BotChase: Integrated Unsupervised Learning with Decision Tree Classifier for Graph-Based Bot Detection

V. Krishna Sahithi[1], R. Jyothika[1], S. Preethi[1], Dr. A. R. Siva Kumaran[2]

[1]UG Student, [2]Professor, [1,2]Department of Computer Science and Engineering

[1,2]Malla Reddy Engineering College for Women (UGC-Autonomous), Maisammaguda, Secunderabad, Telangana, India

ABSTRACT

Bot detection using machine learning (ML), with network flow-level features, has been extensively studied in the literature. However, existing flow-based approaches typically incur a high computational overhead and do not completely capture the network communication patterns, which can expose additional aspects of malicious hosts. Recently, bot detection systems that leverage communication graph analysis using ML have gained attention to overcome these limitations. A graph-based approach is rather intuitive, as graphs are true representation of network communications. To overcome from the issues arisen from existing models, this project uses supervised and unsupervised algorithms, and these algorithms will be trained and generate a model and this model will be applied on new request data to identify whether request is normal or attack. Using unsupervised (K-means) algorithm, we will separate dataset into Bot (attack) and BENIGN (normal) records. K-means will arrange similar records in one cluster, and we will filter out all those records which has a smaller number of requests. All high request number of records will consider as BOT or attack. After separating records, it uses graph-based features extraction technique to extract features from dataset. Dataset will be passed to graph and each IP will be consider as VERTEX and then connect source and destination with edges. Edges will have weight based on its incoming and outgoing link connections. To get edge weight we will calculate between_ness centrality, incoming edge weight, outgoing edge weight and alpha_centrality weight. After all this calculation we will extract in_degree, out_degree, in_degree_weight, out_degree_weight, between_ness, clustering and alpha_centrality as features. Any record which has high number of connections will mark its label as 1 (BOT) otherwise 0 (normal). After features extraction from graph, we will go for normalization to get mean values of each feature. Normalized features will be used to train decision tree classifier and this model can be used to predict type of future requests.

Keywords: Bot detection, machine learning, DoS attack, K-means clustering

1. INTRODUCTION

Undoubtedly, organizations are constantly under security threats, which not only costs billions of dollars in damage and recovery, it often also detrimentally affects their reputation. A botnet-assisted attack is a widely known threat to these organizations. According to the U.S. Federal Bureau of Investigation, “Botnets caused over $9 billion in losses to U.S. victims and over $110 billion globally. Approximately 500 million computers are infected each year, translating into 18 victims per second.” The most infamous attack, called Rustock, infected 1 million machines, sending up to 30 billion spam emails a day [1]. Hence, it is imperative to defend against these botnet-assisted attacks. A botnet is a collection of bots, agents in compromised hosts, controlled by botmasters via command and control (C2) channels. A malevolent adversary controls the bots through botmaster, which could be distributed across several agents that reside within or outside the network. Hence, bots can be used for

tasks ranging from distributed denial-of-service (DDoS) to massive-scale spamming, to fraud and identify theft. While bots thrive for different sinister purposes, they exhibit a similar behavioral pattern when studied up-close. The intrusion kill-chain [2] dictates the general phases a malicious agent goes through in-order to reach and infest its target.

Anomaly-based methods are widely used in both detection [3], [4]. They first establish a baseline of normal behavior for the protected system and model a decision engine. The decision engine determines and alerts any divergence or statistical deviations from the norm as a threat. Machine learning (ML) [3] is an ideal technique to automatically capture the normal behavior of a system. The use of ML has boosted the scalability and accuracy of anomaly-based IDSs [4]. The most widely employed learning paradigms in ML include supervised and unsupervised. Supervised learning uses labeled training datasets to create models. It is employed to learn and identify patterns in the known training data. However, labeling is non-trivial and typically require domain experts to manually label the datasets [3]. This can be cumbersome and prone to error, even for small datasets. On the other hand, unsupervised learning uses unlabelled training datasets to create models that can discriminate between patterns in the data.

An important step prior to learning, or training a ML model, is feature extraction. These features act as discriminators for learning and inference, reduce data dimensionality, and increase the accuracy of ML models. The most commonly employed features in bot detection are flow-based (e.g., source and destination IPs, protocol, number of packets sent and/or received, etc.). However, these features do not capture the topological structure of the communication graph, which can expose additional aspects of malicious hosts. In addition, flowlevel models can incur a high computational overhead, and can also be evaded by tweaking behavioral characteristics e.g., by changing packet structure [5].

Graph-based features, derived from flow-level information, which reflect the true structure of communications, interactions, and behavior of hosts, are an alternate that overcome these limitations. We show that incorporating graph-based features into ML yields robustness against complex communication patterns and unknown attacks. Moreover, it allows for cross-network ML model training and inference. The major contributions of this paper are as follows:

- We propose BotChase, an anomaly-based bot detection system that is protocol agnostic, robust to zero-day attacks, and suitable for large datasets.
- We employ a two-phased ML approach that leverages both supervised and unsupervised learning. The first phase filters presumable benign hosts. This is followed by the second phase on the pruned hosts, to achieve bot detection with high precision.
- We propose feature normalization (F-Norm) on top of graph-based features in BotChase and evaluate various ML techniques. Our graph-based features, inspired from the literature and derived from network flows, undergo F-Norm to overcome severe topological effects. These effects can skew bot behavior in different networks, exacerbating ML prediction. Furthermore, these features allow to combine data from different networks and promote spatial stability [6] in the ML models.
- We compare the performance of our graph-based features with flow-based features from BotMiner [7] and BClus [8] in a prototype implementation of BotChase. Furthermore, we compare BotChase with BotGM [9] and the end-to-end system proposed for BClus.
- We evaluate the BotChase prototype system in an online setting that recurrently trains and tests the ML models with new data. We also leverage the Hoeffding Adaptive Tree (HAT) [10] classifier for incremental learning. This is crucial to account for changes in network traffic and host behavior.

2. LITERATURE SURVEY

Botnet detection has been an active area of research that has generated a substantial body of work. Common botnet detection approaches are passive. They assume successful intrusions and focus on identifying infected hosts (bots) or detecting C2 communications, by analyzing system logs and network data, using signature- or anomaly-based techniques. Signature-based techniques have commonly been used to detect pre-computed hashes of existing malware in hosts and/or network traffic. They are also used to isolate IRC-based bots by detecting bot-like IRC nicknames [12], and to identify C2-related DNS requests by detecting C2-like domain names [13].

Metadata such as regular expressions based on packet content and target IP occurrence tuples [14] is an example of what could be employed in a signature and pattern detection algorithm. More generally, signature-based techniques have been employed to identify C2 by comparison with known C2 communication patterns extracted from observed C2 traffic, and infected hosts by comparison with static profiles and behaviours of known bots [15]. However, they can be easily subverted by unknown or modified threats, such as zero-day attacks and polymorphism [15], [16]. This undermines their suitability to detect sophisticated modern botnets.

On the other hand, anomaly-based techniques use heuristics to associate certain behaviour and/or statistical features extracted from system or network logs, with bots and/or C2 traffic. C2 occurs at the early stages of a botnet's lifecycle, thus its detection is deemed essential to prevent malicious activities. Most existing anomaly-based C2 detection techniques are based on the statistical features of packets and flows [7], [12], [17]–[27]. Works like [17], [18] are focused on specific communication protocols, such as IRC, providing narrow-scoped solutions. Whereas, BotMiner [7] is a protocolindependent solution, which assumes that bots within the same botnet are characterized by similar malicious activities and communication patterns. This assumption is an over simplification, since botnets often randomize topologies [15] and communication patterns as we observe in newer malwares, such as Mirai [28]. Other works, such as [23], [27], leverage ML and traffic-based statistical features, for detecting C2 with low error rates. However, such techniques require that all flows are compared against each other to detect C2 traffic, which incurs a high computational overhead. In addition, they are unreliable, as they can be evaded with encryption and by tweaking flow characteristics [5]. Therefore, it is evident that a non-protocol-specific, more efficient, and less evadable detection method is desired.

Anomaly-based bot detection solutions that do not focus on detecting C2 per se, but rather identify bots by observing and analyzing their activities and behaviour, address some of the aforementioned issues. Graph-based approaches, where host network activities are represented by communication graphs, extracted from network flows and host-to-host communication patterns, have been proposed in this regard [5], [9], [29]– [40]. Le et al. [37] present a strong case for leveraging SelfOrganizing Maps (SOMs) in the context of bot detection with recall rates beyond 90%. However, SOM remains an unsupervised learning algorithm that ultimately requires manual expertise to distinguish unknown network traffic.

BotGM [9] builds host-to-host communication graphs from observed network flows, to capture communication patterns between hosts. A statistical technique, the inter-quartile method, is then used for outlier detection. Their results exhibit moderate accuracy with low false positives (FPs) based on different windowing parameters. However, BotGM generates multiple graphs for every single host. That is, for every pair of unique IPs, a graph is constructed, such that every node in the graph

represents a unique 2-tuple of source and destination ports, with edges signifying the time sequence of communication. However, this entails a high overhead.

Khanchi et al. [41] propose botnet detection on non-stationary stream of data using incremental non-overlapping window. They propose to use a team of genetic programs to predict on records in a stream of data. These records are archived to build a data subset to further train the classifier. The true labels of the records are requested from human operators as long as the label budget is met. However, if records from minor classes are targeted for true label queries, minor classes are promoted aggressively, reducing the performance of major classes.

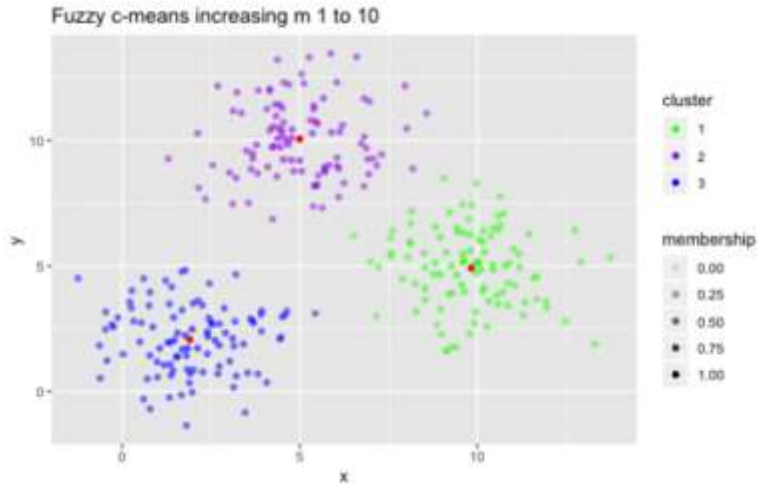
Chowdhury et al. [39] use ML for clustering the nodes in a graph, with a focus on dimensionality and topological characterization. Their assumption is that most benign hosts will be grouped in the same cluster due to similar connection patterns, hence can be eliminated from further analysis. Such a crucial reduction in nodes effectively minimize detection overhead. However, their graph-based features are plagued by severe topological effects (cf., Section IV). They use statistical means and user-centric expert opinion to tag the remaining clusters as malicious or benign. However, leveraging expert opinion can be cumbersome, error prone and infeasible for large datasets. Recently, rule-based host clustering and classi- fication [40] have been proposed, where pre-defined thresholds are used to discriminate between benign and suspicious hosts. Unfortunately, relying on static thresholds make the technique prone to evasion and less robust to ML-based outlier detection.

Big Data has received a lot of attention lately, which is also often paired with streaming. Employing ML in a streaming context [42] undoubtedly yields better result than batching. Keen statistical techniques, such as concept drifts and ADWIN windowing, help surmount the challenges facing classification in data streaming. However, retraining the ML model only when a concept drift occurs may require specific threshold tuning, which does not generalize.

3. EXISTING SYSTEM

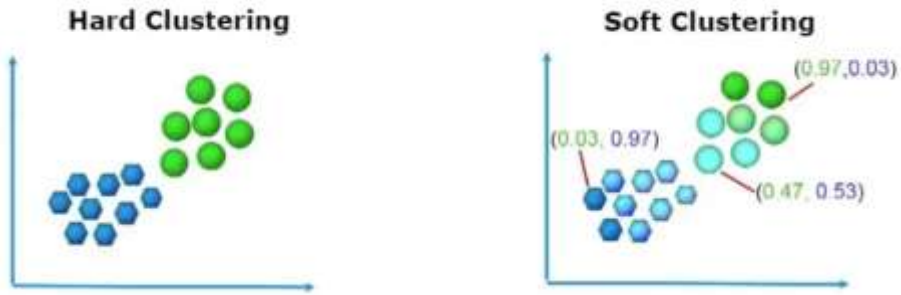
Fuzzy C-Means Clustering (FCM) Algorithm

Fuzzy logic principles can be used to cluster multidimensional data, assigning each point a membership in each cluster center from 0 to 100 percent. This can be very powerful compared to traditional hard-threshold clustering where every point is assigned a crisp, exact label. This algorithm works by assigning membership to each data point corresponding to each cluster center on the basis of distance between the cluster center and the data point. More the data is near to the cluster center more is its membership towards the particular cluster center. Clearly, summation of membership of each data point should be equal to one.



It is an unsupervised clustering algorithm that permits us to build a fuzzy partition from data. The algorithm depends on a parameter m which corresponds to the degree of fuzziness of the solution. Large values of m will blur the classes and all elements tend to belong to all clusters. The solutions of the optimization problem depend on the parameter m. That is, different selections of m will typically lead to different partitions. Given below is a gif that shows the effect of the selection of m obtained from the fuzzy c-means.

K-Means versus Fuzzy C-Means



Let us compare these two powerful algorithms to get a clear idea of where the fuzzy c-means algorithm fits in.

Attribution to a cluster: In fuzzy clustering, each point has a probability of belonging to each cluster, rather than completely belonging to just one cluster as it is the case in the traditional k-means. In Fuzzy-C Means clustering, each point has a weighting associated with a pardticular cluster, so a point doesn't sit "in a cluster" as much as has a weak or strong association to the cluster, which is determined by the inverse distance to the center of the cluster.

Speed: Fuzzy-C means will tend to run slower than K means, since it's actually doing more work. Each point is evaluated with each cluster, and more operations are involved in each evaluation. K-Means just needs to do a distance calculation, whereas fuzzy c means needs to do a full inverse-distance weighting.

Personal Opinion: FCM/Soft-K-Means is "less stupid" than Hard-K-Means when it comes to elongated clusters (when points otherwise consistent in other dimensions tend to scatter along a particular dimension or two).

Steps in Fuzzy C-Means

Assume a fixed number of clusters k.

Initialization: Randomly initialize the k-means μk associated with the clusters and compute the probability that each data point xi is a member of a given cluster k, P (point xi has label k|xi, k).

Iteration: Recalculate the centroid of the cluster as the weighted centroid given the probabilities of membership of all data points xi

Termination: Iterate until convergence or until a user-specified number of iterations has been reached (the iteration may be trapped at some local maxima or minima).

K-Nearest Neighbor (KNN) Algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Steps

1. Get labeled data: The labeled data consists of features and labels. Features are the characteristics or the property of the object whereas labels are the class of the object with those features.
2. Convert labeled data to encoded data: Usually computations are based on numerical form so we convert the data to numeric form by encoding them.
3. Create feature set: Creating a set of features by packing the features.
4. Split the data for train and test: The data are split training and testing. Usually, 80% for training and 20% for testing but can select based on need.
5. Train the classifier: Training the classifier with the training data by specifying the value of k. Use k =3 for binary classification, i.e., two labels classification. If used k =1 then it is simply a nearest neighbor classifier.
6. Test the classifier: Testing the classifier with the testing data.
7. Evaluate: Evaluating the classifier using confusion matrix and its evaluation metrics i.e., accuracy, precision, recall, et cetera.

Disadvantages of existing system

- Apriori specification of the number of clusters.

- With lower value of β we get the better result but at the expense of a greater number of iterations.
- Euclidean distance measures can unequally weight underlying factors.

4. PROPOSED SYSTEM

4.1 CTU dataset

The CTU-13 is a dataset of botnet traffic that was captured in the CTU University, Czech Republic, in 2011. The goal of the dataset was to have a large capture of real botnet traffic mixed with normal traffic and background traffic. The CTU-13 dataset consists in thirteen captures (called scenarios) of different botnet samples. On each scenario we executed a specific malware, which used several protocols and performed different actions. Each scenario was captured in a pcap file that contains all the packets of the three types of traffic. These pcap files were processed to obtain other type of information, such as NetFlows, WebLogs, etc.

4.2 K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training.
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

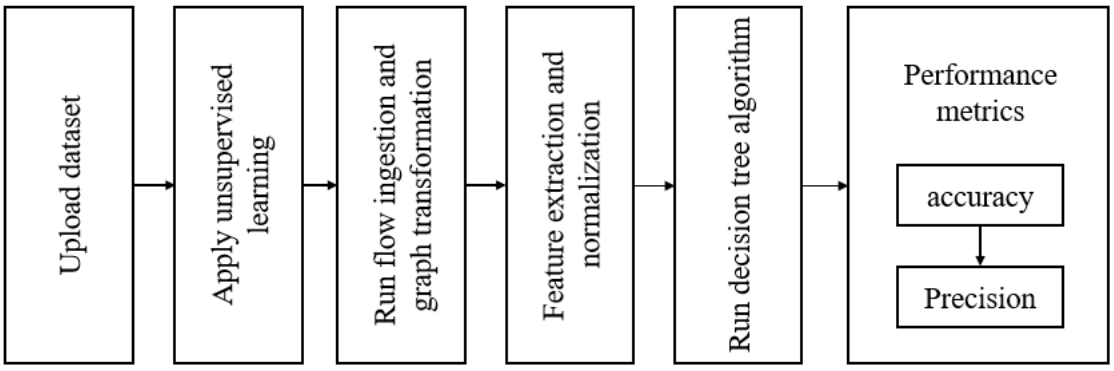


Fig. 4.1: Block diagram of proposed system.

The algorithm takes the unlabelled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering

algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

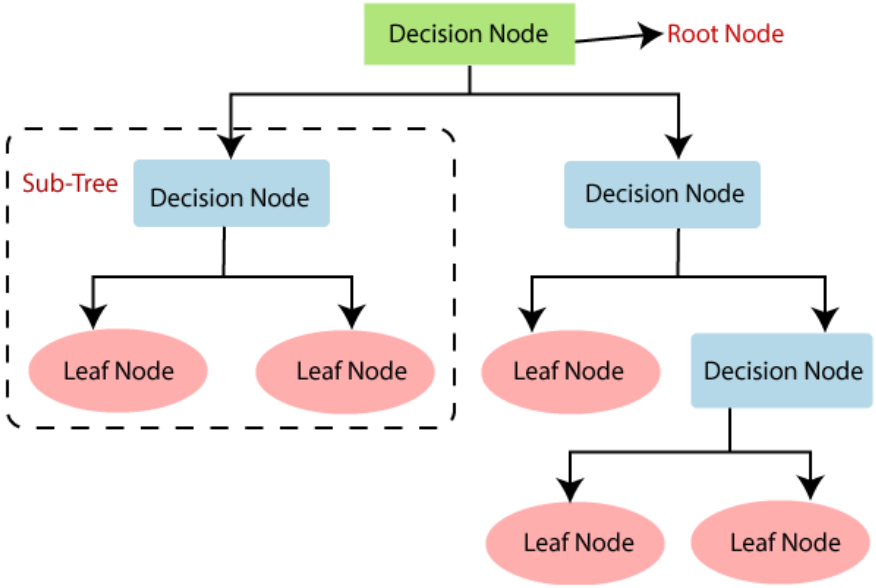
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

Decision Tree Algorithm

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed based on features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is called a decision tree because, like a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- To build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
- A decision tree simply asks a question and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

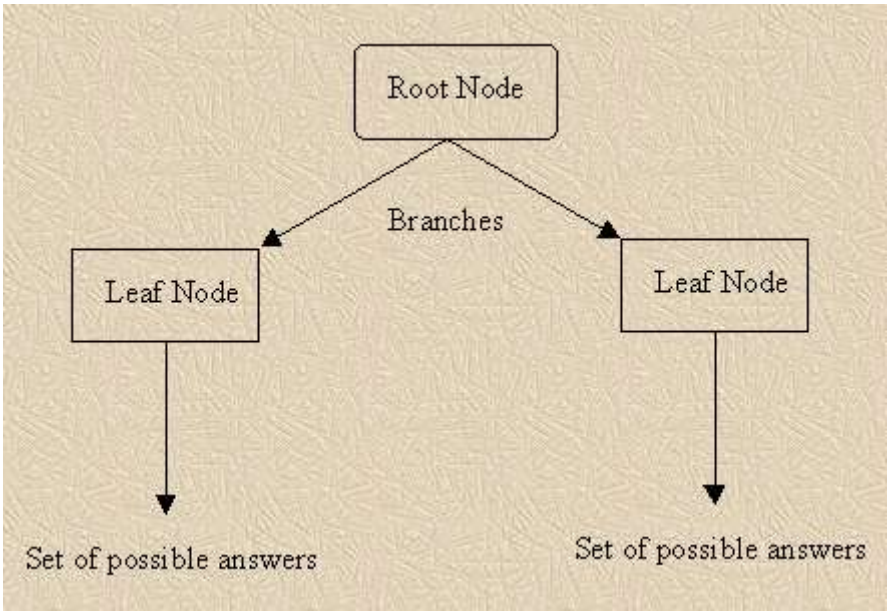


Features of Decision Tree Learning

- Method for approximating discrete-valued functions (including boolean).
- Learned functions are represented as decision trees (or if-then-else rules).
- Expressive hypotheses space, including disjunction.
- The decision tree is robust to noisy data.

Diagram

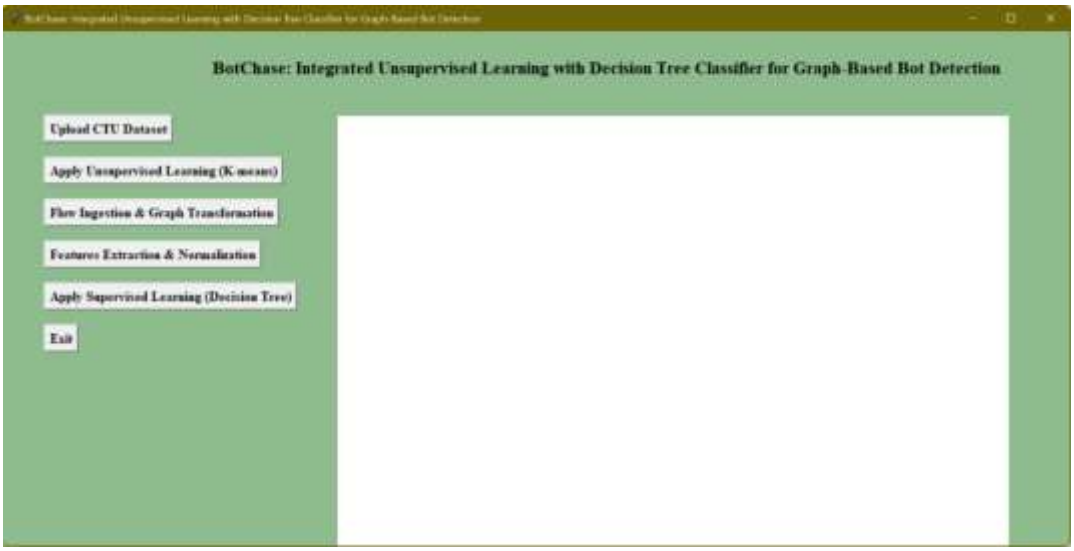
- Each non-leaf node is connected to a test that splits its set of possible answers into subsets corresponding to different test results.
- Each branch carries a particular test result's subset to another node.
- Each node is connected to a set of possible answers.



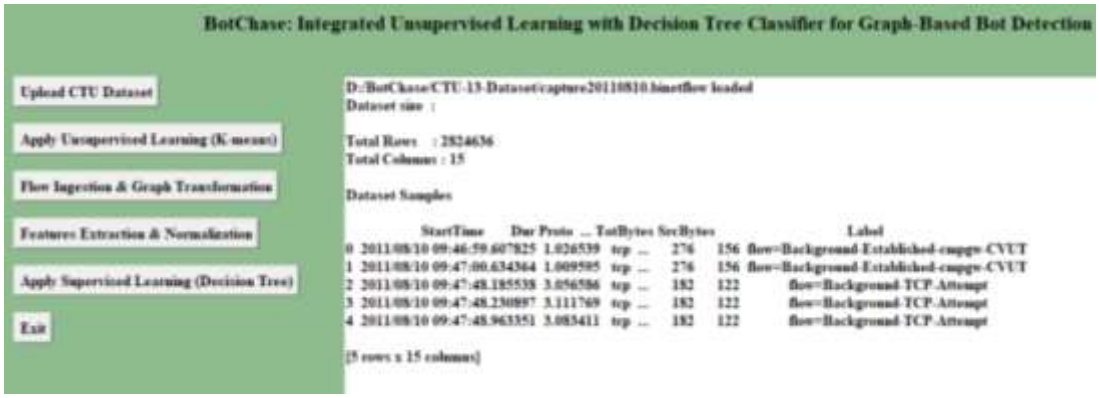
Advantages of proposed system

- Simple to understand and to interpret.
- Requires little data preparation.
- The cost of using the tree is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data.
- Able to handle multi-output problems.

5. RESULTS AND DISCUSSION



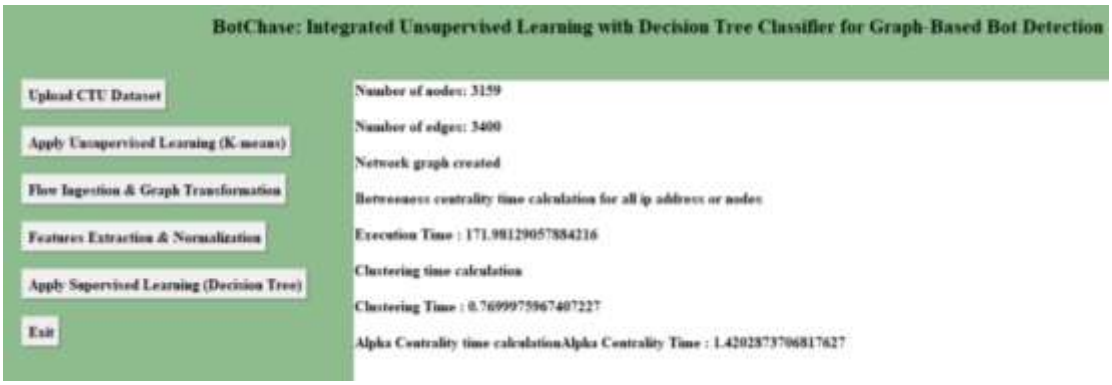
In above screen click on 'Upload CTU Dataset' button and upload dataset



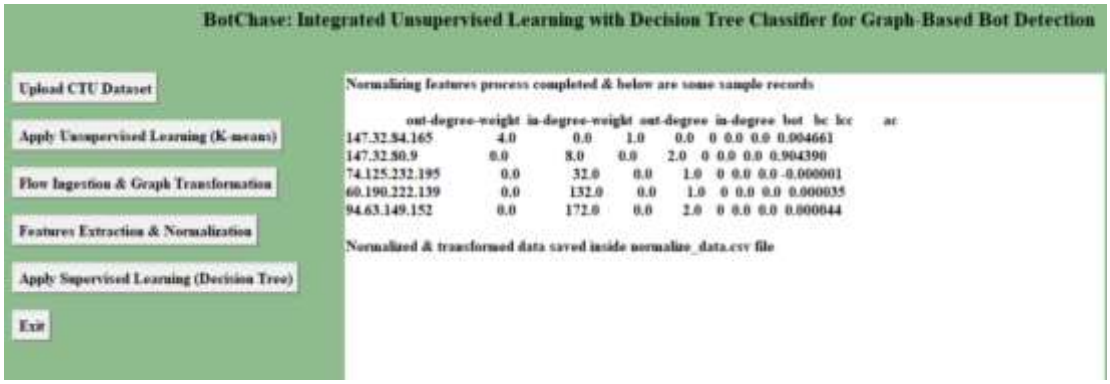
In above screen dataset contains total 2824636 records and each record contain 15 columns and below it I am displaying some dataset records. Now click on 'Apply Unsupervised Learning (K-means) to separate Bot & Benign Data' button to remove benign records

In above screen we can see dataset size before removing benign records and after removing benign records. By removing some benign records, we can reduce dataset size. Now click on 'Run Flow Ingestion & Graph Transformation' button to generate graph

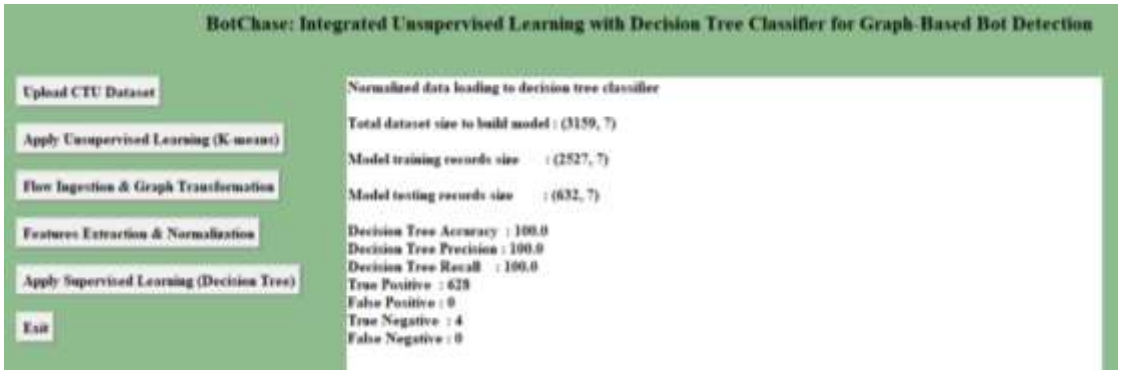
In above screen we can see progress bar which indicates graph-based features extraction and while applying this technique it will open 2 empty windows and you just close those 2 empty windows to get below screen



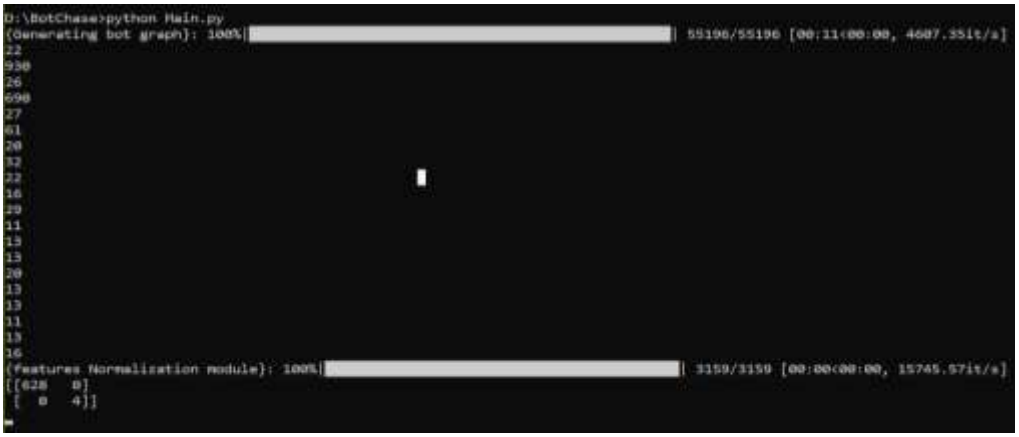
In above screen we can see total nodes and edges generated and time taken to calculate between_ness, alpha centrality, and clustering. Now we generate graphs and now click on 'Features Extraction & Normalization' button to extract features and to perform normalization on extracted features.



In above screen after normalization, I am displaying few records with out_degree, in degree and weight details. In above screen 'bc' refers to between_ness and 'lcc' refers to clustering and 'ac' refers to alpha centrality. All normalized records are saved inside 'normalize_data.csv' file and you can open and see that file from code folder. Now click on 'Run Decision Tree Algorithm' button to generate training model with decision tree classifier and to calculate metrics such as accuracy, precision etc.



In above screen after normalization, we got total records as 3159 with 7 columns (in_degree, out_degree, weight etc.) and application split total records into train size as 2527 and test size as 632. After building train model we apply test records and got accuracy as 100%. Below screen showing normalization progressing steps



6. CONCLUSION

In this paper, we proposed BotChase, a system that is capable of efficiently transforming network flows into an aggregated graph model. It leverages two ML phase to differentiate bots from benign

hosts. Using the results of the phase favour DT, showcasing high TPs and low FPs. BotChase is also able to detect bots that rely on different protocols, proves robust against unknown attacks and cross-network ML model training and inference. Flow-based features employed in BotChase under perform in comparison to graph-based features. BotChase also outperforms an end-to-end system that employs flow-based features, and performs well against the graph-based BotGM system. In an online setting, BotChase leverages HAT for incremental learning to process data on-the-fly. While the model takes longer to converge, it exhibits superior classification performance in its final state. To implement this project, we are using CTU-13 dataset and this dataset contains 13 observation and each observation contains PCAP files (This file contains all network information) and capture file (this file contains extracted information from PCAP files, and this file contains data such as source address, destination address, time, packet size etc.). To generate graph, we need to used capture file not PCAP file, so I downloaded some observation and saved inside 'CTU-13-dataset' folder.

REFERENCES

[1] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring payper-install: the commoditization of malware distribution," in USENIX Security, 2011, p. 13.

[2] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," Information Warfare & Security Research, vol. 1, no. 1, p. 80, 2011.

[3] R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," Journal of Internet Services and Applications, vol. 9, no. 1, pp. 1–99, 2018.

[4] G. Creech and J. Hu, "A Semantic Approach to Host-Based Intrusion Detection Systems Using Contiguous and Discontiguous System Call Patterns," IEEE Trans. on Computers, vol. 63, no. 4, pp. 807–819, 2014.

[5] B. Venkatesh, S. H. Choudhury, S. Nagaraja, and N. Balakrishnan, "BotSpot: fast graph-based identification of structured P2P bots," Journal of Computer Virology and Hacking Techniques, vol. 11, no. 4, pp. 247–261, 2015.

[6] Y. Jin et al., "A modular machine learning system for flow-level traffic classification in large networks," ACM TKDD, vol. 6, no. 1, p. 4, 2012.

[7] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection," in USENIX Security, 2008, pp. 139–154.

[8] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," Computers & Security, vol. 45, pp. 100–123, 2014.

[9] S. Lagraa et al., "BotGM: Unsupervised graph mining to detect botnets in traffic flows," in IEEE CSNet, 2017, pp. 1–8.

[10] A. Bifet and R. Gavalda, "Adaptive learning from evolving data streams," ` in Intl. Sym. on Intelligent Data Analysis, 2009, pp. 249–260.

[11] A. Abou Daya, M. Salahuddin, N. Limam, and R. Boutaba, "A GraphBased Machine Learning Approach for Bot Detection," in IFIP/IEEE IM, 2019.

[12] J. Goebel and T. Holz, "Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation," HotBots, vol. 7, p. 8, 2007.

[13] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using dnsbl counter-intelligence," SRUTI, vol. 6, pp. 49–54, 2006.

[14] M. Hagan, B. Kang, K. McLaughlin, and S. Sezer, "Peer based tracking using multi-tuple indexing for network traffic analysis and malware detection," in IEEE PST, 2018, pp. 1–5.

[15] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A taxonomy of botnet behavior, detection, and defense," IEEE COMST, vol. 16, no. 2, pp. 898–924, 2014.
[16] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion detection systems," Computer Networks, vol. 31, no. 8, pp. 805–822, 1999.
[17] J. R. Binkley and S. Singh, "An Algorithm for Anomaly-based Botnet Detection," SRUTI, vol. 6, p. 7, 2006.
[18] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-Scale Botnet Detection and Characterization," HotBots, vol. 7, p. 7, 2007.
[19] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in Botnet detection. Springer, 2008, pp. 1–24.
[20] R. Villamar´ın-Salomon and J. C. Brustoloni, "Identifying botnets using ´ anomaly detection techniques applied to dns traffic," in IEEE CCNC, 2008, pp. 476–481.
[21] W. Lu, M. Tavallaee, G. Rammidi, and A. A. Ghorbani, "BotCop: An online botnet traffic classifier," in IEEE Comm. Networks and Serv. Research, 2009, pp. 70–77.
[22] H. R. Zeidanloo, A. B. Manaf, P. Vahdani, F. Tabatabaei, and M. Zamani, "Botnet detection based on traffic monitoring," in ICNIT, 2010, pp. 97– 101.
[23] S. Saad et al., "Detecting P2P botnets through network behavior analysis and machine learning," in IEEE PST, 2011, pp. 174–180.
[24] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, "Detecting stealthy P2P botnets using statistical traffic fingerprints," in IEEE/IFIP DSN, 2011, pp. 121–132.
[25] W. Lu, G. Rammidi, and A. A. Ghorbani, "Clustering botnet communication traffic based on n-gram feature selection," Computer Comm., vol. 34, no. 3, pp. 502–514, 2011.
[26] H. Choi and H. Lee, "Identifying botnets by capturing group activities in DNS traffic," Computer Networks, vol. 56, no. 1, pp. 20–33, 2012.
[27] D. Zhao et al., "Botnet detection based on traffic behavior analysis and flow intervals," Computers & Security, vol. 39, pp. 2–16, 2013.
[28] M. Antonakakis et al., "Understanding the Mirai Botnet," in USENIX Security, 2017, pp. 1093–1110.
[29] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in RAID, 2007, pp. 276–295.
[30] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, "BotGrep: Finding P2P Bots with Structured Graph Analysis," in USENIX Security, vol. 10, 2010, pp. 95–110.
[31] J. Franc¸ois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting botnets using mapreduce," in IEEE Intl. Workshop on Info. Forensics and Security, 2011, pp. 1–6.
[32] H. Hang, X. Wei, M. Faloutsos, and T. Eliassi-Rad, "Entelecheia: Detecting p2p botnets in their waiting stage," in IEEE/IFIP Networking, 2013, pp. 1–9.
[33] K. Henderson et al., "Rolx: structural role extraction & mining in large graphs," in ACM KDD, 2012, pp. 1231–1239.
[34] Q. Ding, N. Katenka, P. Barford, E. Kolaczyk, and M. Crovella, "Intrusion as (anti) social communication: characterization and detection," in ACM KDD, 2012, pp. 886–894.
[35] J. Franc¸ois, S. Wang, and T. Engel, "BotTrack: tracking botnets using NetFlow and PageRank," in Intl. Conf. on Research in Networking, 2011, pp. 1–14.
[36] P. Jaikumar and A. C. Kak, "A graph-theoretic framework for isolating botnets in a network," Security and communication networks, vol. 8, no. 16, pp. 2605–2623, 2015.
[37] D. C. Le, A. N. Zincir-Heywood, and M. I. Heywood, "Data analytics on network traffic flows for botnet behaviour detection," in IEEE SSCI, 2016, pp. 1–7.

[38] D. Zhuang and J. M. Chang, "Peerhunter: Detecting peer-to-peer botnets through community behavior analysis," in IEEE DSC, 2017, pp. 493–500.

[39] S. Chowdhury et al., "Botnet detection using graph-based feature clustering," Journal of Big Data, vol. 4, no. 1, p. 14, 2017.

[40] D. Zhuang and J. M. Chang, "Enhanced peerhunter: Detecting peer-topeer botnets through network-flow level community behavior analysis," IEEE TIFS, vol. 14, no. 6, pp. 1485–1500, 2019.

[41] S. Khanchi, A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "On botnet detection with genetic programming under streaming data label budgets and class imbalance," Swarm and evolutionary computation, vol. 39, pp. 123–140, 2018.

[42] P. Mulinka and P. Casas, "Stream-based machine learning for network security and anomaly detection," in ACM Workshop on Big Data Analytics and ML for Data Comm. Networks, 2018, pp. 1–7.