Detecting Network Intrusion Anomalies through Egonet-based Data Mining with Apache Spark

Se Ho Kwak
Department of Computer Science
Pomona College
Claremont, CA, USA
skai2018@mymail.pomona.edu

Nathan Paik
Department of Computer Science
Pomona College
Claremont, CA, USA
npab2018@mymail.pomona.edu

Enyue Lu
Department of Computer Science
Salisbury University
Salisbury, MD, USA
ealu@salisbury.edu

***Abstract*— Network intrusions often contain dangerous breaches to network security systems and their data. We design an anomaly detection system to identify network intrusions. Our proposed detection method is inspired by the use of egonets in the Oddball Algorithm but differs by the extracted features and the anomaly classification procedure. The detection process follows the generalized design: create a k-nearest-neighbors graph from a network dataset; extract each node's egonet's edge weights, number of edges, and total eigenvector centrality sum; compare each node's egonet's features through pairwise comparisons; and define a median "truth" line from the comparison and label nodes as anomalous based on their distance from the line. We have achieved an anomaly detection accuracy score of up to 92.9% with the eigenvector centrality score vs. edge weight feature comparison. We parallelize our algorithm by implementing Resilient Distributed Datasets in Apache Spark.**

Keywords—Apache Spark, anomaly detection, egonet, network intrusion

I. Introduction

Network Intrusion Detection Systems (NIDS) are necessary to detect potentially harmful intrusions on network traffic. These systems help to detect unforeseen and unpredictable security breaches [1]. Unsupervised anomaly detection is used to detect anomalous network intrusions by using only the given dataset itself. There are few parallelization methods that have been introduced in the anomaly detection field. Our feature-based extraction method takes inspiration from the Oddball algorithm [2]. Furthermore, while our methodology does not entirely follow the Oddball algorithm, much of our idea and structure comes from the Oddball algorithm's use of egonets. In the Oddball algorithm, an egonet was constructed for each node in a social network dataset. Our method extracts the total number of edges and total weights of all edges in the egonet similar to the Oddball method, but we use the total eigenvector centrality measure of nodes in the egonet rather than the principal eigenvalue of the egonet [2]. Furthermore, the Oddball algorithm classifies anomalies by their distance from the expected "normal" line. However, we identify anomalies by their distance from a median "ground truth" line that is different from their expected truth line. We examine the effectiveness of a feature-based extraction method to detect network intrusion anomalies in the Network Security Laboratory-KDD (NSL-KDD) dataset while parallelizing the process through Apache Spark.

II. Background Information

A. Network Data

We use the NSL-KDD dataset, which is an enhanced version of the KDD-99 cup dataset [3]. Unlike the KDD-99 cup dataset, the NSL-KDD does not contain redundant or duplicate records. Each connection record in the dataset contains 41 attributes. We limit each connection record to contain 18 attributes that include the five basic attributes (duration, protocol type, service, src bytes, dst bytes) and thirteen other secondary attributes which describe "content features" such as the number of failed login attempts [1]. Furthermore, we limit the attack types to only Denial of Service types.

B. Ego-Networks

Our work revolves around each node's egonet. A node's egonet or ego-network, in our instance, is the collection of immediate neighbors from the node and all the edges that exist among those neighbors also [2]. This gives us a better representation of which nodes are most heavily connected and reveals patterns between node connections. Egonets are the basis of our anomaly detection method.

C. Graph Similarity Measure

Because the NSL-KDD dataset contains only single connection vectors with attributes as nodes, we artificially create edges between each connection node. We implement the Euclidean distance to get the vector "distance" between each node, and the radial basis function to interpolate and assign a similarity value for the multidimensional data. The following radial basis function represents the artificial edge weights between nodes:

$$w_{i,j} = \frac{1}{exp(\| \vec{v_i} - \vec{v_j} \|^2)}$$

III. Methodology

To preprocess the graph and illustrate figures we use a python script. However, we implement the rest of our work in Scala for Apache Spark. We make our process parallelizable with Spark by implementing all our algorithms through

Resilient Distributed Datasets (RDD), the fundamental data structure of Apache Spark. These RDDs allow for us to work with distributed memory through in-memory computations in large clusters in a fault tolerant manner [4].

A. Graph Creation

We utilize different sizes of the NSL-KDD network dataset (1,000 samples, 5,000 samples, 10,000 samples). With these datasets, each edge connecting two vertices represents the similarity measure between the two vertices. This similarity measure is based on two functions. First, we calculate the Euclidean distance between the two vertices, and then, we apply the radial basis function using the Euclidean distance as the vector distance to give the complete similarity measure. This creates a complete graph of our dataset with each node connected to all other nodes with edges based on similarity. Therefore, to reduce the size of the graph to work with, we implement a k-nearest-neighbors graph (kNN graph). Each sample takes a different k-value according to how many total nodes are in the graph.

B. Feature Extraction

From the kNN graph, we calculate each vertex's eigenvector centrality measure. Then we organize each node into their respective egonets, which consist of the node's immediate neighboring connections and all the edge connections amongst the neighboring nodes. We extract different features to analyze for each egonet, which include each egonet's total number of edges, total sum of edge weights, and total eigenvector centrality measure.

IV. Results

A. Graphic

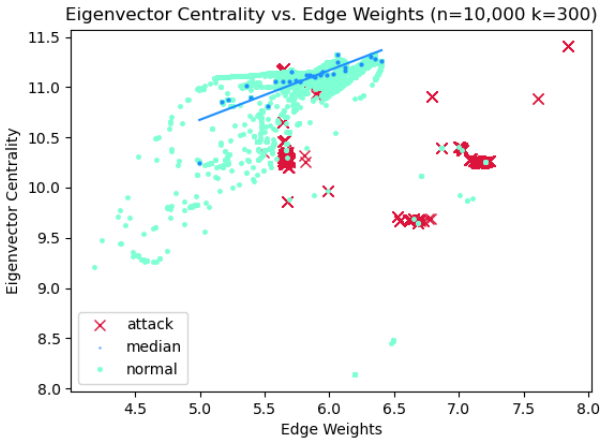

Fig. 1. This graph represents the pairwise comparison between eigenvector centrality and total edge weights for 10,000 nodes and a k value of 300.

B. Anomaly Score and Accuracy

Given each feature of the egonets, we pair features and analyze them with each other. We perform three paired analyses per sample: total edge weights vs. total number of edges, total eigenvector centrality measure vs. total weight, and total eigenvector centrality measure vs. total number of edges.

We partition all the vertices into equal parts based off one of the measures from our pairwise comparison and use the central point of each partition as our "median" values for our ground truth line. We take the vertices that are farthest from the median truth line and label those as anomalies. Our accuracy is determined by the following equation where TP is the number of true positives, FP is the number of false positives, TN is the number of true negatives, and FN is the number of false negatives. The following equation represents the accuracy:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The accuracies of each pairwise analysis with its respective node sample sizes are represented in Table I below.

TABLE I. Detection Accuracy

Comparison type	Eigenvector centrality vs. Total edge weight	Eigenvector centrality vs. Total number of edges	Total edge weight vs. Total number of edges
1,000 nodes K = 40	92.2%	90.6%	86.2%
5,000 nodes K = 150	92.24%	87.44%	84.88%
10,000 nodes K = 300	92.9%	85.76%	87.62%

V. Conclusion and Future Work

From the results in Table I, we achieve up to 92.9% accuracy with our anomaly detection algorithm. Our best approach for anomaly detection accuracy is to apply the eigenvector centrality vs. total edge weight comparison. The Fig. 1 graph shows how attack types are separated from normal types after running our pairwise comparison. A possible future work includes improving our algorithm through supervised machine learning by enhancing our anomaly classifying technique to be more sophisticated rather than just taking the farthest points from the median "ground truth" line. Another possible work would be finding an algorithm to optimize the k-value for our kNN graphs. We successfully parallelized the detection process on a local cluster and the cluster in the high-performance computing laboratory at Salisbury University. Our runtimes for our detection algorithm on the 5,000-node sample and 10,000-node sample are 2.2 minutes and 20 minutes respectively using a 6-node cluster. We are in the process of optimizing the runtime and improving the efficiency on our GPU enabled clusters.

Acknowledgment

This work is funded by NSF CCF-1757017 under Research Experiences for Undergraduates Program.

REFERENCES

[1] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009, July). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1-6). IEEE.

[2] Akoglu, L., Tong, H., & Koutra, D. (2015). Graph based anomaly detection and description: a survey. Data mining and knowledge discovery, 29(3), 626-688.

[3] "NSL-KDD Dataset." *University of New Brunswick*, UNB, www.unb.ca/cic/datasets/nsl.html.

[4] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., ... & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)* (pp. 15-28).