

Machine Learning-Based Intrusion Detection System for Network Security

Produced by : Mouhssine ANNOURI

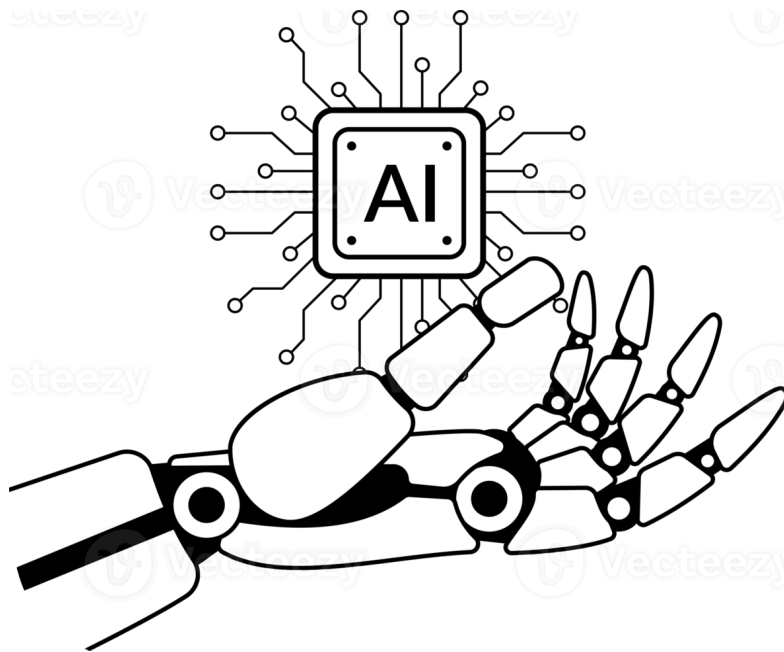


Table of Contents

Table of Contents.....	2
Introduction.....	3
Problem Statement.....	3
Data collection.....	4
Downloading files.....	6
Data preprocessing.....	6
Adding Columns.....	6
Adding Attack Types.....	7
Categorical Features.....	7
Mapping Categorical Values.....	7
Data Correlation.....	8
Modeling.....	8
Data Preparation.....	8
Model Training.....	8
Evaluation.....	9
Testing.....	9

Introduction

With the ever-increasing reliance on interconnected systems, cyber threats pose a significant challenge to network security. Intrusion Detection Systems (IDS) play a critical role in safeguarding networks by identifying malicious activity and potential security breaches. Traditional rule-based IDS approaches struggle to adapt to the evolving landscape of cyberattacks.

This project proposes the development of a machine learning-based IDS capable of effectively identifying and flagging suspicious network traffic. By leveraging the power of machine learning algorithms, the system will be able to learn and adapt to new attack patterns, offering a more robust defense against emerging threats.

This document outlines the project's objective, which is to develop and implement a machine learning-based IDS. It will detail the chosen algorithms, data preparation methods, and evaluation metrics used to assess the system's effectiveness in detecting potential security breaches within network traffic.

Problem Statement

Our objective is to develop a system capable of analyzing network traffic and identifying malicious activity.

This system, known as a network intrusion detector, will function as a predictive model. It will learn to differentiate between normal network connections and suspicious ones, which may indicate intrusion attempts or cyberattacks. The goal is to create a model that classifies network connections as either normal or belonging to a specific type of attack (denial-of-service, unauthorized access, etc.).

Attackers can target computer systems in four main ways:

Denial-of-Service (DoS): These attacks aim to overwhelm a system with traffic, making it unavailable to legitimate users. An example is a SYN flood attack.

Remote to Local (R2L): These attacks focus on gaining unauthorized access to a system from a remote location. An example is using password guessing techniques.

User to Root (U2R): These attacks exploit vulnerabilities to escalate privileges from a regular user account to a powerful administrator account (often called "root"). Buffer overflow attacks are commonly used for this purpose.

Probing: These attacks involve gathering information about a system, such as identifying open ports for vulnerabilities. Port scanning is a typical probing technique.

Data collection

To achieve our goals, we leveraged the [KDD Cup 1999](#) dataset

The data is organized into several files, including :

- **kddcup.names** : A list of features.
- **kddcup.data.gz** : The full data set
- **kddcup.data_10_percent.gz** : A 10% subset.
- **kddcup.newtestdata_10_percent_unlabeled.gz**
- **kddcup.testdata.unlabeled.gz**
- **kddcup.testdata.unlabeled_10_percent.gz**
- **corrected.gz** : Test data with corrected labels.
- **training_attack_types** : A list of intrusion types.
- **typo-correction.txt** : A brief note on a typo in the data set that has been corrected

The following features are included in the dataset:

<i>feature name</i>	<i>description</i>	<i>type</i>
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete

wrong_fragment	number of ``wrong" fragments	continuous
urgent	number of urgent packets	continuous

Table 1: Basic features of individual TCP connections.

<i>feature name</i>	<i>description</i>	<i>type</i>
hot	number of ``hot" indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised" conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root" command attempted; 0 otherwise	discrete
num_root	number of ``root" accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot" list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest" login; 0 otherwise	discrete

Table 2: Content features within a connection suggested by domain knowledge.

<i>feature name</i>	<i>description</i>	<i>type</i>
count	number of connections to the same host as the current connection in the past two seconds	continuous
	<i>Note: The following features refer to these same-host connections.</i>	
error_rate	% of connections that have ``SYN" errors	continuous

error_rate	% of connections that have ``REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	<i>Note: The following features refer to these same-service connections.</i>	
srv_serror_rate	% of connections that have ``SYN" errors	continuous
srv_rerror_rate	% of connections that have ``REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Table 3: Traffic features computed using a two-second time window.

Downloading files

We downloaded the previously mentioned files.

```
[6] # downloading files
import wget
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.names'
wget.download(url, 'kddcup.names')
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.newtestdata_10_percent_unlabeled.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.testdata_unlabeled.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/kddcup.testdata_unlabeled_10_percent.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/corrected.gz'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types'
wget.download(url)
url = 'http://kdd.ics.uci.edu/databases/kddcup99/typo-correction.txt'
wget.download(url)
```

Now we can proceed to data preprocessing

Data preprocessing

Adding Columns

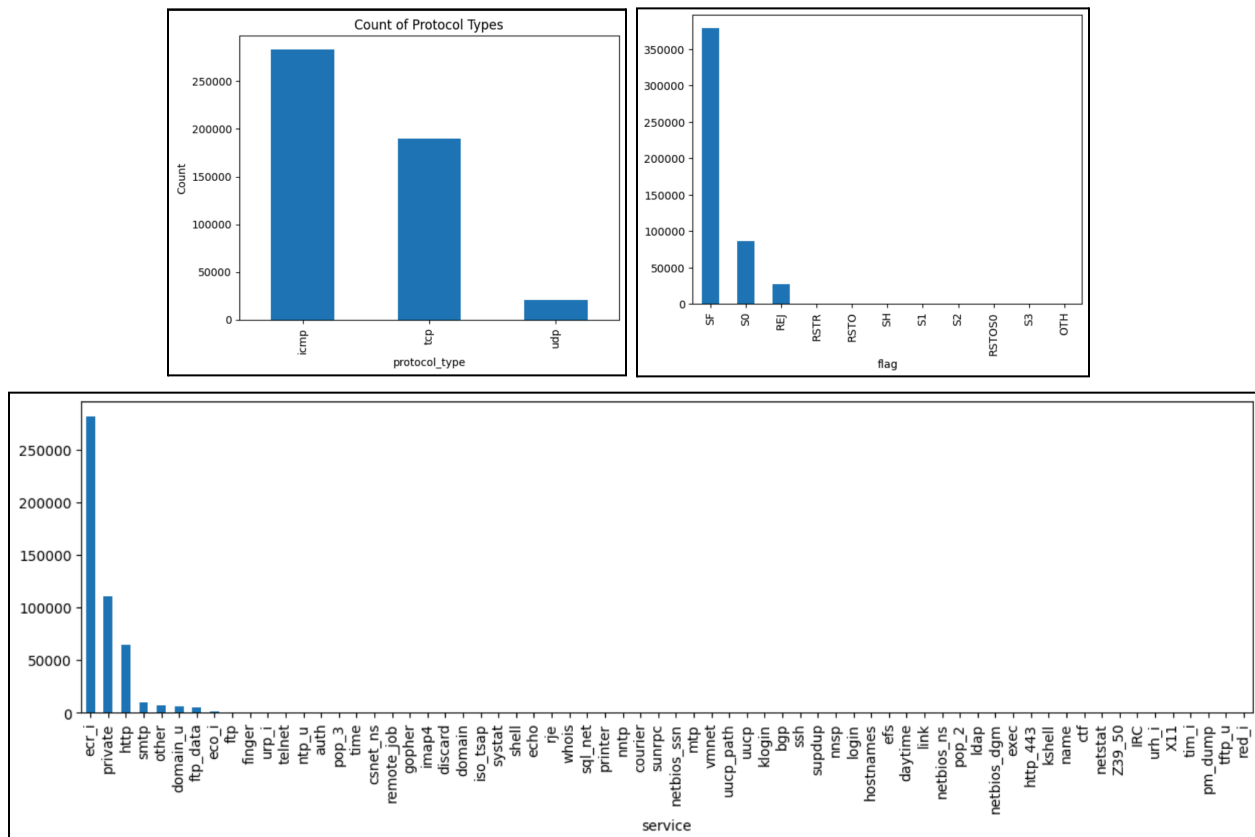
First, we created a variable called `columns` that includes all the feature names from the `kddcup.names` file, along with the target column.

Adding Attack Types

Next, we defined a variable named `attacks_types` that encompasses all the intrusion types listed in the `training_attack_types` file. We then loaded the data into a dataframe using the predefined columns and target variable.

Categorical Features

For the categorical features, we identified three columns: `flag`, `protocol_type`, and `service`. The possible values for these columns are:



Mapping Categorical Values

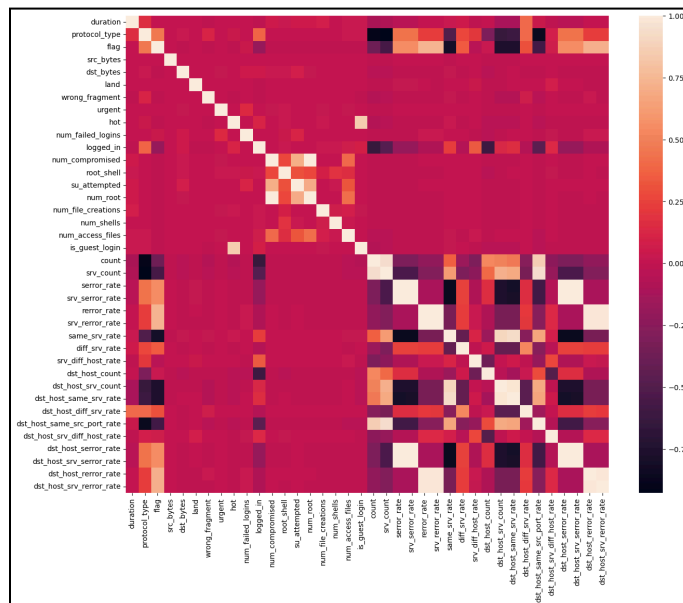
Next, we mapped each value within the categorical columns (`flag`, `protocol_type`, and `service`) to a numeric value starting from 0.

```
#flag feature mapping
fmap = {'SF':0, 'SO':1, 'REJ':2, 'RSTR':3, 'RSTO':4, 'SH':5, 'S1':6, 'S2':7, 'RSTOS0':8, 'S3':9, 'OTH':10}
df['flag'] = df['flag'].map(fmap)
```

```
pmap = {'icmp':0, 'tcp':1, 'udp':2}
df['protocol_type'] = df['protocol_type'].map(pmap)
```

Data Correlation

We then analyzed the correlation to gain an overview of the relationships between the features. This helped us identify which features are more significant and which can be disregarded in the analysis. Below is the heatmap of the correlation:



Based on the results of the heatmap, we identified and removed 8 features that were highly correlated with others. These features will be ignored in the subsequent analysis to improve model performance and reduce redundancy.

Modeling

Data Preparation

1. **Normalization:** We normalized the continuous data to ensure they are on a comparable scale.
2. **Data Splitting:** The data was split into training (70%) and testing (30%) sets.

```
df = df.drop(['target'], axis=1)
print(df.shape)

# Target variable and train set
Y = df[['Attack Type']]
X = df.drop(['Attack Type'], axis=1)

sc = MinMaxScaler()
X = sc.fit_transform(X)

# Split test and train data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
print(X_train.shape, X_test.shape)
print(Y_train.shape, Y_test.shape)
```

✓ 0.5s

Model Training

We used the following models for training:

- Decision Tree
- Logistic Regression
- Random Forest
- Gradient Boosting Classifier
- Support Vector Machine

Evaluation

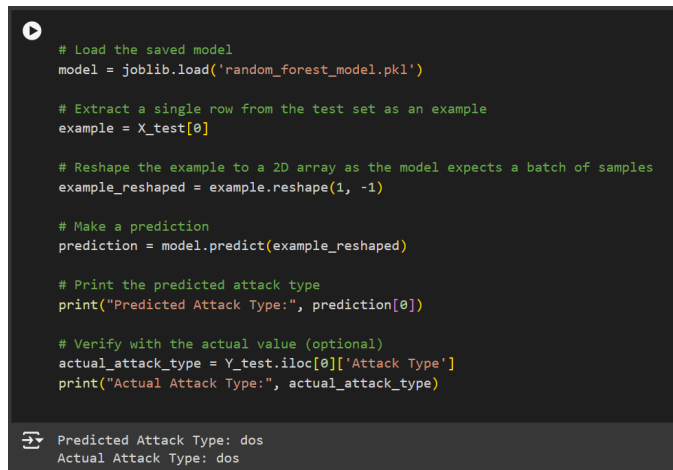
To evaluate the performance of the models, we used the following metrics:

- Accuracy Score
- Precision Score
- F1 Score
- Recall

Among these models, the Random Forest model achieved the highest scores, so we chose to save it.

Testing

After selecting the Random Forest model, we saved and reloaded it for testing. The model worked perfectly, as shown in the screenshot below:



```
# Load the saved model
model = joblib.load('random_forest_model.pkl')

# Extract a single row from the test set as an example
example = X_test[0]

# Reshape the example to a 2D array as the model expects a batch of samples
example_resaped = example.reshape(1, -1)

# Make a prediction
prediction = model.predict(example_resaped)

# Print the predicted attack type
print("Predicted Attack Type:", prediction[0])

# Verify with the actual value (optional)
actual_attack_type = Y_test.iloc[0]['Attack Type']
print("Actual Attack Type:", actual_attack_type)
```

Predicted Attack Type: dos
Actual Attack Type: dos