databases. Springer; 2008. p. 313–325
3. Seijo-Pardo B, Porto-Díaz I, Bolón-Canedo V, Alonso-Betanzos A. Ensemble feature selection: homogeneous and heterogeneous approaches. Knowl Based Syst. 2017;118:124–39.
4. Kalousis A, Prados J, Hilario M. Stability of feature selection algorithms: a study on high-dimensional spaces. Knowl Inf Syst. 2007;12(1):95–116.
5. Zuech R, Hancock J, Khoshgoftaar TM. Feature popularity between different web attacks with supervised feature selection rankers. In: 2021 20th IEEE international conference on machine learning and applications (ICMLA). IEEE; 2021. p. 30–37
6. Sharafaldin I, Lashkari AH, Ghorbani AA. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. 2018. p. 108–116
7. CICIDS2017 Dataset. 2020. https://www.unb.ca/cic/datasets/ids-2017.html. Accessed 28 Aug 2020

8. Leevy JL, Khoshgoftaar TM, Bauder RA, Seliya N. A survey on addressing high-class imbalance in big data. J Big Data. 2018;5(1):1–30.
9. Soltysik RC, Yarnold PR. Megaoda large sample and big data time trials: separating the chaff. Optim Data Anal. 2013;2:194–7.
10. Cao M, Chychyla R, Stewart T. Big data analytics in financial statement audits. Account Horiz. 2015;29(2):423–9.
11. OWASP Top Ten webpage. 2020. https://owasp.org/www-project-top-ten/. Accessed 10 Aug 2021
12. Sarhan M, Layeghy S, Portmann M. An explainable machine learning-based network intrusion detection system for enabling generalisability in securing iot networks. arXiv preprint arXiv:2104.07183 2021
13. Leevy JL, Hancock J, Zuech R, Khoshgoftaar TM. Detecting cybersecurity attacks across different network features and learners. J Big Data. 2021;8(1):1–29.
14. Fitni QRS, Ramli K. Implementation of ensemble learning and feature selection for performance improvements in anomaly-based intrusion detection systems. In: 2020 IEEE international conference on industry 4.0, artificial intelligence, and communications technology (IAICT). IEEE; 2020. p. 118–124.
15. Beechey M, Kyriakopoulos KG, Lambotharan S. Evidential classification and feature selection for cyber-threat hunting. Knowl Based Syst. 2021;226:107120.
16. Hua Y. An efficient traffic classification scheme using embedded feature selection and lightgbm. In: 2020 information communication technologies conference (ICTC). IEEE; 2020. p. 125–130.
17. Zhang H, Huang L, Wu CQ, Li Z. An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset. Comput Netw. 2020;177: 107315.
18. CSE-CIC-IDS2018 Dataset. 2020. https://www.unb.ca/cic/datasets/ids-2018.html. Accessed 28 Aug 2020
19. Arlot S, Celisse A, et al. A survey of cross-validation procedures for model selection. Stat Surv. 2010;4:40–79.
20. Kohavi R, et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai. Montreal, Canada; 1995. p. 14, 1137–1145 .
21. Zuech R, Hancock J, Khoshgoftaar TM. Investigating rarity in web attacks with ensemble learners. J Big Data. 2021;8(1):1–27.
22. Scikit-learn website. 2020. https://scikit-learn.org/stable/. Accessed 30 Jan 2021
23. Myles AJ, Feudale RN, Liu Y, Woody NA, Brown SD. An introduction to decision tree modeling. J Chemom J Chemom Soc. 2004;18(6):275–85.
24. Raileanu LE, Stoffel K. Theoretical comparison between the gini index and information gain criteria. Ann Math Artif Intell. 2004;41(1):77–93.
25. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32.
26. Breiman L. Bagging predictors. Mach Learn. 1996;24(2):123–40.
27. CatBoost home page. 2020. https://catboost.ai/. Accessed 28 Aug 2020
28. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A. Catboost: unbiased boosting with categorical features. In: Advances in neural information processing systems. 2018. p. 6638–6648
29. LightGBM GitHub website. 2020. https://github.com/microsoft/LightGBM. Accessed 28 Aug 2020
30. Natekin A, Knoll A. Gradient boosting machines, a tutorial. Front Neurorobot. 2013;7:21.
31. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T-Y. Lightgbm: a highly efficient gradient boosting decision tree. In: Advances in neural information processing systems. 2017. p. 3146–3154
32. Chen T, Guestrin C. Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd international conference on knowledge discovery and data mining. 2016. p. 785–794
33. Guo C, Berkhahn F. Entity embeddings of categorical variables. 2016. arXiv preprint arXiv:1604.06737
34. Scikit-learn Documentation—Feature Selection. 2020. https://scikit-learn.org/stable/modules/feature_selection.html. Accessed 16 Aug 2021
35. Zien A, Krämer N, Sonnenburg S, Rätsch G. The feature importance ranking measure. In: Joint European conference on machine learning and knowledge discovery in databases. Springer; 2009. p. 694–709 .
36. Scikit-learn Documentation - chi2 Feature Selection. 2020. https://scikit-learn.org/stable/modules/generated/sklearn-.feature_selection.chi2.html. Accessed 16 Aug 2021
37. Vergara JR, Estévez PA. A review of feature selection methods based on mutual information. Neural Comput Appl. 2014;24(1):175–86.
38. Mohammad AH. Comparing two feature selections methods information gain and gain ratio on three different classification algorithms using arabic dataset. J Theor Appl Inf Technol 2018; 96(6)
39. info_gain Pypi project. 2020. https://pypi.org/project/info-gain/. Accessed 16 Aug 2021.
40. Bradley AP. The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recogn. 1997;30(7):1145–59.
41. Day JD, Zimmermann H. The OSI reference model. Proc IEEE. 1983;71(12):1334–40.
42. Lashkari AH, Draper-Gil G, Mamun MSI, Ghorbani AA. Characterization of tor traffic using time based features. In: ICISSp, 2017. p. 253–262
43. Draper-Gil G, Lashkari AH, Mamun MSI, Ghorbani AA. Characterization of encrypted and vpn traffic using time-related. In: Proceedings of the 2nd international conference on information systems security and privacy (ICISSP). 2016. p. 407–414
44. OWASP A2:2017-Broken Authentication. 2020. https://owasp.org/www-project-top-ten/2017/A2_2017-Broken-_Authentication. Accessed 10 Aug 2021
45. OWASP A10:2017-Insufficient Logging & Monitoring. 2020. https://owasp.org/www-project-top-ten/2017/A10_2017--Insufficient_Logging%2526Monitoring. Accessed 10 Aug 2021