Journal of Big Data

RESEARCH Open Access

# Impact of class distribution on the detection of slow HTTP DoS attacks using Big Data

Chad L. Calvert* and Taghi M. Khoshgoftaar

*Correspondence:
ccalver3@fau.edu
Florida Atlantic University,
777 Glades Road, Boca Raton,
FL, USA

**Abstract**

The integrity of modern network communications is constantly being challenged by more sophisticated intrusion techniques. Attackers are consistently shifting to stealthier and more complex forms of attacks in an attempt to bypass known mitigation strategies. In recent years, attackers have begun to focus their attack efforts on the application layer, allowing them to produce attacks that can exploit known issues within specific application protocols. Slow HTTP Denial of Service attacks are one such attack variant, which targets the HTTP protocol and can imitate legitimate user traffic in order to deny resources from a service. Successful mitigation of this attack type requires network analysts to evaluate large quantities of network traffic to identify and block intrusive traffic. The issue, is that the number of legitimate traffic instances can far outnumber the amount of attack instances, making detection problematic. Machine learning techniques can be used to aid in detection, but the large level of imbalance between normal (majority) and attack (minority) instances can lead to inaccurate detection results. In this work, we evaluate the use of data sampling to produce varying class distributions in order to counteract the effects of severely imbalanced Slow HTTP DoS big datasets. We also detail our process for collecting real-world representative Slow HTTP DoS attack traffic from a live network environment to create our datasets. Five class distributions are generated to evaluate the Slow HTTP DoS detection performance of eight machine learning techniques. Our results show that the optimal learner and class distribution combination is that of Random Forest with a 65:35 distribution ratio, obtaining an AUC value of 0.99904. Further, we determine through the use of significance testing, that the use of sampling techniques can significantly increase learner performance when detecting Slow HTTP DoS attack traffic.

**Keywords:** Class imbalance, Slow HTTP DoS, Class imbalance, Big Data

## Introduction

The continued rise of network and cloud-based applications allow for companies to provide robust and dynamic services to their customers across the world both cheaply and efficiently. In 2018, nearly 80% of enterprises have begun to experiment with apps running on cloud or network based platforms [1]. Due to the growing reliance on such services, the security of the networks hosting these applications is more paramount than ever. Attackers are very much aware of the importance these applications hold for various companies, making them obvious targets for malicious exploits. Industry domains ranging from telecommunications to software publishing are finding themselves under constant bombardment from cyber attacks such as *distributed denial of service (DDoS)* attacks [2, 3]. According to the NETSCOUT Threat Intelligence Report for 2018, the overall number of DDoS attacks has increased by nearly 26% over the previous year [4]. Although continued efforts are in effect to mitigate such attacks, attackers are constantly evolving their attack methodologies to avoid the newest detection techniques. In recent years, attackers have moved towards the implementation of application layer attacks, as these have proven harder to detect as opposed to traditional network layer based approaches [5]. These attacks target specific application protocols being utilized by the target, many of which are prone to proven exploits.

One such protocol with known exploits is the *Hypertext Transfer Protocol (HTTP)*, which is often the target of attacks such as Slowloris and Slow POST DoS attacks [6, 7]. Both of these attack variants are easily executed by attackers, and are difficult to detect as the resulting traffic is difficult to differentiate from normal HTTP communications [8]. Because of this, these attacks can prove extremely detrimental to any service being hosted on a network utilizing this application protocol.

One component to mitigating network attacks is continuous network traffic analysis, which is an important aspect of maintaining proper network security. During this process, organizations often encounter enormous amounts of normal traffic during daily operations. Parsing this big data for the occurrences of malicious traffic can be a daunting effort, as the total number of traffic instances encountered can easily range in the millions. To aid in this detection, analysts often turn to machine learning techniques. However, depending on the type of attack, the resulting traffic may only contain a relatively low number of attack instances. This can lead to a relatively high level of class imbalance, where the minority class (attack) instances are vastly fewer than the majority class (normal) instances. As the size of the dataset increases, this issue is often exacerbated as the gap between number of minority and majority instances increases [9]. This occurrence can be even more predominant in network attack datasets, considering the large amount of network communications which can be collected. The issue of high or severe class imbalance can prove especially problematic when attempting to apply machine learners to detect attack instances. Since so few instances may exist, there may not be enough discernible behavioral patterns to correctly identify the other malicious instances within the remaining data. One approach to mitigate the issue of class imbalance is the use of data sampling techniques, which aims to change the ratio of minority and majority instances by selecting a subset of the original data, thus lessening the perceived severity between the two classes.

In this work, we evaluate the benefit of applying data sampling to a severely imbalanced big dataset from the cyber security domain. Our collected dataset consists of two Slow HTTP DoS attacks (Slowloris and Slow POST), implemented on a live production network. The resulting dataset consists of approximately 1.89 million instances, with 6646 total minority (attack) instances. As the minority class ratio is less than 0.01%, we consider this dataset severely imbalanced [10]. Although the minimum number of instances to define a dataset as "big" data has not formally been defined, other works on the use of big data have utilized datasets far smaller than ours, some as low as 100,000 instances [10]. Therefore, as our dataset contains well over 1 million instances, we consider our dataset as big data.

As our dataset consists of an over representation of the majority (normal) class, we utilize *random undersampling (RUS)* to create a total of five class distributions: 99:1, 90:10, 75:25, 65:35, and 50:50. RUS was chosen as our data sampling technique due to its relative success in mitigating the effects of class imbalance in other works [11, 12]. These five sampling ratios were applied to eight learners and were evaluated using 5 runs of twofold cross validation with 10 iterations. The learners evaluated were: *5-Nearest Neighbor (5NN)*, *Naïve Bayes (NB)*, *Multilayer Perceptron (MLP)*, *Support Vector Machines (SVM)*, JRIP using *Repeated Incremental Pruning to Produce Error Reduction (RIPPER)*, *Random Forest (RF)*, C4.5 decision trees, and *Logistic Regression (LR)*. The detection performance of all learners were evaluated using *Area Under the Receiver Operating Characteristics (ROC) Curve (AUC)*.

Our results show that the use of RUS offered significant performance increases when compared to the performance using no data sampling. Overall, we suggest that the combination of RF with a class ratio of 65:35 is the optimal choice, achieving an AUC mean of 0.99904.

The remainder of this paper is organized as follows. In "Attack background" section, we detail the Slow HTTP DoS attack methods collected in our dataset. In "Related works" section, we discuss related works associated with utilized data sampling on highly imbalanced big data. "Methods/experimental" section outlines our collection procedure and our empirical design. In "Results and discussion" section, we discuss our findings. Lastly, in "Conclusion" section, we conclude our works and identify future endeavors.

## Attack background

Slow HTTP DoS attacks focus on targeting the application layer, in this instance the HTTP protocol, and can be implemented through several variations such as Slowloris, and Slow POST [13]. In a Slowloris attack, an attacker begins their initial communications with a web server, with the goal of leaving the connection open as long as possible. The server will continue to send new requests to the attacker attempting to further the communication. The attacker, on the other hand, simply sends partial packet headers back to the server, never fully resolving the communication. The behavior is represented in Algorithm 1. This behavior continues until either the server experiences a communication time-out or the attacker ends the attack. Server resources are eventually depleted so that future legitimate connections are unable to be fulfilled, due to the attacker's stranglehold on server communications.

In a Slow POST attack, an attacker sends legitimate POST requests to a server. The packet information associated with this request specifies a content-length header value of an extreme size. The server then waits for the entire message body specified by the header value to be received. Throughout the interaction, the attacker will continue to send data at a rate as low as one byte per transmission. To bypass the server's client idle timeout, the data is transmitted in regular time intervals. Therefore the server's resources are made unavailable until the request is completed. The behavior of this attack can be seen in Algorithm 2.

**Algorithm 1** Slowloris Connection

```
1: procedure SLOWLORIS
2:     for $i = 1$ to number of connections do
3:         CONNECT($hostName$)
4:         SEND($GETRequest$)
5:     while Server requests communication do
6:         header ← random header value
7:         SEND(header)
```

**Algorithm 2** POST Connection

```
1: procedure POST
2:     for $i = 1$ to number of connections do
3:         CONNECT($hostName$)
4:         CONNECT($identifyPOSTvariable$)
5:         SEND($POSTRequest$)
6:     while content-length header value is not met do
7:         header ← partial value
8:         SEND(header)
```

The effects of both of the aforementioned attack variants can be exacerbated depending on the attack method of execution. These attacks can easily be implemented in a distributed fashion using multiple hosts, or can even be configured to enact multiple attack instances from a single machine depending on the tools being utilized. This allows the attacker to perform a potentially devastating attack while using very little in terms of their own personal resources.

For both Slow HTTP attack variants, numerous tools and scripts exist that are publicly and freely available to attackers to utilize. A benefit of these tools is that many are capable of making multiple connections from a single machine, lessening the need to execute distributed attacks if the attacker's resources are limited. However, attackers with ample resources can still choose to run these tools in a distributed fashion, making a small army of hosts even more dangerous.

*Low orbit ion cannon (LOIC)* Slow [14] is a variant of the popular LOIC DoS tool, which specifically focuses on performing Slow HTTP attacks such as Slow HTTP POST. Although this tool is easily accessible, it does offer a limited range of configurable options, such as the inability to adjust the content-length value or target specific pages on a server.

*R U Dead Yet (R.U.D.Y.)* [15] is another Slow DoS tool which allows the user to select the number of connections as well as target specific pages/forms on a server. However, the attacker is still unable to specify the timeout or content-length. What is of interest is how R.U.D.Y. targets specific POST values within a form. Before executing the attack, R.U.D.Y. will scan the target URL for a POST form, then display the available POST variables to the user for specific targeting.

*Open web application security project (OWASP)* Switchblade [16] is a stress test tool used for testing the availability, performance, and capacity of a web application. The tool was made publicly available in 2010 and offers a robust amount of configurable attack options. Switchblade allows users to set the number of connections, connection rate, timeout value, content-length, and user agent values. This allows for attackers to enact very targeted attacks with varied conditions.

Suites such as SlowHTTPTest [17], provide highly configurable attack scenarios which give attackers a high level of control when executing their attacks. Attackers are able to select the number of connections, connection rate, length of follow up data, and more to fine tune their attack setups. SlowHTTPTest is a Linux-based application, but can easily be executed on Windows platforms utilizing the Cygwin environment for added flexibility in execution.

Several other script-based tools also exist, such as Slowloris.pl [18], Slowloris.py [19], and PyLoris [20]. As these tools are written in common languages such as Perl and Python, they are easily modifiable by attackers to fit customized attack criteria. They are also lightweight tools which can be executed from nearly any environment.

## Related works

In our literature review, few works specifically relating to the problem of class imbalance within cyber security big datasets have been identified. However, in work performed by Roy et al. [21], a study was conducted with the aim of addressing a series of common challenges that may affect machine learning based approaches to malware detection in the Android domain. For this work, a dataset was collected consisting of real-world Android malware apps, totalling approximately 430,000 instances. The data contained 8500 minority instances with an imbalance ratio of 50:1. Based upon these specifications, the data is considered imbalanced, but not severely. We also view the data as a large dataset, but not specifically big data. The challenges addressed, and how they are tested, represent interesting questions relating to how they may affect other domains, such as Network Based Intrusion Detection. Six total research questions were posed. The most relevant question related to our work, asked if performance degrades as the class ratio approaches more real-world levels (for Android domain, approx. 1:100). Six class distribution ratios were evaluated (1:1, 1:5, 1:10, 1:20, 1:50, and 1:100) using the *K-nearest neighbor (KNN)* classifier. As the imbalance increases, the *area under the precision recall curve (AUPRC)* performance does decrease substantially (decreasingly steadily from 0.964 to 0.456). Overall, this work does well to address several domain specific challenges, but does have several shortcomings. Most obvious is the fact that only one classifier (KNN) was evaluated. Further, no specific value for K was given. The findings in this work could have been better verified through the use of a wider range of machine learners.

Other works have attempted to address the issue of severely imbalanced big data, albeit in a differing domain. Bauder et al. [22] conducted an empirical study to evaluate the impact of varying class distributions on learner behavior for Medicare fraud big data. In this work, the focus is placed on the detection of Medicare Part B provider fraud. A novel approach to mapping fraud labels based on known fraudulent providers is also proposed. The resulting dataset is considered severely imbalanced, containing very few fraudulent instances. Seven class distributions were created using RUS (99.9:0.1, 99:1, 95:5, 90:10, 75:25, 65:35, and 50:50) to evaluate the performance of six machine learning methods (NB, LR, KNN, SVM, C45, and RF). Their results indicated that a learner and distribution combination of RF with a 90:10 class ratio produced the best results with an AUC value of 0.87302.

Rio et al. [23] address the issue of class imbalance by focusing on the use of *random oversampling (ROS)* techniques. Their experiments utilized data from the ECBDL'14 Big Data competition [24], which relates to the Bioinformatics domain. This dataset consisted of nearly 32 million instances and less than 2% minority class instances. A MapReduce [25] version of ROS was utilized to produce oversampled datasets with rates of 105%, 115%, 130%, 140%, 150%, 160%, 170%, and 180%. The resulting datasets were evaluated using the RF algorithm. Their results suggested that setting the oversampling ratio to a value that balances *true positive rate (TPR)* and *true negative rate (TNR)* can lead to better performance in terms of accuracy. Fernadez et al. [26] also investigated class imbalance in big data using the MapReduce framework. They compared RUS and ROS using MapReduce with two subsets of the ECBDL'14 dataset, one with 12 million instances and the other with less than 1 million. The authors examined the performance of RF and decision trees, using both Apache Spark and Apache Hadoop MapReduce frameworks. Key observations from this work are that models using Apache Spark generally performed better than models using Hadoop. Further, ROS performed better with more MapReduce partitions while RUS performed better with less partitions, indicating that the number of Hadoop partitions impacts performance. Lastly, Apache Spark-based RF and decision tree algorithms produced better results with RUS compared to ROS.

## Methods/experimental

This section outlines the procedure for the execution of our Slow HTTP DoS attacks and our associated data collection process. Our network topology and normal network usages are defined as well. We also detail the machine learning methods applied to our collected dataset.

### Data collection

Our capture framework allows for us to perform our attacks within a real-world network environment servicing numerous active users. Our campus network consists of hosts from classrooms, labs, and offices. To facilitate our network usages, we utilize a series of switches, servers, and routers capable of servicing on and off campus users. A Cisco *adaptive security appliance (ASA)* is used as a firewall and is capable of providing us with *virtual private network (VPN)* capabilities. An Apache web server was established to be utilized as a student resource portal, and also serves as the target for our attacks. This server is configured with CentOS 6.8 and runs on a Dell 2950 Poweredge with two quad-core Intel Xeon 5300 processors and 16 GBs of memory. Figure 1 shows our architecture in more detail.

The Apache student resource server is configured using WordPress, and hosts lecture material, assignments, assessments, and other content required by student users. General, normal traffic related to course work may consist of, but is not limited to, downloads, uploads, website navigation, and other communications with the web server. Within the context of our network usages, students both locally on our network and from online may request course material from our server concurrently. Our

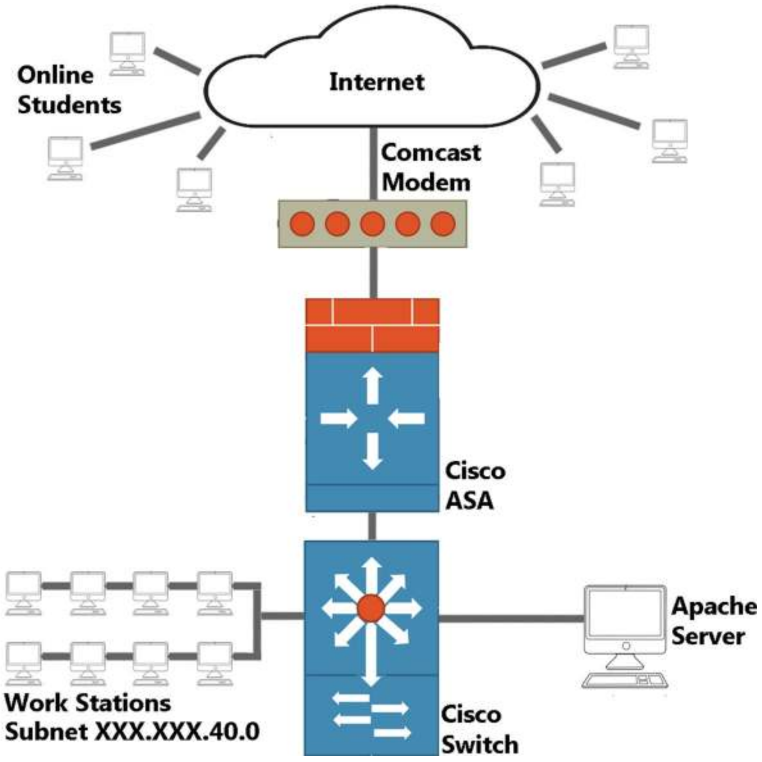


**Fig. 1** Topology of campus network

extended network also supports other faculty and student usages ranging from virtualization to audio/video streaming.

For our attack implementation we opted to enact two common Slow HTTP DoS attack variants to represent a broader range of attack traffic, the first being a Slowloris attack, based on targeting HTTP GET communications. For our Slowloris collection, our attacks were performed on a physical host machine using popular attack tools, rather than through simulation. Several tools were compared, such as SlowHTTPTest [17], PyLoris [20], and Slowloris.py [19]. This was done to identify if any one tool produced different traffic patterns. Based on our comparison, the traffic across these tools produced similar traffic, and therefore only one tool was utilized in our experiments. For our experiments, we chose the Slowloris.py attack script, as it allowed for the most verbose configuration options.

To represent different levels of Slowloris attacks, we ran three different attack scenarios with varying configurations. As some attacks are performed in a "low and slow" [27] manner, we opted to run a series of attacks in a stealthy configuration. To represent this stealthy attack, in which an attacker is attempting to avoid detection through the use of minimal connections, we enacted one attack instance using a single connection with a random connection interval between 1 and 20 s. A more moderate level attack was performed, executing 125 connections using a random connection interval between 1 and 10 s. Lastly, to represent a non-stealthy attack, we incorporated 250 connections and a random connection interval of 1 and 5 s. Each attack ran from a single host machine for 1 h, and targeted our student resource server.

For our second Slow HTTP DoS attack, we enacted as series of Slow POST attacks. For this implementation, we used OWASP Switchblade 4.0 as our attack tool. As with our Slowloris experiment, we once again ran tests using a wide range of popular attack tools, such as LOIC Slow [14], R.U.D.Y [15], Railgun [28], and Torshammer [29]. We settled on Switchblade as it provided us with the most configurable options, and the resulting traffic behavior from all tools tested were identical. Like our Slowloris implementation, our attacks were run from a single physical host machine utilizing multiple connections.

To more fully represent varying attack conditions in a real-world attack, we once again chose to implement varying scenarios ranging from stealthy to non-stealthy attack configurations. In total, nine different attack configurations were implemented, separated into three distinct categories. Our stealthy set of configurations utilized single connections to prolong potential detection. We scaled up the connection amount to 125 to represent a moderate level attack. Lastly, we increased the number of connections to 250 to represent a non-stealthy attack. For each of these three categories, we ran three attacks with varying timeout values: 10 s, 50 s, and 100 s. For all attacks, we used a content-length value of 1000. This value was chosen as it was a default value across most attack tools. All attacks targeted the same php form element on our web server, and ran for one hour each, resulting in a total of nine hours of attack traffic.

*Full packet captures (FPCs)* [30] allow us to observe all traffic communications as they are received. The key issue with analyzing full packets is that it can be quite resource intensive to analyze all available packet features. To lessen this resource impact, we opted to use Netflow [31] features extracted from our FPCs. Netflow traffic refers to a high-level summary of network communications. A Netflow record is identified based upon the standard 5-tuple attribute set that makes up a conversation: source IP, destination IP, source port, destination port, and transport protocol. Based upon which Netflow standard is being implemented, other attribute fields can also be produced. We utilize the IPFIX [32] standard for our flow extraction. The resulting features can be seen in Table 1.

Once all Netflow instances were extracted, we labeled any instances correlating to any of our attack implementations as "attack" and all remaining traffic collected from our network as "normal". In total, we collected 1,892,161 instances, with 6646 representing "attack" instances.

## Machine learning methods

Eight classification algorithms were selected to build predictive models based on our collected datasets: KNN, NB, MLP, SVM, C4.5 decision trees, RF, JRIP, and LR. This variety of learners was selected to broaden the scope of our analysis. All models were built using the WEKA machine learning toolkit [33].

KNN [34] is an instance-based learning algorithm that contains all available instances and assigns new instance labels based upon distance functions. These distance functions utilize K, which represents the amount of closest instances to the test instance to decide its label. For our study, K is assigned a value of 5.

**Table 1 Description of selected Netflow features**

| Feature name | Description |
|---|---|
| Protocol | Transport-layer protocol number of flow |
| Packets | Number of packets in flow |
| Bytes | Number of bytes in flow |
| Flags | Logical OR of TCP flag fields of flow |
| Initial Flags | TCP flags in initial packet |
| Session Flags | All TCP flags in entire connection |
| Attributes | Flow attributes [SFTC] |
| Duration | Duration length (in milliseconds) of flow |
| Payload Bytes | Size of payload measured in bytes |
| Payload Rate | Non-overhead packet data per second |
| Packets/Second | Number of packets per second |
| Bytes/Second | Number of bytes per second |
| Bytes/Packet | Number of bytes per packet |
| Class | Class label (attack or normal) |

NB [35] is based upon Bayes' theorem and is well suited for datasets with high dimensionality. The algorithm works upon the assumption that features are independent, and utilizes this premise to calculate the posteriori probability that an instance is a member of a specific class. Although a simple premise, Naive Bayes has been known to outperform other more sophisticated classification methods.

MLPs [36], also referred to simply as artificial neural networks, utilize neurons (perceptrons) to compute an individual value from multiple inputs using non-linear transformations. Although a single neuron can be rather limiting, MLPs use these neurons as building blocks to create much larger networks. MLPs can also utilize several hidden layers to help transform inputs into usable values by the outer layers.

SVMs [37] are discriminate classifiers used as supervised learning models with associated learning algorithms. SVMs utilize hyper-planes to separate instances in a dataset into two distinct groups, and assign new instances to one class or another. The aim is for the SVM to identify the most optimal hyper-plane with the largest gap between class instances as possible.

The C4.5 decision tree [38] is a tree-based learning algorithm. In WEKA, this is the implementation of the J48 decision tree. In these algorithms a decision tree is built based on the training data. Each branch of the tree represents a feature in the data which divides the instances into more branches based on the values which that feature can take. The leaves represent the final class label. The C4.5 algorithm uses a normalized version of Information Gain to decide the hierarchy of features in the final tree structure. Information Gain is based on the decrease in entropy after a dataset is split on a feature. The aim is to find the feature that returns the most homogeneous branches.

RF is an ensemble learning method, created from the construction of multiple decision tress. RF can be used for both classification and regression tasks and is a popular choice due to its typically strong performance and relative simplicity [39].

JRIP uses a proportional ruler learner called RIPPER . It uses association rules with *reduced error pruning (REP)* and splits the training data into a growing and pruning set

[40]. At each simplification step, the pruning operator with the greatest REP is chosen. Simplification ends when pruning increases the error on the set.

LR represents a machine learning algorithm utilized for classification scenarios consisting of two outcomes [41]. Unlike linear regression, in LR a sigmoidal function is used to give us class probability values, either being 0 or 1. We utilize Weka's default 'ridge' parameter value. This represents the penalized maximum likelihood estimation with a quadratic penalty function.

### Metrics

To evaluate each model, we utilize AUC [42]. The ROC curve graphs the TPR and *false positive rate (FPR)* of the model. TPR represents the percentage of the attack instances that are correctly predicted as attacks. FPR represents the percentage of the normal data which is wrongly predicted as attack data. The ROC curve is built by plotting TPR vs FPR as the classifier decision threshold is varied. Higher AUC values tend to correlate to higher TPR and lower FPR, both of which are preferred outcomes.

We utilize twofold cross validation to evaluate our AUC values. Using twofold cross validation divides the data into 2 non-overlapping parts, representing our folds. For each iteration, the first part is reserved as test data and the remaining part is used as training data. We opted to use twofold, as using higher fold counts can take far more computational time depending on the learner being evaluated. Our final AUC values are calculated by aggregating the AUC values of the models being tested for each of 2 parts of the data. In order to decrease the bias of randomly selected folds, we applied five runs of twofold cross validation with 10 iterations to provide each performance value.

In total, our collected dataset consists of 1,898,807 instances, with 1,892,161 normal (majority) instances and 6646 attack (minority) instances. This results in a severe class imbalance of approximately 0.004%. High levels of class imbalance can often lead to a classifier biasing the majority class [43]. To counteract this bias, we utilize RUS to randomly select a subset of instances from the majority class, while keeping all instances of the minority class [44]. We generate five class distribution ratios of normal instances to attack instances from which to build our models. The ratios created are 99:1, 90:10, 75:25, 65:35, and 50:50.

To further reinforce the results of our performance metric, we perform significance testing to evaluate the impact each learner and class distribution has on AUC values. We use *ANalysis Of VAriance (ANOVA)* [45] as a means to determine if our factors are equal. As we are evaluating the significance of both learner and distribution impact, we perform two-way ANOVA [46]. We also implement a post hoc analysis via Tukey's *honestly significant difference (HSD)* test [47]. A Tukey's HSD test determines if factor means are significantly different from one another. Factor means are grouped based upon their similarity. Means within the same letter grouping represent no significant difference, whereas means in different letter groupings do indicate a significant difference.

## Results and discussion

This section evaluates the results of our learners and the varying combinations of class imbalance ratios to determine which learner performs best, as well as the significance of using data sampling to improve performance on big datasets. As stated previously,

**Table 2 Cross validation results**

| Ratio | 5NN | RF | C4.5 | LR | MLP | SVM | JRIP | NB |
|---|---|---|---|---|---|---|---|---|
| 99:1 | *0.99894* | 0.99480 | 0.98168 | 0.98535 | 0.79018 | 0.98689 | 0.95682 | 0.96982 |
| 90:10 | 0.99871 | 0.99856 | *0.99889* | 0.98958 | 0.98396 | 0.98820 | 0.98962 | 0.96851 |
| 75:25 | 0.99875 | 0.99889 | *0.99890* | 0.99376 | 0.98816 | 0.99095 | 0.99288 | 0.97327 |
| 65:35 | 0.99877 | *0.99904* | 0.99867 | 0.99338 | 0.98919 | 0.99138 | 0.99290 | 0.97464 |
| 50:50 | 0.99862 | *0.99905* | 0.99844 | 0.99420 | 0.99074 | 0.99150 | 0.99437 | 0.97581 |
| None | *0.99826* | 0.98850 | 0.96145 | 0.98223 | 0.75768 | 0.98150 | 0.89414 | 0.97273 |

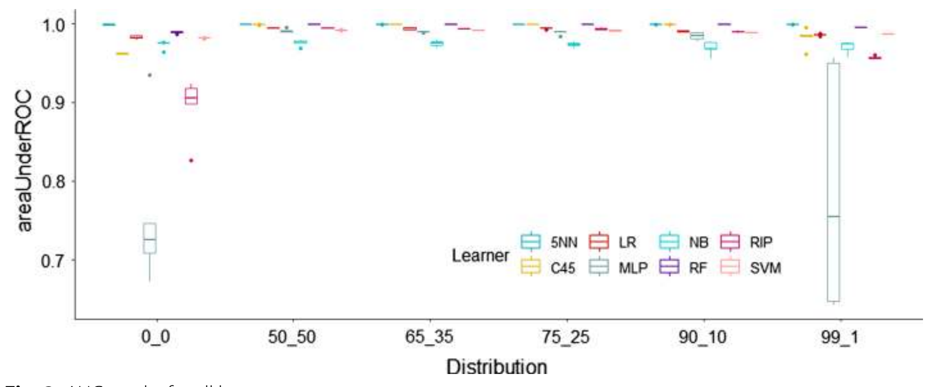
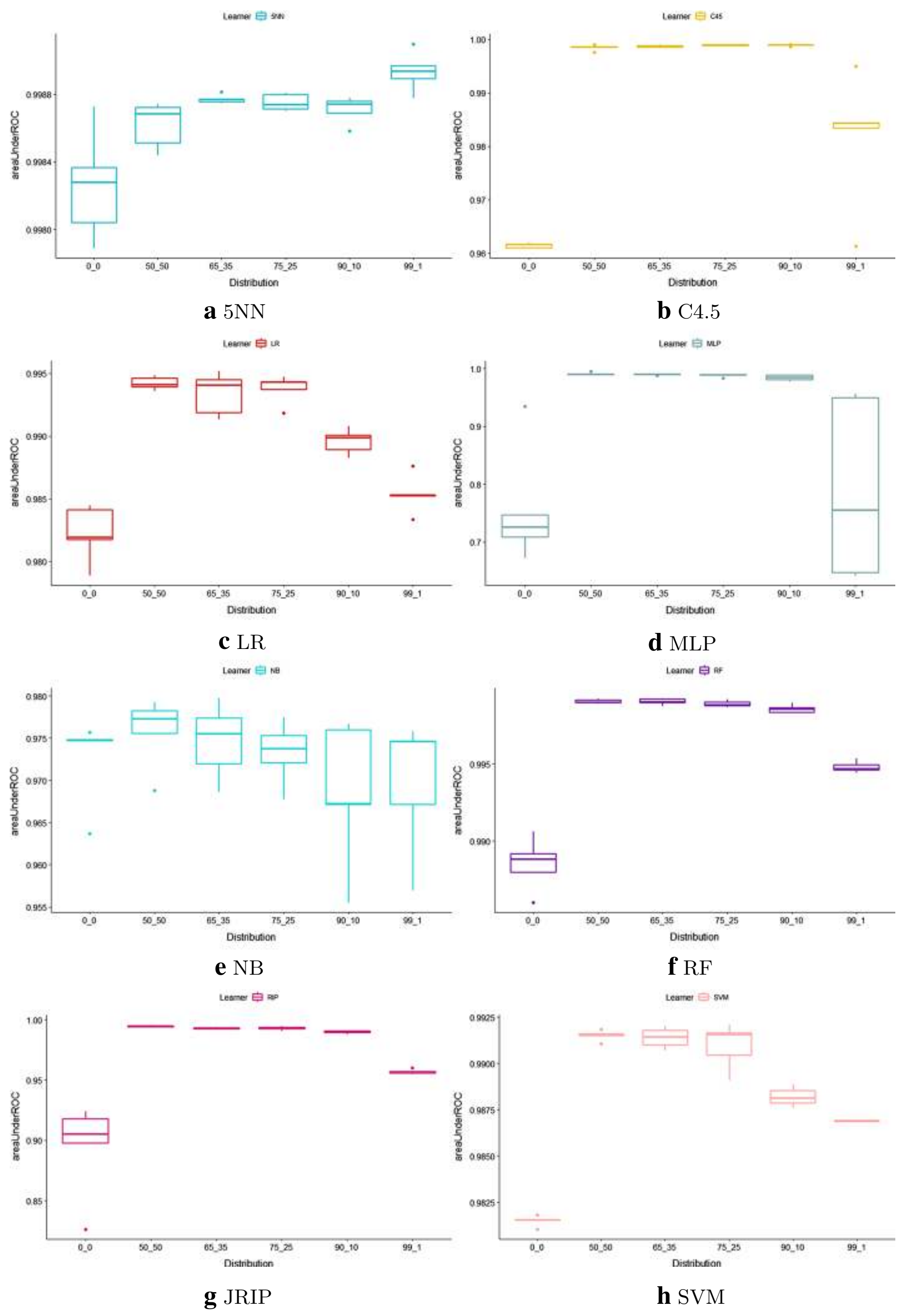


**Fig. 2** AUC results for all learners

we created a total of five class distributions based upon our original dataset. The ratios used are 50:50, 65:35, 75:25, 90:10, and 99:1. We have also included results relating to the original non-sampled dataset, which we refer to as "None", to further represent any performance difference between using and not-using RUS. For all class distributions, we evaluate each of the following eight learners: 5NN, RF, C4.5, LR, MLP, SVM, JRIP, and NB. For each learner and class ratio combination, 5 runs of twofold cross validation are applied. The average AUC values of the resulting runs are shown in Table 2. The table displays the results for each of the five class imbalance ratios and the non-sampled "None" dataset. It should be noted, that most of our learners achieve high AUC values in regards to the detection of our Slow HTTP attach dataset. This has been attributed in part to the feature set used by Netflow, which has proven to be highly discriminative in prior works for these types of attack instances [48, 49]. The highest AUC value of 0.99905 is achieved with RF and a class ratio of 50:50. The second highest AUC of 0.99904 is also achieved by RF, but with a class ratio of 65:35. As the ratios become more imbalanced, C4.5 achieves higher performance, with AUC values of 0.99890 and 0.99889 corresponding to the ratios of 75:25 and 90:10 respectively. When observing the most imbalanced ratio of 99:1, 5NN performs best with an AUC value of 0.99894. Interestingly, 5NN also performs best for the original dataset with no sampling applied, obtaining an AUC value of 0.99826. The worst learners, on average, for all class distributions were MLP and NB. These results are further visualized in Figs. 2 and 3.

The overall high AUC metrics are certainly encouraging when attempting to determine the validity of the learner's ability to detect Slow HTTP attack traffic. However,



**a** 5NN **b** C4.5 **c** LR **d** MLP **e** NB **f** RF **g** JRIP **h** SVM

**Fig. 3** AUC results by learner

when evaluating the significance of using sampling techniques to improve classifier performance on big data, the initial benefits are not as easily discerned from our cross validation results alone. Therefore, to further evaluate any potential performance gains using RUS, we perform significance testing using ANOVA and post hoc analysis with Tukey's HSD. Table 3 displays our two-factor ANOVA test results. The factors

**Table 3 ANOVA Results**

| | *Df* | Sum Sq | Mean Sq | F value | Pr (> F) |
|---|---|---|---|---|---|
| Distribution | 5 | 0.09756 | 0.01951 | 25.571 | $< 2e-16$ |
| Learner | 7 | 0.15021 | 0.02146 | 28.123 | $< 2e-16$ |
| Distribution:learner | 35 | 0.25811 | 0.00738 | 9.665 | $< 2e-16$ |
| Residuals | 192 | 0.14650 | 0.00076 | | |

**Table 4 Tukey's HSD learner results**

| Learner | AUC | AUC std | r | Min | Max | Group |
|---|---|---|---|---|---|---|
| 5NN | 0.99867 | 0.00025 | 30 | 0.99789 | 0.99910 | a |
| RF | 0.99647 | 0.00399 | 30 | 0.98601 | 0.99925 | a |
| C4.5 | 0.98967 | 0.01503 | 30 | 0.96096 | 0.99908 | ab |
| LR | 0.98975 | 0.00483 | 30 | 0.97891 | 0.99514 | ab |
| SVM | 0.98840 | 0.00364 | 30 | 0.98105 | 0.99205 | ab |
| NB | 0.97243 | 0.00600 | 30 | 0.95555 | 0.97969 | b |
| JRIP | 0.97012 | 0.03982 | 30 | 0.82635 | 0.99529 | b |
| MLP | 0.91665 | 0.12416 | 30 | 0.64263 | 0.99468 | c |

being considered are Class Distribution and Learner. Based on a significance value of 0.05, we can conclude the following:

1. The distribution p-value of $< 2e-16$ indicates that the class distribution ratio is associated with AUC performance
2. The learner p-value of $< 2e-16$ indicates that the chosen machine learner is associated with AUC performance
3. The p-value for the interaction between distribution and learner of $< 2e-16$ indicates that this interaction effect is statistically significant.

To further understand the significance of each specific learner and each class ratio, we utilize a Tukey's HSD test to separate our learners and ratios into groups based upon their significant differences. Learners and ratios belonging to the different letter groupings indicate that they are significantly different. Table 4 displays the significance groupings of each learner's performance across all class distributions. From this we identify four levels of performance groupings amongst our learners. Both 5NN and RF were significantly better than other learners and placed in group 'a'. C4.5, LR, and SVM following behind in the second grouping 'ab'. NB and JRIP were categorized into group 'b', with MLP residing at the lowest performing group 'c'. Similar to Tables 4, 5 displays the result groupings of our Tukey's HSD test, but this time focusing on each class distribution across all learners. Here only two groupings were identified. Distributions of 50:50, 65:35, 75:25, and 90:10 were assigned to the top performing group 'a'. The most severe imbalance ratio of 99:1 as well as the original non-sampled dataset were relegated to group 'b'. Figures 4 and 5 have been provided to visualize the resulting groupings. Table 6 represents all results from our Tukey's HSD test organized by learner.

**Table 5 Tukey's HSD class distribution results**

| Distribution | AUC | AUC std | r | Min | Max | Group |
|---|---|---|---|---|---|---|
| 50:50 | 0.99284 | 0.00736 | 40 | 0.96879 | 0.99921 | a |
| 65:35 | 0.99225 | 0.00777 | 40 | 0.96864 | 0.99925 | a |
| 75:25 | 0.99194 | 0.00825 | 40 | 0.96778 | 0.99917 | a |
| 90:10 | 0.98950 | 0.01018 | 40 | 0.95555 | 0.99904 | a |
| 99:1 | 0.95806 | 0.08238 | 40 | 0.64263 | 0.99910 | b |
| None | 0.94206 | 0.08457 | 40 | 0.67230 | 0.99873 | b |

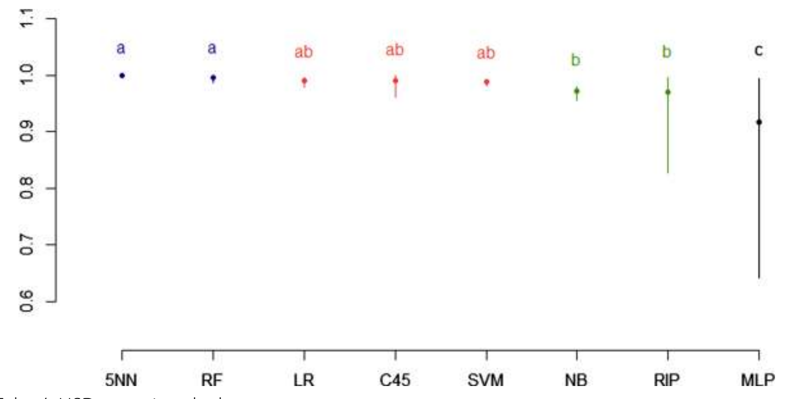


**Fig. 4** Tukey's HSD groupings by learner

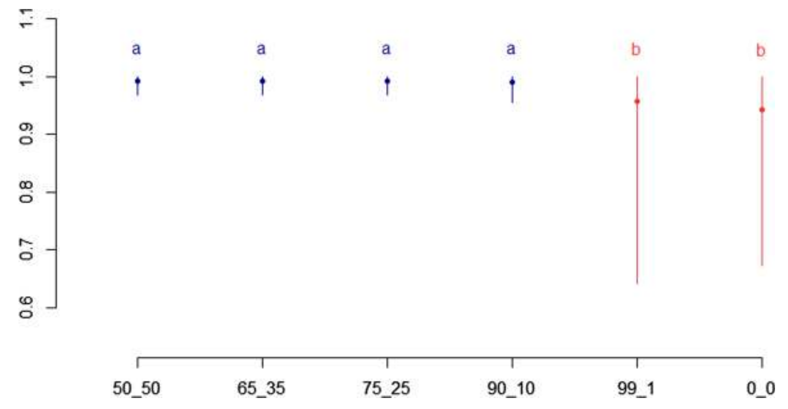


**Fig. 5** Tukey's HSD groupings by distribution

Based on these results, we recommend an optimal learner and ratio combination be chosen from their respective group 'a's. As RF, assigned to learner group 'a', achieved better performance than 5NN for the distributions assigned to distribution group 'a', we select RF as the best overall learner based upon our significance testing. In regards to class distribution, both group 'a' ratios of 50:50 and 65:35 provided comparable performance when using RF. When evaluating which distribution is the most optimal, we must also consider the type of attack being detected. When detecting standard flood based DDoS attacks, the significance of missing a few individual attack instances is not necessarily substantial. Therefore, it may be preferred to choose a distribution that produces less computational overhead to make detection more efficient. As the 50:50 class distribution produces a smaller dataset, evaluation on said dataset will be significantly faster than distributions with more traffic instances. However, considering that we are evaluating Slow HTTP DoS attacks, the significance of each attack instance is more important. This is due to that fact that each individual instance can be held for an immense amount of time, draining resource availability over longer periods of time. Such attacks also do not have to be performed in a distributed fashion to be successful, making the identification of each individual attack instance more important. For such attack types where each missed instance could ultimately result in significant losses or ramifications, we make the argument that a ratio of 65:35 is a better choice, as it retains a larger number of instances from the majority class. By keeping a larger number of majority instances, we allow for a better representation of normal traffic behavior. This allows for more confidence that our learner performance is more accurate of results that may be found in a real-world scenario.

Regarding the overall results obtained from our significance analysis, it would seem to indicate that there is little to no difference between the distribution ratios within group 'a'. However, this conceived lack of significance between these ratios could be attributed to the initially high performance values originally obtained. As the AUC values are so high, there is little room for variances between each performance value. It is possible,

**Table 6 Tukey's HSD by learner results**

| Learner | Distribution | AUC | Group | Learner | Distribution | AUC | Group |
|---|---|---|---|---|---|---|---|
| 5NN | 99:1 | 0.99894 | a | MLP | 50:50 | 0.99074 | a |
| | 65:35 | 0.99877 | a | | 65:35 | 0.98919 | a |
| | 75:25 | 0.99875 | a | | 75:25 | 0.98816 | a |
| | 90:10 | 0.99871 | a | | 90:10 | 0.98396 | a |
| | 50:50 | 0.99862 | a | | 99:1 | 0.79018 | c |
| | None | 0.99826 | a | | None | 0.75768 | c |
| RF | 50:50 | 0.99905 | a | SVM | 50:50 | 0.99150 | a |
| | 65:35 | 0.99904 | a | | 65:35 | 0.99138 | a |
| | 75:25 | 0.99889 | a | | 75:25 | 0.99095 | a |
| | 90:10 | 0.99856 | a | | 90:10 | 0.98820 | a |
| | 99:1 | 0.99480 | a | | 99:1 | 0.98689 | a |
| | None | 0.98850 | a | | None | 0.98150 | a |
| C4.5 | 75:25 | 0.99890 | a | JRIP | 50:50 | 0.99437 | a |
| | 90:10 | 0.99889 | a | | 65:35 | 0.99290 | a |
| | 65:35 | 0.99867 | a | | 75:25 | 0.99288 | a |
| | 50:50 | 0.99844 | a | | 90:10 | 0.98962 | a |
| | 99:1 | 0.98168 | a | | 99:1 | 0.95682 | ab |
| | None | 0.96145 | ab | | None | 0.89414 | b |
| LR | 50:50 | 0.99420 | a | NB | 50:50 | 0.97581 | a |
| | 75:25 | 0.99376 | a | | 65:35 | 0.97464 | a |
| | 65:35 | 0.99338 | a | | 75:25 | 0.97327 | a |
| | 90:10 | 0.98958 | a | | None | 0.97273 | a |
| | 99:1 | 0.98535 | a | | 99:1 | 0.96982 | a |
| | None | 0.98223 | a | | 90:10 | 0.96851 | a |

that a similar dataset utilizing a feature set other than Netflow, may produce a wider range of performance values and hence show greater significance between class distributions. Still, the fact remains that even with such high AUC values, a clear significant difference was determined when evaluating the most severe class ratio of 99:1 and the original dataset. This shows that the use of RUS and the chosen class distribution can provide a statistically significant performance increase.

## Conclusion

As application layer attacks become more prevalent and more sophisticated, the necessity for quick identification is paramount. However, successful detection and mitigation commonly require network analysts to sift through immense amounts of traffic instances in order to find those corresponding to malicious behavior. Furthermore, as the number of attack instances can be substantially less than those of legitimate users, successfully identifying illegitimate traffic can be like finding a needle in a haystack.

In this work, we implement data sampling techniques with the intent of increasing the detection performance of Slow HTTP DoS attacks on severely imbalanced big datasets. Our collection procedure for collecting representative real-world big data from a production network is also outlined. The collected data consists of over 1.8 million total instances, with less than 0.01% attack instances, thus resulting in a severely imbalanced dataset. Five class distributions were generated from this dataset, in an attempt to mitigate the severe level of class imbalance between the majority and minority instances. We evaluated eight machine learners on each of the produced distributions to identify the optimal learner and class ratio combinations. From our results, we determined that RF combined with a 65:35 class distribution was the best combination, achieving an AUC cross validation value of 0.99904. This pairing is also preferred due to the fact that more of the majority instances were retained, as compared to other learner/distribution combinations. This allows for the resulting dataset to better represent normal traffic behaviors for more accurate detection. Furthermore, we performed significance testing using two-factor ANOVA and Tukey's HSD. These results showed that both the choice of learner and class distribution make a statistically significant impact on AUC values.

Future works aim to evaluate our methodology using other cyber security related big datasets. We also plan to examine other performance metrics beyond AUC values, such as Geometric Mean and AUPRC.

**Abbreviations**

5NN: 5-nearest neighbor; ANOVA: analysis of variance; ASA: adaptive security appliance; AUC: area under the curve; AUPRC: area under the precision recall curve; DDoS: distributed denial of service; FPC: full packet capture; FPR: false positive rate; HSD: honestly significant difference; HTTP: hypertext transfer protocol; KNN: K-nearest neighbor; LOIC: low orbit ion cannon; LR: logistic regression; MLP: multilayer perceptron; NB: Naive Bayes; OWASP: open web application security project; R.U.D.Y.: R U Dead Yet; REP: reduced error pruning; RF: random forest; RIPPER: repeated incremental pruning to produce error reduction; ROC: receiver operating characteristics; ROS: random oversampling; RUS: random undersampling; SVM: support vector machine; TNR: true negative rate; TPR: true positive rate; VPN: virtual private network.

**Acknowledgements**

The authors would like to thank the various members of the Data Mining and Machine Learning Laboratory, Florida Atlantic University, for assistance with the reviews. The authors would also like to give special thanks to Clifford Kemp for his data collection efforts.

**Authors' contributions**

CLC performed the literature review and drafted the manuscript. TMK worked with CLC to develop the article's framework and focus. TMK introduced this topic to CLC. All authors read and approved the final manuscript.

**Funding**
None.

**Availability of data and materials**
Not available now.

**Ethics approval and consent to participate**
Not applicable.

**Consent for publication**
All authors have consented to the publication of this work.

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

Received: 24 May 2019 Accepted: 16 July 2019
Published online: 30 July 2019

## References

1. Columbus L. Roundup Of cloud computing forecasts and market estimates, 2018. https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/#427f5980507b. Accessed 20 March 2019.
2. Security response: the continued rise of DDoS attacks. Technical report. 2014. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-continued-rise-of-ddos-attacks.pdf. Accessed 22 Feb 2019.
3. Liu Y, Zhang H, Yang Y. A dos attack situation assessment method based on qos. In: Proceedings of 2011 international conference on computer science and network technology, IEEE, 2011. pp. 1041–5.
4. Modi H. NETSCOUT threat intelligence report. Technical report. 2018. https://www.netscout.com/sites/default/files/2019-02/SECR_001_EN-1901%20-%20NETSCOUT%20Threat%20Intelligence%20Report%202H%202018.pdf. Accessed 20 March 2019.
5. Durcekova V, Schwartz L, Shahmehri N. Sophisticated denial of service attacks aimed at application layer. In: 2012 ELEKTRO, IEEE, 2012. pp. 55–60.
6. Yevsieieva O, Helalat SM. Analysis of the impact of the slow http dos and ddos attacks on the cloud environment. In: 2017 4th international scientific-practical conference problems of infocommunications. Science and technology (PIC SI&T), IEEE, 2017. pp. 519–23.
7. Hirakaw T, Ogura K, Bista BB, Takata T. A defense method against distributed slow http dos attack. In: 2016 19th international conference on network-based information systems (NBiS)), IEEE, 2016. pp. 519–23.
8. Marquette S. Types of DDoS attacks 2017. https://www.esecurityplanet.com/network-security/types-of-ddos-attacks.html. Accessed 22 Feb 2019.
9. Arellano P. Making decisions with data—still looking for a needle in the Big Data Haystack? 2017. https://www.birst.com/blog/making-decisions-data-still-looking-needle-big-data-haystack/. Accessed 5 Apr 2019.
10. Leevy JL, Khoshgoftaar TM, Bauder RA, Seliya N. A survey on addressing high-class imbalance in big data. J Big Data. 2018;5(1):42.
11. Prusa J, Khoshgoftaar TM, Dittman DJ, Napolitano A. Using random undersampling to alleviate class imbalance on tweet sentiment data. In: 2015 IEEE international conference on information reuse and integration, IEEE, 2015. pp. 197–202.
12. Dai D, Hua S. Random under-sampling ensemble methods for highly imbalanced rare disease classification. In: International conference on data mining 2016, CSREA, 2016. pp. 54–8.
13. Kumar G. Denial of service attacks—an updated perspective. In: Systems science & control engineering, 2016. pp. 285–94.
14. LOIC SLOW. https://sourceforge.net/projects/loicslow/. Accessed 5 Mar 2019.
15. R.U.D.Y. https://sourceforge.net/projects/r-u-dead-yet/. Accessed 5 Mar 2019.
16. OWASP Switchblade. https://www.owasp.org/index.php/OWASP_HTTP_Post_Tool. Accessed 5 Mar 2019.
17. SlowHTTPTest. http://www.r00tsec.com/2012/01/slowhttptest-application-layer-dos.html. Accessed 10 Mar 2019.
18. Slowloris.pl. https://github.com/llaera/slowloris.pl. Accessed 10 Mar 2019.
19. Slowloris.py. https://github.com/gkbrk/slowloris. Accessed 8 Mar 2019.
20. PyLoris. https://sourceforge.net/projects/pyloris/. Accessed 8 Mar 2019.
21. Roy S, DeLoach J, Herndon N, Cargea D, Ou X, Ranganath VP, Lit H, Guevara N. Experimental study with real-world data for android app security analysis using machine learning. In: Proceedings of the 31st annual computer security applications conference, ACM, 2015. pp. 81–90.
22. Bauder R, Khoshgoftaar TM. The effects of varying class distribution on learner behavior for medicare fraud detection with imbalanced big data. In: Health Information Science and Systes 2018. Berlin: Springer; 2018. pp. 1–14.
23. Rio Sd, Benitex JM, Herrer F. Analysis of data preprocessing increasing the oversampling ratio for extremely imbalanced big data classification. In: 2015 IEEE Trustcom/BigDataSE/ISPA, IEEE; 2015. pp. 180–5.
24. "EcBDL"14 Big Data Competition. http://cruncher.ncl.ac.uk/bdcomp/. Accessed 10 Apr 2019.
25. Dean J, Ghemawat S. Mapreduce: simplified data processing on large clusters. Commun ACM. 2018;51(1):107–13 ACM.
26. Fernandez A, del Rio S, Chawla NV, Herrera F. An insight into imbalanced big data classification: outcomes and challenges. Comp Intellig Syst. 2017;3:105–20 Springer.

27. "Radware's ddos handbook: The ultimate guide to everything you need to know about ddos attacks. https://security.radware.com/WorkArea/DownloadAsset.aspx?id=793. Accessed 2 Feb 2019.
28. Railgun. https://github.com/rapid7/metasploit-framework/wiki/How-to-use-Railgun-for-Windows-post-exploitation. Accessed 8 Mar 2019.
29. Hammer Tor's. https://sourceforge.net/projects/torshammer/. Accessed 5 Mar 2019.
30. Koch M. Implementing full packet capture. In: SANS institute information security reading room, SAN6; 2016. pp. 1–27.
31. Zhenqi W, Xinyu W. Netflow based intrusion detection system. In: 2008 international conference on multiMedia and information technology, IEEE; 2008. pp. 825–8.
32. Claise B, Trammell B, Aitken P. Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information. CISCO. Technical report, Cisco (September 2013). https://tools.ietf.org/html/rfc7011. Accessed 15 Mar 2019.
33. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The weka data mining software: an update. SIGKDD Explor Newsl. 2009;11:10–8 ACM.
34. Guo G, Wang H, Bell D, Bi Y, Greer K. Knn model-based approach in classification. In: IJCAI workshop on empirical methods in artifcial intelligence, IBM; 2001. pp. 41–6
35. Rish I. An empirical study of the naive bayes classifier. In: On the move to meaningful internet systems 2003: CoopIS, DOA, and ODBASE. OTM 2003. Lecture notes in computer science. Berlin: Springer; 2003. pp. 986–96.
36. Mubarek AM, Adali E. Multilayer perceptron neural network technique for fraud detection. In: 2017 international conference on computer science and engineering (UBMK), IEEE; 2017. pp. 383–7.
37. Campbell C, Ying Y. Learning with support vector machines. Williston: Morgan & Claypool Publishers; 2011.
38. Kohavi R, Quinian JR. Data mining tasks and methods: classification: decision-tree discovery. In: Handbook of data mining and knowledge discovery. Oxford University Press, Inc; 2002. pp. 267–76.
39. Khoshgoftaar TM, Golawala M, Van Hulse J. An empirical study of learning from imbalanced data using random forest. In: 19th IEEE international conference on tools with artificial intelligence (ICTAI 2007), IEEE; 2007. pp. 310–7.
40. Choudhury S, Bhowal A. Comparative analysis of machine learning algorithms along with classifiers for network intrusion detection. In: 2015 international conference on smart technologies and management for computing, communication, controls, energy and materials (ICSTM), IEEE; 2015. pp. 89–95.
41. Shukla P, Rai R. Ara-mac: Attacker identification using logistic regression. In: 2017 international conference on recent innovations in signal processing and embedded systems (RISE), IEEE; 2017. pp. 124–8.
42. Seliya N, Khoshgoftaar TM, Hulse JV. A study on the relationships of classifier performance metrics. In: 2009 21st IEEE international conference on tools with artificial intelligience, IEEE; 2009. pp. 59–66.
43. Johnson JM, Khoshgoftaar TM. Survey on deep learning with class imbalance. In: Journal of Big Data. 2019. pp. 1–54.
44. Van Hulse J, Khoshgoftaar TM, Napolitano A. Experimental perspectives on learning from imbalanced data. In: Proceedings of the 24th international conference on machine learning, ACM; 2007. pp. 935–42.
45. Chandrakantha L. Learning anova concepts using simulation. In: Proceedings of the 2014 zone 1 conference of the American Society for Engineering Education, IEEE; 2014. pp. 1–5.
46. Pandis N. Two-way analysis of variance: Part 2. In: American journal of orthodontics and dentofacial orthopedics, AAO; 2016. pp. 137–9.
47. Tukey JW. Comparing individual means in the analysis of variance. Biometrics. 1949;5:99–114 International Biometric Society.
48. Calvert C, Khoshgoftaar TM, Kemp C, Najafabadi MM. Detection of slowloris attacks using netflow traffic. In: 24th ISSAT international conference on reliability and quality in design, ISSAT; 2018. pp. 1–6.
49. Calvert C, Kemp C, Khoshgoftaar TM, Najafabadi MM. Detecting slow http post dos attacks using netflow features. In: The 32nd international FLAIRS conference, FLAIRS; 2019. pp. 1–4.

## Publisher's Note