Web Bot Detection System Based on Divisive Clustering and K-Nearest Neighbor Using Biostatistics Features Set

Rizwan Ur Rahman, SECURE, Center of Excellence for Cyber Security, VIT University, Bhopal, India*
Deepak Singh Tomar, Maulana Azad National Institute of Technology, Bhopal, India

ABSTRACT
Web bots are destructive programs that automatically fill the web form and steal the data from websites. According to numerous web bot traffic reports, web bots traffic comprises more than 50% of the total web traffic. An effective guard against the stealing of data from websites and automated web form is to identify and confirm the human user presence on websites. In this paper, an efficient k-nearest neighbor algorithm using hierarchical clustering for web bot detection is proposed. The proposed technique exploits a novel taxonomy of web bot features known as biostatistics features. Numerous attack scenarios for web bot attacks such as automatic account registration, automatic form filling, bulk message posting, and web scrapping are created to imitate the zero-day web bot attacks. The proposed technique is evaluated with a number of experiments using standard evaluation parameters. The experimental result analysis demonstrates that the proposed technique is extremely efficient in differentiating human users from web bots.

KEYWORDS
Clustering, k-NN, Supervised Learning, Web Bot, Web Robot, Web Security

INTRODUCTION
At the present time, the human society is enormously dependent on the Internet, the most important source of communication. For this reason, the accessibility of the Internet is very significant for the growth of the civilized society. For example, the expansion and the success of Internet have changed the way of working of conventional services such as marketing, banking, and electoral system. All of these conventional services are now rapidly replacing by efficient web based applications. On the other hand, the intrinsic vulnerabilities of web application give possibility for a range of attacks on the web based applications.

For instance, web bot is a class of web security attack that creates an enormous threat to the security of every web application and web service (Heartfield et al., 2013). Principally, the web bot is a script or program which is developed to execute completely automated and repetitive task on web applications. The name bot is derived from the word robot, so it is also known as web robot. The intention of the development of web bots can either be bad or good. The functionality and the actions classify the web bot to be good web bot or bad web bot (Gilani et al., 2016). The most extensively used good web bot is web crawler. Its purpose is to index web sites and web applications in search...
engines (Thelwall, 2001). On the other hand, bad web bots are developed to carry out a range of destructive tasks (Rahman & Tomar, 2018).

The bad web bots are the reason of majority of security attacks on web sites. These web sites attacks are vary from small cyber crime like click fraud, Backlinks creations, and mass registration to big crimes like stealing of credit card information and credential stuffing (Thelwall et al., 2009). According to global web bot traffic report, web bots traffic comprises about fifty percent of the total web traffic (Zelfman, 2017). The distribution of web traffic is shown in Figure 1. From the figure it is clear that the thirty percent comes from good web bots and remaining twenty percent comes from bad web bots (Wang et al., 2014).

When the report critically examined, it illustrates the particular web sites such as small, medium and large are further exposed to web bot attacks in the year 2014. Thus, bad bots attacked categorically to all of these web sites. As depicted in the subsequent figure (Figure 2), the proportion of web traffic coming from bad web bots is steadily thirty percent, irrespective of its size.

In reality, bad bots for instance ScrapeBox (Shin et al., 2011) and XRumer (Hayati et al., 2012) are created for generating the Backlinks, web scraping, content scraping, form spamming, and automated registration of web services such as mailing. Web Bot defense mechanism can be categorize into two main approaches namely preventive approach and detective approach. Preventive approach requires direct human participation such as Turing test in form of CAPTCHA (Rahman & Tomar, 2012). However, advanced web bot such as XRumer is capable of bypassing this preventive approach generally used by web applications through solving CAPTCHA using optical character recognition (OCR). In fact, it has been reported in the year 2008, the web bot XRumer effectively evaded Google and Hotmail CAPTCHA to make enormous number of accounts with these web services. Similarly, Decaptcha application defeated CAPTCHA of Wikipedia just about twenty five percent of the time (Bursztein & Bethard, 2009).

The other approach is the detective approach which does not require direct human participation. It can be categorize further into two categories namely basic detection approach and advanced detection approach also known as analytical learning. In basic approach web log analysis is performed using string processing. On the other hand, in advanced detection approach machine learning algorithms are exploited. Current research on web bot detection primarily focusing on machine learning approaches, which accomplish significantly enhanced results than the other approaches. This is because, machine
Learning approach rely on set of procedures that is appropriately engineered. Even so, employing these traditional detection methods alone is not adequate in the cyber world. To increase the security practices, the web applications, and services has to additionally utilize analytical learning for regular monitoring of new web bot detection (Mahmood & Afzal, 2013).

However, there are problems associated with the traditional analytical detection system for instance, the limited processing speed. The security related data such as web access logs etc is generating at an exponential rate (Cybenko & Landwehr, 2012).

This paper introduces Web Bot Detection System based on Divisive Clustering and K-Nearest Neighbor Using Biostatistics Features Set.

Proposed system introduces novel taxonomy of feature set known as Biostatistics features. Biostatistics features are related to individual user activities on web applications and consist of a combination of distinctive actions that makes an individual. In this proposed technique, an improved and efficient clustering algorithm known as divisive clustering using 2-medoid is proposed and implemented for web bot detection. For generating the dataset, web bot automated tools including XRumer, Magic Submitter, Selenium, Botchief, and Autoit are configured.

These web bot tools enable the attacker to automatically post comment to a massive number of target websites, automatic account registration etc. Apart from these well known web bots, web bot attack scenario is created and installed to imitate the zero-day attack. This web bot attack scenario is able to create automatic account registration, bulk message posting, and can be used for data scrapping. The proposed system is also evaluated by performing a number of experiments over generated and real dataset. The performance metrics exploited for the evaluation of the proposed technique are Accuracy, Sensitivity, Specificity, Recall, F-Measure, and Matthews Correlation Coefficient.

Figure 2. Web bot Traffic according to size of web sites

| Size of Web Sites | Percentage of Requests |
|-------------------|------------------------|
| Large web sites   | 10000 to 100000 request/day |
| Medium web sites | 1000 to 10000 request/day |
| Small web sites  | 10 to 100 request/day    |

**Related Work**

A number of supervised machine learning and unsupervised machine learning models have been proposed in the past. These machine learning models are based on a range of web bot features. A noteworthy contribution in web bot detection has come from the work of Tan and Kumar (Tan & Kumar, 2004). In this work, the authors try to identify web bots by exploiting feature vectors taken from numerous characteristics of web sessions and then by examining web bots navigational patterns. They derived 25 features from web access logs such percentage of images requested, percentage of HTML pages requested, average time between two HTML pages requests etc. At last, they applied
C4.5 decision tree algorithm over dataset having human and web bot sessions with 25 different navigational features. Using C4.5 decision tree algorithm they were able to achieve 90% accuracy.

Bomhardt and Gaul (Bomhardt & Gaul, 2005) incorporated feature vectors for instance, percentage of response codes such as code 200, code 404 etc. Then, they trained the model using neural networks with required feature vectors. Stassopoulou and Dikaiakos (Stassopoulou & Dikaiakos, 2006) introduces new features such as maximum sustained click rate, total percentage of image requests, duration of session, pdf/ps requests, and responses codes. Finally, they trained the model using naive bayes algorithm to classify the sessions of a web access log as web bot or human and they were able to accuracy of 95%.

The work of Bhattarai et al (Bhattarai et al., 2009) particularly focuses on spam bot detection. They used features particularly for spam bot detection such as Post comment similarity, Stop words ratio, Redundancy ratio, and Sentence count. They test their model using number supervised learning methods including Naïve Bayes Classifier, SVM, Logistic Regression, and Decision Tree. They were able to achieve maximum 86% accuracy using Decision Tree classifier Gianvecchio et al (Gianvecchio et al., 2008) used 16 number of chat bots varying from simple to advanced bots. They employed two different classifiers to detect chat bots. The entropy based classifier is used to identify unknown chat bots, while the Bayesian classifier is efficient in identifying known chat bots.

Stevanovic et al (Stevanovic et al., 2012) proposed two added features: Percentage of sequential HTTP requests, Standard deviation of depth of requested page, Percentage of sequential HTTP requests. They employed seven different classifiers such as C4.5, RIPPER, Naïve Bayesian, Bayesian Network, k-NN, SVM and Neural Networks. They were able to achieve nearly 100% accuracy using Neural Networks, C4.5, RIPPER, and k-NN algorithms. Doran and Gokhale (Doran & Gokhale, 2016) proposed real time web bot detection model using Discrete Time Markov Chain (DTMC) Model. They were able to achieve F1 score of 75%. The work of Lagopoulos et (Lagopoulos et al., 2017) is mainly focused on detection of web bot in academic publishing domain. For this purpose they introduced novel features set such as Total Topics, Unique Topics, Page Similarity, Page Variance, and Boolean Page Variance. They employed four different models namely SVM, Gradient Boosting Model, Multi Layer Perceptron, and Extreme Gradient Boosting (XGB) model. They were able to achieve F-Measure of 91.81% and accuracy of 91.33%, using Gradient Boosting model. The comparative analysis of web bot detection methods is presented in literature table (Table 1).

Table 1. Literature Table: Summary of web bot detection

| S. No. | Year | Paper | Learning Approach | Type of Web Bot Detected | Feature Set | Results |
|-------|------|-------|-------------------|--------------------------|-------------|---------|
| 1     | 2002 | Tan et al | Decision Tree(C4.5 Algorithm) | Web Crawler | 25 different features | Able to achieve 90% Accuracy |
| 2     | 2005 | Bomhardt and Gaul | Neural Networks | Web Crawler | Response codes such as code 200, code 404 etc | Able to achieve Recall 94.7% and precision 95.4% |
| 3     | 2006 | Stassopoulou and Dikaiakos | Bayesian network | Web Crawler | Maximum sustained click rate, total percentage of image requests, duration of session, pdf/ps requests, and responses codes. | Able to achieve 95% Accuracy |

Table 1 continued on next page
However, all of these existing detection techniques have two limitations. First of all, these detection techniques are domain specific. For instance, work of Bhattarai is specific for detecting spam bots in blog sites; therefore they have chosen feature vectors related to spam post comment similarity and stop words ratio. In the same way, work of Lagopoulos is specific for detecting web bots in academic publishing; therefore they have chosen feature vectors related to their domain such as unique topics and page variance. Similarly, other authors work on chat bots and navigational patterns.

Second of all, these detection techniques are not scalable. Generally, these detection techniques have limitation that they operated their programs on single machine with limited computing power and insufficient storage. Moreover, security related data comes in numerous forms such as semi structured web access logs and sometimes unstructured data also.

In proposed web bot detection system, an attempt is made to give solution to these above mentioned problems. As far as the first problem is concerned, the framework is exploiting novel taxonomy of feature vectors known as Biostatics features. These features set are related to human users of web applications. In this proposed technique, the human users are differentiated from web bots rather than differentiating web bots form human users. When human user interacts with web applications they have certain patterns, for instance, duration spent on particular web page, duration spent on particular form fields such as text box and drop down, and movement among these form fields with

| S. No | Year | Paper | Learning Approach | Type of Web Bot Detected | Feature Set | Results |
|-------|------|-------|-------------------|--------------------------|-------------|---------|
| 4     | 2009 | Bhattarai et al | Naïve Bayes Classifier, SVM, Logistic Regression, Decision Tree | Spam Bot | Post comment similarity, Stop words ratio, Redundancy ratio, Sentence count | Able to achieve 86% Accuracy with Decision Tree Classifier. |
| 5     | 2011 | Gianvecchio et al | Entropy based classifier, Bayesian Classifier | Chat Bot | Message size, Inter message delay | For particular dataset, able to detect 100% of all periodic bots and detects 95% of random chat bots |
| 6     | 2012 | Stevanovic et al | 7 Classifiers: C4.5 RIPPER, Naïve Bayesian, Bayesian Network, k-NN, SVM and Neural Networks | Web Crawler | Percentage of sequential HTTP requests, Standard deviation of depth of requested page | Able to achieve nearly 100% accuracy using Neural Networks, C4.5, RIPPER and k-NN algorithms |
| 7     | 2016 | Doran and Gokhale | Discrete Time Markov Chain (DTMC) Model | Web Crawler | | Able to achieve F1 score of 75% |
| 8     | 2017 | Lagopoulos et al | 4 Classifiers: SVM, Gradient Boosting Model, Multi Layer Perceptron, eXtreme Gradient Boosting (XGB) | Web Crawler | Total Topics, Unique Topics, Page Similarity, Page Variance, Boolean Page Variance | Able to achieve F-Measure of 91.81% and accuracy of 91.33%, using Gradient Boosting |
tab key or else by means of the mouse, scrolling speed in vertical and horizontal directions, and typing pace. By using this taxonomy, specific feature set such as chat bot feature set and spam bot feature set are not required.

As far as the second problem is concerned, the proposed technique is constructing Binary Tree Dendrogram which is a compact representation of a dataset, all entries in a leaf node corresponds to a cluster that consumes number of data vectors within the threshold criteria. This technique is also memory efficient since it just stores a small number of abstracted vectors (medoids) instead of the complete dataset.

BIOSTATISTICS FEATURES FOR HUMAN CHARACTERIZATION

This section of the paper, presents features that are required for human characterization or human identification while they interact with web applications. These human characterizations include, time spent on particular page, pattern of surfing the web pages and how they fill web forms and post comments. In first part, details of web bots and automated programs such as Botchief, Selenium and AutoIt is provided and then description of how the web bot data and human user data is collected from the web applications. Lastly, the web application usage of web bots and humans are characterized in terms of biostatistics features that are collected.

DATA COLLECTION AND GATHERING

The Collection and Generation of log data are essentially, assembling the data from various sources in the database. To carry out this objective a variety of huge volume of logs are collected from numerous data sources. These sources of data comprises of live Institutional Web Application logs and Experimental Setup.

In first step the Apache server web access logs from live Institutional Web Application is collected; covering a time interval of one month. The Apache web server produces enormous quantity of important information about user access and errors. Particularly, the Apache web server records all the web requests serve by the web server.

In second step, the testbed is created for collecting the information log of malicious activities of web bots. In this testbed, isolated virtual machines are installed. Subsequently, in every virtual machine; samples of web bots and automated programs are installed and configured. These web bots include XRumer, Magic Submitter, Botchief, Selenium, and AutoIt.

XRumer is an automated web bot software designed for spamming on comment portion of blogs and web forums. This automated software is mainly used for search engine optimization and is developed by Botmaster Labs. It is capable of entering and registering into web forums and posting to it with the objective of enhancing the search engine rank. This automated software can sidestep security mechanisms generally used by many web application and forums, for instance web account registration, numerous types of CAPTCHAs before posting. It uses HTTP proxies in an effort to make it further complex for web administrators to block comments and posts by client IP.

Like XRumer, Magic Submitter is automated web bot software which submits website contents to different places on the World Wide Web to boost the website traffic in extremely short span of time.

Botchief is the most advanced automated web bot that is used to create massive accounts, web scraping including content scraping and web form submission. It is also used to collect and analyze web data, synchronize online accounts, upload and download data on websites.

Although, Selenium (Wang et al., 2016) was developed as an automated framework for testing the web applications. However, it is also exploited for executing the variety of web bot attacks such as Web Scrapping and Spaming etc.

AutoIt (Brand & Balvanz, 2005) is a simple scripting language for Windows based programs that simulate user input actions including mouse actions, mouse clicks and keystroke actions. It is
different from the other scripting languages, in a way that it can interact with the web applications by accurately using the mouse buttons and keyboard shortcuts.

Aside from these standard web bots, Form Spamming bot attack scenario is developed and executed in testbed. This Form spamming bot is a malicious program implemented as web browser in C#.NET for automated account creations. The main objective is to identify zero-day web bot attacks, since there is no signature of zero-day attack is available. The description of the developed spamming bot attack scenario is given below.

- **Name of the Attack:** Form Spamming Bot
- **Possible Attacker:** Web Programmer, Web Application Developer
- **Target:** Web Application, Web Server
- **Possible Vulnerabilities:** Insufficient Application Layer Protection, Invalidated Forwards and Redirects, Absence of Bot Protection
- **Possible Threats:** Spamming, Automatic account creation, Brute force attacks, Dictionary attacks, Phishing attack, Web Scraping, and Data Scraping
- **Resources Affected:** Wastage of server memory, credential stuffing, and sensitive data leakage

**EXTRACTION OF FEATURE VECTORS**

Extraction of Feature Vectors is an important and a necessary step in efficient, high dimensionality detection methods. It is usually a vital step for data processing prior to the application of machine learning algorithms. For precise and rapid classification, feature vectors have been extracted together from web access log (basic information of users), and from the implemented parser (human user actions) to exploit features brought by these data sources. Features vectors from these data source are typically related to the maliciousness, for instance spamming, injection and scrapping. The experimental results shall demonstrate how the grouping of features from the two data sources can facilitate improving the efficiency of the overall framework.

As mention in the previous section, each and every one of these available web bot detection methods are helpful for identifying simple web bots that cannot generate any human-like activities. These methods are inefficient to spot advanced web bots for instance, XRumer, Magic Submitter, and machine learning web bots, which could browse the web application by raising mouse and keyboard events. The feature vectors used in these existing detection methods are not enough capable to detect above mentioned advanced web bot by using web browsing behavior and web access log analysis.

In order to solve this problem, a new feature set known as Biostatistics features is introduced. Biostatistics features are based on events generated from mouse and keyboard when human users interact with the web applications. A new taxonomy of feature set (Biostatistics Features and Non Biostatistics Features) is proposed and implemented in this paper. Biostatistics Features are related to individual application user activity patterns consist of a combination of distinctive actions that makes an individual (Zhu, 2007).

These features include the identification and verification of human actions and their patterns. Rather than giving attention to the outcome of application user, biostatistics features give attention to, how an application users performing the specific actions. For instance, when filling out web form whether the form fields such as Email, Date of Birth are entered properly and correctly, but in what manner users enter form fields. These user action patterns include typing pace, duration spent on particular web page, duration spent on particular form fields, and movement among the form fields with tab key or else by means of the mouse.

On the other hand, the Non Biostatistics Features are general and not related to user actions. These features include IP Address, and user agent, request type, and response code. The detail of feature set has been given in Table 2 and Table 3 on each feature that has been used.


| S. No | Feature                        | Abbreviation | Definition                                                                                                                                                                                                 | Type (Basic/ Derived) | Unit |
|-------|--------------------------------|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|------|
| 1     | Input Source                   | IS           | It Indicates the type of device that generated the event. It has seven constant values for example, 1 for mouse, 6 for keyboard, 0 for unknown device.                                                        | Basic                 | Dimensionless |
| 2     | Click Pressure                 | CP           | The degree of pressure exerted at the time of clicking the device. It has a value between 0.0 for minimum click pressure and 1.0 for maximum click pressure.                                                   | Basic                 | Dimensionless |
| 3     | Horizontal Scrolling Amount    | HSA          | It is the distance traverse by mouse when scrolling in horizontal direction.                                                                                                                                | Basic                 | Pixel |
| 4     | Vertical Scrolling Amount      | VSA          | It is the distance traverse by mouse when scrolling in vertical direction.                                                                                                                                   | Basic                 | Pixel |
| 5     | Horizontal Scrolling Speed     | HSS          | It is the speed of mouse when scrolling in horizontal direction. It can be calculated as $HSS = \frac{HSA}{(T_2-T_1)}$ Where T2 and T1 are respectively time stamps when wheel stop moving and start moving. | Derived               | Pixel/ms |
| 6     | Vertical Scrolling Speed       | VSS          | It is the speed of mouse when scrolling in vertical direction. It can be calculated as $VSS = \frac{VSA}{(T_2-T_1)}$ Where T2 and T1 are respectively time stamps when wheel stop moving and start moving. | Derived               | Pixel/ms |
| 7     | Inter-Distance                 | ID           | Distance traverse by all the mouse move events.                                                                                                                                                               | Basic                 | Pixel |
| 8     | Time-Span                      | TS           | Difference between the last and first mouse move events. It can be calculated as $TS = T_2-T_1$ Where T2 and T1 are respectively time stamps when last mouse event and first mouse event occur. | Derived               | ms    |
| 9     | Mean Velocity                  | MV           | It is the average or mean velocity of mouse. It can be calculated as $MV=\frac{ID}{TS}$                                                                                                                     | Derived               | Pixel/ms |
| 10    | Keystroke Length               | KL           | It is the time between the key-press and key-release for a single key.                                                                                                                                        | Derived               | ms    |
| 11    | Inter Keystroke Length         | IKL          | Inter Keystroke Length is the time period between the initial and the final of ‘m’ consequent key presses.                                                                                                     | Derived               | ms    |
| 12    | Mean Typing Speed              | MTS          | The average or mean typing speed of a series of key strokes in character per milliseconds.                                                                                                                 | Derived               | char/ms |
| 13    | Session Length                 | SL           | It is total time spent on particular session. It can be calculated as time difference between the last and first request.                                                                               | Derived               | second |
| 14    | Inter Request Time             | IRT          | It is time spent by the user or web bot between two successive page requests. Users generally look out for complete HTML page rendering and hyperlinks prior to make any other request. Web bots are very quick as compared to human in finding hyperlinks in the returned HTML page. Additionally, web bots are also capable of visiting hyperlinks in a HTML page without waiting for the HTML page to finish complete rendering. | Derived               | second |
| 15    | Standard Deviation Time        | SDT          | It is Inter request time variance of all the requests in a session. For example, if there are 10 requests in sessions, then it is the average of squared differences from the mean value. Web bots are capable of making requests at constant period whereas a human user has to wait for hyperlinks makes a larger difference in the time between the web page requests as compared to web bots. Standard Deviation Time is calculated using the formula given below. $A_{IRT} = \sqrt{\frac{\sum_{i=1}^{n} (IRT - IRT_{mean})^2}{n-1}}$ | Derived               | second |
Table 3. Non Biostatistics Features

| S. No | Feature                        | Definition                                                                 | Type (Basic/Derived) | Unit |
|-------|--------------------------------|---------------------------------------------------------------------------|-----------------------|------|
| 1     | Distinct IP                    | The IP address of the web application user.                               | Basic                 |      |
| 2     | Request Type                   | The type of web request.                                                  | Basic                 |      |
| 3     | Content Type                   | The type of web content.                                                  | Basic                 |      |
| 4     | Content Size                   | Size of content in Bytes.                                                 | Basic                 |      |
| 5     | User Agent                     | The User Agent gives the information about type of application, operating system, and web browser. | Basic                 |      |
| 6     | Response Code                  | It point out whether the particular web request has been completed successfully or not. | Basic                 |      |
| 7     | Sum of Requests                | The total sum of requests in a particular session.                        | Derived               |      |
| 8     | Fraction of Error Requests     | It is the percentage or the Fraction of Error Requests.                   | Derived               |      |

**Table 2 continued**

Table 2 continued

| S. No | Feature                        | Abbreviation | Definition                                                                 | Type (Basic/Derived) | Unit |
|-------|--------------------------------|--------------|---------------------------------------------------------------------------|-----------------------|------|
| 16    | Entropy of Inter Request Time  | ENT          | Entropy indicates the uncertainty related to random variable. It points out that the request to a web application was random or directed. A random request to the web application will have larger entropy. Entropy of Inter Request Time is calculated using the formula given below [36].

\[
\text{Entropy (IRT)} = -\sum_{i=1}^{n} IRT_i \log(IRT_i)
\]

| 16    | Derived                        | second       |                                            |                       |      |

**PROPOSED TECHNIQUE**

This section will give a detailed description about the divisive hierarchical clustering algorithm using 2-medoid that is used to create smaller number of significant data vectors from a big dataset in a form of compact binary tree dendrogram. With the smaller number of data vectors, the k-nearest neighbor can achieve shorter searching time and improved classification performance.

In k-NN algorithm, the training dataset contains n number of vectors and each vector has m number of components i.e., features, where k stands for the number of nearest neighbors. Further each vector has its own predefined class label such as Web Bot and Human User.
In web bot detection method based on k-NN, the accuracy of prediction mostly rely on the selection of k value and the type of similarity function exploited for computation. The k-NN algorithm for web bot detection with k=1 and k=3 is shown in Figure 3.

The main advantage of using k-Nearest Neighbors classification algorithm in web bot detection is its efficiency and effectiveness, when dealing with large dataset of web requests. An added advantage of using this algorithm to other learning methods such as Neural Network and Support Vector Machine; is that its capability of dealing with the scalability issue.

Figure 3. k-NN Algorithm for Web Bot Detection with k=1 and k=3

However, there is a major limitation associated with this technique. The k-NN algorithm has to compute the similarity function or distance with all the instances of training data vectors at each prediction, which is time-consuming if there are huge numbers of training data vectors.

To overcome this limitation of k-NN algorithms, Compact Binary Tree Dendrogram is constructed with divisive clustering using 2-medoid algorithm as training phase. As a result, the time complexity of searching the k-nearest neighbor is reduced based on the binary tree dendrogram. Subsequently the time of similarity function computation is also decreased largely. The detail description of the binary tree dendrogram construction and efficient k-nearest neighbor query is presented in the following section.
DIVISIVE CLUSTERING ALGORITHM FOR COMPACT BINARY TREE DENDROGRAM CONSTRUCTION

Hierarchical clustering algorithms divides data vectors in levels by constructing a tree structure known as Dendrogram. This dendrogram is not a plane or flat set of clusters; rather it consists of multilevel hierarchy. In this multilevel hierarchy, clusters at single level are connected to the clusters at the next level. It gives the flexibility to the model to choose the suitable level in clustering.

Fundamentally, there are two ways in which hierarchical clustering can be achieved. Agglomerative hierarchical clustering: This is also known as bottom up hierarchical clustering in which each data vector starts in its own singleton cluster, and couple of clusters are fused as we go up the hierarchy (Arockiam et al., 2012).

Divisive hierarchical clustering: This is also known as top down hierarchical clustering. In this algorithm, initially all the data vectors are in one cluster. Then, the cluster is parted by means of a partitioning clustering algorithm such as k-means. This process is executed recursively until each vector comes in its own singleton cluster. This algorithm has the added advantage over bottom up clustering in terms of efficiency. For instance, in a number of cases entire hierarchy or complete dendrogram need not to generate up to single vector leaves.

At first, the Divisive clustering using 2-Means Algorithm is applied on dataset for generating the binary tree dendrogram. The algorithm starts from the top with each and every one of the vectors in single cluster. The cluster is parted using 2-Means algorithm. This process is repeated recursively and in parallel until every vector comes in singleton cluster of its own or given threshold criteria is achieved. It has been observed that using 2-Means algorithm for partitioning; it is becoming extremely sensitive if outliers and noise are present. Since, it calculates the mean of the vectors which may not be from the data set. In other words, it aims to minimize the sum squared errors.

So, in order to solve this problem Medoid is used instead of means. A Medoid of a vector set is a vector, whose average similarity to every one of the vectors in given data set is maximal. Specifically, Medoid is the most middle vector in the data set (Singh & Chauhan, 2011). This method is more robust and insensitive to outliers and noise.

Let \( \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_n \) be dataset of n vectors with similarity function SIM. Mathematically, Medoid can be defined as

\[
V_{medoid} = \text{argmax}_{V \in \{\vec{V}_1, \vec{V}_2, \ldots, \vec{V}_n\}} \sum_{i=1}^{n} \text{SIM}(CG, V_i)
\]

\[
CG = \sum_{i=1}^{n} V_i / n \quad \text{and} \quad \text{SIM is Similarity Function}
\]

For example, suppose dataset contains three vectors \( \vec{V}_1, \vec{V}_2, \) and \( \vec{V}_3 \) in 2D space having x, y components are (5,6), (7,6), and (15,15) respectively. To find out the medoid of a given dataset following steps are executed.

Step 1: Compute the Center of Gravity \( \text{CG} = \sum_{i=1}^{n} V_i / n \) of given dataset. It comes out to be \((5+7+15)/3, (6+6+15)/3) = (9, 9)\).

Step 2: Compute the similarity function of every vector with Center of Gravity that is \( \text{SIM}(CG, V_1), \text{SIM}(CG, V_2), \text{and} \text{SIM}(CG, V_3) \). Suppose using one of the similarity functions these values comes out to be \( \text{SIM}(CG, V_1) = 0.166, \text{SIM}(CG, V_2) = 0.211, \text{and} \text{SIM}(CG, V_3) = 0.105 \).

Step 3: Medoid of given data set is the vector for which the similarity function is maximized. In this case \( \text{SIM}(CG, V_2) = 0.211 \) is maximum, so the medoid is \( V_2 \).

Figure 4. Difference between medoid and mean in a 2D Space
Figure 4. illustrates the difference between medoid and mean in a 2D Space. The set of vectors in the left make a cluster, whereas the leftmost vector is an outlier. In both the figures the vector represented by red square is center of the cluster computed from mean and medoid respectively. Mean of the vectors is largely influenced by the outliers and hence can never represent the correct cluster centre. On the other hand medoid is more robust to the outliers and appropriately represents the centre of cluster.

While computing Mean and Medoid in the clustering algorithm the most important point is the selection of similarity or distance measure. Clustering algorithms need an accurate description of the proximity between two data vectors, in terms of either distance or similarity. The similarity measure is inversely related to the distance measure. As a result, greater the value of the distance, lesser is the value of similarity and vice versa. A range of similarity measures are available in literature and extensively applied. In order to prove the effectiveness of proposed clustering algorithm; three different similarity measures are exploited and these are given below (Strehl et al., 2000).

- Euclidian Distance Similarity
- Manhattan Distance Similarity
- Cosine Similarity

**EUCLIDIAN DISTANCE SIMILARITY**

This Similarity is based on Euclidian Distance which is the actual straight line distance between two vectors in Euclidean space. It is the most common distance measure employed in clustering algorithms.

For measuring Euclidian Distance Similarity between web requests, given two web request represented by their vectors $\overrightarrow{V_a}$ and $\overrightarrow{V_b}$ respectively, the Euclidean Distance Similarity $\text{SIM}_E$ between two vectors is defined as

$$\text{SIM}_E\left(\overrightarrow{V_a}, \overrightarrow{V_b}\right) = \frac{1}{1 + \text{ED}\left(\overrightarrow{V_a}, \overrightarrow{V_b}\right)}$$

Where,

$$\overrightarrow{V_a} = \left(F_{1,a}, F_{2,a}, \ldots, F_{i,a}\right)$$

$$\overrightarrow{V_b} = \left(F_{1,b}, F_{2,b}, \ldots, F_{i,b}\right)$$

$F_{i,a}$ is the $i$th Feature of Vector $\overrightarrow{V_a}$ and $\text{ED}\left(\overrightarrow{V_a}, \overrightarrow{V_b}\right)$ is the Euclidean Distance between vectors $\overrightarrow{V_a}$ and $\overrightarrow{V_b}$ and can be calculated from the formula given below.

$$\text{ED}\left(\overrightarrow{V_a}, \overrightarrow{V_b}\right) = \sqrt{\sum_{i=1}^{n} \left(F_{i,a} - F_{i,b}\right)^2}$$
MANHATTAN DISTANCE SIMILARITY

This Similarity is based on Manhattan Distance which is measured along the axes at ninety degree angles between two vectors in Euclidean space. The Manhattan Distance Similarity $SIM_M$ between two vectors is defined as

$$SIM_M(V_a, V_b) = \frac{1}{1 + MD(V_a, V_b)}$$

$MD(V_a, V_b)$ is the Manhattan Distance between vectors $V_a$ and $V_b$ and can be calculated from formula given below.

$$MD(V_a, V_b) = \sum_{i=1}^{n} |F_{i,a} - F_{i,b}|$$

COSINE SIMILARITY

Once web requests are symbolized as vectors, the similarity between the two vectors is the correlation between the two vectors. It is computed as the cosine of the angle between two vectors. For this reason, it is referred as cosine similarity. For measuring the Cosine Similarity between web requests, given two web request represented by their vectors $V_a$ and $V_b$ respectively, the Cosine Similarity $SIM_C$ between two vectors is defined as dot product of $V_a$ and $V_b$ and it is calculated from the formula given below.

$$SIM_C(V_a, V_b) = \frac{V_a \cdot V_b}{\sqrt{\sum_{i=1}^{n} F_{i,a}^2} \sqrt{\sum_{i=1}^{n} F_{i,b}^2}}$$

All these Similarity functions ($SIM_E$, $SIM_M$, $SIM_C$) has a final value of a real number between 0 to 1. A zero value indicates that the vectors are dissimilar entirely while value of one show that the vectors are identical virtually.

The pseudo code divisive hierarchical clustering using 2-medoid for Compact Binary Tree Dendrogram Construction is given below.
2-MEDOID ALGORITHM

Algorithm: 2-Medoid
Input:
D: Dataset D= \{ \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_n \} Containing n vectors. Each vector has m Components (Features set) \{F_1,F_2,\ldots,F_m\}
Output:
Set of 2 Clusters
Method:
1. Arbitrarily divide the Dataset D into two clusters
   \[ D_1 = \{ \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_{n/2} \} \text{ and } D_2 = \{ \vec{V}_{n/2+1}, \vec{V}_{n/2+2}, \ldots, \vec{V}_n \} \]
2. Calculate the Medoid of each cluster as
   \[ V_{medoid} = \text{argmax}_{V \in \{ V_1, V_2, \ldots, V_n \}} \sum_{i=1}^{n} \text{SIM}(CG, V_i) \]
   \[ CG = \sum_{i=1}^{n} V_i / N \text{ and SIM is Similarity Measure} \]
   \[ \text{REPEAT} \]
   \[ 3. \text{ Assign each vector to the cluster which the vector is most similar based on cosine similarity using the formula below} \]
   \[ \text{SIM}_C(\vec{V}_a, \vec{V}_b) = \frac{\vec{V}_a \cdot \vec{V}_b}{\| \vec{V}_a \| \| \vec{V}_b \|} = \frac{\sum_{i=1}^{n} F_{i,a} F_{i,b}}{\sqrt{\sum_{i=1}^{n} F_{i,a}^2} \sqrt{\sum_{i=1}^{n} F_{i,b}^2}} \]
   \[ 4. \text{ If there is reassignment of vector} \]
   \[ 5. \text{ THEN recalculate the Medoid of each cluster} \]
   \[ 6. \text{ UNTIL Convergence} \]

DIVISIVE HIERARCHICAL CLUSTERING USING 2-MEDOID

Algorithm: Divisive clustering using 2-Medoid
Input:
D: Dataset D= \{ \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_n \} Containing n vectors. Each vector has m Components (Features set) \{F_1,F_2,\ldots,F_m\}
Threshold Parameter (\Theta)
Output:
Compact Binary Tree Dendrogram
Method:
1. Execute 2-Medoid Algorithm on Dataset D= \{ \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_n \}
2. For each of the resultant Cluster \( C_i \) and \( C_j \)
   a. Recursively execute 2-Means Procedure on vectors in \( C_i \) and \( C_j \)
3. Until each vector is in its own singleton cluster \textbf{OR} SIM < \Theta

The 2-medoid algorithm will take the dataset as input and divide the dataset into two clusters. This procedure is repeated recursively until each vector comes in its own singleton cluster or threshold criteria is achieved. The value of threshold parameter can be set on similarity function which has
range of 0 to 1. The final outcome of the algorithm is Compact Binary Tree Dendrogram. An example of binary tree dendrogram with height $h = 4$ is shown in Figure 5.

Figure 5. Binary Tree Dendrogram with Height $h = 4$

At root the complete dataset represented by its medoid is present and next level the dataset is parted into two clusters represented by their corresponding medoids. The advantages of creating Compact Binary Tree Dendrogram are as follows:

- This Tree Dendrogram is a compact representation of a dataset, all entries in a leaf node corresponds to a cluster that consumes number of data vectors within the threshold criteria.
- Every node in a tree represents a cluster made from sub-clusters, which is represented by its medoid.
- It is able to handle outliers and noise efficiently, since it using medoid instead of mean.
- It is also memory efficient since it just stores a small number of abstracted vectors (medoids) instead of the complete dataset.
- A leaf node has no pointer link to any other nodes. Every leaf node cluster should satisfy the threshold criteria, i.e., similarity of every data vector with its medoid in cluster should be less than the threshold parameter ‘$\Theta$’.

A key parameter in constructing the Binary tree dendrogram is the ‘$\Theta$’ threshold since it finds out the size of the binary tree, so that the binary tree can fit into the primary memory. If ‘$\Theta$’ is very small, then the number of cluster nodes will be large, as a result, the algorithm may run out of main memory before even processing every data vectors. On the other hand if ‘$\Theta$’ is large then false positives will increase.

In proposed technique, the value ‘$\Theta$’ is determined intuitively based on the number of vectors, features of vectors, and range of each feature vector.
EFFICIENT K-NEAREST NEIGHBOR QUERY ALGORITHM

To compute the nearest neighbors query, a straightforward brute force search could be exploited. Nevertheless, in order to handle big volume of data with high dimensions, brute force will become expensive algorithm.

In order to evaluate the efficiency of the binary tree dendrogram, numerous experiments have been performed with a variety of datasets, and multiple k values. In first step, k-nearest neighbor query search will be performed using a brute force approach. In second step, k-nearest neighbor query search as will be performed to accomplish the key objective of this research, that is, to efficiently search and speed-up k-nearest neighbor query.

BRUTE FORCE ALGORITHM K-NEAREST NEIGHBOR QUERY

Brute force query search also known as exhaustive search is a generic problem solving method in which similarity of unknown vector is computed with every data vector in training dataset. The pseudo code of k-NN Algorithm using Brute Force is given below and cosine similarity is chosen for similarity function.

```
Algorithm: Brute Force k-NN
Input:
  a. D: Dataset D= {V_1, V_2, ..., V_n} Containing n vectors. Each vector has m Components (Features set) {F_1, F_2, ..., F_m}
  b. Unknown Vector V_{Unknown}
  c. Value of K
Output:
Classification Result; Vector V_{Unknown} is either classified as Web Bot or Legitimate User
Method:
1. Load the training Dataset D = {V_1, V_2, ..., V_n} in memory.
2. For i = 1 to m:
   Calculate the Similarity Function SIM(V_i, V_{Unknown})
   Using the formula below
   \[ SIM = \cos(\theta) = \frac{V_i \cdot V_{Unknown}}{|V_i||V_{Unknown}|} \]
3. Construct set S of K maximum similarity obtained. Each one of these similarity correspond to previously classified vectors.
4. Return the majority class label in set S.
```

K-NEAREST NEIGHBOR QUERY USING BINARY TREE DENDROGRAM

The dendrogram which is obtained from hierarchical clustering is a binary tree in which every node is cluster represented by its medoid. Starting with the first cluster i.e., the root node, the algorithm moves down the tree recursively, it goes right cluster of the tree or left cluster depending on whether the similarity function of unkown vector with medoid of left cluster is greater less than the similarity function of unkown vector with medoid of right cluster (SIM(Medoid_Left, V_{Unknown}) > SIM(Medoid_Right, V_{Unknown})).
It will maintain set S of K maximum similarity obtained and at the end, it will return the majority class label in set S.

The pseudo code of k-NN Algorithm using Binary Tree Dendrogram is given below and cosine similarity is chosen for similarity function.

**BINARY TREE VS. TERNARY TREE**

In Ternary Tree Dendrogram construction, 3-medoid algorithm is executed recursively instead of 2-medoid. A ternary tree similar to binary tree is a data structure wherein every node has at most three children, generally represented as Left, Mid and Right. For searching a k-nearest neighbor query in ternary tree dendrogram, the data set is divided into three parts and remove two third of the search space at each level.

At first glance, it seems that k-nearest neighbor query in ternary tree dendrogram will take less execution time, since it makes \( \log_3(n) \) recursive calls. On the other hand, k-nearest neighbor query in binary tree dendrogram will take more execution time, since it makes \( \log_2(n) \) recursive calls. By deeply examining the algorithm (k-NNBinaryTreeDendrogram), the following recursive formula for counting the comparisons in k-nearest neighbor query in binary tree dendrogram can be written as.

\[
F(n) = F\left(\frac{n}{2}\right) + 2
\]

Where the initial condition is:

\[
F(1) = O(1) = 1
\]
Similarly, the following recursive formula for counting the comparisons in k-nearest neighbor query in ternary tree dendrogram can be written as.

\[ F(n) = F\left(\frac{n}{3}\right) + 4 \]

Because at each iteration k-nearest neighbor query in ternary tree makes four comparisons as compared to k-nearest neighbor query in binary tree which only makes two comparisons.

To solve the recursive formula of k-nearest neighbor query in ternary tree, the recursive formula has re-written for \( n/3, n/9, \ldots, 3 \).

\[
F\left(\frac{n}{3}\right) = F\left(\frac{n}{9}\right) + 4 \\
F\left(\frac{n}{9}\right) = F\left(\frac{n}{27}\right) + 4 \\
F(9) = F(3) + 4 \\
F(3) = F(1) + 4
\]

Next, adding the left and the right sides of the above equations, the following equation is obtained:

\[
F(n) + F\left(\frac{n}{3}\right) + F\left(\frac{n}{9}\right) + \ldots + F(9) + F(3) = F\left(\frac{n}{3}\right) + F\left(\frac{n}{9}\right) + F\left(\frac{n}{27}\right) + F(3) + F(1) + (4 + 4 + \ldots + 4)
\]

In above equation number of 4’s on the right side is \( \log_3(n) \).

Lastly, the equal terms have been cancelled out and simplifying the remaining equation, the following expression is obtained for k-nearest neighbour query in ternary tree:

\[ F(n) = F(1) + 4 \log_3(n) \]

Similarly, the following expression is also obtained for k-nearest neighbour query in binary tree:

\[ F(n) = F(1) + 2 \log_2(n) \]

Hence, the evaluation of k-nearest neighbour query in binary tree and ternary tree is dependent on the evaluation of the expressions of \( 2 \log_3(n) \) and \( \log_2(n) \). From the basic property of logarithm the expression \( 2 \log_3(n) \) can be further formulated as \( \frac{2}{\log_2(3)} \times \log_2(n) \).
Because the value of \( \frac{2}{\log_2(3)} \approx 1.26 \) is greater than one, k-nearest neighbour query in ternary tree takes more time than k-nearest neighbour query in binary tree. Consequently, quaternary tree (4-medoid algorithm) will take more time than ternary tree. As a result the best partition of dataset is dividing the dataset into two parts at each level.

**HANDLING THE SCALABILITY ISSUE**

Scaling to large datasets is a complex task, one of the key challenges being the unfeasibility of loading the dataset into the primary memory of a particular computer. For instance, the size of the web access logs or other logs is greater than 1000 GBs. Fitting such high volume of data in primary memory is even more difficult task. When handling the large amounts of data, feasible methods are dimensionality reduction on the data, keeping the dataset on the secondary memory and loading the part of dataset in the primary memory, distribute the dataset on numerous machines and exploiting a distributed nearest neighbor query algorithm. Proposed technique attempt to address scalability issue in the following way:

- The proposed technique does not load the complete dataset into the main memory. It creates Tree Dendrogram in form of index which just stores a small number of abstracted vectors (medoids) instead of the complete dataset. Every node in a tree represents a cluster made from sub-clusters, which is represented by its medoid.
- When training dataset changes in real-time the proposed technique does not create the tree dendrogram from the scratch. It just follows the simple insertion procedure of new data vector in binary tree dendrogram which is given below.

Step 1: The new data vector is will always be a part of dataset i.e., it will always present at root level.
Step 2: Next, it will compare the similarity with medoid of left cluster and medoid of right cluster, whichever is the higher value it will belong to that cluster.
Step 3: This procedure is repeated recursively until it finds the leaf node.
Step 4: When newly inserted vector finally reach the leaf node, there can be two possible cases.
  - Step 4(a): First, if the leaf cluster can take in the new vector without violating the threshold criteria (Θ), then the insertion procedure will be completed and terminated.
  - Step 4(b): Second, if the entry in the leaf node violates the threshold criteria (Θ). In this case, that particular cluster leaf needs to split further using 2-medoid until threshold criteria is achieved.

**PERFORMANCE EVALUATION OF PROPOSED TECHNIQUE AND EXPERIMENTAL RESULTS**

To evaluate the performance of the proposed web bot detection technique, numbers of experimentations are carried out. The detailed explanation of different Datasets, Evolution Metrics, and Result Analysis are presented in subsequent section.

**DATA SET**

For the purpose of experimentation, two data sets are exploited; first Dataset (D1) is obtained from the access log of an institutional website covering a time period of two months. Second Dataset (D2) is generated from testbed wherein the samples of numerous web bots are installed. These two datasets
are exploited together to evaluate the proposed framework. The distribution of datasets D1 and D2 are shown in Figure 6. Table 4 shows distributions of the total number of vectors (web sessions) with web bot requests and legitimate web request in the datasets.

Table 4. Distribution of Dataset D1 and D2

| Dataset | Dataset Source | Total number of web request | Total number of vectors (web sessions) | Total number of web bot requests | Total number of legitimate requests |
|---------|----------------|----------------------------|---------------------------------------|---------------------------------|-----------------------------------|
| D1      | Institutional Website Access Log | 521588                     | 19283                                 | 1927                            | 17356                             |
| D2      | Testbed        | 46354                      | 21059                                 | 10817                           | 10242                             |

As a data cleaning process, the vectors i.e., the identified web sessions having web requests less than ten are removed from the experimental datasets. Because, in these data vectors; it is not possible to calculate the required features such as Inter Request Time, Standard Deviation Time, Entropy of Inter Request Time, and so on. These data vectors become insignificant as they do not have any valuable information to identify the web request.

EVALUATION PARAMETERS AND METRICS

The most essential parameters involved in the performance evaluation of detection framework are True Positive, False Positive, True Negative, and False Negative. The detail description of these parameters in context to web bot detection is presented in Table 5.

Table 5. Evaluation Parameters

| S. No | Symbol | Parameter       | Definition                                           |
|-------|--------|-----------------|------------------------------------------------------|
| 1     | TP     | True Positive   | Web Bots are correctly identified as web bots        |
| 2     | FP     | False Positive  | Legitimate requests are incorrectly identified as web bots |
| 3     | TN     | True Negative   | Legitimate requests are correctly identified as legitimate requests |
| 4     | FN     | False Negative  | Web Bots are incorrectly identified as legitimate requests |
The performance metrics exploited for the evaluation of the framework are Accuracy, Sensitivity, Specificity, Recall, F-Measure, and Matthews Correlation Coefficient (Powers, 2011). To attain extremely high accuracy can be straightforward by cautiously choosing the sample data size. If the only accuracy as a measure for evaluating the performance of the framework; then framework would become biased. Therefore, to overcome this problem, Sensitivity and Specificity metrics which are independent of the volume of dataset are also considered.

On the other hand, accuracy and f-measure are extensively employed in validating the supervised machine learning algorithms, in some cases they can be misleading. Since, they do not completely take the four parameters in their final score calculation. To keep away from these unsafe misleading results, there is an added evaluation matrix known as Matthews Correlation Coefficient (MMC). It considers all the four parameters and is usually regarded as unbiased measure which could be exploited even though the training classes are of extremely different sizes. It gives numeric value from −1 to +1. A MMC value of +1 corresponds to ideal prediction, value of 0 is random prediction, and value of −1 corresponds to complete disagreement between the observation and prediction (Baldi et al., 2000). The detailed description of these metrics is presented in Table 6.

### Table 6. Evaluation metrics for learning algorithms

| S. No | Measure                          | Formula                                      | Significance                                      |
|-------|----------------------------------|----------------------------------------------|--------------------------------------------------|
| 1     | Sensitivity (Recall)             | Sen = Rec = \( \frac{TP}{P} = \frac{TP}{TP + FN} \)   | The percentage of correctly identified web bots.  |
| 2     | Specificity                      | Spe = \( \frac{TN}{N} = \frac{TN}{TN + FP} \)     | The percentage of correctly identified legitimate web requests. |
| 3     | Accuracy                         | Acc = \( \frac{TP + TN}{TP + TN + FP + FN} \)     | The ability to distinguish between web bots and legitimate web request. |
| 4     | Precision                        | Pre = \( \frac{TP}{TP + FP} \)               | The percentage of web requests identified as web bots that are actually web bots. |
| 5     | F-Measure                        | \( F_1 = \frac{2 \times Rec \times Pre}{Rec + Pre} = \frac{2 \times TP}{2 \times TP + FP + FN} \) | Balanced measure that fuses precision and recall. |
| 6     | Matthews Correlation Coefficient | MMC = \( \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \) | It is unbiased measure that considers all the four parameters. It is generally taken into consideration when the training classes are of extremely different sizes. |

### RESULT ANALYSIS

In order to validate the accuracy and efficiency of proposed technique, 5-fold cross validation process is employed. In this process, the experimental dataset is divided into 5 approximately equivalent sizes of data subsets. Subsequently single data subset is utilized for testing the model and remaining 4 data parts are utilized for training the model. This procedure is continued 5 times in order to utilize
each data part at least one time. The Evaluation Metrics such as sensitivity and specificity is then the average of all the sensitivity and specificity computed in every fold.

Table 7. Result comparison of web bot detection using Brute Force Algorithm for k-Nearest Neighbor query with varying k values

| k-NN Classifier | Sensitivity | Specificity | Accuracy |
|-----------------|-------------|-------------|----------|
| k=1             | 85.40%      | 97.55%      | 96.18%   |
| k=5             | 82.55%      | 97.31%      | 95.58%   |
| k=11            | 89.36%      | 98.05%      | 97.04%   |
| k=15            | 87.59%      | 97.79%      | 96.62%   |
| k=19            | 85.92%      | 97.61%      | 96.29%   |

By means of 5-fold cross validation scheme, Table 7 shows a result comparison of the kNN algorithm for web bot detection with different values of k. By exploiting kNN algorithm with k=1, the achieved Sensitivity, Specificity and Accuracy are 85.40%, 97.55%, 96.18% respectively. When the value of k is 5, the Sensitivity of 82.55%, Specificity of 97.31%, and the Accuracy of 95.58% are achieved. For this specific dataset, when k=11, it gives the best result than other values. In this particular k value Accuracy reaches 97.04% and Specificity 98.05%. As a result, such value of k is used for further analysis. Figure 7 shows a result comparison of k-NN algorithm with varying k values, k = 1, 5, 11, 15 and 19

Figure 7. Result comparison of Brute Force k-NN algorithm with different k values

Different similarity and distance measures namely Euclidean Distance, Manhattan Distance, and Cosine Similarity are also employed in same training and test data sets using k-NN algorithm. The results are presented in Table 8 and result comparison is shown in Figure 8. The Accuracies are 86%, 85% respectively when Euclidean Distance and Manhattan Distance are employed. Again, for this specific dataset, when Cosine similarity is employed, it gives the best result than other measures. In
this case, Accuracy reaches 97%, Sensitivity reaches 89.36% and Specificity 98.05%. Hence, better Accuracy, Sensitivity, and Specificity can be achieved with cosine similarity measure.

Table 8. Result comparison of the Brute Force k-NN algorithm with different similarity measures

| k-NN Classifier (k=11) | Euclidean Distance | Manhattan Distance | Cosine Similarity |
|------------------------|--------------------|--------------------|-------------------|
| Sensitivity            | 65.21%             | 62.90%             | 89.36%            |
| Specificity            | 89.90%             | 89.24%             | 98.05%            |
| Accuracy               | 86.26%             | 85.34%             | 97.04%            |

Figure 8. Result comparison of k-NN algorithm with different similarity measures

A significant characteristic of the cosine similarity is its independence of vector length. This property is very helpful in web bot detection. Since, web bots are automated programs which, perform tasks that structurally repetitive, at a constant rate.

To prove the effectiveness of cosine similarity in web bot detection empirically, three web requests have been taken out along with three features namely Inter-distance, Time-Span, and Mean Velocity. First request belongs to Human, second belongs to the Web Bot and last is Unknown which is actually Web Bot. The feature values of these three web request are presented in Table 9.

Table 9. Feature value of Web Requests

| Features                  | Inter-Distance | Time-Span | Mean Velocity       |
|---------------------------|----------------|-----------|---------------------|
| Human                     | 575 Pixel      | 2328 ms   | 0.246993127 Pixel /ms |
| Web Bot                   | 2819 Pixel     | 1628 ms   | 1.731572482 Pixel /ms |
| Unknown                   | 571 Pixel      | 457 ms    | 1.249452954 Pixel /ms |
Then, the similarity is calculated using three similarity measures namely \( \text{SIM}_E \), \( \text{SIM}_M \), and \( \text{SIM}_C \). The results are presented in Table 10.

**Table 10. Similarity Measures of Web Requests**

| Similarity       | SIM(Unknown, Web Bot) | SIM(Unknown, Human) |
|------------------|-----------------------|---------------------|
| Euclidean Similarity | 0.000394367           | 0.000534187         |
| Manhattan Similarity | 0.000198166           | 0.000237883         |
| Cosine similarity | 0.988584929           | 0.793841111         |

From the similarity values given in Table 13 following results can be inferred. Euclidean Similarity of Unknown with Human (0.000534187) is greater than Euclidean Similarity of Unknown with Web Bot (0.000394367). Hence, Unknown is incorrectly identified as Human. In the same way, Manhattan Similarity of Unknown with Human (0.000237883) is greater than Manhattan Similarity of Unknown with Web Bot (0.000198166). Thus, in this case also Unknown is incorrectly identified as Human. On the other hand, Cosine Similarity of Unknown with Human (0.793841111) is less than Cosine Similarity of Unknown with Web Bot (0.988584929). As a result, in this case the Unknown is correctly identified as Web Bot.

**Table 13. Experimental results of k-NN search query using Binary Tree Dendrogram for different datasets D1, D2, (D1+D2)**

| Dataset | Sensitivity | Specificity | Precision | Accuracy | F1-Score | MCC   |
|---------|-------------|-------------|-----------|----------|----------|-------|
| D1      | 79.17%      | 92.17%      | 78.84%    | 88.66%   | 79.00%   | 71.24%|
| D2      | 83.33%      | 94.27%      | 83.70%    | 91.41%   | 83.52%   | 77.71%|
| D1+D2   | 81.20%      | 93.21%      | 81.20%    | 90.02%   | 81.20%   | 74.41%|

Listed in Table 13 are the 5-fold cross validations experimental results achieved by the execution of k-NN search query using Binary Tree Dendrogram for different datasets D1, D2 and combine D1+D2.

From dataset D2 which is generated dataset, a maximum MCC of 77.71% is achieved with 91.41% accuracy. In this, the sensitivity and specificity are respectively 83.33%, 94.27% with 83.52% F-Score. It should be noted that, in case of dataset D1 which is obtain from live and real web application a maximum MCC of 71.24% is achieved with 88.66% accuracy. In this, the sensitivity and specificity are respectively 79.17%, 92.17% with 79.00% F-Score. These results indicate that the proposed web bot features set with k-NN search query using Binary Tree Dendrogram are enough capable to detect with high accuracy and MCC value in logarithmic time as opposed to linear time.

**CONCLUSION**

The detection of malicious bots is one of the most contemporary research areas in cyber security. As the various security related data for instance, web access logs and other logs are generating at enormous speed by web applications and web services, the present detection techniques are not enough capable to protect against new type of web bot attacks. Most of the current detection techniques are developed for particular type of web bots. In addition to that, the conventional data bases are incompetent of
storing the security data in a single machine. In this paper, scalable technique for web bot detection has been proposed. The proposed technique is capable of detecting variety of web bots including form spamming bots and web scraping bots.

The following four important conclusions were drawn from this research:

- The proposed technique exploits 2-medoid divisive clustering algorithm for constructing Binary Tree Dendrogram which is a compact representation of a dataset, all entries in a leaf node corresponds to a cluster that consumes number of data vectors within the threshold criteria. This technique is also memory efficient since it just stores a small number of abstracted vectors (medoids) instead of the complete dataset. Searching k-NN query using binary tree takes O(log(n)) time as opposed to linear time in simple k-NN query.
- The proposed taxonomy of web bot feature set is come out to be very effective in distinguishing human users from web bots. With the help of proposed taxonomy web bot detection technique has become independent of specific bot types and their feature set.
- The detection technique is evaluated through series of experiments on set of dataset collected from institutional website and generated dataset from test bed. From generated dataset from testbed, a maximum MCC of 77.71% is achieved with 91.41% accuracy. In this, the sensitivity and specificity are respectively 83.33%, 94.27% with 83.52% F-Score which is almost equivalent to accuracy obtained from generated dataset.
- Different similarity measures such as Euclidean Distance similarity, Manhattan Distance similarity, and Cosine Similarity are employed on datasets using k-NN algorithm. The Accuracies are 95.45%, 94.77% respectively when Euclidean Distance and Manhattan Distance are employed. Again, for this specific dataset, when Cosine similarity is employed, it gives the best result than other measures. In this case, Accuracy reaches 97.04% Sensitivity reaches 89.36% and Specificity 98.05%.
REFERENCES

Ahmed, A., Awad, E., & Traore, I. (2007). A new biometric technology based on mouse dynamics. *IEEE Transactions on Dependable and Secure Computing, 4*(3), 165–179.

Arockiam, L., Baskar, S. S., & Jeyasimman, L. (2012). Clustering techniques in data mining. *Asian Journal of Information Technology, 11*(1), 40–44.

Bhattarai, A., Rus, V., & Dasgupta, D. (2009, March). Characterizing comment spam in the blogosphere through content analysis. In *Computational Intelligence in Cyber Security, 2009. CICS’09. IEEE Symposium on* (pp. 37-44). IEEE.

Bland, J. M., & Altman, D. G. (1996). Statistics notes: Measurement error. *BMJ (Clinical Research Ed.), 312*(7047), 1654.

Bomhardt, C., Gaul, W., & Schmidt-Thieme, L. (2005). Web robot detection-preprocessing web logfiles for robot detection. In *New developments in classification and data analysis* (pp. 113–124). Springer.

Bours, P. (2012). Continuous keystroke dynamics: A different perspective towards biometric evaluation. *Information Security Technical Report, 17*(1-2), 36–43.

Bours, P., & Fullu, C. J. (2009, September). A login system using mouse dynamics. In *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP’09. Fifth International Conference on* (pp. 1072-1077). IEEE.

Brand, J., & Balvanz, J. (2005, November). Automation is a breeze with autoit. In *Proceedings of the 33rd annual ACM SIGUCCS conference on User services* (pp. 12-15). ACM.

Bursztein, E., & Bethard, S. (2009, August). Decaptcha: breaking 75% of eBay audio CAPTCHAs. In *Proceedings of the 3rd USENIX conference on Offensive technologies* (p. 8). USENIX Association.

Cybenko, G., & Landwehr, C. E. (2012). Security analytics and measurements. *IEEE Security and Privacy, 10*(3), 5–8.

Doran, D., & Gokhale, S. S. (2016). An integrated method for real time and offline web robot detection. *Expert Systems: International Journal of Knowledge Engineering and Neural Networks, 33*(6), 592–606.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security, 28*(1-2), 18-28.

Gianvecchio, S., Xie, M., Wu, Z., & Wang, H. (2008, July). Measurement and Classification of Humans and Bots in Internet Chat. In *USENIX security symposium* (pp. 155-170). USENIX.

Gilani, Z., Wang, L., Crowcroft, J., Almeida, M., & Farahbaksh, R. (2016, April). Stweeler: A framework for twitter bot analysis. In *Proceedings of the 25th International Conference Companion on World Wide Web* (pp. 37-38). International World Wide Web Conferences Steering Committee.

Han, J., Pei, J., & Kamber, M. (2011). *Data mining: concepts and techniques*. Elsevier.

Hayati, P., Firoozeh, N., Potdar, V., & Chai, K. (2012, September). How much money do spammers make from your website? In *Proceedings of the CUBE International Information Technology Conference* (pp. 732-739). ACM.

Jin, J., Offutt, J., Zheng, N., Mao, F., Koehl, A., & Wang, H. (2013, June). Evasive bots masquerading as human beings on the web. In Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on (pp. 1-12). IEEE.

Kolakowska, A. (2013, June). A review of emotion recognition methods based on keystroke dynamics and mouse movements. In *Human System Interaction (HSI), 2013 The 6th International Conference on* (pp. 548-555). IEEE.

Lagopoulos, A., Tsoumakas, G., & Papadopoulos, G. (2017). *Web Robot Detection in Academic Publishing*. arXiv preprint arXiv:1711.05098.

Mahmood, T., & Afzal, U. (2013, December). Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools. In *Information assurance (ncia), 2013 2nd national conference on* (pp. 129-134). IEEE.
Powers, D. M. (2011). *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. Academic Press.

Rahman, R., Tomar, D. S., & Das, S. (2012, May). Dynamic image based captcha. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on* (pp. 90-94). IEEE.

Rahman, R. U., & Tomar, D. S. (2018). Botnet Threats to E-Commerce Web Applications and Their Detection. In Improving E-Commerce Web Applications Through Business Intelligence Techniques (pp. 48-81). IGI Global.

Shin, Y., Gupta, M., & Myers, S. A. (2011, March). The Nuts and Bolts of a Forum Spam Automator. LEET.

Singh, S. S., & Chauhan, N. C. (2011, May). K-means v/s K-medoids: A Comparative Study. In *National Conference on Recent Trends in Engineering & Technology (Vol. 13)*. Academic Press.

Spam swine break next-gen captchas. (n.d.). http://www.theregister.co.uk/2008/10/03/captcha_break

Stassopoulou, A., & Dikaiakos, M. D. (2006, August). Crawler detection: A Bayesian approach. In *Internet Surveillance and Protection, 2006. ICISP'06. International Conference on* (pp. 16-16). IEEE.

Stevanovic, D., An, A., & Vlajic, N. (2012). Feature evaluation for web crawler detection with data mining techniques. *Expert Systems with Applications, 39*(10), 8707–8717.

Strehl, A., Ghosh, J., & Mooney, R. (2000, July). Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI 2000)* (Vol. 58, p. 64). AAAI.

Tan, P. N., & Kumar, V. (2004). Discovery of web robot sessions based on their navigational patterns. In *Intelligent Technologies for Information Analysis* (pp. 193–222). Springer.

Thelwall, M. (2001). A web crawler design for data mining. *Journal of Information Science, 27*(5), 319–325. doi:10.1177/016555150102700503

Traore, I., Woungang, I., Obaidat, M. S., Nakkabi, Y., & Lai, I. (2012, November). Combining mouse and keystroke dynamics biometrics for risk-based authentication in web environments. In *Digital Home (ICDH), 2012 Fourth International Conference on* (pp. 138-145). IEEE.

Wang, W., Zheng, Y., Xing, X., Kwon, Y., Zhang, X., & Eugster, P. (2016, November). Webranz: web page randomization for better advertisement delivery and web-bot prevention. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 205-216). ACM.

Zelfman, I. (2017, Jan. 24). Bot traffic report 2016. *Imperva Incapsula Blog.*

Zhu, S. R. (2007). Authentication based on feature of hand-written signature. *Journal of Central South University of Technology, 14*(4), 563–567.