Utilizing the UNSW-NB15 dataset with Principal Component Analysis and Random Forest on Apache Spark to Identify Malicious Threats

Ryan Miller, Sam Nelson, Esteban Paredes, Queshana Sealey

CAP 6789: Big Data Analytics

**Abstract**

In 2019, Forbes published that 90% of the world’s data had been created in the past two years with 2.5 quintillion bytes of data created daily (Marr, 2019). As the internet expands horizontally with global adoption and vertically in applicable uses – the data found therein becomes increasingly valuable. While this newfound value is certainly an asset to the global community, it also presents itself as a liability as nefarious actors would attempt to seize the data for ill-gotten-gain. In fact, in 2021 there has been a cyber attack experienced every eleven seconds on average (Embroker, 2021). Computer scientists, data analyst, and cybersecurity experts worldwide work diligently to outpace the would-be-attackers to protect users from harm. The introduction of the UNSW-NB15 dataset in 2015 provided researchers from various fields the opportunity to test their methods and algorithms for the detection of malicious actors on real-world data. Employing Apache Spark and the UNSW-NB15 dataset, this paper utilizes Principal Component Analysis (PCA) for dimensionality reduction and the Random Forest machine learning classification algorithm to identify malicious attacks. The primary focus of this research is to determine which Apache Spark parameters will most efficiently execute the program, how many Principal Components is optimal, and how many trees to include in the Random Forest algorithm.

# Introduction

Cybersecurity is an ever-changing field. The first computer worm, known as Creeper, was a harmless experiment in 1971 by a developer working on the ARPANET to show a program can be moved from one computer to another on its own. In 1973, the first cybersecurity-oriented program, named Reaper, was created to detect and delete the Creeper worm. In the 1980’s and 1990’s, with the advent of the internet, network attacks, the number of viruses spread, and the antivirus industry all exploded to a global scale. This trend has continued to expand over the past decades. The global cybersecurity market size in 2020 was $153.16 billion due to the growing security needs for Cloud-based technologies, Internet-of-Things, and various computers in most every home in the world. Fortune Business Insights reports the cybersecurity market is projected to grow to $366.10 billion by 2028 (Insights, F.B., 2021). With the growing needs for protection against network attacks, the UNSW-NB15 dataset was generated to evaluate Network Intrusion Detection Systems in a more modern environment of network traffic. The UNSW-NB15 dataset serves as the basis for our research.

This research paper proposes a multilayered machine learning approach to detecting network attacks utilizing Principal Component Analysis for Dimensionality Reduction and the Random Forest classification algorithm. We utilized Apache Spark 2.4.0 in our research on the UNSW-NB15 dataset.

The Random Forest classification is an ensemble of the Decision Tree classification algorithm. The decision tree algorithm uses rules to create a hierarchical tree to classify an observation. The Random Forest algorithm creates multiple decision trees (forest) which increases the prediction performance. Each observation is labeled based on the prediction that occurs the most in the forest.

Apache Spark is an analytics engine for large-scale data processing. It is an open-source computing framework unifies streaming, batch, and interactive big data workloads to unlock new applications. The capabilities of Spark include implicit data parallelism and fault tolerance across nodes (Zaharia, et al., 2016). The Big Data environment of today allowed us to take advantage of this distributed compute engine built on top of the Hadoop environment.

Apache Spark requires a cluster manager and distributed storage system, in our case, Hadoop 2.6.0. Spark gave us the capability to parallelize computing resources in clusters of computers and scale up or down with ease. The Spark ML framework is a distributed machine-learning framework built on top of Spark Core and was used throughout our algorithm for string encoding, PCA, feature vectorization, classification, hyperparameter tuning, and model evaluation.

# Related Works

Intrusion Detection Systems (IDS) have been proposed for mitigating security intrusions for the past few decades. With high-speed network traffic and the overwhelming size of networks today, proposed methodologies still face the challenge of building scalable, adaptable, and lightweight IDS.

Waskle, Parashar, and Singh developed an efficient IDS incorporating principal component analysis and random forest classification on the KDD’99 dataset. The techniques they applied were intended to improve upon the detection of malicious activities on the internet. Compared with lone classification algorithms Naïve Bayes, Decision Tree, and Support Vector Machine (SVM), their results are superior to their benchmarks with a lower performance time and error rate, and a higher accuracy rate (Waskle, Parshar, & Singh, 2020).

In the conference paper “Analysis of UNSW-NB15 Dataset Using Machine Learning Classifiers” (Dickson et al, 2021), the UNSW-NB15 data is analyzed and compared with the NSL-KDD dataset for performance and accuracy using machine learning classifiers. In “Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models” (Moustafa et al, 2017), the research team authoring the paper also assess the performance of their proposed framework for a scalable anomaly detection system (ADS) using these two datasets. The volume of network traffic data generated across host systems with multiple devices, software, sensors, platforms, and other connected sources necessitates the use of Big Data analytical techniques for real time decision support of monitoring entities seeking to distinguish between normal and abnormal instances (Moustafa et al, 2017).

Aggregation of network flows focusing on interarrival times and packet sizes in Moustafa, Turnbull, and Choo’s analysis of IoT, namely MQTT, DNS, and HTTP, protocols yielded a high detection accuracy for bot detection in their IDS. The tools utilized included a testbed for the UNSW-NB15 dataset, Raspberry Pi, Node-Red for connection to the AWS IoT hub with an MQTT broker, MySQL, the tcpdump tool, and either an extractor module with Bro-IDS or packet analyzer, depending on the protocol (Moustafa, Turnbull, & Choo, 2019). The authors used an AdaBoost ensemble learning algorithm, using the Naïve Bayes, Decision Tree, and Artificial Neural Network machine learning methods, with statistical flow features for their Network Intrusion Detection System to classify potential attacks on the network.

Moustafa, Slay, and Misra wrote a paper continuing the theme of identifying zero-day attacks in 2021 titled “Generalized Outlier Gaussian Mixture Technique Based on Automated Association Features for Simulating and Detecting Web Application Attacks”. Prior to the research conducted in this paper there were many options for protecting against malicious attacks using blacklists based on the signatures of known attack systems. These same options failed to identify zero-day attacks that had previously not been identified as malicious. Prior to access to the UNSW-NB15 dataset, researchers ran simulations to test system’s abilities to identify zero-day attacks. The researchers in this paper used the UNSW-NB15 dataset to develop an Outlier Gaussian Mixture (OGM) technique for improve identification of previously unidentified threats. Utilizing a Gaussian Mixture Model the researchers combined two Probably Density Functions to estimate random distributions. The results were then compared against the simulated data and illustrated on a Q-Q plot. The OGM method was then tested on the original dataset against Cart, K-nearest neighbor, Support Vector Machine, and Random Forest with the OGM DR of 95.56% coming in highest and the False Alarm Rate (FAR) coming in lowest at 4.43%. When compared to the simulated datasets the results were similar although slightly improved, indicating the value in simulation techniques for identifying and training threats (Moustafa, Slay, & Misra, 2021).

Utilizing the Association Rule Mining (ARM) approach, the researchers wrote “Flow Aggregator Module for Analysing Network Traffic” applied their methodologies to the UNSW-NB15 dataset to test whether or not they could “efficiently correlate the most-repeated observations without losing any observations.” The concept is based on using techniques and tools that are accustomed to handling big data – such as MySQL along with statistical and machine learning methods. By clustering the source and destination IP addresses the researchers were able to observe the flows occurring between the two sources. To reduce data distortion commonly experienced in sampling methods the researchers employed the ARM technique enabling the processing of large numbers of network flows. Ultimately, it was concluded that the ARM technique was effective when abnormal instances are already identified, otherwise the sampling technique was recommended (Moustafa, Creech, & Slay, 2018).

A network forensic methodology for monitoring and investigating network attacks was proposed by Moustafa et al in 2018. This is done using a scheme that retains network traffic data, identifies important features using the chi-square statistic, then determines which events are anomalous with a novel correntropy-variation technique. The traffic data is captured using tcpdump before being stored in a MySQL databased for further analysis. The chi-square statistic is then used to identify important features by measuring the correlation of two variables, of which the highest ranked are identified as important. Finally, the correntropy-variation technique estimates the similarity between attack samples and normal flows, and establishes a variation threshold which is used to identify attacks. This technique is applied on the UNSW-NB15 dataset, and its performance is compared against three other approaches for the same analysis. The results revealed that the scheme proposed by the authors outperformed the aforementioned techniques in accuracy and reduced occurrences of false alarms (Moustafa et al., 2018).

Elmrabit et al. (2020) looks at twelve techniques to determine which ones work well for detecting cyber-security threats. They examine six classical supervised machine learning algorithms and six deep learning algorithms. They used the UNSW-NB15 dataset, CICIDS-2017 dataset, and ICS cyber-attack dataset. For most datasets, Random Forest preformed the best based on accuracy precision, recall, and AUC. However, this could be due to the fact the deep learning algorithms need large amounts of data to perform well. The Naïve Bayes classification haws the lower performance in terms of accuracy, precision, recall, and AUC (Elmrabit, Zhou, Li, & Zhou, 2020, p. 125-129).

# The Data

Moustafa and Slay created the UNSW-NB15 dataset to evaluate Network Intrusion Detection Systems using the IXIA PerfectStorm tool to simulate normal and abnormal traffic within a network. The simulation included three virtual servers, one spreading nine families of attacks, and was conducted in early 2015 capturing 100 GBs in pcap files divided into 1000 MBs using the tcpdump tool. Data was extracted using Argus, Bro-IDS tools, and twelve C# algorithms with output routed to SQL Server 2008. The features captured included flow, basic, content, time, general purpose, connection, and labelled features. The attack simulation captured is over 2.5 million records long and 12.6% of the distribution are attacks classified as any of the following: Fuzzers, Analysis, Backdoor, DoS, Exploits, Generic (including block-ciphers), Reconnaissance, Shellcode, and Worms. (Moustafa & Slay, 2015)

The UNSW-NB15 dataset is commonly utilized in published research whose material cover (but is not limited to) intrusion detection methods, statistical analysis, big data, and machine learning. It is one of several available benchmark datasets of hybrid normal/abnormal labeled network traffic, which also include:

* DARPA
* KDD’99
* NSL-KDD
* DEFCON
* CAIDA
* LBNL
* CDX
* Kyoto
* ISCX 2012
* UNS ISCX

(Dickson et al, 2021)

This dataset was then made available for public use on the research.unsw.edu.au, with 2,540,047 records distributed across 4 csv files. We combined these files containing the attack data set into a unified csv which we have used for our work.

According to our observations, as well as the research summary provided, there are 49 resulting features (or attributes) that are contained in this publicly available data set. Our group’s assessment of each of these columns is shown in the following chart:

|  |  |  |  |
| --- | --- | --- | --- |
| **Column No.** | **Column Name** | **Data Type** | **Group 4 Descriptions** |
| 1 | srcip | nominal | The source IP address is where the packet is sent from |
| 2 | sport | integer | The source port identifies the process that sent the packet |
| 3 | dstip | nominal | The destination IP address is where the packet is being sent |
| 4 | dsport | integer | The destination port identifies the process that receives the packet |
| 5 | proto | nominal | These are communication protocols, each with a different set of rules on how it communicates with data between nodes |
| 6 | state | nominal | Events and actions associated with a connection |
| 7 | dur | Float | Record connection time (in seconds) |
| 8 | sbytes | Integer | Number of data bytes from source to destination |
| 9 | dbytes | Integer | Number of data bytes from destination to source |
| 10 | sttl | Integer | Value for the period of time that a packet should exist before being discarded |
| 11 | dttl | Integer | Value for the period of time that a packet should exist before being discarded |
| 12 | sloss | Integer | Packets from the source that needed to be retransmitted or did not make it to the destination |
| 13 | dloss | Integer | Packets from the destination that either didn't make it to the source or needed to be retransmitted. |
| 14 | service | nominal | This attribute depicts the service which was utilized for the server request. |
| 15 | Sload | Float | Source bits per second is the size of the transmitted data being uploaded from some source. All but 18 rows stay within 32 bits. |
| 16 | Dload | Float | Destination bits per second is the size of the transmitted data being received/downloaded from some source. |
| 17 | Spkts | integer | The number of packets (containing control information and user data) in a network transfer from some source to some destination. |
| 18 | Dpkts | integer | The number of packets (containing control information and user data) in a network transfer from some destination to some source. |
| 19 | swin | integer | An 8-bit integer that determines the maximum amount of data that can be sent before the source must wait for the destination to acknowledge. |
| 20 | dwin | integer | An 8-bit integer that determines the maximum amount of data that can be sent before the destination must wait for the source to acknowledge. |
| 21 | stcpb | integer | A TCP sequence number in range 0 to 2^32 from the source used to keep track of how much data is sent. |
| 22 | dtcpb | integer | A TCP sequence number in range 0 to 2^32 from the destination used to keep track of how much data is sent. |
| 23 | smeansz | integer | Average packet size from the source |
| 24 | dmeansz | integer | Average packet size from the destination |
| 25 | trans\_depth | integer | Used to track the current deepest transaction. This is meant to cope with missing requests and responses. |
| 26 | res\_bdy\_len | integer | Response Body Length. |
| 27 | Sjit | Float | Source jitter in milliseconds. High jitter indicates a poor connection or transfer speed and the loss of information. |
| 28 | Djit | Float | Desitnation jitter in milliseconds. High jitter indicates a poor connection or transfer speed and the loss of information. |
| 29 | Stime | Timestamp | Record start time. |
| 30 | Ltime | Timestamp | Record last time. Combined with start time total record time can be computed. |
| 31 | Sintpkt | Float | The amount of time in milliseconds between the receipt of one packet and the next for the source. |
| 32 | Dintpkt | Float | The amount of time in milliseconds between the receipt of one packet and the next for the destination. |
| 33 | tcprtt | Float | The round-trip time of synchronize packet - synchronize acknowledgement - acknowledgement packets; the sum of synack and ackdat. |
| 34 | synack | Float | Time between the synchronize packet and the synchronize acknowledgement packet return. |
| 35 | ackdat | Float | Time between the synchronize acknowledgement packet and the acknowledgement packet return. |
| 36 | is\_sm\_ips\_ports | Binary | A test to identify if source and destination IP addresses and port numbers are equal (1) or else (0). |
| 37 | ct\_state\_ttl | Integer | These values pertain to the 6 states of Australia, with the value of 0 likely representing an unknown or unclear source and destination. |
| 38 | ct\_flw\_http\_mthd | Integer | Within the http service traffic data represented by the record, this field would represent the number of Get and Post methods that are transmitted over the server. |
| 39 | is\_ftp\_login | Binary | FTP access is usually password protected with user logins, and this field indicates whether there was such an access requirement (if an ftp session was even utilized). |
| 40 | ct\_ftp\_cmd | integer | If an ftp session was utilized, this would represent the number of flows where a command was used. The value doesn't go higher than 8 in CSV 1. |
| 41 | ct\_srv\_src | integer | Sorts within the last time (which appears to be column 30, not 26) to find when two characteristics matched out of 100 connections. The resulting value should never be higher than 100, minimum 0. In this column, checks for service (column 14) and source address (column 1) being the same. |
| 42 | ct\_srv\_dst | integer | Same as ct\_srv\_src (41), only with service (14) and destination address (3) being the same within the last time. Once more, last time is (30) not (26). |
| 43 | ct\_dst\_ltm | integer | Same as ct\_srv\_src (41), only with destination address (3) being the same across 100 connections. There being only 6 IP addresses used in the testing, a high value here is understandable. |
| 44 | ct\_src\_ ltm | integer | Similar to as ct\_dst\_ltm(43), only with source address (1) being the same across 100 connections. There being only 6 IP addresses used in the testing, a high value here is again understandable. |
| 45 | ct\_src\_dport\_ltm | integer | Similar to ct\_srv\_src (41), only with source address (1) and destination port (4) being the same within the last time (30). This would represent traffic being sent where it started. |
| 46 | ct\_dst\_sport\_ltm | integer | Similar to ct\_src\_dport\_ltm (45) above, but with destination address (3) and the source port (2) being the same. I'd think we'd get similar results ge as ct\_src\_dport\_ltm, since both are sending to where the traffic originated. |
| 47 | ct\_dst\_src\_ltm | integer | Similar idea here to ct\_srv\_src (41) and ct\_src\_dport\_ltm (45), only this time very specifically the same source (1) and the destination (3) address. Its like sending mail with a return to sender slip already on it. |
| 48 | attack\_cat | nominal | This column will list one of the nine families the the attack records are sorted into. These attack families vary in their style and effects, ranging from exploiting a known security vulnerability to merely gathering security data. |
| 49 | Label | binary | A binary field that will show as 1 if the record pertains to one of the attack families. Otherwise, it will show as 0. This field should not be used for testing purposes, as your basically giving the answer away here. |

For the data set provided, there is a binary column known as Label (49) which registers as 0 if the record was a benign traffic flow, and 1 if the record was classified as one of the 9 attack families. This would be the class variable for the data set for a binary assessment of normal or abnormal data. The attack\_cat (48) nominal feature would be a more specific identification of which of the attack families the abnormal records would be associated with. According to the UNSW-NB15 Network Data Set document (Moustafa & Slay, 2015), the nine families of attacks are Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms.

Please see below for a description of each the attack families. (Sarhan et al, 2020, p.8)

|  |  |
| --- | --- |
| **Class** | **Description** |
| Benign | Normal unmalicious flows. |
| Fuzzers | An attack in which the attacker sends large amounts of random data which cause a system to crash and also aim to discover security vulnerabilities in a system. |
| Analysis | A group that presents a variety of threats that target web applications through ports, emails and scripts. |
| Backdoor | A technique that aims to bypass security mechanisms by replying to specific constructed client applications. |
| DoS | Denial of Service is an attempt to overload a computer system’s resources with the aim of preventing access to or availability of its data. |
| Exploits | Are sequences of commands controlling the behavior of a host through a known vulnerability. |
| Generic | A method that targets cryptography and causes a collision with each block-cipher. |
| Reconnaissance | A technique for gathering information about a network host and is also known as a probe. |
| Shellcode | A malware that penetrates a code to control a victim’s host. |
| Worms | Attacks that replicate themselves and spread to other computers. |

Normal benign data accounts for 87.4% of the total data set. The abnormal records identified as attacks accounted for 12.6% of the data set. We have listed the following counts of each attack based on our analysis of the data:

|  |  |
| --- | --- |
| **Attack Type** | **Attack Count** |
| Generic | 215,481 |
| Exploits | 44,525 |
| Fuzzers | 24,246 |
| DoS | 16,353 |
| Reconnaissance | 13,987 |
| Analysis | 2,677 |
| Backdoor | 2,329 |
| Shellcode | 1,511 |
| Worms | 174 |

# Preprocessing

Before conducting our PCA analysis, we conducted preliminary preprocessing of the UNSW-NB15 data set. After our evaluation, it was determined the following nominal columns would be redundant to include in the principal component analysis, due to their impacts being measured as integer values in other features of the data set. For instance, srcip (Source IP address) is compared to the dstip (Destination IP address) and the port features (sport and dsport) in order to produce binary comparisons columns such as is\_sm\_ips\_ports, ct\_srv\_src, ct\_srv\_dst, t\_src\_ ltm etc.

* srcip (Source IP address)
* sport (Source port number)
* dstip (Destination IP address)
* dsport (Destination port number)
* Stime (record start time)
* Ltime (record last time)
* ct\_state\_ttl (No. for each state according to specific range of values for source/destination time to live)
* attack\_cat (The name of each attack category)

The feature “Label” was identified as our target feature and encoded as 0 for normal and 1 for attack records. Label was excluded from vector assembler input columns. This left us with a total of 40 features to be analyzed.

The following 3 nominal columns were to be included, but had StringIndexer and OneHotEncoder functionality applied to them to produce corresponding Index and Vector columns.

* proto (Transaction protocol)
  + protoIndex
  + protoVec
* state (Indicates to the state and its dependent protocol, e.g. ACC, CLO …and (-) (if not used state)
  + stateIndex
  + stateVec
* service (http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if not much used service)
  + serviceIndex
  + serviceVec

The resulting vector columns took the place of the original nominal columns, resulting in 39 features:

"protoVec", "stateVec", "sbytes", "dbytes", "sttl", "dttl", "sloss", "dloss", "serviceVec", "Sload", "Dload", "Spkts", "Dpkts", "swin", "dwin", "stcpb", "dtcpb", "smeansz", "dmeansz", "trans\_depth", "res\_bdy\_len", "Sjit", "Djit", "Sintpkt", "Dintpkt", "tcprtt", "synack", "ackdat", "is\_sm\_ips\_ports", "ct\_flw\_http\_mthd", "is\_ftp\_login", "ct\_ftp\_cmd", "ct\_srv\_src", "ct\_srv\_dst", "ct\_dst\_ltm", "ct\_src\_ ltm", "ct\_src\_dport\_ltm", "ct\_dst\_sport\_ltm", "ct\_dst\_src\_ltm"

# Algorithm Used

Because our dataset was very large, with over 2.5 million rows and nearly 50 columns of data comprising over one-hundred million data points, it was important to first execute a Principal Component Analysis (PCA) on our dataset. PCA is method for reducing the dimensionality of data by identifying multicollinearity within the data through the use of a covariance matrix (Jaadi, 2021). The result of a PCA is that the independent variables, in this case nearly fifty in number, are reduced to a series of principal components that are comprised of nearly all of the information but with much less noise – or relationship between the variables.

The first step in a PCA is to standardize the data in order to eliminate the overweighting that will occur if the variance within certain variables has a greater magnitude than others (Jaadi, 2021). Once the data is standardized, the machine computes a covariance matrix to illustrate the relationship between each variable (Jaadi, 2021). A positive value in the covariance matrix indicates a correlated relationship, and a negative value indicates an inverse relationship (Jaadi, 2021).

The next step is to compute Eigenvectors and Eigenvalues to determine the composition of the principal components. Essentially, an analysis is run by the machine to determine the new axis with the greatest variance, then the next largest variance – and so on, until the appropriate number of principal components has been produced. As Zakira Jaadi points out, “an important thing to realize here is that, the principal components are less interpretable and don’t have any real meaning since they are constructed as linear combinations of the initial variables” (Jaadi, 2021). Below is an example of a PCA being run to determine the maximum variance axis:

Chart, scatter chart

Description automatically generated

Source: A Step-by-Step Explanation of Principal Component Analysis (PCA), (Jaadi, 2021)

During this process, a feature vector is creating to determine which principal components should be retained. Because of this, you cannot use principal components to construct a regression equation like you could if you used the original variables. However, what you give up in clarity you gain in efficiency. The principal components are ranked in order of greatest amount of variance explained to least with the optimal number of components selected based on the amount of variance you wish for you model to account for (Jaadi, 2021).

The last step is to utilize the selected principal components in machine learning algorithms to produce a cleaner model with less noise – or correlation. After analyzing the variance explained by each principal component, we determined that six was the optimal number of components to use for our algorithm.

In this paper, we selected the Random Forest algorithm to work with our principal components. Random Forest (RF) is a machine learning algorithm that specifically known for its ability to work with classification problems (Yiu, 2019). Although there are many of machine learning algorithms that can handle classification problems, RF is particularly adept at solving classification problems because it enables the machine to rely on large combinations of methods and solutions yielded by the ensemble in order to determine the best solution. RF is an ensemble learning model with attributes chosen from all attributes to best segment observations. Each attribute is chosen randomly and splits the observations by the tree nodes. In RF classification models, each iteration of the model rearranges the attribute segmentation order and trains a selected number of trees to take a majority vote on unseen samples.

RF is a type of decision tree algorithm (Yiu, 2019). When utilizing an RF algorithm, varying numbers of decision trees are created with each tree constructed of different selection criteria that tests different sets of data. Because each tree operates with a different decision process, the ensemble, or collection of trees, returns largely uncorrelated results (Yiu, 2019). Furthermore, because a PCA was employed to remove the correlation in the dataset, the end result was mostly uncorrelated components being analyzed by mostly uncorrelated trees. Trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance (Hastie, Tibshirani, & Friedman, 2008).

Model diversity within the trees is important because the model is sensitive to small changes in the data being test (Yui, 2019). To combat this, RF permits sampling data with replacement so that each tree may have the same data point included multiple times. If this was not permitted, each tree would largely test the same data and the results would be more correlated than if replacement is employed. In this sense, the data is not divided into smaller pieces and analyzed by different trees but rather the same number of data points, equal to the sum total of the entire training dataset, is analyzed by each tree – however since the partitions were created with replacement the data analyzed by each tree is not identical.

The last component in constructing an ensemble is to select random features with which to analyze the data. Since each tree is constructed of different, random features – even if the data being analyzed is similar the errors encountered by each tree will be different. As such, the aggregate decision of the ensemble will yield uncorrelated errors and explain a larger portion of the correlation within the dataset being tested. In this paper, six principal components were computed from 35 of the original UNSW-NB15 dataset features and 4 features were encoded numerically from string datatypes. The principal components were tested against RF algorithms that included five, ten, twenty, and thirty trees.

**Results**

The assessment of the RF algorithm first started with a determination of the optimal core count to be used by the Spark application running the data analysis. Core count represents the number of worker threads to be utilized, which provide the CPU, memory, and storage resources to Spark applications. Each worker contains an executor, a JVM process that performs tasks which are allocated to the worker by the Driver Program. In order to specify the exact number of such cores, local was used as the master URL when the application was initiated. In this manner we conducted three tests at core counts of 4, 12, and 20 in order to gauge the optimal parameters for performance and efficiency. The results, shown below, indicated diminishing returns beyond a count of 12 cores, where a run time of 35 seconds was achieved:

|  |  |  |  |
| --- | --- | --- | --- |
| **# of Cores Used** | **4** | **12** | **20** |
| Time to Run Application | 57 s | 35 s | 35 s |
| Memory Used | 0 | 0 | 0 |
| # of Spark Tasks Run | 138 | 390 | 623 |
| # of Completed Tasks | 138 | 390 | 623 |

Based on the metrics relating to application runtime and # of tasks, we have assessed a quantity of 12 cores as being the optimal set to run our application on. This was decided given that both the 12-core and 20-core applications ran the same length of time at 35 seconds, but the 12-core app generated significantly fewer tasks, which results in fewer resources being utilized, a major consideration in a live environment.

Chart

Description automatically generated with low confidence

A confusion matrix provides the results of the algorithm in order to calculate the effectiveness of the algorithm’s performance. In our case, we were most concerned with the Type II error – or the error that results in a malicious source being approved as benign – which is represented in the bottom left cell of the confusion matrix. Ten trees resulted in the lowest Type II error, but we were not content to simply make our selection on the basis of this outcome.

We calculated the Accuracy which ranged from .9857 with five trees to .9875 with thirty trees with a reduction in marginal increase as the number of trees increased. Due to the large number of benign sources in the dataset, all values of trees returned high accuracy. Because overall Accuracy was less important to us than Type II error and all accuracies were in fact very high, we did not change our mind on the number of trees to use being ten after examining accuracy. Next, we analyzed Precision and Recall. Because we are most concerned with not allowed malicious sources into the network, we preferred Recall over Precision and Accuracy. Again, ten trees had the most preferred performance. We then calculated and examined the F1 Measure, which penalizes performances with extreme outcomes. In this case, ten, twenty, and thirty trees resulted in F1 Measures that were comparable and did not alter our decision to use ten trees.

To finalized our selection, we examined the Area Under the Curve for the Receiving Operator Characteristics. Essentially, the AUC-ROC is assumed to be .5 in binary cases as a 50/50 guess is probabilistically 50%. The higher the AUR-ROC, the more accurate the model. Again, ten cores had the higher AUC-ROC. Last, we calculated the False Positive Rate and False Negative Rate as illustrated. Twenty trees resulted in the lowest FPR and ten trees resulted in the lowest FNR. Again, because we were mainly concerned with failing to identify malicious sources, we continued with our selection of ten trees.

In conclusion, after calculating the confusion matrices and the corresponding measures, our team assessed a count of **ten trees** as *the most effective and accurate version* of the Random Forest algorithm to employ for our dataset. Our second choice would have been twenty trees, with thirty trees coming in a distant third but well ahead of last place five trees. As mentioned in question 3, our optimized core count was for a total of 12 to conduct the application process. Given our AUC value of 0.7304 for our chosen parameters, our usage of the Random Forest Algorithm yielded statistically significant results vs the average value of 0.5. Our achievement of an approximately 1.1% false negative rate is also an overwhelming reduction in the total number of malicious actions that could have made it through the algorithm.

The performance analysis also highlighted Scala’s capacity to scale up resources to minimize runtimes, with expected diminishing returns. Though challenging, it was made clear that combining and optimizing applications of both Random Forest and Scala demonstrated significant synergy. Additional testing and modifications could yield even more optimal results, though such likelihoods are not to dimmish the work and effort put in by the team to achieve our current values.

# Conclusion

As cybersecurity continues to grow, there will be a need to protect against network attacks. Using the UNSW-NB15 dataset, we proposed a multilayered machine learning approach to detect network attacks. To perform the analysis, we used Principal Component Analysis to reduce the dimensions and the Random Forest classification algorithm. The Spark ML framework is a distributed machine-learning framework built on top of Spark Core and was used throughout our algorithm for string encoding, PCA, feature vectorization, classification, hyperparameter tuning, and model evaluation. The algorithms were implemented using Apache Spark due to its ability to parallelize computing resources. We selected the appropriate dimensions and then used PCA to select a subset of features. After, we ran the Random Forest algorithm with a different number of trees, components, and cores. Based on this analysis, the optimal number of trees is ten and the optimal number of components is six.

# Future Work

This research paper proposed a multilayered machine learning pipeline utilizing Principal Component Analysis for Dimensionality Reduction and the Random Forest classification algorithm to improve accuracy in detecting network attacks. Within our approach, additional preprocessing techniques and feature elimination may prove beneficial for better model interpretability and reduced multicollinearity. Principal component analysis (PCA) revealed that the first two principal components explained approximately 99% of the variance in the data which is significant and may have led us to overfitting. In the future, combining our algorithm with boosting algorithms and other classification frameworks for an ensemble voting classifier approach should significantly improve our predictive capabilities. In addition, we vetted our hyperparameters with five-fold cross validation and a limited parameter grid. A larger grid of Random Forest and PCA hyperparameters, and any additional pipeline stages, should be experimented with for robustness.

References

Dickson A., Thomas C. (2021). Analysis of UNSW-NB15 Dataset Using Machine Learning Classifiers. In: Thampi S.M., Piramuthu S., Li KC., Berretti S., Wozniak M., Singh D. (eds) Machine Learning and Metaheuristics Algorithms, and Applications. SoMMA 2020. Communications in Computer and Information Science, vol 1366. Springer, Singapore. <https://doi.org/10.1007/978-981-16-0419-5_16>

Elmrabit, N., Zhou, F., Li, F., & Zhou, H. (2020, June). Evaluation of machine learning algorithms for anomaly detection. In 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security) (pp. 1-8). IEEE.

Embroker. (2021, November 19). 2021 Must-know Cyber Attack Statistics and

Trends. Embroker. (2021, November 19). Retrieved December 4, 2021, from https://www.embroker.com/blog/cyber-attack-statistics/.

Hastie, T., Tibshirani, R., & Friedman, J. (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

Insights, F.B. (2021). Cyber Security Market Size, Share & COVID-19 Impact Analysis, By Component (Solution and Services), By Deployment Type (Cloud and On-Premise), By Enterprise Size (Small & Medium Enterprise and Large Enterprise), By Industry (BFSI, IT and Telecommunications, Retail, Healthcare, Government, Manufacturing, Travel and Transportation, Energy and Utilities and Others) and Regional Forecast, 2021-2028. Report ID:FBI101165.

Marr, B. (2019, September 5). How much data do we create every day? the mind-blowing stats everyone should read. Forbes. Retrieved December 4, 2021, from https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=50a715c360ba.

Moustafa, N. and Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 2015 Military Communications and Information Systems Conference (MilCIS). 2015, pp. 1-6, doi: 10.1109/MilCIS.2015.7348942.

Moustafa N., Creech G., Slay J. (2017). Big Data Analytics for Intrusion Detection System: Statistical Decision-Making Using Finite Dirichlet Mixture Models. In: Palomares Carrascosa I., Kalutarage H., Huang Y. (eds) Data Analytics and Decision Support for Cybersecurity. Data Analytics. Springer, Cham. <https://doi.org/10.1007/978-3-319-59439-2_5>.

Moustafa, N., Creech, G., & Slay, J. (2018). Flow aggregator module for analysing network traffic. *Advances in Intelligent Systems and Computing*, 19–29. <https://doi.org/10.1007/978-981-10-7871-2_3>

Moustafa, N., Misra, G., & Slay, J. (2021). Generalized outlier gaussian mixture technique based on automated association features for simulating and detecting web application attacks. *IEEE Transactions on Sustainable Computing*, *6*(2), 245–256. <https://doi.org/10.1109/tsusc.2018.2808430>

Moustafa, Nour, et al. "A Network Forensic Scheme Using Correntropy-Variation for Attack Detection." IFIP International Conference on Digital Forensics. Springer, Cham, 2018.

Moustafa, N., Turnbull, B. andChoo, K.R. (2019, June). An Ensemble Intrusion Detection Technique Based on Proposed Statistical Flow Features for Protecting Network Traffic of Internet of Things. IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4815-4830, June 2019, doi: 10.1109/JIOT.2018.2871719.

Sarhan, M., Layeghy, S., Moustafa, N., & Portmann, M. (2020, November 18). NetFlow datasets for Machine Learning-based network intrusion detection systems. arXiv.org. Retrieved November 11, 2021, from <https://arxiv.org/abs/2011.09144>.

Waskle, S., Parashar, L. and Singh, U. (2020). Intrusion Detection System Using PCA with Random Forest Approach. 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020, pp. 803-808, doi: 10.1109/ICESC48915.2020.9155656.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., . . . Stoica, I. (2016, November). Apache Spark: a unified engine for big data processing. Communcations of the ACM, 59(11), 56-65. doi:10.1145/2934664.