NETWORK INTRUSION DETECTION SYSTEM

A PROJECT REPORT

Submitted By

Dharu Piraba Muguntharaman dm5596
Mohanna R S mr6552
Shamyukta Rajagopal sr6626

New York University

May 13, 2023

ABSTRACT

Ubiquitous networks, wireless sensor networks, and web technologies have all contributed to an exponential rise in computer usage in today's networked society. This has led to an enormous amount of data that needs to be processed and evaluated, which is more than can be done with generally available hardware and software. In order to successfully reduce the quantity of data volumes and dimensions, we intend to make use of big data solutions like Apache Spark. These technologies can aid in the detection and prevention of network security problems by evaluating vast quantities of intricate and varied data from numerous sources. They can specifically be used to create ultra-high-dimensional data models that precisely evaluate online stream data, enabling us to anticipate and identify intrusions and attacks in real-time. We will use a variety of research techniques, including data mining, machine learning, and statistical analysis, to draw conclusions from the data and create predictive models in order to accomplish this goal. In doing so, we intend to help advance the creation of network security technologies that will better safeguard our digital assets in the data-driven society of today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Network security is an essential concern for businesses of all sizes. Ensuring the security of computer networks is crucial in the linked world of today. The goal of this project is to create a highly accurate classifier that can successfully differentiate between ”good” or typical network connections and ”bad” network connections (intrusions or attacks). This will be accomplished by treating the issue as a supervised learning task and classifying network flows as either benign or malicious using the labels included in the dataset. Using Apache Spark Structured Streaming, which offers distributed storage and processing capabilities, data transformation and purification procedures are implemented to guarantee the accuracy of the data analysis. The data is then categorized using a range of techniques that make distinctions between the different sorts of attacks offered in the dataset. The dataset’s relevant subset of attributes is chosen as predictors to categorize specific network flows. To ascertain their viability, conventional machine learning algorithms like k-nearest neighbors and random forest are utilized. The effectiveness of different machine learning methods is also investigated using Spark MLlib, an extendable machine learning framework included into Apache Spark. This project attempts to improve network security and safeguard against cyberthreats that can compromise critical data and interfere with corporate operations. It does this by creating a highly accurate classifier.

1.1 Problem Statement

The need for businesses to install efficient Intrusion Detection Systems (IDS) to defend their networks has increased due to the exponential development of online activities and the ongoing escalation of cyber threats. The massive amounts of data produced by modern networks have proven difficult for traditional IDS systems to handle, which frequently makes it impossible to identify and address potential threats in real-time. The goal of this project is to provide an IDS solution that makes use of Apache Spark's capabilities to process massive amounts of data in real-time. The goal of our work is to create a system that is more capable than conventional IDS systems of detecting and responding to possible threats by utilizing the distributed storage and processing capabilities of Spark. In order to evaluate network traffic data and spot possible threats as they emerge, sophisticated machine learning models and algorithms will be developed and put to use. By offering businesses a strong and effective IDS solution that can defend against cyberattacks and reduce the danger of data breaches, this initiative ultimately seeks to contribute to the establishment of a more secure online environment.

1.2 Objectives

The primary objective of this project is to create an intrusion detection system (IDS) that is capable of quickly identifying and preventing network security breaches. This will be accomplished through real-time analysis of massive amounts of network traffic data to spot any suspicious behaviour that might point to an impending incursion or attack.

In order to achieve this, we will make use of Apache Spark's capabilities to create an effective and scalable IDS system that can handle and analyze enormous amounts of data in real-time. In order to enhance intrusion detection accuracy and lower false positives, we will create cutting-edge machine learning models and algorithms. Additionally, we will strengthen network security and defend against online dangers that can jeopardize important data and interfere with business operations. The specific goals of this project are to:

- Create an IDS system that is reliable and expandable and can quickly process and analyze massive amounts of network traffic data.
- Use advanced machine learning models and algorithms to increase intrusion detection accuracy and decrease false positives.
- By identifying and responding to possible threats in real-time, you can improve network security. Defend against online dangers that could jeopardize confidential information and interrupt business operations.

A more secure online environment will be developed as a result of meeting these goals, which will also give businesses a strong tool to defend against cyberattacks and reduce the danger of data breaches.

CHAPTER 2

PROPOSED SYSTEM

2.1 Prototype Pipeline : Stream Processing and Transformation

The project's streamlining and data flow chart outlines various procedures that must be taken in order to efficiently identify and stop security breaches in a network environment. The KDD dataset is used to perform the first phase, which is to collect the required data. The network traffic data in this dataset can be used to train and test the model. The data is then broadcast into Kafka, a distributed streaming platform that enables real-time data processing. To handle massive amounts of data in real-time, which is essential for effective intrusion detection, is a crucial step. Once the data is in Kafka, it goes through feature engineering, which is the process of choosing and modifying the data to increase intrusion detection accuracy and decrease false positives. This stage entails locating and selecting relevant features from the data and converting them into a format that the machine learning model can utilize.

The machine learning model receives the preprocessed data and uses it to make

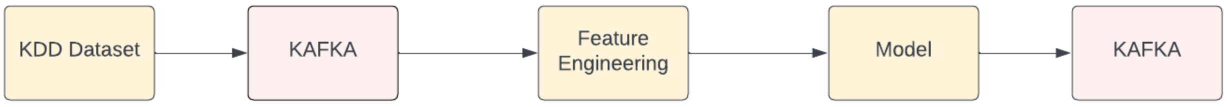


FIGURE 2.1: Prototype pipeline

predictions. In order to effectively categorize network traffic as benign or malicious, a variety of algorithms and approaches are used in the model development process. Enhancing network security and defending against online threats that could compromise critical data and interfere with corporate operations are the ultimate goals of this stage. Finally, the model's output is streamed back into Kafka so that it may be evaluated and in real-time. This guarantees that any potential hazards are found and addressed as soon as feasible. This data flow chart and streamlining technique are intended to make effective intrusion detection and prevention possible in a network setting.

2.2 Architecture design

The above diagram illustrates the architecture design of the proposed system. The NSL-KDD dataset is used as the input data for this project. This data is then fed for data preprocessing where unrelated features are removed and missing values are handled. This preprocessed data is then fed for the Feature engineering process. This process helps in identifying relevant features important for building the model. The next phase is doing classification using machine learning methods. This entails using feature-engineered, preprocessed data to train various machine learning models that will categorize network traffic as either normal or suspicious. Here, two techniques are used: Gaussian mixture clustering and random forest classifiers, and K-means clustering with random forest classifiers. The effectiveness of these two approaches for this particular dataset is assessed and compared. The model is then examined to determine how well it performs. Analyzing the model's accuracy, precision, recall, and F1 score is part of this process. Any modifications required to enhance the performance of the model are done in accordance with the evaluation. Overall, this architecture design diagram displays a thorough and iterative procedure for creating a powerful machine learning intrusion detection system.

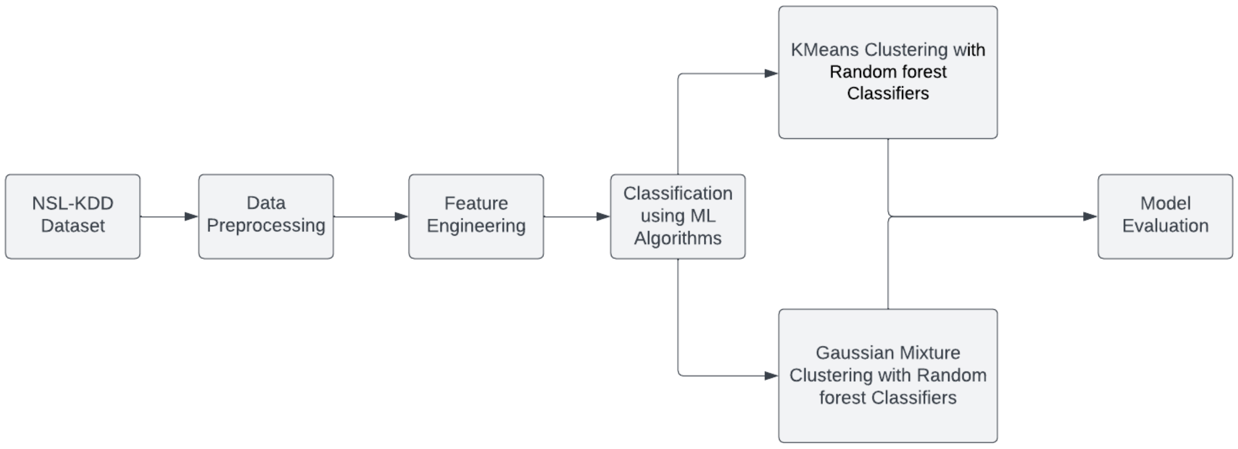

NSL-KDD Dataset
Data Preprocessing
Feature Engineering
Classification using ML Algorithms
KMeans Clustering with Random forest Classifiers
Gaussian Mixture Clustering with Random forest Classifiers
Model Evaluation

FIGURE 2.2: DataFlow diagram

CHAPTER 3

TECHNOLOGIES USED

3.1 KAFKA

Apache Kafka is a distributed streaming platform that is open-source and made to handle enormous amounts of real-time data processing. It is used to create streaming applications and real-time data pipelines that are able to process and analyze enormous amounts of data in a distributed and fault-tolerant way. Data from various sources, such as databases, sensors, and other systems, can be handled by Kafka, and it can stream that data to various locations, such as analytics systems or data warehouses. Kafka is a popular option for creating real-time data processing pipelines because it offers features including scalable and fault-tolerant message storage, high throughput, and low latency.

3.2 Pyspark

Apache Spark is a speedy and versatile distributed computing technology, and Pyspark is its Python API. Scala, Java, and Python are just a few of the many programming languages supported by Spark, which enables large-scale data processing and analytics. Data scientists may create and run Spark code in Python with the help of Pyspark's user-friendly Spark interface. Large dataset processing, machine learning, and real-time data streaming are all prominent applications for Pyspark.

3.3 Pandas & Matplotlib

Popular open-source Python package Pandas is used for data analysis and manipulation. In addition to data cleansing, transformation, and analysis, it offers data structures for effectively handling and working with massive datasets. Another well-known open-source Python toolkit for data visualization is Matplotlib. It offers a variety of interactive visualizations that may be customized, allowing data scientists to explore and share their data's insights. Pandas and Matplotlib offer a potent collection of Python tools for data analysis and visualization.

3.4 SPARK ML

The machine learning part of Apache Spark, an open-source distributed computing system used for big data processing, is called Spark ML. Data scientists may create and hone machine learning models on huge datasets using Spark ML, which offers a scalable and distributed machine learning framework. In addition to other tasks, Spark ML offers a number of methods for classification, regression, clustering, and collaborative filtering. In addition, it offers tools for feature engineering, model validation, and data preprocessing, making it a complete toolkit for creating machine learning pipelines. To create end-to-end data processing pipelines, Spark ML is frequently used with other Spark components like PySpark and Spark SQL.

CHAPTER 4

IMPLEMENTATION

4.1 Dataset Description

In the fields of intrusion detection and network security, the NSL-KDD dataset is a frequently used benchmark dataset. It is an enhanced version of the intrusion detection system evaluation dataset known as the KDD Cup 1999. The 1998 DARPA Intrusion Detection Evaluation Program dataset used in the KDD Cup 1999 has certain limitations and shortcomings. To solve some of the problems with the original KDD Cup 1999 dataset, the NSL-KDD dataset was created. It offers an environment that is more rigorous and realistic for testing intrusion detection methods. As indicated by the ”NSL” in NSL-KDD, which stands for ”Network Security Laboratory,” it was developed in a controlled network security lab environment.
The network traffic data in the NSL-KDD dataset represents various attacks and everyday activities. The total dataset is about 4GB . The dataset consists of several parts, including a training set and a test set.

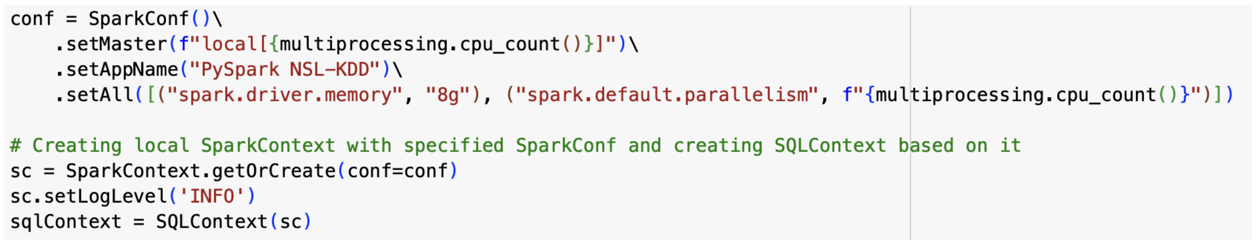
1. Training set: This portion of the dataset contains approximately 4.9 million network connections, which were generated to simulate various types of attacks and normal behavior. Each connection is described by 41 features, including protocol type, service type, source and destination IP addresses,

source and destination port numbers, etc. The connections in the training set are labeled as either an attack or normal.

2. Test set: The test set contains around 311,028 network connections, which are also described by the same 41 features as the training set. However, the labels for the connections in the test set are not initially provided. This portion of the dataset is used to evaluate the performance of intrusion detection systems and machine learning algorithms.

4.2 Importing necessary libraries and data

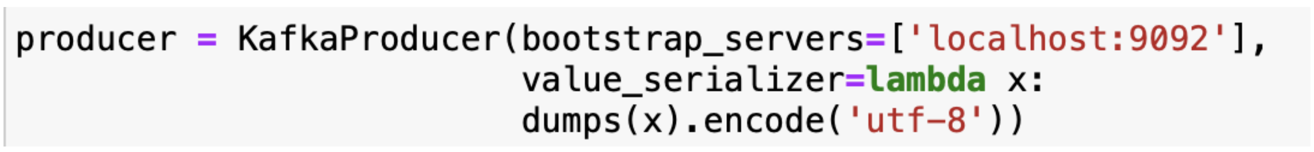
The code begins with importing various libraries such as os, math, itertools, multiprocessing, pandas, numpy, matplotlib.pyplot, and seaborn. These libraries provide functionalities for data manipulation, visualization, and parallel processing.



```
conf = SparkConf()\
    .setMaster(f"local[{multiprocessing.cpu_count()}]")\
    .setAppName("PySpark NSL-KDD")\
    .setAll([("spark.driver.memory", "8g"), ("spark.default.parallelism", f"{multiprocessing.cpu_count()}")])

# Creating local SparkContext with specified SparkConf and creating SQLContext based on it
sc = SparkContext.getOrCreate(conf=conf)
sc.setLogLevel('INFO')
sqlContext = SQLContext(sc)
```

FIGURE 4.1: Creating Spark Context and SQL Context

Kafka acts as a data streaming platform that can handle millions of events per second and store data for longer periods of time, allowing security teams to analyze data and detect potential security threats in real-time. In a typical IDS implementation, network data is collected from various sources, such as network devices or servers, and sent to a Kafka topic using a Kafka producer.



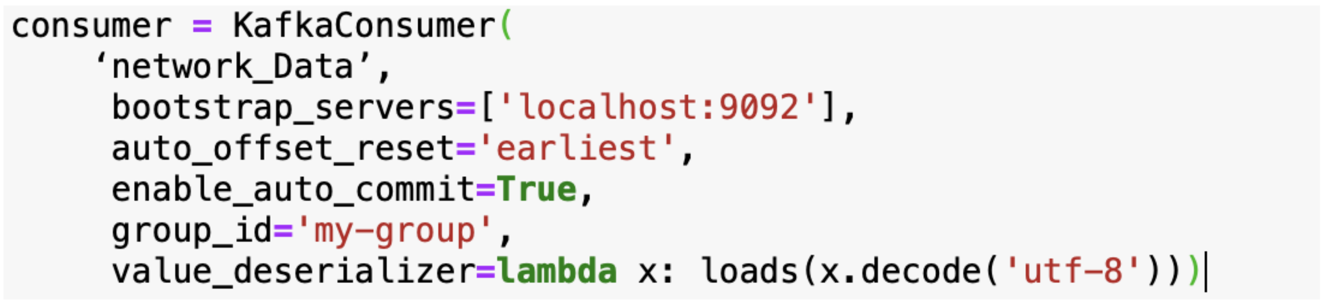
```
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
                         value_serializer=lambda x:
                         dumps(x).encode('utf-8'))
```

FIGURE 4.2: Initialize Kafka Producer

Kafka consumer is then used to subscribe to a Kafka topic, receive messages, and process them in real-time using network intrusion detection algorithms. The consumer is used to receive the data from the Kafka topic and analyze it to detect any malicious traffic patterns.

The consumer application is configured to process the messages, filter the data, apply network intrusion detection algorithms to the incoming network traffic data to detect any malicious patterns or anomalies.

If any malicious traffic patterns or anomalies are detected, the consumer application generates alerts. The alerts can be sent via email, text message, or other notification channels.

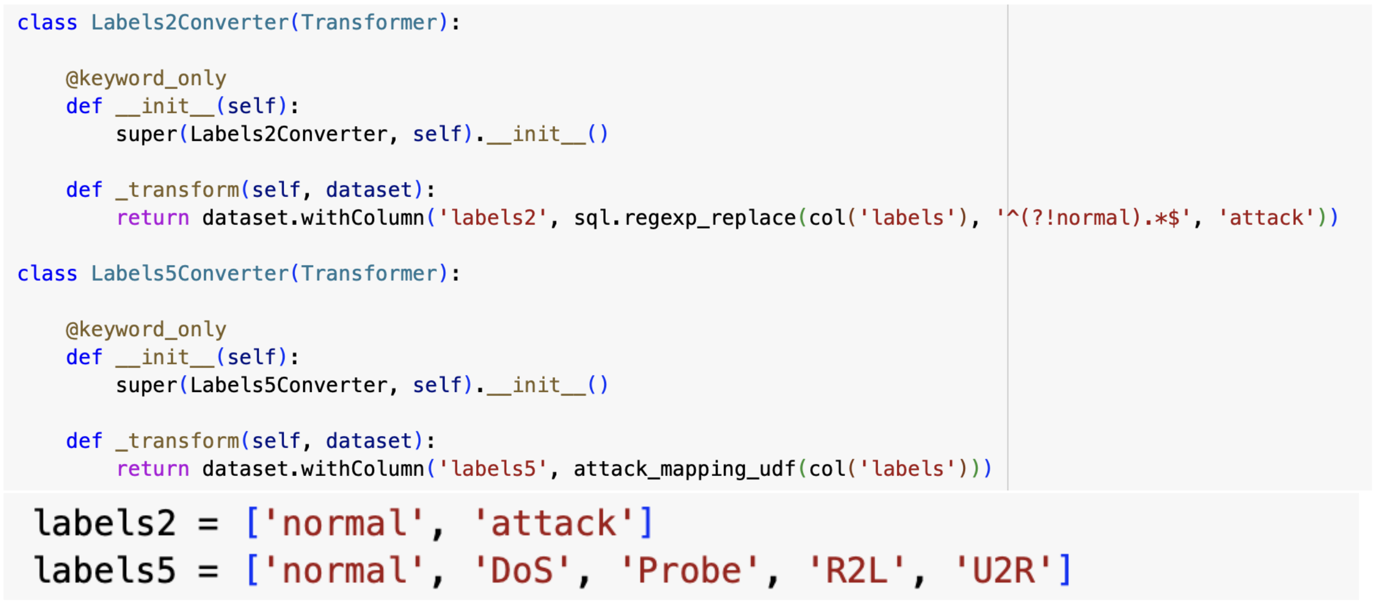


```
consumer = KafkaConsumer(
    ‘network_Data’,
     bootstrap_servers=['localhost:9092'],
     auto_offset_reset='earliest',
     enable_auto_commit=True,
     group_id='my-group',
     value_deserializer=lambda x: loads(x.decode('utf-8')))
```

FIGURE 4.3: Kafka Consumer

4.3 Data Preprocessing

Based on the values in the ”labels” column, the network connections are initially divided into the ”normal” and ”attack” classes. This classification aids in separating authentic network activity from potentially malicious activities. Denial of Service (DoS), Probe, Remote-to-Local (R2L), and User-to-Root (U2R) are the next four major divisions into which the attacks are divided. Based on their traits and tactics, these categories aid in classifying various attacks.
Each type of attack is given an index after categorization in order to speed up data processing and analysis. The dataset also includes an identity (ID) column. Individual network connections are identified more simply via the ID column.



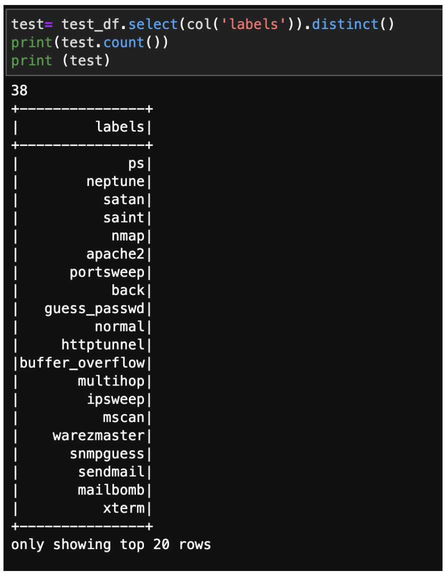
```
class Labels2Converter(Transformer):

    @keyword_only
    def __init__(self):
        super(Labels2Converter, self).__init__()

    def _transform(self, dataset):
        return dataset.withColumn('labels2', sql.regexp_replace(col('labels'), '^(?!normal).*$', 'attack'))

class Labels5Converter(Transformer):

    @keyword_only
    def __init__(self):
        super(Labels5Converter, self).__init__()

    def _transform(self, dataset):
        return dataset.withColumn('labels5', attack_mapping_udf(col('labels')))
```

```
labels2 = ['normal', 'attack']
labels5 = ['normal', 'DoS', 'Probe', 'R2L', 'U2R']
```

FIGURE 4.4: Categorizing training data into attack types and normal connections

4.4 Exploratory data analysis

For gaining a deep understanding of our dataset, its variables, and their relationships we have performed EDA . During the Exploratory Data Analysis (EDA) phase, one of the initial steps in data preparation involves dividing the network connections into two main classes: normal and attack. The division is based on the ’labels’ column, which indicates whether a network connection is classified as normal or corresponds to an attack. This step helps in understanding the distribution and proportion of normal connections versus the different types of attacks present in the dataset. After segregating the attacks from the normal connections, the attacks are further categorized into four main categories from the total of 38 attacks based on their characteristics (23 in training data and additional 15 in test data). These categories are: Denial of Service (DoS) , Probe , Remote-to-Local (R2L , USER TO ROOT) . Figure 4.6



```
test= test_df.select(col('labels')).distinct()
print(test.count())
print (test)
```

```
38
+---------------+
|         labels|
+---------------+
|             ps|
|        neptune|
|          satan|
|          saint|
|           nmap|
|        apache2|
|      portsweep|
|           back|
|   guess_passwd|
|         normal|
|     httptunnel|
|buffer_overflow|
|       multihop|
|        ipsweep|
|          mscan|
|    warezmaster|
|      snmpguess|
|       sendmail|
|       mailbomb|
|          xterm|
+---------------+
only showing top 20 rows
```

FIGURE 4.5: List of all attack types in dataset

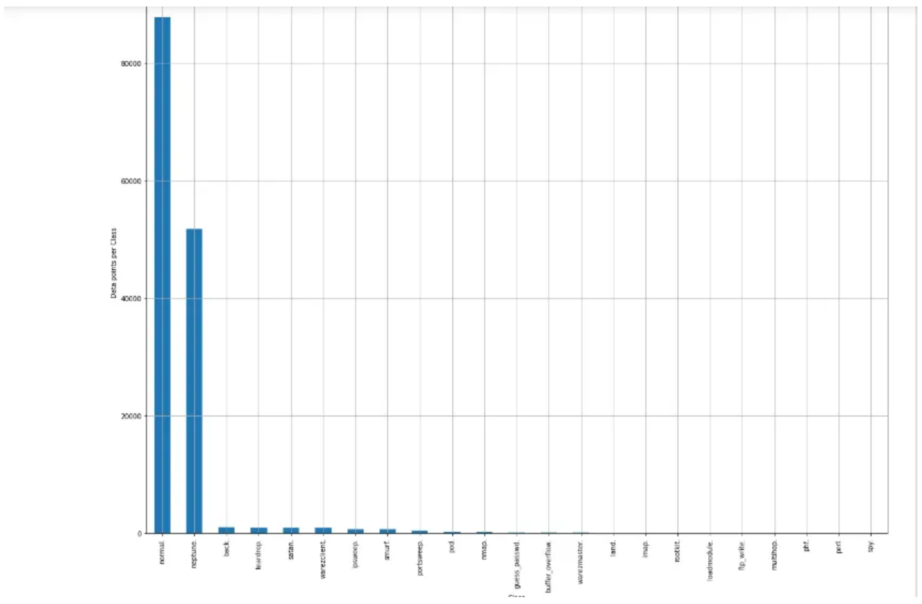


FIGURE 4.6: Distribution of 38 attack types

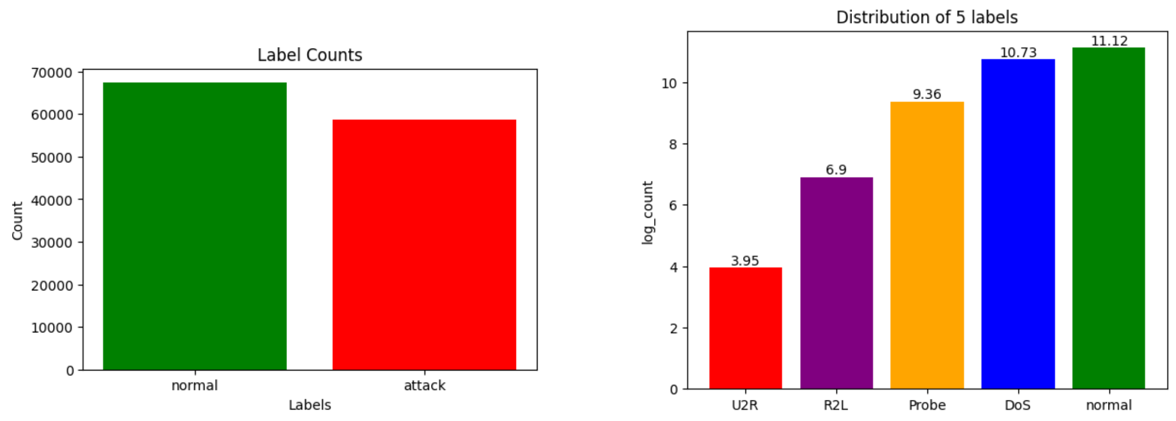


FIGURE 4.7: Training Data distribution for 4 major type of attacks and normal connection

By categorizing the attacks into different types and analyzing their frequencies and impact, we gained a better understanding of their nature. We applied this categorization step to both the training and testing data, and observed that they exhibited similar patterns. For the Labels5 converter, we took the logarithm and visualized the data because U2R attacks were scarce in number and couldn't be effectively represented without this transformation.

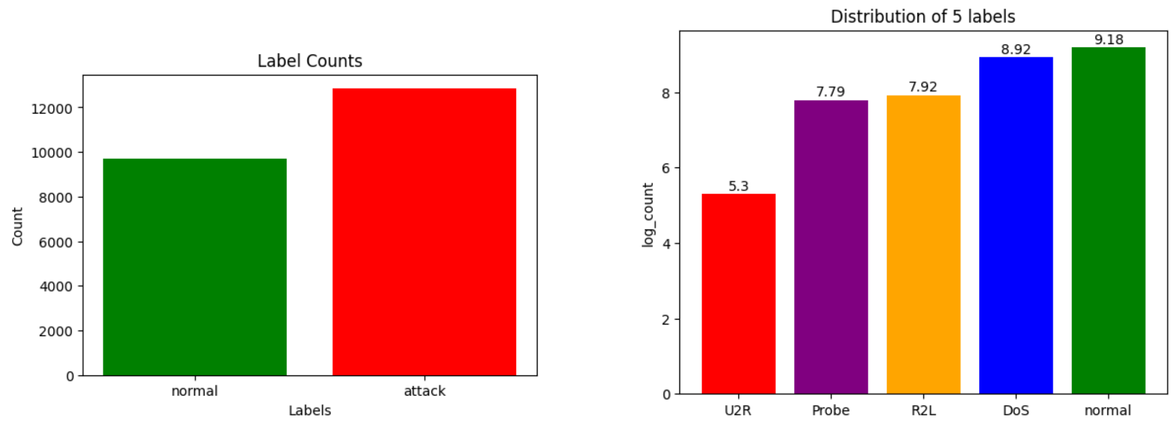


FIGURE 4.8: Testing Data distribution for 4 major type of attacks and normal connection

We examined the data distribution across different protocol types during our analysis, notably TCP, UDP, and ICMP. According to our research, TCP was more vulnerable to attacks than the other protocols. Our analysis also showed that DoS assaults were the most common across all protocols.

This analysis suggests that in order to successfully reduce the risks associated

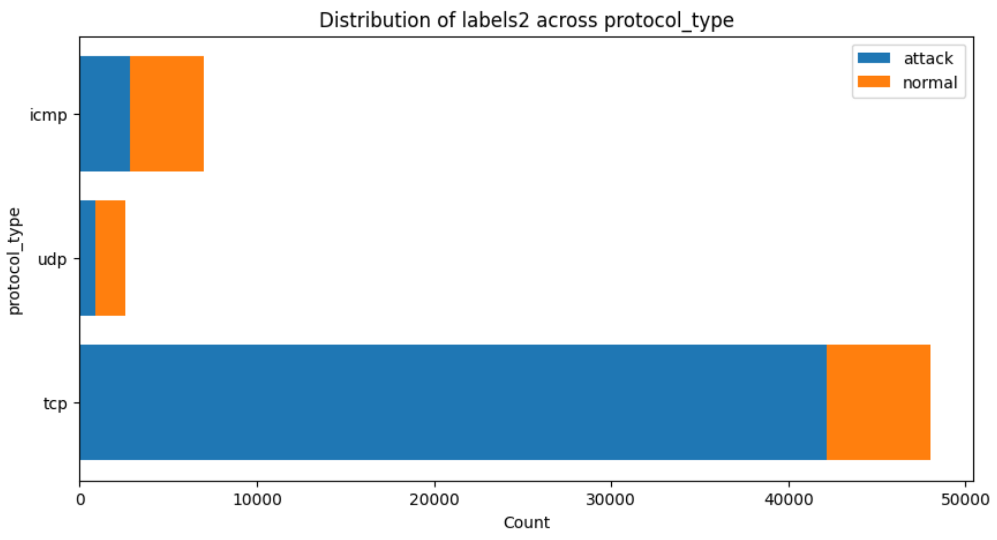


FIGURE 4.9: Distribution of normal and attack connection labels across protocol_type

with assaults, especially DoS attacks, TCP-based networks may require extra security measures. Organizations can prioritize the deployment of strong security solutions to protect TCP-based network infrastructures by becoming aware of this vulnerability.

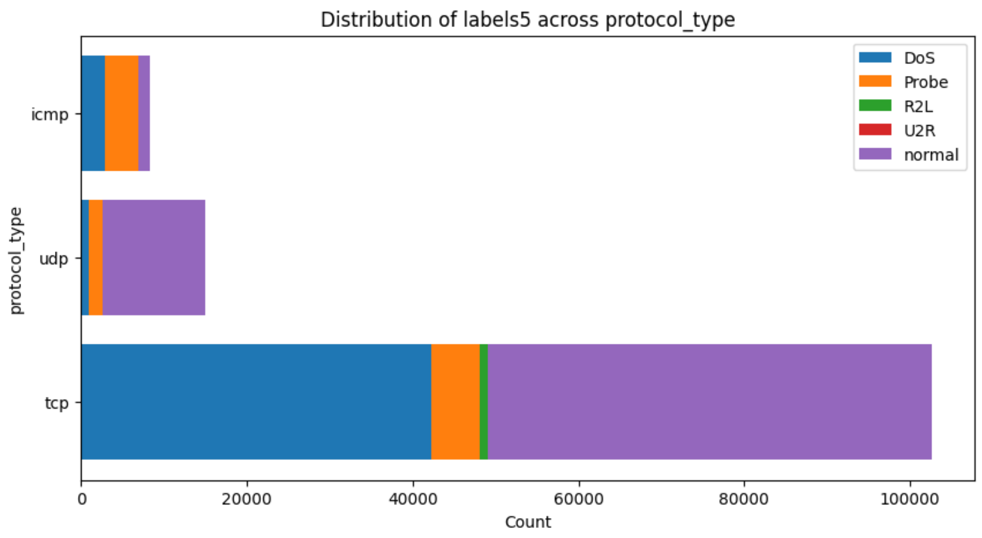


FIGURE 4.10: Distribution of normal and 4 attack types across protocol_type

We made an important network connection discovery. We found that it can be misleading to solely base a network connection's authenticity and success on its status or flags. According to our analysis, connections that appear to be successful based on their status or flags may really still be hiding possible attacks. This study underlines the importance of undertaking thorough analysis of network data in addition to relying merely on connections' success rates. We effectively detect possible assaults that could go undetected otherwise by accounting for extra elements, such as the flags associated with network connections. Adopting a holistic strategy that goes beyond surface-level indications and carefully examines network connections is crucial to ensuring robust network security.

We further examined the distribution of normal connections and attacks across 70 different types of services, including BGP, SSH, Telnet, and others. Our analysis

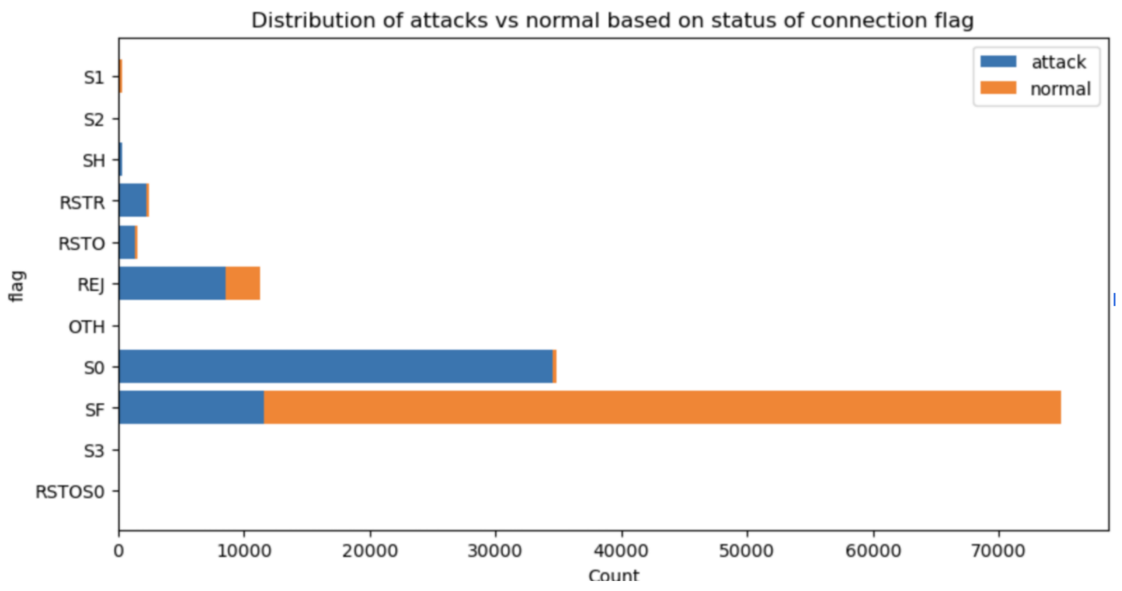


FIGURE 4.11: Distribution of attacks vs normal based on flags

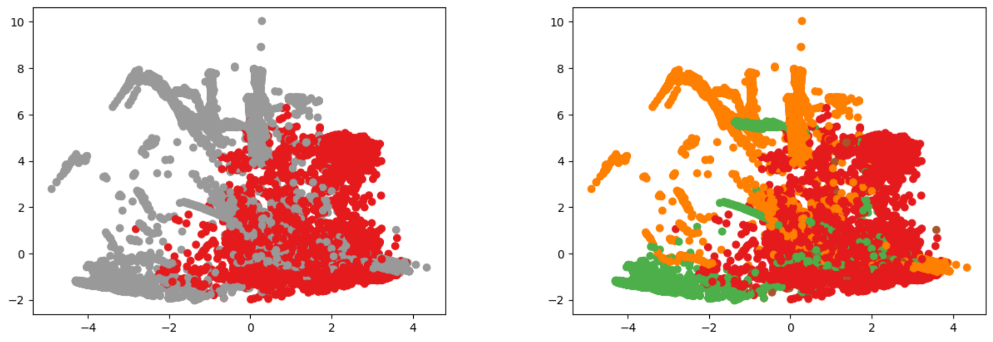
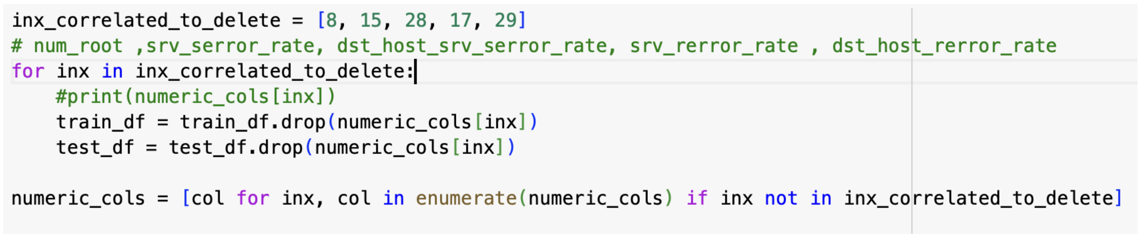


FIGURE 4.12: Distribution of dataset based on attacks vs normal

revealed insights into how attacks were distributed among these services. This analysis helps us gain a deeper understanding of the vulnerabilities associated with different services and enables us to enhance our intrusion detection capabilities for better network security.

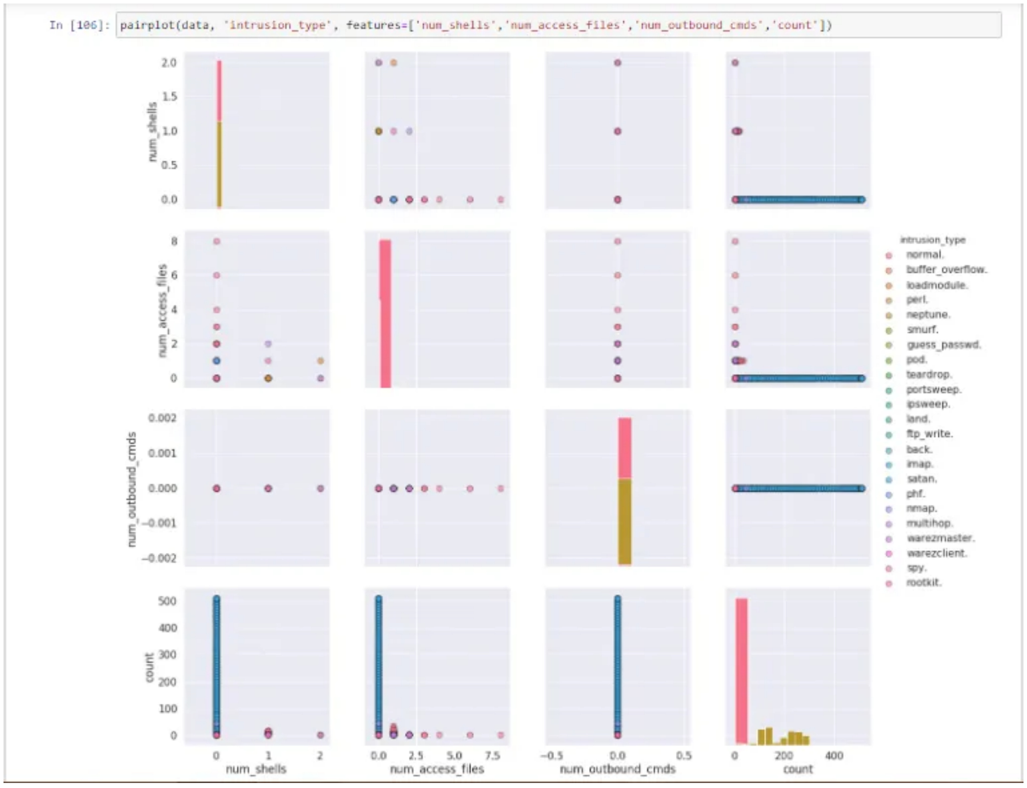
PCA (Principal Component Analysis) algorithm is employed for visualizing the data (Fig 4.8). By using PCA, we can effectively reduce the dimensionality of the data and visualize it in a lower-dimensional space . In the first graph, we plot the 'attack' and 'normal' labels to gain insights into their distribution and separability. The second graph focuses on visualizing the four distinct types of attacks (DoS, Probe, R2L, U2R) in comparison to normal connections.

Pearson correlation measures the linear relationship between two variables.It assumes that the variables are normally distributed and the relationship between them is linear. Spearman correlation measures the monotonic relationship between two variables. It assesses the strength and direction of the relationship without assuming linearity. Spearman correlation is robust to outliers and can handle non-linear relationships.



```
inx_correlated_to_delete = [8, 15, 28, 17, 29]
# num_root ,srv_serror_rate, dst_host_srv_serror_rate, srv_rerror_rate , dst_host_rerror_rate
for inx in inx_correlated_to_delete:
    #print(numeric_cols[inx])
    train_df = train_df.drop(numeric_cols[inx])
    test_df = test_df.drop(numeric_cols[inx])

numeric_cols = [col for inx, col in enumerate(numeric_cols) if inx not in inx_correlated_to_delete]
```

FIGURE 4.13: Identifying and deleteing highly corelated features



```
In [106]: pairplot(data, 'intrusion_type', features=['num_shells','num_access_files','num_outbound_cmds','count'])
```

FIGURE 4.14: Pairplot

The provided function selects four features namely num_shells , num_access_files, num_outbound_cmds , count from our dataset and generates a grid of 16 bivariate plots. Each plot in the grid shows the relationship between two different features such as normal vs buffer_overflow, resulting in a total of 16 plots with various combinations of the selected features.

Finally we have constructed a heap matrix to find highly correlated features in the dataset for improving the accuracy . By calculating both Pearson and Spearman correlations, we gained insights into different aspects of the relationship between the numeric columns. IT provides a more comprehensive understanding of the relationships within the data, capturing both linear and non-linear associations.

Since correlation coefficients are unitless values, the heatmap does not have a

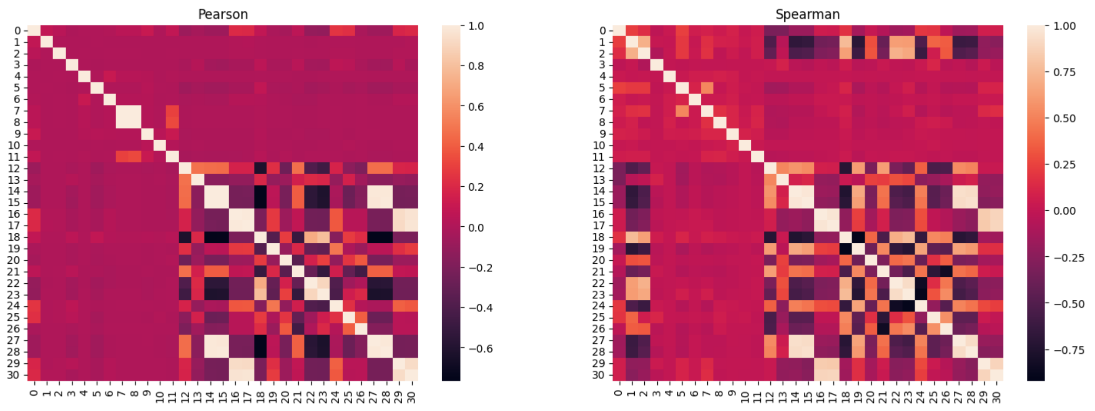
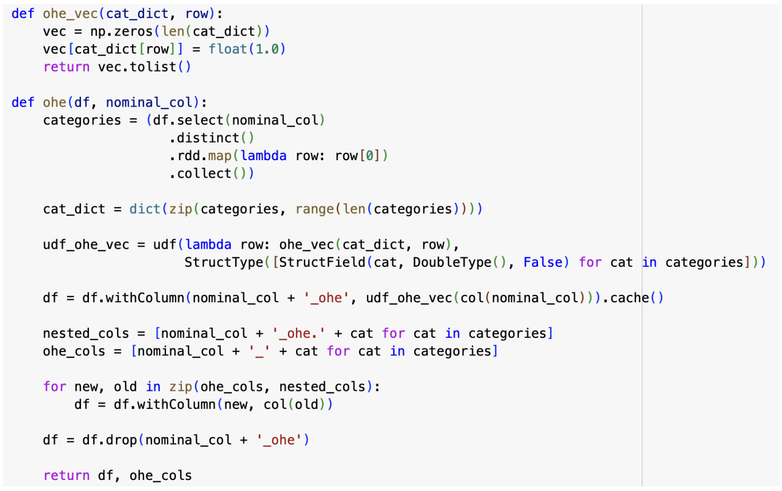


FIGURE 4.15: Pearson Spearman Correlation

specific unit. Instead, it provides a visual representation of the correlation strength, where higher values indicate a stronger positive or negative correlation, and lower values indicate a weaker or no correlation.

4.5 One Hot Encoding for categorical variables

In order to handle categorical variables, we have implemented a custom function for One Hot Encoding (OHE). One Hot Encoding is a technique used to transform categorical variables into a binary vector representation that can be easily interpreted by machine learning algorithms. Our custom function takes a categorical variable as input and returns a binary matrix representation using OHE. This enables us to incorporate categorical information into our analysis and improve the predictive power of our models.



```python
def ohe_vec(cat_dict, row):
    vec = np.zeros(len(cat_dict))
    vec[cat_dict[row]] = float(1.0)
    return vec.tolist()

def ohe(df, nominal_col):
    categories = (df.select(nominal_col)
                    .distinct()
                    .rdd.map(lambda row: row[0])
                    .collect())

    cat_dict = dict(zip(categories, range(len(categories))))

    udf_ohe_vec = udf(lambda row: ohe_vec(cat_dict, row),
                      StructType([StructField(cat, DoubleType(), False) for cat in categories]))

    df = df.withColumn(nominal_col + '_ohe', udf_ohe_vec(col(nominal_col))).cache()

    nested_cols = [nominal_col + '_ohe.' + cat for cat in categories]
    ohe_cols = [nominal_col + '_' + cat for cat in categories]

    for new, old in zip(ohe_cols, nested_cols):
        df = df.withColumn(new, col(old))

    df = df.drop(nominal_col + '_ohe')

    return df, ohe_cols
```

FIGURE 4.16: One hot encoding

4.6 Feature Engineering

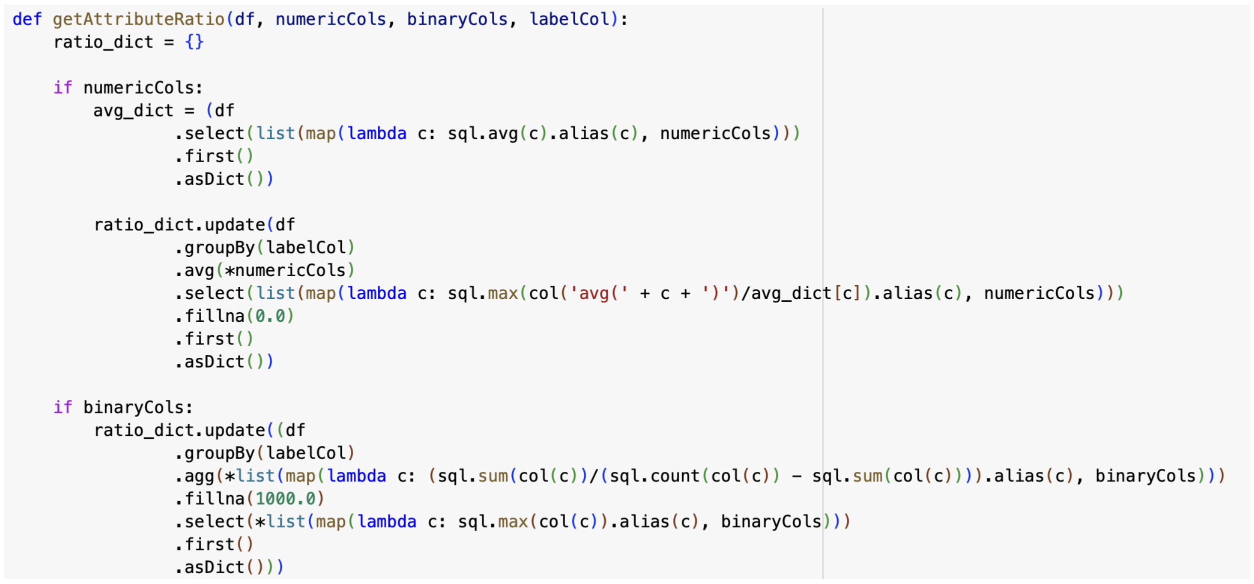
The proposed feature selection approach incorporates the AR (Average Ratio) metric, which quantifies the discriminative power of each feature by assessing the ratio between the average between-class distance and the average within-class distance.The actions we take to put this strategy into practice are as follows:

1. Preprocessing: Preprocessing the dataset in the first step entails removing redundant and unnecessary characteristics. The use of common preprocessing methods like normalization, discretization, and encoding helps to ensure the accuracy of the data and improve the efficiency of subsequent analysis.

2. AR Computation: The AR value is calculated for each feature in the dataset using the AR metric algorithm. In this calculation, the distances between examples of the same class as well as between instances of different classes are measured. We calculate an AR value for each feature, which represents the strength of the feature's discrimination, by analyzing these distances.

3. Selection and Ranking: The features are ranked using the obtained AR values in descending order. The top-k features with the greatest AR values can be found using this ranking, which highlights their better discriminatory power. The final group of features to be used for intrusion detection is chosen from among these top-ranked features.

This method allows us to evaluate each feature's ability to discriminate and pinpoint the subset that will provide the most useful information for intrusion detection. The chosen features, which have higher AR values, are anticipated to considerably improve the performance by helping to accurately identify and classify probable incursions.



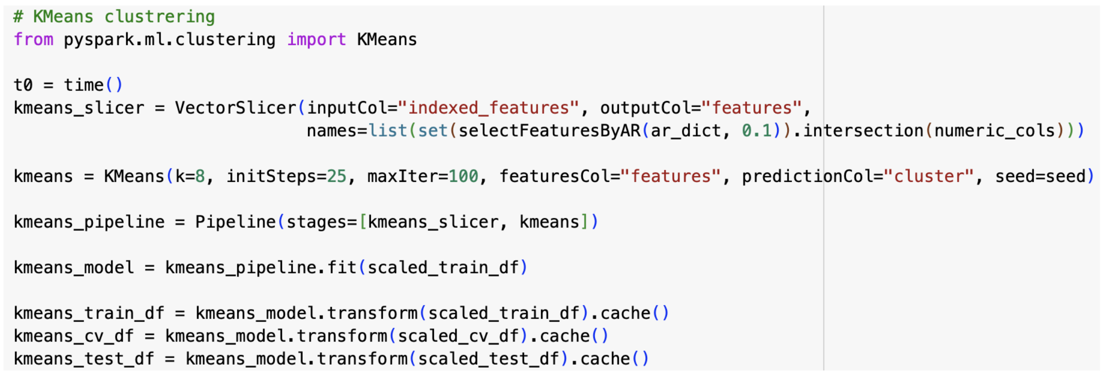
```
def getAttributeRatio(df, numericCols, binaryCols, labelCol):
    ratio_dict = {}

    if numericCols:
        avg_dict = (df
                .select(list(map(lambda c: sql.avg(c).alias(c), numericCols)))
                .first()
                .asDict())

        ratio_dict.update(df
                .groupBy(labelCol)
                .avg(*numericCols)
                .select(list(map(lambda c: sql.max(col('avg(' + c + ')')/avg_dict[c]).alias(c), numericCols)))
                .fillna(0.0)
                .first()
                .asDict())

    if binaryCols:
        ratio_dict.update((df
                .groupBy(labelCol)
                .agg(*list(map(lambda c: (sql.sum(col(c))/(sql.count(col(c)) - sql.sum(col(c)))).alias(c), binaryCols)))
                .fillna(1000.0)
                .select(*list(map(lambda c: sql.max(col(c)).alias(c), binaryCols)))
                .first()
                .asDict()))
```

FIGURE 4.17: Finding Attribute ratio for all features

4.7 KMeans clustering with Random Forest Classifiers

Due to their capacity to accurately identify complicated patterns and anomalies in network data, machine learning techniques, such as KMeans clustering and Random Forest classifiers, are widely used in network intrusion detection. These methods are excellent at identifying new or undiscovered attack types by extracting pertinent information from unprocessed network data. KMeans clustering and other unsupervised learning techniques reveal underlying data structures, while Random Forest classifiers accurately categorize data by learning from labeled examples. KMeans clustering scale effectively with big datasets, adapt to changing attack patterns, and minimize false positives by differentiating between legitimate network activity and malicious activity. Utilizing these methods, businesses may improve network security, identify potential risks instantly, safeguard sensitive data from unauthorized access, and compromise.

To streamline the execution, a pipeline is constructed to sequentially execute the



```python
# KMeans clustrering
from pyspark.ml.clustering import KMeans

t0 = time()
kmeans_slicer = VectorSlicer(inputCol="indexed_features", outputCol="features",
                             names=list(set(selectFeaturesByAR(ar_dict, 0.1)).intersection(numeric_cols)))

kmeans = KMeans(k=8, initSteps=25, maxIter=100, featuresCol="features", predictionCol="cluster", seed=seed)

kmeans_pipeline = Pipeline(stages=[kmeans_slicer, kmeans])

kmeans_model = kmeans_pipeline.fit(scaled_train_df)

kmeans_train_df = kmeans_model.transform(scaled_train_df).cache()
kmeans_cv_df = kmeans_model.transform(scaled_cv_df).cache()
kmeans_test_df = kmeans_model.transform(scaled_test_df).cache()
```

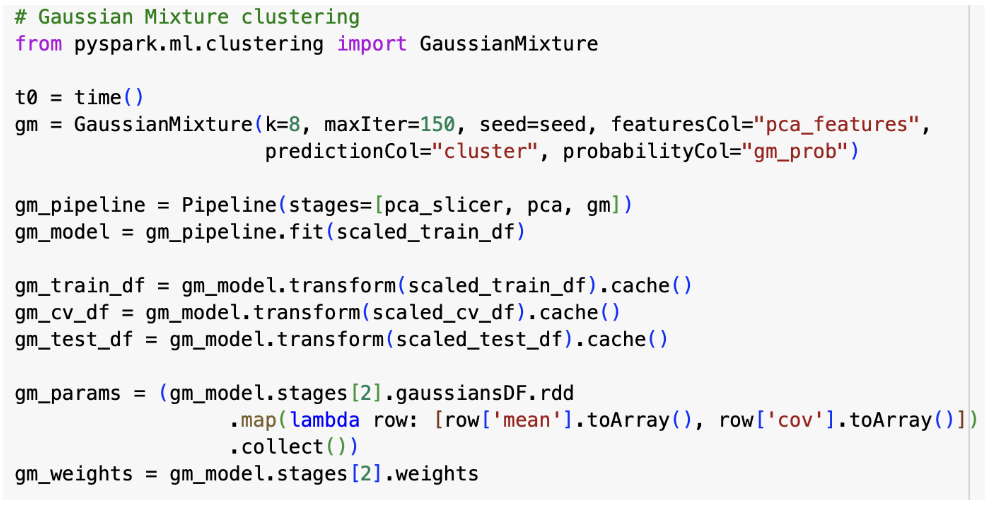
FIGURE 4.18: Defining the K Means clustering model

VectorSlicer and KMeans stages. This ensures the seamless application of feature selection and clustering on the data.The pipeline is then fitted to the training dataset, resulting in a trained KMeans model. This model captures the underlying patterns and similarities in the data, enabling the assignment of instances to their respective clusters.Subsequently, the trained KMeans model is applied to the training, cross-validation, and test datasets using the transform method. This process generates transformed dataframes with an additional ”cluster” column, indicating the assigned cluster for each instance. To optimize computational efficiency, the transformed data frames are cached, allowing for swift access and reuse in subsequent analyses.

4.8 Gaussian Mixture clustering with Random Forest Classifiers

Gaussian Mixture Models (GMM) are preferred in intrusion detection due to their ability to detect anomalies without labeled data, interpret results through distinct clusters or probabilistic distributions, handle large and high-dimensional datasets efficiently, and adapt to different intrusion scenarios by incorporating feature engineering techniques. They offer flexibility, scalability, and the potential to capture complex attack patterns.

By combining feature selection, Principal Component Analysis (PCA), and



```
# Gaussian Mixture clustering
from pyspark.ml.clustering import GaussianMixture

t0 = time()
gm = GaussianMixture(k=8, maxIter=150, seed=seed, featuresCol="pca_features",
                     predictionCol="cluster", probabilityCol="gm_prob")

gm_pipeline = Pipeline(stages=[pca_slicer, pca, gm])
gm_model = gm_pipeline.fit(scaled_train_df)

gm_train_df = gm_model.transform(scaled_train_df).cache()
gm_cv_df = gm_model.transform(scaled_cv_df).cache()
gm_test_df = gm_model.transform(scaled_test_df).cache()

gm_params = (gm_model.stages[2].gaussiansDF.rdd
                  .map(lambda row: [row['mean'].toArray(), row['cov'].toArray()])
                  .collect())
gm_weights = gm_model.stages[2].weights
```

FIGURE 4.19: Defining the Guassian Mixture clustering model

GMM clustering, the approach aims to identify patterns and clusters within the dataset. The GMM algorithm is configured with specific parameters, and a pipeline is created to execute the feature selection, PCA, and GMM stages in sequence. The pipeline is then fitted to the training data, resulting in a trained GMM model. The model is applied to the training, cross-validation, and test datasets, generating transformed datasets with assigned clusters and probabilities.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Peformance Metrics

To measure how “accurate” or superior your classifiers are at predicting the class label of tuples. The classifier evaluation measures are accuracy (also known as recognition rate), sensitivity (or recall), specificity, precision, F1.

- **Accuracy :** It represents the percentage of images in each class that are correctly classified.

$$Accuracy = (TP + FN)/(TP + TN + FP + FN)$$

- **F1 Score :** It aggregates the precision and recall of a classifier into a single metric by taking harmonic mean.

$$F1Score = (precision * recall)/(precision + recall)$$

- **Precision :** It is the ratio of true positive to total positives.

$$Precision = TP/(TP + FP)$$

- **Recall :** It represents the model’s ability to correctly predict positives from actual positives.

$$Recall = TP/(TP + FN)$$

5.2 Model Results:

Model	**Training Accuracy**	**Testing Accuracy**	**False Alarm Rate**	**F1 Score**
K-Means Clustering	90.09%	99.83%	0.009	0.99
Guassian Mixture Clustering	99.58%	89.18%	0.12	0.90

TABLE 5.1: Result Comparison of Classification Models

The performance scores of all the models are similar on both the train and test data, indicating that they are not overfitting. This concludes the fascinating case study that involved using the KDD dataset and applying various machine learning techniques to construct a Network Intrusion Detection System. The system can accurately differentiate between good and bad connections with high precision while minimizing the number of false positives.

CHAPTER 6

Big Data Analytics in Intrusion Detection System:

Big Data encompasses a vast amount of structured and unstructured data from diverse sources, which cannot be efficiently processed using traditional applications. This needs real-time processing and presents difficulties in the interpretation of the data. Big Data is important because of its sheer mass as well as the useful information that organizations can derive from it. It not only makes it possible to find new insights, but it also offers a thorough comprehension of underlying patterns. However, due to redundant qualities and records, intrusion detection in Big Data analytics is challenging. Dimension reduction is therefore essential for improving the effectiveness of classifiers by increasing performance, efficiency, and computational complexity for anomaly detection. Big Data analytics brings significant benefits to network intrusion detection systems by enabling scalable and real-time analysis of network data, applying advanced analytics techniques, detecting anomalies, integrating diverse data sources, facilitating rapid incident response, and providing scalable storage and processing capabilities. These capabilities empower organizations to effectively detect, prevent, and mitigate network intrusions and enhance overall network security.

CHAPTER 7

Limitations

When using Spark Streaming and an external database like Apache Ignite, there are constraints related to intrusion detection systems (IDS) in the big data context.

1. Limitation of Spark Streaming: One drawback of Spark Streaming is that, when aggregate functions are utilized, multi-stream joins are not supported. This means that using the Spark Streaming framework to carry out complicated actions that require aggregating data from numerous streams is difficult. For successful intrusion detection, multi-stream joins are necessary for identifying correlations and trends among several data sources. The analytical capabilities of IDS in big data contexts are constrained by the inability to make such joins using aggregate functions.

2. Utilizing External Database: Using an external database like Apache Ignite to store the processed and transformed data introduces some limitations. While Apache Ignite offers scalability and fast in-memory storage, it adds complexity to the overall system architecture. Integrating Spark Streaming with Apache Ignite requires additional development effort and may introduce additional points of failure or performance bottlenecks. Furthermore, maintaining consistency and synchronization between the streaming data in Spark Streaming and the data stored in Apache Ignite can be a challenge.

CHAPTER 8

Future Work

Ensembling approaches can be effectively utilized to enhance the detection rate in intrusion detection systems. The system can benefit from the various perspectives and advantages of each unique intrusion detection algorithm or technique by integrating several of them, which enhances detection performance overall. To further improve the evaluation of our intrusion detection system it is advised to test the effectiveness of various intrusion detection algorithms or approaches on a test dataset with a schema comparable to our own data in order One potential benchmark dataset for assessing the performance of our system is the CSE-CIC-IDS2018 dataset, which is often utilized in the research community. This evaluation procedure enables a thorough comparison of several algorithms and methodologies, allowing us to identify the most efficient methods for our specific intrusion detection needs.

Furthermore, it is advised to create an end-to-end pipeline that incorporates the Zeek monitor logs with a data engineering pipeline in order to obtain higher resolution and real-time monitoring. By delivering stream queries to a Kafka message broker, this pipeline should be created to process and analyze streaming data. We can improve the resolution and accuracy of our intrusion detection system by utilizing Kafka's capabilities and bringing effective data engineering approaches, such as data preprocessing, feature engineering, and real-time analysis, into practice. This real-time pipeline enables us to detect and respond to potential intrusions promptly, ensuring a proactive approach to network security.

CHAPTER 9

Conclusion

Through exploratory data analysis, we gained insights into the distribution of network connections, protocol types, and the prevalence of different attack categories. We also performed data preprocessing, including feature engineering and encoding, to prepare the dataset for further analysis.

We investigated a number of machine learning algorithms, such as ensemble approaches like Random Forest Classifiers, clustering with KMeans and Gaussian Mixture Models. These techniques allowed us to detect and classify network intrusions, leveraging the power of big data analytics. Based on our evaluation criteria, which included accuracy, precision, recall, and F1 score, we found that KMeans performed better than Gaussian Mixture Clustering overall in terms of accuracy. This suggests that KMeans performed better at precisely identifying instances of legitimate or malicious network connections. The high accuracy that KMeans was able to attain shows that it can successfully segregate and find patterns in the dataset. In addition, we talked about how feature selection and dimensionality reduction are crucial for enhancing the effectiveness and performance of intrusion detection systems. We can improve the precision and scalability of the detection process by locating pertinent features and simplifying the dataset.

REFERENCES

1. Dahiya, P. and Srivastava, D.K. (2018) ‘Network intrusion detection in big dataset using Spark’, Procedia Computer Science, 132, pp. 253–262. doi:10.1016/j.procs.2018.05.169.

2. Sulaiman, N.S. (2020) ‘Big Data Analytic of Intrusion Detection System’, International Journal of Advanced Trends in Computer Science and Engineering, 9(1.4), pp. 450–453. doi:10.30534/ijatcse/2020/6391.42020.

3. Al-Jarrah, O.Y. et al. (2014b) ‘Machine-learning-based feature selection techniques for large-scale network intrusion detection’, 2014 IEEE 34th International Conference on Distributed Computing Systems Workshops [Preprint]. doi:10.1109/icdcsw.2014.14.

4. ‘Feature selection using attribute ratio in NSL-KDD Data’ (2014) International Conference Data Mining, Civil and Mechanical Engineering (ICDMCME’2014), Feb 4-5, 2014 Bali (Indonesia) [Preprint]. doi:10.15242/iie.e0214081.

5. thinline72 (no date) Thinline72/NSL-KDD: Pyspark solution to the NSL-KDD dataset: Https://www.unb.ca/cic/datasets/nsl.html, GitHub. Available at: https://github.com/thinline72/nsl-kdd (Accessed: 12 May 2023).

6. Singh, S. (2020) Building an intrusion detection model using KDD Cup’99 Dataset, Medium. Available at: https://medium.com/analytics-vidhya/building-an-intrusion-detection-model-using-kdd-cup99-dataset-fb4cba4189ed (Accessed: 12 May 2023).

7. ‘Cyber-security: The use of Big Data Analytic model for network intrusion detection classification’ (2019) Computer Engineering and Intelligent Systems [Preprint]. doi:10.7176/ceis/10-7-02.

8. ‘A review of network intrusion detection in the Big Data Era: Challenges and future trends’ (2015) Networking for Big Data, pp. 217–236. doi:10.1201/b18772-17.

9. Umair, M.B. et al. (2022) ‘A network intrusion detection system using hybrid multilayer deep learning model’, Big Data [Preprint]. doi:10.1089/big.2021.0268.

10. Bagui, S. and Li, K. (2021) ‘Resampling imbalanced data for network intrusion detection datasets’, Journal of Big Data, 8(1). doi:10.1186/s40537-020-00390-x.