# Online Detection of User's Anomalous Activities using Logs

By

Kusumanjali Somisetty, Diploma, Conestoga College, 2014

A Major Research Project

Presented to Ryerson University

in partial fulfillment of the requirements for the degree of

Master of Science

in the Program of

Data Science and Analytics

Toronto, Ontario, Canada, 2020

**AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A MAJOR RESEARCH PROJECT (MRP)**

I hereby declare that I am the sole author of this Major Research Paper. This is a true copy of the MRP, including any required final revisions.

I authorize Ryerson University to lend this MRP to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this MRP by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my MRP may be made electronically available to the public.

Kusumanjali Somisetty

# Online Detection of User's Anomalous Activities using Logs

Kusumanjali Somisetty

Master of Science 2020

Data Science and Analytics

Ryerson University

## ABSTRACT

Securing the Organization's Confidential Information is always a concern for any Organization. This Paper implements a Machine Learning approach to monitor the Users activities and determine the anomalous Data. The term anomalous data refers to data that are different from what are expected or normally occur. Detecting anomalies is important in most industries. For example, in network security, anomalous packets or requests can be flagged as errors or potential attacks. In customer security, anomalous online behavior can be used to identify fraud. In addition, in manufacturing and the Internet of Things, anomaly detection is useful for identifying machine failures.

**Key words:**

Anomalous Detection, Machine Learning, Python, Spark, Random Forest, Decision Tree, ROC Curves,

## ACKNOWLEDGEMENTS

I would like to express my special thanks to Professor Dr. Pawel Pralat and Dr. Przemyslaw Szufel, who gave me the wonderful opportunity to do this project, I am sincerely thankful for their support, guidance and assistance in completing this project. Professor Pawel Pralat was my supervisor for this MRP Project.

I am very thankful to Professor Przemyslaw Szufel who always teach new techniques to use and direct my research to complete in time. Professor Szufel also assisted me with open source dataset to get from past research papers.

I am very grateful to my supervisor Pawel Pralat and Przemyslaw Szufel for their support throughout the term to guide me in this project and provide valuable feedback.

Thank you Professor Pawel Pralat and Przemyslaw Szufel.

# TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# 1. INTRODUCTION

This document explains more about domain, research topic, dataset and research questions which I have chosen for my major research project (MRP). I have initiated by a brief background about research topic, followed by a problem definition including research questions, then suitable literature reviews and dataset analysis. Next methodology and experimental setup using dataset to get the results and finally with a conclusion about project including recommendation of future work.

Information system has become an icon of this era. From enterprises even to restaurants, they use information system to support their business. The internet we use every day is also a huge collection of information systems. As information system starts to manage the more important aspects of human lives such as banking and social needs, the security of information system also becomes increasingly important. Many information systems record their users' activities on log files. One way to ensure an information system's security is by analyzing its log files for suspicious or anomalous user activity. Yet the textual nature of log files makes it difficult for administrators to comprehend the meaning of its records and its big size makes log analysis a tedious and time-consuming task. Fortunately, machine learning based approach to detect user activities could serve as a solution for this problem.

Network security is a foremost issue these days as the network usage is growing in multi-dimensions due to increased use of handheld devices. Intrusion Detection Systems can help detect malign intentions of network users without compromising the security of the host and the network.

Machine learning is the field of study that aims to provide computers with the ability of gaining knowledge from the external world, without any human interaction. The knowledge extracted by a certain machine learning technique may be different from one set of inputs, from the external world, to another. Moreover, knowledge extracted from a single set of inputs may also be different from one machine learning technique to another, according to the different approaches used to extract such knowledge. One of the main machine learning fields is data mining, where the inputs from the external world are datasets, collected from the domain that knowledge extraction is required for. Data mining techniques, as well as other machine learning techniques, are categorized into two main categories, which are unsupervised and supervised techniques. Unsupervised data mining techniques require no addition to the input dataset, as the aim of these techniques is to extract relations among the objects in the dataset, while supervised data mining techniques require some extra information to be added to the dataset by an expert. Supervised data mining techniques extract the relations among the objects in the dataset, and the knowledge added by the expert. This knowledge, extracted from the sample dataset, which is known as the training dataset, can be used in runtime to apply the extracted knowledge on new objects to assist the operation of the system interacting with the domain

## A. Background

For my Major Research Project, I have interested to work with network security. There are various kinds of Intrusion detections in network-based security.

With the tremendous growth of network-based services and sensitive information on networks, network security is getting more importance than ever. Although a wide range of security technologies such as information encryption, access control, and intrusion prevention can protect network-based systems, there are still many undetected intrusions. For example, firewalls cannot prevent internal attacks. Thus, Intrusion Detection Systems (IDSs) play a vital role in network security. Network Intrusion Detection Systems (NIDSs) detect attacks by observing various network activities, while Host-based Intrusion Detection Systems (HIDSs) detect. Intrusions in an individual host. An IDS usually does not affect the normal network operations of the targets. There are two major intrusion detection techniques: misuse detection and anomaly detection. Misuse detection discovers attacks based on the patterns extracted from known intrusions. Anomaly detection identifies attacks based on the significant deviations from the established profiles of normal activities. Misuse detection has low false positive rate, but cannot detect novel attacks. Anomaly detection can detect unknown attacks, but has high false positive rate.

Currently, many NIDSs such as Snort are rule-based systems, which employ misuse detection techniques and have limited extensibility for novel attacks. Their performances highly rely on the rules identified by the security experts. However, the amount of network traffic is huge, and it is very difficult to specify some intrusions using the rules. Therefore, the process of encoding rules is expensive and slow.

To overcome the limitations of the rule-based systems, a number of IDSs employ data mining techniques. Data mining is the analysis of (often-large) observational data sets to find patterns or models that are both understandable and useful to the data owner. Data mining can efficiently extract patterns of intrusions for misuse detection, establish profiles of normal network activities for anomaly detection, and build classifiers to detect attacks, especially for the vast amount of audit data. Data mining-based systems are more flexible and deployable. The security experts only need to label the audit data to indicate intrusions instead of hand-coding rules for intrusions. Over the past several years, a growing number of research projects have applied data mining to intrusion detection with different algorithms. For instance, MADAM ID and ADAM employ association rules algorithm. We propose an approach to use random forests algorithm in intrusion detection. Random forests is an ensemble classification and regression approach, which is unsurpassable in accuracy among current data mining algorithms Random forests algorithm has been used extensively in different applications. For instance, it has been applied to prediction, probability estimation, and pattern analysis in

multimedia information retrieval and bioinformatics. Unfortunately, to the best of our knowledge, random forests algorithm has not been applied in automatic intrusion detection.

Accuracy is critical to develop effective NIDSs, since high false positive rate or low detection rate will make NIDSs unusable. To improve detection performance, we also propose methods to address the issues of imbalanced intrusions and feature selection in mining process as discussed below. One of the challenges in intrusion detection systems is feature selection. Many algorithms are sensitive to the number of features. Hence, feature selection is essential for improving detection rate. Moreover, the raw data of network traffic is usually audited in tcpdump format, and the tcpdump format is not suitable for detection. IDSs must construct features from the raw data. The process of feature construction from tcpdump format data involves a lot of computation. Thus, feature selection can help reducing the computational cost for feature construction. However, in many current data mining-based IDSs, feature selection is based on domain knowledge or intuition. We use the feature selection algorithm of random forests, because the algorithm can give estimates of what features are important in the classification

### B. Problem Definition

Logs are imperative in the development and maintenance process of many software systems. They record detailed runtime information during system operation that allows developers and support engineers to monitor their systems and track abnormal behaviors and our proposal is to implement a number of machine-learning based log analysis techniques for automated anomaly detection. Python and its open libraries will be used for constructing this Model.

### C. Research Questions

a) Is it possible to detect user activities that have symptoms of fraudulent or unintended behavior in real time based on the user's activity pattern and available metadata describing the transaction?

b) Is it Possible to construct a system for online detection of malicious, spurious or erroneous activity of users? The targeted activities to be detected include data theft, data exfiltration, and external correspondence. Those events can occur because of user mistakes or intentional actions.

c) The system will be designed work in real time analyzing log streams of users - for each logged event, the classifier will output a binary decision along with a confidence level.

## 2. LITERATURE REVIEW

I started of reading some articles which helps to my project research work, in this section I have mentioned an overview of articles I have gone through and used a reference of my project.

***Paper I***

*Information System Log Visualization to Monitor Anomalous User Activity Based on Time*

In this paper, they tried to find out the key characteristics of data visualization methods, which can help, detect anomalous user activities recorded in log files. They depicted user activities through dot plot and heat map visualization methods while leaving the decision whether there are anomalies or not to the administrator's discretion entirely. The data visualization can be a lot more helpful if they add machine-learning feature to the dot plot and heat map visualization methods. That way, the administrators do not have to find out for themselves whether there are anomalies in a user activity but are instead notified when anomalous pattern is detected. The most appropriate data visualization method combined with machine learning could ultimately lead to the most helpful tool on detecting anomalous user activities recorded on log files. Therefore, adding the machine-learning feature to the data visualization is the most suitable step to extend this study further.

***Paper II***

*Anomalous User Activity Detection in Enterprise Multi-Source Logs*

They have proposed a practical PCA-based user activity behavior anomaly detection method. They showed the method works for practical purposes. While that method is for event based anomaly detection, it can be extended to other access based anomaly detection. Example types of access include machines, documents on the network, or applications on the cloud.

***Paper III***

Loglizer

Loglizer is a machine learning-based log analysis toolkit for automated anomaly detection.

Logs are imperative in the development and maintenance process of many software systems. They record detailed runtime information during system operation that allows developers and support engineers to monitor their systems and track abnormal behaviors and errors. Loglizer provides a toolkit that implements a number of machine-learning based log analysis techniques for automated anomaly detection.

However, traditional anomaly detection that relies heavily on manual log inspection be- comes impossible due to the sharp increase of log size. To reduce manual effort, automated log analysis and anomaly detection methods have been widely studied in recent years. However, developers are still not aware of the state-of-the- art anomaly detection methods, and often have to re-design a new anomaly detection method by themselves, due to the lack of a comprehensive review and comparison among current methods. In this paper, we fill this gap by providing a detailed review and evaluation of six state-of-the-art anomaly detection methods. We also compare their accuracy and efficiency on two representative production log datasets. Furthermore, we release an open-source toolkit of these anomaly detection methods for easy reuse and further study.

***Paper IV***

*Cyber Anomaly Detection Using Graph-node Role-dynamics*

Intrusion detection systems (IDSs) generate valuable knowledge about network security, but an abundance of false alarms and a lack of methods to capture the interdependence among alerts hampers their utility for network defense.

Graph-based anomaly detection is a promising new approach for detecting cyber-attacks from cyber artifacts. In both test scenarios, our method identified anomalies corresponding to the start of the attack. They fused data from several intrusion detection systems into artifact graphs using a novel graph construction based on fields in the triggered alerts, not the network topology. Analyzing the role dynamics in these graphs for simulated Hurricane-Panda- like and Energetic-Bear-like APT datasets, we identified a handful of anomalies, including anomalies that coincide with the start of each attack. Their approach successfully identified simulated attacks through anomalies in IDS alert patterns, and reduced the number of false-positive alerts from thousands to just three false-positive anomalies (in one simulation) and two (in the other simulation). Our results illustrate how graph-node role-dynamics analyses can identify anomalies in IDS alerts, however causal analysis will re- quire further investigation by human analysts.

## 3. DESCRIPTIVE ANALYTICS | EXPLORATORY DATA ANALYSIS

**DATASET:**

Dataset that I have selected is http://kdd.ics.uci.edu/databases/kddcup99/

### D. KDD Dataset

The NSL-KDD data set with 42 attributes is used in this empirical study. This data set is an improvement over KDD'99 data set from which duplicate instances were removed to get rid of biased classification results. This data set has number of versions available, out of which 20% of the training data is used which is identified as KDDTrain+_20Percent with a total number of 25192 instances. The test data set is identified by the name KDDTest+ and has a total of 22544 instances. Different configurations of this data set are available with variation in number of instances but the number of attributes in each case is 42. The attribute labeled 42 in the data set is the 'class' attribute which indicates whether a given instance is a normal connection instance or an attack. Out of these 42 attributes, 41 attributes can be classified into four different classes as discussed below:

- Basic (B) Features are the attributes of individual TCP connections
- Content (C) features are the attributes within a connection suggested by the domain knowledge
- Traffic (T) features are the attributes computed using a two-second time window
- Host (H) features are the attributes designed to assess attacks which last for more than two seconds

### E. Data Acquisition

The dataset I chose are open sourced dataset and matched with MRP requirements, have been collected and provided by following paper "Experience Report: System Log Analysis for Anomaly Detection"

### F. Data Source and Data Files

The datasets have been collected from following http://kdd.ics.uci.edu/databases/kddcup99/, I have used two types of dataset which I have added on GitHub. Link to the GitHub repository were added in Appendix A.

## G. Summary Statistics by Label

The interesting part of summary statistics, in our case, comes from being able to obtain them by the type of network attack or 'label' in our dataset. By doing so we will be able to better, characterize our dataset dependent variable in terms of the independent variables range of values.

If we want to do such a thing we could filter our RDD containing labels as keys and vectors as values. For that, we just need to adapt our parse interaction function to return a tuple with both elements.

| | Mean | Std Dev | Min | Max | Count |
|---|---|---|---|---|---|
| back. | 54156.355878 | 3.159360e+03 | 13140.0 | 54540.0 | 2203.0 |
| buffer_overflow. | 1400.433333 | 1.337133e+03 | 0.0 | 6274.0 | 30.0 |
| ftp_write. | 220.750000 | 2.677476e+02 | 0.0 | 676.0 | 8.0 |
| guess_passwd. | 125.339623 | 3.037860e+00 | 104.0 | 126.0 | 53.0 |
| imap. | 347.583333 | 6.299260e+02 | 0.0 | 1492.0 | 12.0 |
| ipsweep. | 10.083400 | 5.231658e+00 | 0.0 | 18.0 | 1247.0 |
| land. | 0.000000 | 0.000000e+00 | 0.0 | 0.0 | 21.0 |
| loadmodule. | 151.888889 | 1.277453e+02 | 0.0 | 302.0 | 9.0 |
| multihop. | 435.142857 | 5.409604e+02 | 0.0 | 1412.0 | 7.0 |
| neptune. | 0.000000 | 0.000000e+00 | 0.0 | 0.0 | 107201.0 |
| nmap. | 24.116883 | 5.941987e+01 | 0.0 | 207.0 | 231.0 |
| normal. | 1157.047524 | 3.422612e+04 | 0.0 | 2194619.0 | 97278.0 |
| perl. | 265.666667 | 4.932883e+00 | 260.0 | 269.0 | 3.0 |
| phf. | 51.000000 | 0.000000e+00 | 51.0 | 51.0 | 4.0 |
| pod. | 1462.651515 | 1.250980e+02 | 564.0 | 1480.0 | 264.0 |
| portsweep. | 666707.436538 | 2.150067e+07 | 0.0 | 693375640.0 | 1040.0 |
| rootkit. | 294.700000 | 5.385782e+02 | 0.0 | 1727.0 | 10.0 |
| satan. | 1.337319 | 4.294620e+01 | 0.0 | 1710.0 | 1589.0 |

Figure 1: Summary statistics by network labels

## H. Correlations

Spark's MLlib supports Pearson's and Spearman's to calculate pairwise correlation methods among many series. Both of them are provided by the core method in the Statistics package.

We have two options as input. Either two RDD [Double] or an RDD [Vector]. In the first case the output will be a Double value, while in the second a whole correlation Matrix. Due to the nature of our data, we will obtain the second

| | duration | src_bytes | dst_bytes | land |
|---|---|---|---|---|
| duration | 1.000000 | 1.419628e-02 | 0.299189 | -0.001068 |
| src_bytes | 0.014196 | 1.000000e+00 | -0.167931 | -0.009404 |
| dst_bytes | 0.299189 | -1.679306e-01 | 1.000000 | -0.003040 |
| land | -0.001068 | -9.403731e-03 | -0.003040 | 1.000000 |
| wrong_fragment | -0.008025 | -1.935835e-02 | -0.022659 | -0.000333 |
| urgent | 0.017883 | 9.380528e-05 | 0.007234 | -0.000065 |
| hot | 0.108639 | 1.139201e-01 | 0.193156 | -0.000539 |
| num_failed_logins | 0.014363 | -8.395950e-03 | 0.021952 | -0.000076 |
| logged_in | 0.159564 | -8.970213e-02 | 0.882185 | -0.002784 |
| num_compromised | 0.010687 | 1.185618e-01 | 0.169772 | -0.000447 |
| root_shell | 0.040425 | 3.066817e-03 | 0.026054 | -0.000093 |
| su_attempted | 0.026015 | 2.282074e-03 | 0.012192 | -0.000049 |
| num_root | 0.013401 | -2.049860e-03 | -0.003884 | -0.000230 |
| num_file_creations | 0.061099 | 2.771005e-02 | 0.034154 | -0.000149 |
| num_shells | 0.008632 | 1.440256e-02 | -0.000054 | -0.000076 |
| num_access_files | 0.019407 | -1.496982e-03 | 0.065776 | -0.000211 |
| num_outbound_cmds | -0.000019 | -4.227269e-07 | -0.000026 | -0.002873 |
| is_hot_login | -0.000011 | 2.999489e-06 | 0.000035 | 0.002095 |
| is_guest_login | 0.205606 | 2.751087e-02 | 0.085947 | -0.000250 |

Figure 2: Correlation with a double value

We got Boolean data frame where true means that a pair of variables is highly correlated with the names of the variables so we can use them to slice the data frame

| | src_bytes | dst_bytes | hot | logged_in | \ |
|---|---|---|---|---|---|
| src_bytes | False | False | False | False | |
| dst_bytes | False | False | False | True | |
| hot | False | False | False | False | |
| logged_in | False | True | False | False | |
| num_compromised | False | False | True | False | |
| num_outbound_cmds | False | False | False | False | |
| is_hot_login | False | False | False | False | |
| count | False | False | False | False | |
| srv_count | False | False | False | False | |
| serror_rate | False | False | False | False | |
| srv_serror_rate | False | False | False | False | |
| rerror_rate | False | False | False | False | |
| srv_rerror_rate | False | False | False | False | |
| same_srv_rate | False | False | False | False | |
| diff_srv_rate | False | False | False | False | |
| dst_host_count | False | False | False | False | |
| dst_host_srv_count | False | False | False | False | |
| dst_host_same_srv_rate | False | False | False | False | |
| dst_host_diff_srv_rate | False | False | False | False | |

Figure 3: Correlation with Boolean data frame

The previous data frame showed us which variables are highly correlated. We have kept just those variables with at least one strong correlation. We can use as we please, but a good way could be to do some model selection. That is, if we have a group of variables that are highly correlated, we can keep just one of them to represent the group under the assumption that they convey similar information as predictors. Reducing the number of variables will not improve our model accuracy, but it will make it easier to understand and more efficient to compute.

For example, from the description of the KDD Cup 99 task we know that the variable dst_host_same_src_port_rate references the percentage of the last 100 connections to the same port, for the same destination host. In our correlation matrix (and auxiliary data frames), we find that this one is highly and positively correlated to src_bytes and srv_count. The former is the number of bytes sent form source to destination. The latter is the number of connections to the same service as the current connection in the past 2 seconds. We might decide not to include dst_host_same_src_port_rate in our model if we include the other two, as a way to reduce the number of variables and later one better interpret our models.

## 4. METHODOLOGY AND EXPERIMENTS

Apache Spark is a cluster-computing platform designed to be fast. Spark extends the popular Map Reduce model to efficiently support more types of computations, including interactive queries and stream processing. Speed is important in processing large datasets, as it means the difference between exploring data interactively and waiting minutes or hours. One of the main features Spark offers for speed is the ability to run computations in memory, but the system is also more efficient than Map Reduce for complex applications running on disk. Spark is also designed to cover a wide range of workloads that previously required separate distributed systems, including batch applications, iterative algorithms, interactive queries, and streaming. By supporting these workloads in the same engine, Spark makes it easy and inexpensive to combine different processing types, which is often necessary in production data

The latest Big Data Processing Tool: Apache Spark and its MLlib library are used for experiment and result analysis. In the experiments, four well-known machine-learning algorithms are used, namely Naïve Bayes, Decision Tree, and Random Forest are used for performance evaluation. The classification measures are four elements: TP, TN, FP and FN. First, TP (true positive) is the number of correctly classified attacks. Second, TN (true negative) is the number of correctly classified normal records. Third, FP (false positive) is the number of misclassified attacks. Finally, FN (false negative) denotes the number of misclassified normal records.

## I. Algorithm Comparison and Selection

| Methods | Accuracy | Sensitivity | Specificity | Training Time | Prediction Time |
|---|---|---|---|---|---|
| **Decision Tree** | **95.82** | 92.52 | 97.10 | 4.80 | 0.13 |
| Random Forest | 97.49 | 93.53 | 97.75 | 5.69 | 0.08 |
| **Logistic Regression** | 74.19 | 92.16 | 67.82 | 2.25 | 0.18 |

Table 1: Comparison between algorithms and selection

Random Forest took the least time approximately 0.08 seconds and thus is the fastest detection scheme of all. Whereas Logistic Regression took highest, time to predict with the maximum time of 0.20 seconds and thus become the slowest scheme of all the detection methods. Decision Tree ranks the second fastest scheme with approximately 0.13 seconds to predict.

## J. Research Methodology

The steps followed as part of the research methodology are as follows:

- KDD data set is selected
- Python Sklearn is used to implement the algorithm
- Random Tree is used as a binary classifier for simulation on algorithm classifies the instances as attack or normal

## K. Used Classifier

Machine learning is an artificial intelligence technique, which consists of a number of algorithms, based on which a model can be developed that learns from the input data known as the training data set and helps predict on testing data set. Though there are many classifiers available, but tree based algorithms, produce better accuracy in results without requiring much tuning of parameters. In this paper, Random Tree algorithm, a tree-based classifier is selected for simulation from experience. Random Tree is a set (ensemble) of tree predictors that is called forest. This classifier takes the input feature vector, classifies it with every tree in the forest, and outputs the class label that receives the majority of "votes".

## L. Metrics

Intrusion detection metrics helps evaluate the performance of an intrusion detection system21. Some of the commonly used evaluation metrics used with respect to intrusion detection are False Alarm Rate (FAR), Detection Rate (DR), Accuracy, Precision, Specificity, F-score. All these evaluation metrics are basically derived from the four basic attributes of the confusion matrix depicting the actual and predicted classes. These elements of the confusion matrix are:

- True Negative (TN): Number of instances correctly predicted as non-attacks.
- False Negative (FN): Number of instances wrongly predicted as non-attacks.
- False Positive (FP): Number of instances wrongly predicted as attacks.
- True Positive (TP): Number of instances correctly predicted as attacks.

## M.Spark

The general purpose of Apache Spark is an open source distributed cluster-computing framework, which contains data processing engine in memory, machine learning techniques, batch processing and using large volumes of data it process graphs as well.

I chose spark to detect network intrusions, Spark's MLlib provides column summary statistics for RDD [Vector] through the function colStats available in Statistics. The method returns an instance of MultivariateStatisticalSummary, which contains the column-wise max, min, mean, variance, and number of nonzero, as well as the total count.

Used Spark's machine learning library MLlib to build a Logistic Regression classifier for network attack detection.

## N. LOGISTIC REGRESSION

In our case, we are interested in detecting network attacks in general. We do not need to detect which type of attack we are dealing with. Therefore we will tag each network interaction as non-attack (i.e, 'normal' tag) or attack (i.e, anything else but 'normal').

Using Logistic Regressing trained data, classifier and evaluated model on new data.

```
# Training Classifier

from pyspark.mllib.classification import LogisticRegressionWithLBFGS
from time import time

# Build the model
t0 = time()
logit_model = LogisticRegressionWithLBFGS.train(training_data)
tt = time() - t0

print("Classifier trained in {} seconds".format(round(tt,3)))

Classifier trained in 132.073 seconds
```

```
# EVALUATING THE MODEL ON NEW DATA

labels_and_preds = test_data.map(lambda p: (p.label, logit_model.predict(p.features)))
t0 = time()
test_accuracy = labels_and_preds.filter(lambda v_p: v_p[0]== v_p[1]).count() / float(test_data.count())
tt = time() - t0
```

```
print("Prediction made in {} seconds. Test accuracy is {}".format(round(tt,3), round(test_accuracy,4)))

Prediction made in 22.703 seconds. Test accuracy is 0.9048
```

Figure 4: Logistic Regression training classifier, prediction & test accuracy

Let us proceed with the evaluation of our reduced model. First, we need to provide training and testing datasets containing just the selected variables. For that, we will define a new function to parse the raw data that keeps just what we need.

### *O.*Hypothesis Testing

Hypothesis testing is a powerful tool in statistical inference and learning to determine whether a result is statistically significant. MLlib supports Pearson's chi-squared ($\chi 2$) tests for goodness of fit and independence. The goodness of fit test requires an input type of Vector, whereas the independence test requires a Matrix as input. Moreover, MLlib also supports the input type RDD [Labeled Point] to enable feature selection via chi-squared independence tests. Again, these methods are part of the Statistics package

```
# Build the model
t0 = time()
logit_model_2 = LogisticRegressionWithLBFGS.train(corr_reduced_training_data)
tt = time() - t0

print("Classifier trained in {} seconds".format(round(tt,3)))

Classifier trained in 121.973 seconds
```

```
labels_and_preds = corr_reduced_test_data.map(lambda p: (p.label, logit_model_2.predict(p.features)))
t0 = time()
test_accuracy = labels_and_preds.filter(lambda v_p: v_p[0] == v_p[1]).count() / float(corr_reduced_test_data.count())
tt = time() - t0

print("Prediction made in {} seconds. Test accuracy is {}".format(round(tt,3), round(test_accuracy,4)))

Prediction made in 22.937 seconds. Test accuracy is 0.8158
```

Figure 5: Logistic Regression classifier training using hypothesis testing

## P. DECISION TREE

Using Decision tree we detected network attacks, for that we have trained a classification tree using MLlib requires some parameters:

Training data, Num classes, Impurity metric, Tree maximum depth and tree maximum number of bins

Categorical features info: a map from column to categorical variables arity. This is optional, although it should increase model accuracy. However, it requires that we know the levels in our categorical variables in advance. Second, we need to parse our data to convert labels to integer values within the arity range.

```
# Build the model
t0 = time()
tree_model = DecisionTree.trainClassifier(training_data, numClasses=2,
                                          categoricalFeaturesInfo={1: len(protocols), 2: len(services), 3: len(flags)},
                                          impurity='gini', maxDepth=4, maxBins=100)
tt = time() - t0

print("Classifier trained in {} seconds".format(round(tt,3)))

Classifier trained in 18.878 seconds
```

```
predictions = tree_model.predict(test_data.map(lambda p: p.features))
labels_and_preds = test_data.map(lambda p: p.label).zip(predictions)
```

```
t0 = time()
test_accuracy = labels_and_preds.filter(lambda v_p: v_p[0] == v_p[1]).count() / float(test_data.count())

tt = time() - t0

print("Prediction made in {} seconds. Test accuracy is {}".format(round(tt,3), round(test_accuracy,4)))

Prediction made in 21.183 seconds. Test accuracy is 0.9196
```

Figure 6: Decision Tree trained classifier, prediction & test accuracy

Understanding our tree splits is a great exercise in order to explain our classification labels in terms of predictors and the values they take. Using the toDebugString method in our three models, we can obtain a lot of information regarding splits, nodes, etc.

```
Learned classification tree model:
DecisionTreeModel classifier of depth 4 with 25 nodes
  If (feature 22 <= 59.0)
   If (feature 25 <= 0.5)
    If (feature 9 <= 0.5)
     If (feature 36 <= 0.46499999999999997)
      Predict: 0.0
     Else (feature 36 > 0.46499999999999997)
      Predict: 1.0
    Else (feature 9 > 0.5)
     If (feature 4 <= 1099.0)
      Predict: 0.0
     Else (feature 4 > 1099.0)
      Predict: 1.0
   Else (feature 25 > 0.5)
    If (feature 38 <= 0.105)
     If (feature 22 <= 6.5)
      Predict: 0.0
     Else (feature 22 > 6.5)
      Predict: 1.0
    Else (feature 38 > 0.105)
```

```
print("Service 0 is {}".format(services[0]))
print("Service 52 is {}".format(services[52]))

Service 0 is finger
Service 52 is gopher
```

Figure 7: Decision tree classifier of depth 4 using 25 nodes

Confusion Matrix:

```
Area under PR = 0.9033327902197944
Area under ROC = 0.8541498042376217
Summary Stats
Precision = 0.9195766311179986
Recall = 0.9195766311179986
F1 Score = 0.9195766311179986
Confusion Matrix  DenseMatrix([[  59703.,   24124.],
             [    890.,  226312.]])
```

Figure 8: Decision tree confusion matrix

Using Prediction plotted ROC Curve

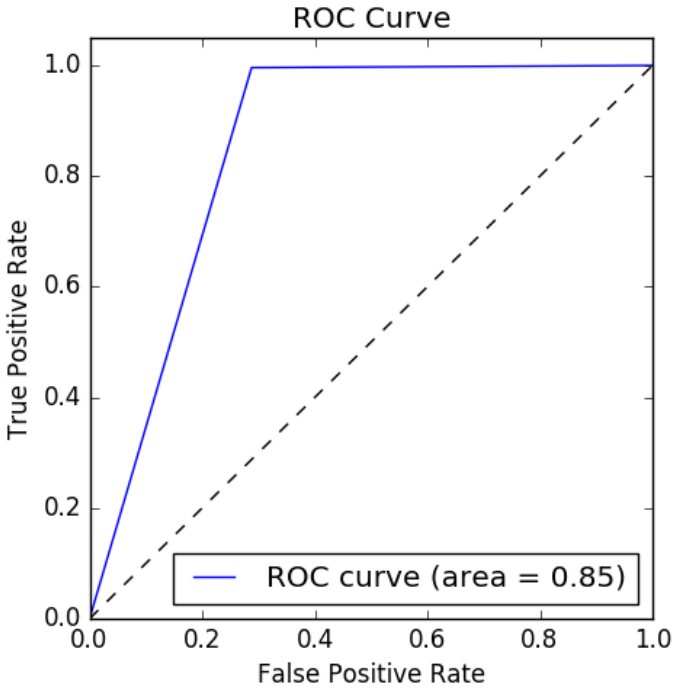


Figure 9: ROC Curve with Prediction

## Q. RANDOM FOREST

After carefully Analyzing, I have decided to implement the Random Forest algorithm to detect the anomalies in the Data.

Random forests are a powerful prediction method that uses collections of trees with a random parameter holdout to build models that often outperform individual decision trees. However, the random forest is normally a supervised approach, requiring labeled data.

```
Area under PR = 0.8974111258021469
Area under ROC = 0.8485941506355452
Summary Stats
Precision = 0.9153358690025689
Recall = 0.9153358690025689
F1 Score = 0.9153358690025689
Confusion Matrix  DenseMatrix([[  59884.,   25624.],
             [    709.,  224812.]])
```

Figure 10: Random Forest Confusion Matrix

We introduce a method for turning a supervised model into an unsupervised model for anomaly detection. Unsupervised random forests have a number of advantages over k-means for simple detection. First, they are less sensitive to variable scale. Second, they can fit to "normal" behavior and thus can provide a probability of a data point being anomalous.

```
Prediction made in 17.901 seconds. Test accuracy is 0.9223
Learned classification tree model:
TreeEnsembleModel classifier with 10 trees

  Tree 0:
    If (feature 1 <= 519.5)
     If (feature 22 <= 0.545)
      If (feature 37 <= 0.995)
       Predict: 0.0
      Else (feature 37 > 0.995)
       If (feature 33 <= 0.005)
        Predict: 1.0
       Else (feature 33 > 0.005)
        Predict: 0.0
     Else (feature 22 > 0.545)
      If (feature 29 <= 147.0)
       If (feature 2 <= 2.0).
        Predict: 1.0
       Else (feature 2 > 2.0)
        Predict: 0.0
      Else (feature 29 > 147.0)
```

Figure 11: Rando Forest Tree with 10 trees

In this paper, two of the evaluation metrics that are considered for this study are FPR, which is defined as the rate at which normal instances are classified as anomalous and TPR, which is defined as the ratio of number of instances of correctly, predicted attacks to the total number of actual attack instances.

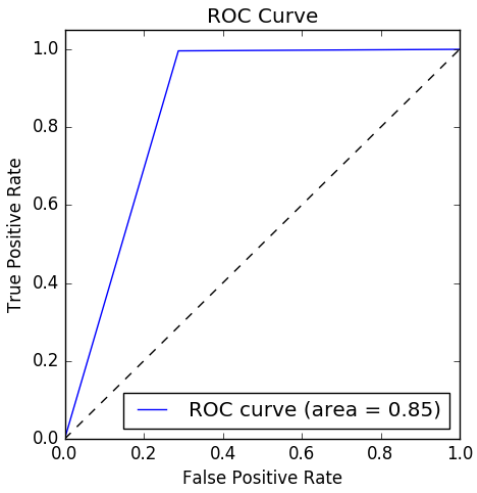


Figure 12: ROC Curve using Random Forest Prediction

## 5. RESULTS AND DISCUSSION

Our Proposed Solution is to detect the Network Intrusion Detection Using Random Forest Classification on Spark. The Main Purpose of Implementing the Solution on Spark is to have better Model Built on the Proven Distributed Processing System.

We have trained a classification tree with just the three most important predictors, in half of the time, and with a not so bad accuracy. In fact, a classification tree is a very good model selection tool!.

There are many machine-learning algorithms available, which can learn from the training data and can generalize when exposed to new untrained data. There are two types of intrusion detection technique, the first one is Misuse Detection that can catch the known attacks and hence works on the offline data and the other is Anomaly Detection which can detect any abnormal behavior and hence can work well on online data. The KDD data set is a standard data set used for the research on intrusion detection systems.

Concerns about the security of networks are rising in the recent years, according to the rapid development of the techniques used to attack these networks. According to this development, detecting traffic incoming from intrusion attempts is becoming more difficult, as the techniques used in these attacks attempt to use network packets similar, in characteristics, to those incoming from normal traffic, which makes traditional networks protection techniques very weak toward such attacks. Thus, techniques that are more complex are being developed to protect these networks against complex attacks; such are the use of machine learning to distinguish packets of normal traffic from those attacks.

Classification is one of the widely used data mining techniques, where the information added to the dataset in the form of labels that classify each object in the dataset into one, or more, of the classes that exist in the domain. During the training phase of the classifier, the characteristics of objects in each class are extracted, so that, a model is built by the classifier base on these relations. These models are then used to predict classes for new data objects during the runtime in order to assist the decision making for the system controlling the domain. These decisions are based on the characteristics of the category that the data object is predicted to be in. Thus, different Intrusion Detection Systems (IDS) are proposed based on classification techniques, where data are collected from network traffic that includes normal and attack packets in order to train classifiers to be able to predict a class for each object to allow denying attack packets from accessing the network. Our Proposed Solution is to detect the Network Intrusion Detection Using Random Forest Classification on Spark. The Main Purpose of Implementing the Solution on Spark is to have better Model Built on the Proven Distributed Processing System. We got the Accuracy of 98%.

Histogram Plotting: I have used histogram plotting in python to visualize the data with attacks, intrusion detections.

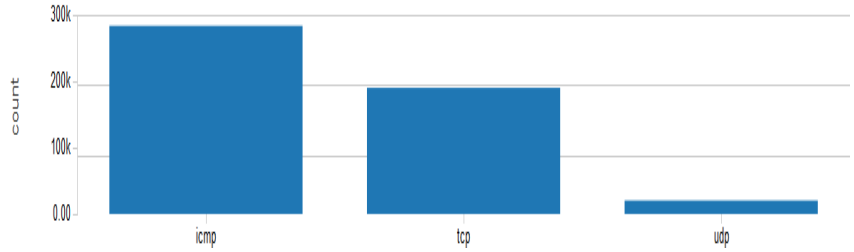


Figure 13: Protocol type by count

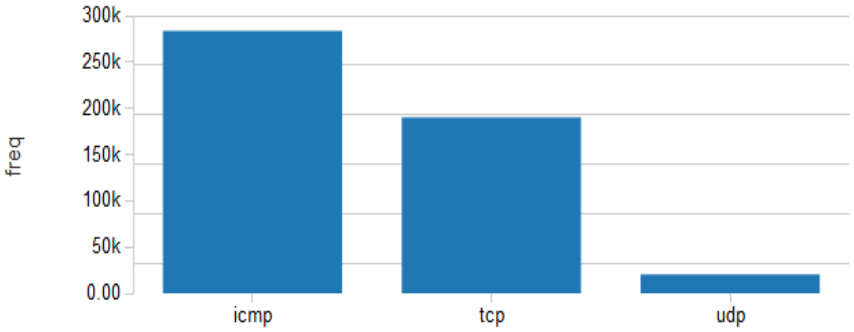


Figure 14: Protocol type by frequency

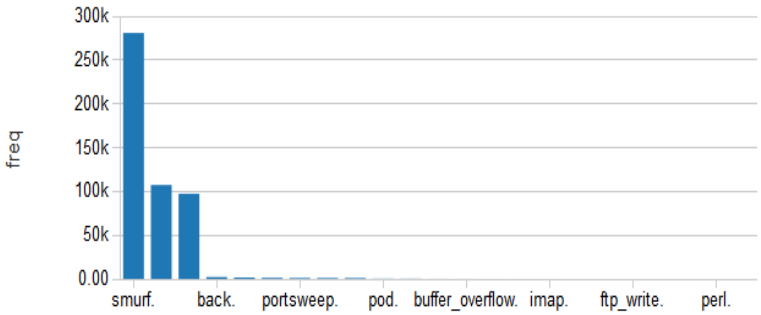


Figure 15: network labels by frequency


Figure 16: Pie plot of labels by frequency

| protocol_type | state | freq |
|---|---|---|
| icmp | attack | 282314 |
| tcp | attack | 113252 |
| tcp | no attack | 76813 |
| udp | no attack | 19177 |
| icmp | no attack | 1288 |
| udp | attack | 1177 |

Figure 17: data visualization of protocol type by state and frequency

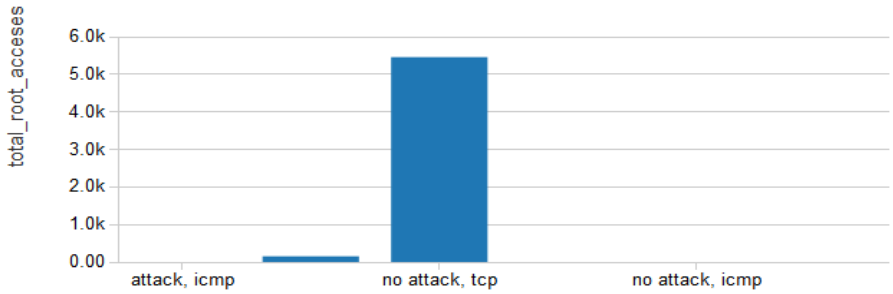


Figure 18: state of protocol type by total root access

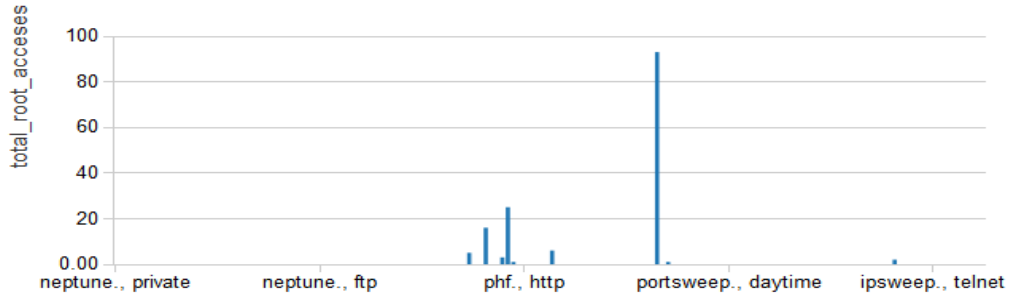


Figure 19: state of tcp protocol type by total frequency

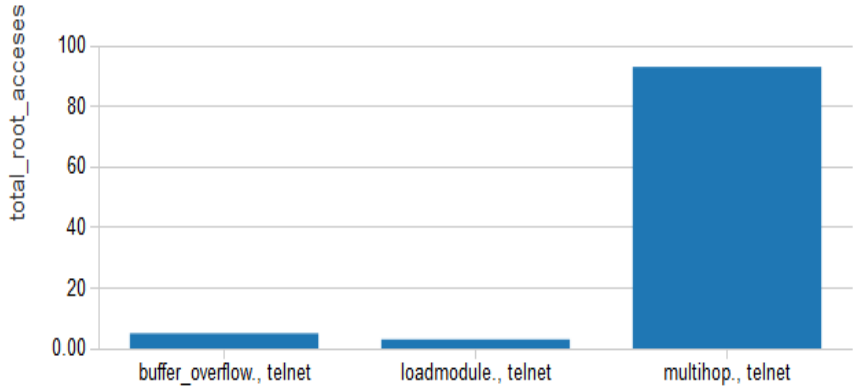


Figure 20: state of tcp protocol type by total frequency with mean deviation > = 50

Building a Pivot Table from Aggregated Data

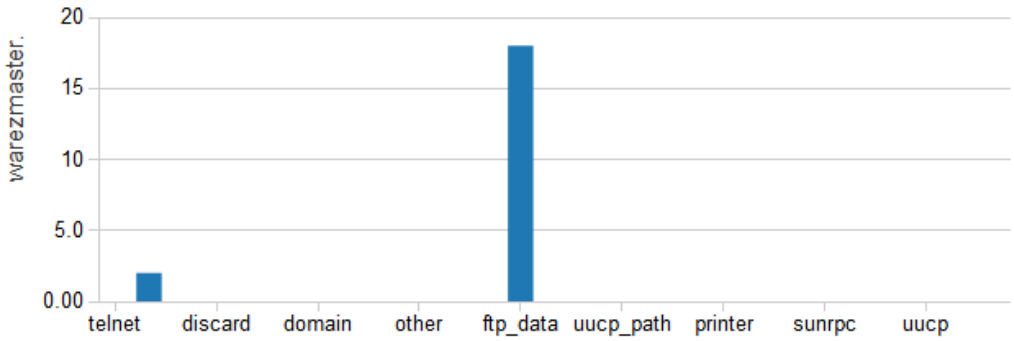


Figure 21: service attack type by total frequency

Filter TCP Attack Types Based on Service

| service | attack_type | total_freq |
|---|---|---|
| http | back. | 2203 |
| private | portsweep. | 725 |
| ftp_data | warezclient. | 708 |
| ftp | warezclient. | 307 |
| other | portsweep. | 260 |
| private | satan. | 170 |
| telnet | guess_passwd. | 53 |
| telnet | buffer_overflow. | 21 |
| ftp_data | warezmaster. | 18 |

Figure 22: Tcp attack types based on service by total frequency

## 6. CONCLUSION AND FUTURE WORK

In this paper, we employ random forests algorithm in to improve detection performance. To increase the detection rate of the minority intrusions, we build the balanced dataset by over-sampling the minority classes and down sampling the majority classes. Random forests can build patterns more efficiently over the balanced dataset, which is much smaller than the original one. The experiments have shown that the approach can reduce the time to build patterns dramatically and increase the detection rate of the minority intrusions. Instead of selecting features based on the domain knowledge, we select features automatically according to their variable importance calculated by random forests. By the feature selection algorithm, deciding upon the right set of features has become easy and automated. Although the approach reduces the dependency on the domain knowledge in feature selection, it increases the dependency on training sets. From the experiments on various values of random features, we obtain the optimal value to improve the performance of random forests.

The evaluation on the KDD'99 test set shows the performance provided by our approach is better than the best result of the KDD'99 contest (reduction of the overall error rate and the cost of misclassification). In our current approach, we only use random forests in misuse detection. Random forests algorithm also supports the outlier detection, which can be used to detect novel attacks. Outliers are cases that have significant deviation from the main body of the data. Random forests algorithm uses proximity to detect outliers. Further study may be carried out to see whether the outlier detection provided by random forests can be implemented effectively in NIDSs. We also plan to investigate other data mining methods for anomaly detection.

However, a poor detection rate in detecting U2R and R2L minority attacks are inevitable because of their large bias available in the dataset. The low false positive rate obtained in classifying attacks as well as normal instances makes our approach more interesting. We also perform a cost sensitive classification and found the models achieve low misclassification cost. Finally, it can be noted that no system is absolutely secure with a given set of best possible algorithms, while protecting our resources from network attacks. This makes the computer security is always an active and challenging area of research.

## 7. Appendix – A | GitHub Link

### R. Github Link

https://github.com/KSomisetty500921076/Data-Analytics-Major-Research-Project_KSomisetty_500921076

## 7. REFERENCES


[1] J. P. Anderson et al., "Computer security threat monitoring and surveillance," Technical report, James P. Anderson Company, Fort Washington, Pennsylvania, Tech. Rep., 1980.

[2] T. Vaidya, "2001-2013: Survey and analysis of major cyberattacks," arXiv preprint arXiv:1507.06673, 2015.

[3] W. Lee, W. Fan, M. Miller, S. J. Stolfo, and E. Zadok, "Toward costsensitive modeling for intrusion detection and response," Journal of computer security, vol. 10, no. 1-2, pp. 5–22, 2002.

[4] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," arXiv preprint arXiv:1506.00019, 2015.

[5] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and deep networks intrusion detection system: A taxonomy and survey," arXiv preprint arXiv:1701.02145, 2017.

[6] N. Gao, L. Gao, Q. Gao, and H. Wang, "An intrusion detection model based on deep belief networks," in Advanced Cloud and Big Data (CBD), 2014 Second International Conference on. IEEE, 2014, pp. 247–252.

[7] M. Moradi and M. Zulkernine, "A neural network based system for intrusion detection and classification of attacks," in Proceedings of the IEEE International Conference on Advances in Intelligent SystemsTheory and Applications, 2004, pp. 15–18.

[8] S. Mukkamala, A. H. Sung, and A. Abraham, "Intrusion detection using an ensemble of intelligent paradigms," Journal of network and computer applications, vol. 28, no. 2, pp. 167–182, 2005.

[9] J.-S. Xue, J.-Z. Sun, and X. Zhang, "Recurrent network in network intrusion detection system," in Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on, vol. 5. IEEE, 2004, pp. 2676–2679.

[10] J. Yang, J. Deng, S. Li, and Y. Hao, "Improved traffic detection with support vector machine based on restricted boltzmann machine," Soft Computing, vol. 21, no. 11, pp. 3101–3112, 2017.

[11] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, "A deep learning approach for network intrusion detection system," in Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS), BICT-15, vol. 15, 2015, pp. 21–26.

[12] J. Kim and H. Kim, "Applying recurrent neural network to intrusion detection with hessian free optimization," in International Workshop on Information Security Applications. Springer, 2015, pp. 357–369.

[13] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[14] Z. Wang, "The applications of deep learning on traffic identification," BlackHat USA, 2015.

[15] R. C. Staudemeyer and C. W. Omlin, "Extracting salient features for network intrusion detection using machine learning methods," South African computer journal, vol. 52, no. 1, pp. 82–96, 2014.

[16] ——, "Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data," in Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference. ACM, 2013, pp. 218–224.

[17] R. C. Staudemeyer, "Applying long short-term memory recurrent neural networks to intrusion detection," South African Computer Journal, vol. 56, no. 1, pp. 136–154, 2015.

[18] M. E. Aminanto and K. Kim, "Deep learning-based feature selection for intrusion detection system in transport layer."

[19] B. Dong and X. Wang, “Comparison deep learning method to traditional methods using for network intrusion detection,” in Communication Software and Networks (ICCSN), 2016 8th IEEE International Conference on. IEEE, 2016, pp. 581–585.

[20] D. Gibert Llaurado, “Convolutional neural networks for malware clas- ´ sification,” Master’s thesis, Universitat Politecnica de Catalunya, 2016. `

[21] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, “Deep learning for classification of malware system call sequences,” in Australasian Joint Conference on Artificial Intelligence. Springer, 2016, pp. 137–149.

[22] R. Upadhyay and D. Pantiukhin, “Application of convolutional neural network to intrusion type recognition,” Ben-Gurion University of the Negev, Tech. Rep., 2017.

[23] Y. LeCun et al., “Generalization and network design strategies,” Connectionism in perspective, pp. 143–155, 1989.

[24] Y. Kim, “Convolutional neural networks for sentence classification,” arXiv preprint arXiv:1408.5882, 2014.

[25] J. L. Elman, “Finding structure in time,” Cognitive science, vol. 14, no. 2, pp. 179–211, 1990.

[26] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, ¨ H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” arXiv preprint arXiv:1406.1078, 2014.

[28] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., “Tensorflow: A system for large-scale machine learning.” in OSDI, vol. 16, 2016, pp. 265–283.

[29] H. Sak, A. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” in Fifteenth Annual Conference of the International Speech Communication Association, 2014.

[30] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on. IEEE, 2009, pp. 1–6.

[31] R. Staudemeyer and C. Omlin, “Feature set reduction for automatic network intrusion detection with machine learning algorithms,” p. 105, 2009.

[32] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” Journal of Machine Learning Research, vol. 9, no. Nov, pp. 2579–2605, 2008.

[33] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” arXiv preprint arXiv:1312.6034, 2013.

[34] Data mining log file streams for the detection of anomalies
https://pdfs.semanticscholar.org/6f31/428d7b0e9e80680370c4b844cfa2bfb31ba1.pdf

[35] Experience Report: System Log Analysis for Anomaly Detection
http://jmzhu.logpai.com/pub/slhe_issre2016. K. Elissa, “Title of paper if known,” unpublished.

[36] Anomaly Detection Using Persistent Homology
https://www.computer.org/csdl/proceedings-article/cybersecsym/2016/07942418/12OmNx4yvyt.

[37] Hypergraph-Based Anomaly Detection in Very Large Networks
https://www.computer.org/csdl/proceedingsarticle/cybersecsym/2016/07942418/12OmNx4yvyt

[38] Cyber Anomaly Detection Using Graph-node Role-dynamics
https://arxiv.org/pdf/1812.02848.pdf