Project Report On

“KDD99 –Intrusion Detection ”

Submitted By :

Arun Joshva Stephenson

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my SME as well as the "IntechHub" team for allowing me to work on the "KDD99 Intrusion Detection System - Classification" project. This project has provided me with valuable research opportunities, enabling me to learn about numerous new concepts and techniques.

References:

I have also used few external resources that helped me to complete this project successfully. Below are the external resources that were used to create this project.

- https://www.google.com/
- https://scikit-learn.org/stable/index.html
- https://www.youtube.com/user/krishnaik06
- https://www.analyticsvidhya.com/
- https://chat.openai.com

INTRODUCTION

Business Problem Framing:

Detecting Cybersecurity Threats with KDD99 Intrusion Detection System

In today's interconnected world, where businesses and organizations rely heavily on information systems and networks, the security of digital assets is of paramount importance.

With the increasing sophistication of cyber threats, organizations need robust cybersecurity solutions to safeguard their data and maintain business continuity.

Problem Statement:

Our business problem revolves around enhancing cybersecurity measures through the implementation of an efficient Intrusion Detection System (IDS).

The primary objective is to develop a classification model that can accurately detect and classify network intrusions, thereby enabling organizations to respond proactively to potential threats.

Conceptual Background of the Domain Problem

Cybersecurity and Network Threats:

- **Cybersecurity:** Cybersecurity refers to the practice of protecting computer systems, networks, and digital data from theft, damage, or unauthorized access. It encompasses a wide range of technologies, processes, and practices designed to safeguard digital assets.
- **Network Threats:** Network threats are malicious activities or events that target computer networks, aiming to compromise their integrity, confidentiality, or availability. Common network threats include malware, phishing, denial of service (DoS) attacks, and intrusion attempts.

2. Intrusion Detection Systems (IDS):

- **Definition:** An Intrusion Detection System (IDS) is a security mechanism that monitors network traffic or system activities for signs of unauthorized or malicious behavior or policy violations.
- **Types of IDS:**
  - **Network-based IDS (NIDS):** NIDS monitors network traffic and looks for suspicious patterns or signatures that indicate intrusion attempts.
  - **Host-based IDS (HIDS):** HIDS operates on individual computer systems or hosts, monitoring system logs, configuration files, and critical system files for signs of intrusion.
- **Detection Methods:**
  - **Signature-Based Detection:** This method involves comparing observed network traffic or system activities to known attack patterns or signatures. If a match is found, it raises an alert.
  - **Anomaly-Based Detection:** Anomaly-based IDS establishes a baseline of normal network or system behavior.

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modeling of the Problem:

We aim to develop a robust and accurate Intrusion Detection System (IDS) using Machine Learning classification techniques to enhance network security and identify network intrusions effectively.

Data Collection Phase : The data was available alredy in kaggle Link: https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data

Data Analysis : After cleaning the data I have done some analysis on the data by using different types of visualizations.

Model Building Phase : After collecting the data, I built a machine learning model. Before model building, have done all data pre-processing steps. The complete life cycle of data science that I have used in this project are as follows:

- ✓ Data Cleaning
- ✓ Exploratory Data Analysis
- ✓ Data Pre-processing
- ✓ Model Building
- ✓ Model Evaluation
- ✓ Selecting the best model
- ✓ Hyper parameter tuning
- ✓ Conclusion

Data Sources and their formats:

We have collected the dataset from the website Kaggle whichis a web platform where open souce Dataset are available. The dimension of the dataset is 4.8M rows and 42 columns including target variable "Target". The particular dataset contains both categorical and numerical data type. The data description is as follows:

Variables	Definition
protocol_type	This column represents the network protocol type used in the communication, such as TCP, UDP, or ICMP.
service	Indicates the network service on the destination, e.g., http, ftp, or smtp.
flag	Describes the status of the connection, like S0 (Connection attempt seen, no reply), SF (Normal establishment and termination), etc.
target	This column typically represents the attack type categories, indicating whether a network instance is a normal connection or part of a specific type of attack (e.g., DoS, Probe, R2L, U2R).
duration	The length of the connection in seconds.
src_bytes.	The number of data bytes sent by the source.
dst_bytes.	The number of data bytes sent by the destination.
wrong_fragment	The number of "wrong" fragments.
urgen.	Urgent packet count.
hot	The number of "hot" indicators.
num_failed_logins	Number of failed login attempts.
num_compromised	Number of compromised conditions.
num_root	Number of "root" accesses.
num_file_creations	Number of file creations.
num_shells	Number of shell prompts.

num_access_files	Number of operations on access control files.
num_outbound_cmds	Number of outbound commands in an ftp session.
count	Connection count to the same host as the current connection.
srv_count	Connection count to the same service as the current connection.
serror_rate	Percentage of connections that have "SYN" errors.
srv_serror_rate	Percentage of connections that have "SYN" errors to the same service.
rerror_rate	Percentage of connections that have "REJ" errors.
srv_rerror_rate	Percentage of connections that have "REJ" errors to the same service.
same_srv_rate	Percentage of connections to the same service.
diff_srv_rate	Percentage of connections to different services.
srv_diff_host_rate	Percentage of connections to different hosts for the same service.
dst_host_count	Count of connections to the same destination host.
dst_host_srv_count	Count of connections to the same destination host on the same service.
num_outbound_cmds	Number of outbound commands in an ftp session.
count	Connection count to the same host as the current connection.
srv_count	Connection count to the same service as the current connection.
serror_rate	Percentage of connections that have "SYN" errors.
srv_serror_rate	Percentage of connections that have "SYN" errors to the same service.
rerror_rate	Percentage of connections that have "REJ" errors.

srv_rerror_rate	Percentage of connections that have "REJ" errors to the same service.
same_srv_rate	Percentage of connections to the same service.
diff_srv_rate	Percentage of connections to different services.
srv_diff_host_rate	Percentage of connections to different hosts for the same service.
dst_host_count	Count of connections to the same destination host.
dst_host_same_srv_rat	Percentage of connections to the same service from the same destination host.
dst_host_diff_srv_rate	Percentage of connections to different services from the same destination host.
dst_host_same_src_port_rat	Percentage of connections from the same source port to the same destination host.
dst_host_srv_diff_host_rate	Percentage of connections from the same source port to different destination hosts.
dst_host_serror_rate:	Percentage of connections to the same destination host that have "SYN" errors.
dst_host_srv_serror_rate	Percentage of connections to the same destination host on the same service that have "SYN" errors.
dst_host_rerror_rate	Percentage of connections to the same destination host that have "REJ" errors.
dst_host_srv_rerror_rate	Percentage of connections to the same destination host on the same service that have "REJ" errors.
dst_host_same_srv_rate	Percentage of connections to the same service from the same destination host.

Data Pre-processing Done:

Data pre-processing is the process of converting raw data into a well-readable format to be used by Machine Learning model. Data pre-processing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we pre-process our data before feeding it into our model. I have used following pre-processing steps:

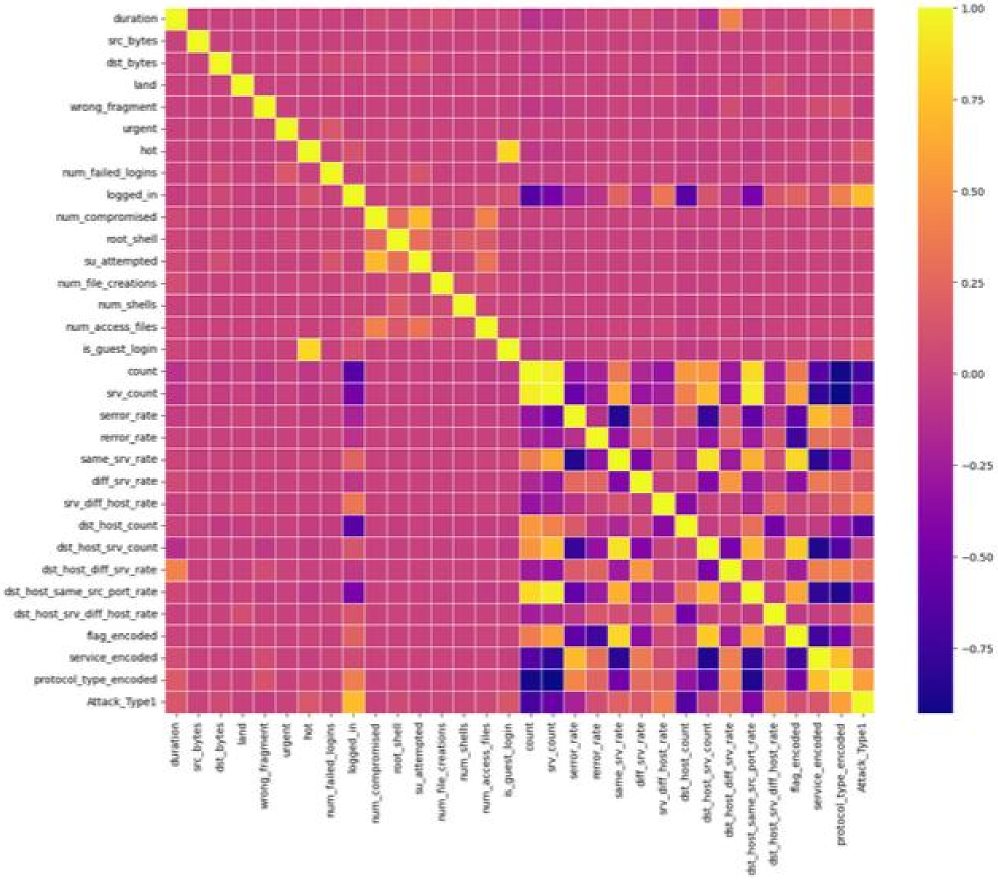
- Importing necessary libraries and loading collected dataset as adataframe.
- Checked some statistical information like shape, number of unique values present, info, unique(), data types, value countfunction etc.
- Checked null valued and found no missing values in the dataset.
- Splitted Numerical and Categorical Variable.
- Performed univariate, bivariate to visualize the data. Visualized each feature using sea born and matplotlib libraries by plotting several categorical and numerical plots.
- Identified outliers using box plots and found outliers but Ignored
- Used correlation coefficient to check the correlation between label and features. With the help of heat map and correlation bar graph was able to understand the Features. With the help of heat map and correlation bar graph was able to understand the Feature vs Label relativity.
- The high correlated variable, so dropped all the highly correlated variable

- Then proceed with Feature Mapping Encoded the columns having object data type using Label Encoder method.
- Separate feature and label data and feature scaling is performedusing Standard Scaler method to avoid any kind of data biasness.
- Performed SMOTE To sampled The data
- Checked the Multicolinearity Problem Within Columns where >5 is considered as Threshold
- Used PCA to Remove Multicolinearity Issue

Data Inputs – Logic – Output Relationships:

The dataset consists of label and features. The features are independent and label is dependent as the values of our independent variables changes as our label varies.

I have checked the correlation between the label and features using heat map and bar plot. Where I got both positive and negative correlation between the label and features. Below is the Heat Map to know the correlation between features and label.
I dropped All the columns which have more that 97% as the information are same



Libraries Required:

- ***import numpy as np:*** It is defined as a python package used forperforming the various numerical computations and processing ofthe multidimensional and single dimensional array elements. The calculations using Numpy arrays are faster than the normal Python array.
- ***Import pandas as pd:*** Pandas is a Python library that is usedfor faster data analysis, data cleaning and data pre-processing. The data-frame term is coming from pandas only.
- ***Import matplotlib.pyplot as plt :*** Matplotlib and Seaborn actsas the backbone of data visualization through Python.
- ***Matplotlib:*** It is a Python library used for plotting graphs with the help of other libraries like Numpy and Pandas. It is a powerful tool for visualizing data in Python. It is used for creating statistical interferences and plotting 2D graphs of arrays.
- ***Import seaborn as sns:*** Seaborn is also a Python library used for plotting graphs with the help of Matplotlib, Pandas and Numpy. It is built on the roof of Matplotlib and is considered as asuperset of the Matplotlib library. It helps in visualizing univariateand bivariate data.
- **from sklearn.preprocessing import LabelEncoder:**It is used to encode alal the Categorical Data
- **from statsmodels.stats.outliers_influence import variance_inflation_factor:**It helps to check the Multicolinearty Issue within Dataset
- **from sklearn.decomposition import PCA:**It helps to Remove Multicolinearity Issue between columns

With the above sufficient libraries, we can perform pre-processing, data cleaning

MODEL/ S DEVELOPMENT AND EVALUATION

Identification of possible Problem-solving approaches(Methods):

I have used both statistical and analytical approaches to solve the problem which mainly includes the pre-processing of the data also used EDA techniques and heat map to check the correlation of independent and dependent features. Encoded data using Label Encoder. Also, before building the model, I made sure that the input data is cleaned and scaled before it was fed into the machine learning models. Machine Learning model pertaining to the feature importance details. Finally created multi Modal Classification models along with evaluation metrics.

For this particular project we need to Intrusion Detection. In this dataset, “Attack_Type” is the target variable, which means our target column is Classificataion in nature so this is a Classification problem. I have used decision Tree and Logistic Regression algorithms and predicted theIntrusion Detection System. By doing various evaluations I have selected Decision Treeas best suitable algorithm to create our final model as it is giving high Accuracy and Good Precision and Recall among all the algorithms used. Performed hyper parameter tuning on best model. Then I saved my final model and loaded the same for predictions.

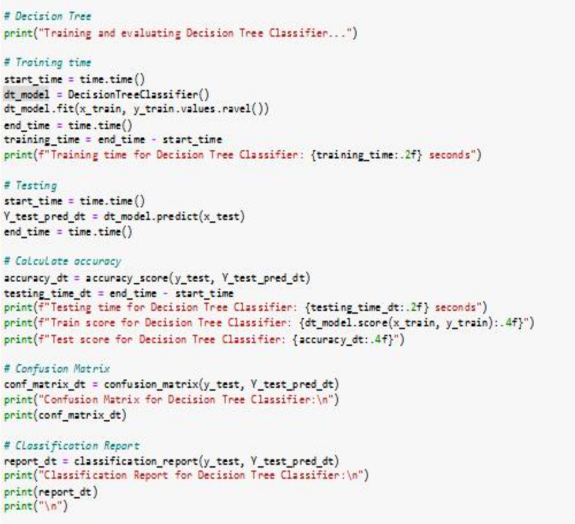
Testing of Identified Approaches (Algorithms):

Since “Attack-Type” is my target variable which is The classes are discrete and non-binary in nature, from this I can conclude that it is a Multiclass-Classification type problem hence I have used following Classificationalgorithms. After the pre-processing and datacleaning I left with 35 columns including target and with the help of feature importance .The algorithms used on training the dataare as follows:

- Decision Tree   Classifier
- Logistic regression

Model Building

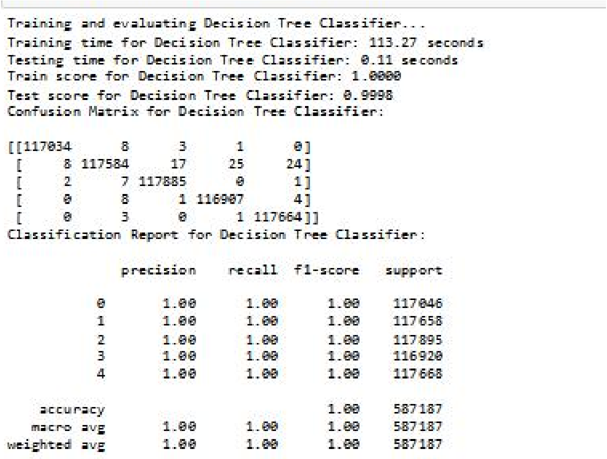
Decision Tree Classifier



```
# Decision Tree
print("Training and evaluating Decision Tree Classifier...")

# Training time
start_time = time.time()
dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train.values.ravel())
end_time = time.time()
training_time = end_time - start_time
print(f"Training time for Decision Tree Classifier: {training_time:.2f} seconds")

# Testing
start_time = time.time()
Y_test_pred_dt = dt_model.predict(x_test)
end_time = time.time()

# Calculate accuracy
accuracy_dt = accuracy_score(y_test, Y_test_pred_dt)
testing_time_dt = end_time - start_time
print(f"Testing time for Decision Tree Classifier: {testing_time_dt:.2f} seconds")
print(f"Train score for Decision Tree Classifier: {dt_model.score(x_train, y_train):.4f}")
print(f"Test score for Decision Tree Classifier: {accuracy_dt:.4f}")

# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, Y_test_pred_dt)
print("Confusion Matrix for Decision Tree Classifier:\n")
print(conf_matrix_dt)

# Classification Report
report_dt = classification_report(y_test, Y_test_pred_dt)
print("Classification Report for Decision Tree Classifier:\n")
print(report_dt)
print("\n")
```

Decision Tree is a decision making tool thatuses a flowchart like tree structure. It observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful output.

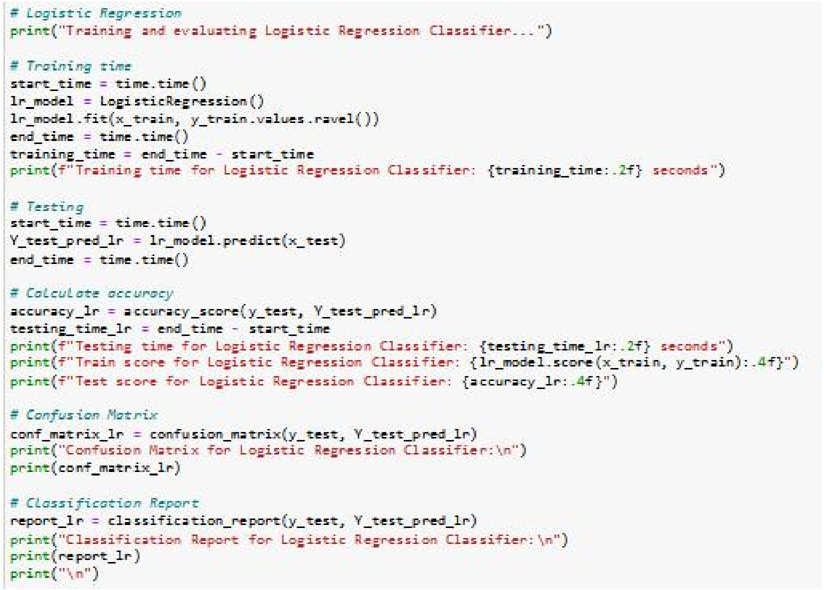
- ➢ Created Decision Tree Classifier model and checked for its evaluation metrics. The model is giving Accuracy-99.98%



```
Training and evaluating Decision Tree Classifier...
Training time for Decision Tree Classifier: 113.27 seconds
Testing time for Decision Tree Classifier: 0.11 seconds
Train score for Decision Tree Classifier: 1.0000
Test score for Decision Tree Classifier: 0.9998
Confusion Matrix for Decision Tree Classifier:

[[117034      8      3      1      0]
 [     8 117584     17     25     24]
 [     2      7 117885      0      1]
 [     0      8      1 116907      4]
 [     0      3      0      1 117664]]
Classification Report for Decision Tree Classifier:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    117046
           1       1.00      1.00      1.00    117658
           2       1.00      1.00      1.00    117895
           3       1.00      1.00      1.00    116920
           4       1.00      1.00      1.00    117668

    accuracy                           1.00    587187
   macro avg       1.00      1.00      1.00    587187
weighted avg       1.00      1.00      1.00    587187
```

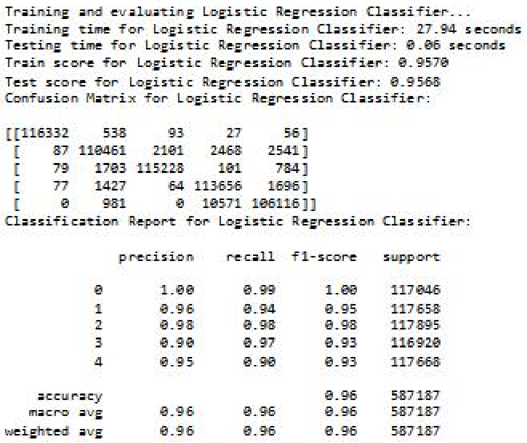
- ***Logistic Regression***

Logistic regression is a widely used statistical method for binary and multi-class classification problems. It models the relationship between a binary or categorical dependent variable and one or more independent variables by estimating probabilities using the logistic function, which allows for efficient classification of data points into different classes.



```
# Logistic Regression
print("Training and evaluating Logistic Regression Classifier...")

# Training time
start_time = time.time()
lr_model = LogisticRegression()
lr_model.fit(x_train, y_train.values.ravel())
end_time = time.time()
training_time = end_time - start_time
print(f"Training time for Logistic Regression Classifier: {training_time:.2f} seconds")

# Testing
start_time = time.time()
Y_test_pred_lr = lr_model.predict(x_test)
end_time = time.time()

# Calculate accuracy
accuracy_lr = accuracy_score(y_test, Y_test_pred_lr)
testing_time_lr = end_time - start_time
print(f"Testing time for Logistic Regression Classifier: {testing_time_lr:.2f} seconds")
print(f"Train score for Logistic Regression Classifier: {lr_model.score(x_train, y_train):.4f}")
print(f"Test score for Logistic Regression Classifier: {accuracy_lr:.4f}")

# Confusion Matrix
conf_matrix_lr = confusion_matrix(y_test, Y_test_pred_lr)
print("Confusion Matrix for Logistic Regression Classifier:\n")
print(conf_matrix_lr)

# Classification Report
report_lr = classification_report(y_test, Y_test_pred_lr)
print("Classification Report for Logistic Regression Classifier:\n")
print(report_lr)
print("\n")
```

- ➢ Created Logistic Regression model and checked for its evaluation metrics. The model is giving Accuracy-96.68%

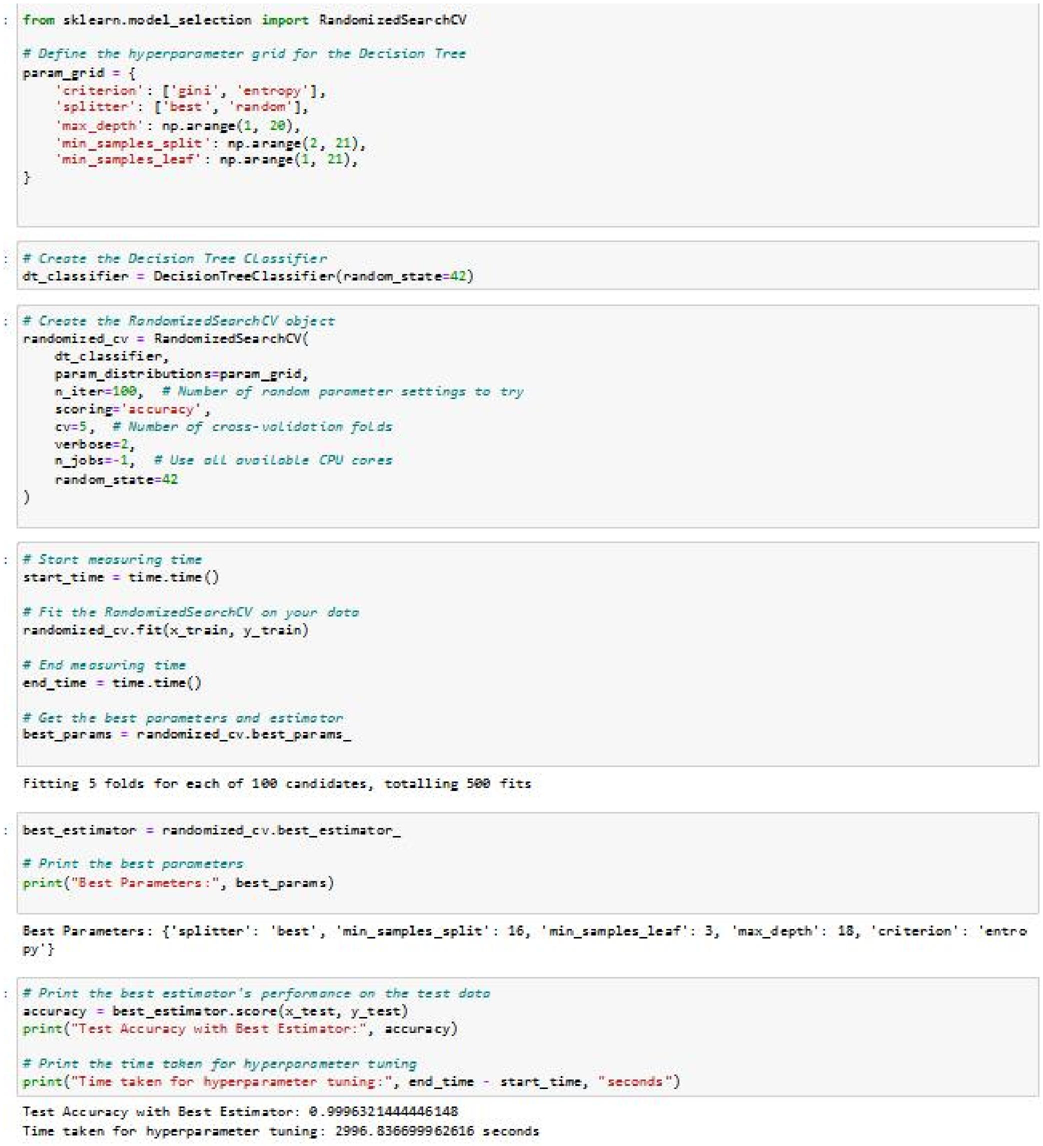


```
Training and evaluating Logistic Regression Classifier...
Training time for Logistic Regression Classifier: 27.94 seconds
Testing time for Logistic Regression Classifier: 0.06 seconds
Train score for Logistic Regression Classifier: 0.9570
Test score for Logistic Regression Classifier: 0.9568
Confusion Matrix for Logistic Regression Classifier:

[[116332    538     93     27     56]
 [    87 110461   2101   2468   2541]
 [    79   1703 115228    101    784]
 [    77   1427     64 113656   1696]
 [     0    981      0  10571 106116]]
Classification Report for Logistic Regression Classifier:

              precision    recall  f1-score   support

           0       1.00      0.99      1.00    117046
           1       0.96      0.94      0.95    117658
           2       0.98      0.98      0.98    117895
           3       0.90      0.97      0.93    116920
           4       0.95      0.90      0.93    117668

    accuracy                           0.96    587187
   macro avg       0.96      0.96      0.96    587187
weighted avg       0.96      0.96      0.96    587187
```

Model Selection

From the above created models, we can conclude that “Decision tree” as the best fitting model as it is giving high Accuracy score,Persision,recall Then I tried to increase our model score by tuning the best model using different types of hyper parameters.

Hyper Parameter Tuning:



```
from sklearn.model_selection import RandomizedSearchCV

# Define the hyperparameter grid for the Decision Tree
param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': np.arange(1, 20),
    'min_samples_split': np.arange(2, 21),
    'min_samples_leaf': np.arange(1, 21),
}
```

```
# Create the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
```

```
# Create the RandomizedSearchCV object
randomized_cv = RandomizedSearchCV(
    dt_classifier,
    param_distributions=param_grid,
    n_iter=100,  # Number of random parameter settings to try
    scoring='accuracy',
    cv=5,  # Number of cross-validation folds
    verbose=2,
    n_jobs=-1,  # Use all available CPU cores
    random_state=42
)
```

```
# Start measuring time
start_time = time.time()

# Fit the RandomizedSearchCV on your data
randomized_cv.fit(x_train, y_train)

# End measuring time
end_time = time.time()

# Get the best parameters and estimator
best_params = randomized_cv.best_params_
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
best_estimator = randomized_cv.best_estimator_

# Print the best parameters
print("Best Parameters:", best_params)
```

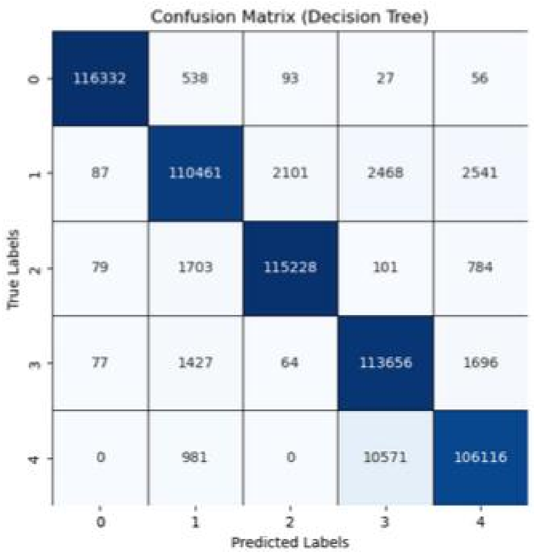
```
Best Parameters: {'splitter': 'best', 'min_samples_split': 16, 'min_samples_leaf': 3, 'max_depth': 18, 'criterion': 'entro
py'}
```

```
# Print the best estimator's performance on the test data
accuracy = best_estimator.score(x_test, y_test)
print("Test Accuracy with Best Estimator:", accuracy)

# Print the time taken for hyperparameter tuning
print("Time taken for hyperparameter tuning:", end_time - start_time, "seconds")
```

```
Test Accuracy with Best Estimator: 0.9996321444446148
Time taken for hyperparameter tuning: 2996.836699962616 seconds
```

- We have successfully incorporated the hyper parameter tuning using best parameters of Decision Tree and the Accuracy of the model has been almost the same which is very good.

Plotting Confusion Matrix:



	0	1	2	3	4
0	116332	538	93	27	56
1	87	110461	2101	2468	2541
2	79	1703	115228	101	784
3	77	1427	64	113656	1696
4	0	981	0	10571	106116

Class 0:

- Actual Class 0 (row 1):
- True Positives (116,332): The model correctly predicted instances belonging to Class 0 as Class 0.
- False Negatives (538): The model incorrectly predicted instances belonging to Class 0 as other classes (Class 1, Class 2, Class 3, or Class 4).

Class 1:

- Actual Class 1 (row 2):
- True Positives (110,461): The model correctly predicted instances belonging to Class 1 as Class 1.
- False Positives (87): The model incorrectly predicted instances not belonging to Class 1 as Class 1.
- False Negatives (2,101): The model incorrectly predicted instances belonging to Class 1 as other classes (Class 0, Class 2, Class 3, or Class 4).

Class 2:

Actual Class 2 (row 3):

- True Positives (115,228): The model correctly predicted instances belonging to Class 2 as Class 2.
- False Positives (1,703): The model incorrectly predicted instances not belonging to Class 2 as Class 2.
- False Negatives (101): The model incorrectly predicted instances belonging to Class 2 as other classes (Class 0, Class 1, Class 3, or Class 4).

Class 3:

Actual Class 3 (row 4):

- True Positives (113,656): The model correctly predicted instances belonging to Class 3 as Class 3.
- False Positives (1,427): The model incorrectly predicted instances not belonging to Class 3 as Class 3.
- False Negatives (64): The model incorrectly predicted instances belonging to Class 3 as other classes (Class 0, Class 1, Class 2, or Class 4).

Class 4:

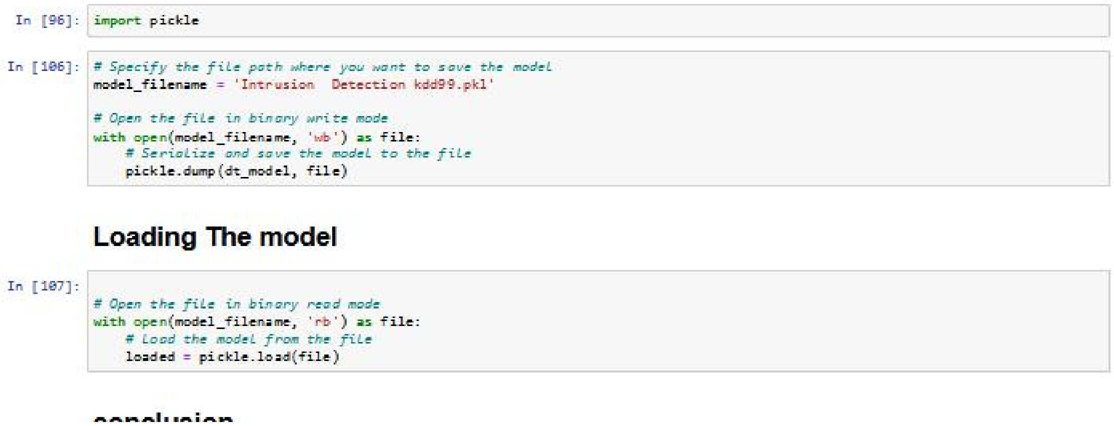
Actual Class 4 (row 5):

- True Positives (106,116): The model correctly predicted instances belonging to Class 4 as Class 4.
- False Positives (0): The model did not incorrectly predict instances not belonging to Class 4 as Class 4.
- False Negatives (9,811): The model incorrectly predicted instances belonging to Class 4 as other classes (Class 0, Class 1, Class 2, or Class 3).

Interpretation:

1. True Positives (TP) are the correct predictions for each class.
2. False Positives (FP) represent instances where the model incorrectly predicted the class.
3. False Negatives (FN) represent instances where the model failed to predict the correct class.

Saving The model:



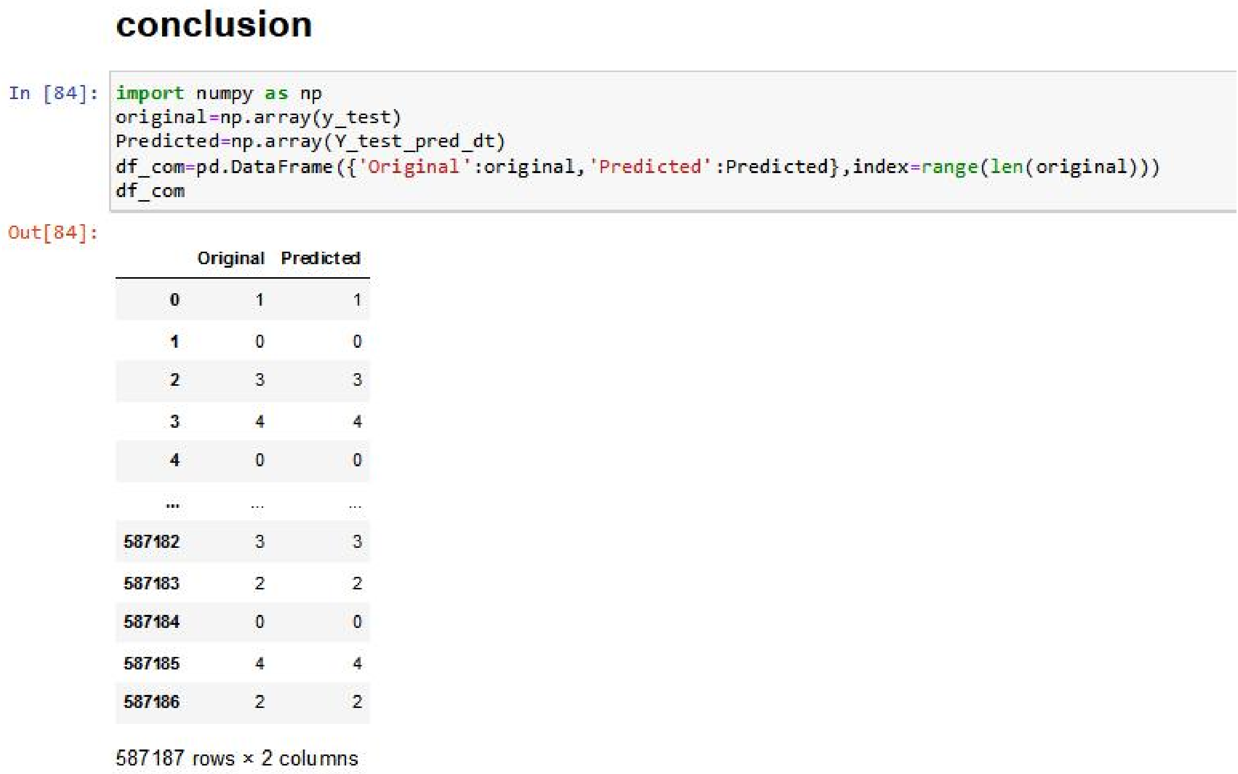
```
In [96]: import pickle
```

```
In [106]: # Specify the file path where you want to save the model
model_filename = 'Intrusion  Detection kdd99.pkl'

# Open the file in binary write mode
with open(model_filename, 'wb') as file:
    # Serialize and save the model to the file
    pickle.dump(dt_model, file)
```

Loading The model

```
In [107]: 
# Open the file in binary read mode
with open(model_filename, 'rb') as file:
    # Load the model from the file
    loaded = pickle.load(file)
```

CONCLUSION



conclusion

```
In [84]: import numpy as np
original=np.array(y_test)
Predicted=np.array(Y_test_pred_dt)
df_com=pd.DataFrame({'Original':original,'Predicted':Predicted},index=range(len(original)))
df_com
```

Out[84]:

	Original	Predicted
0	1	1
1	0	0
2	3	3
3	4	4
4	0	0
...	...	...
587182	3	3
587183	2	2
587184	0	0
587185	4	4
587186	2	2

587187 rows × 2 columns

Key Findings and Conclusions of the study

The case study aims to give an idea of applying Machine Learning algorithms to predict Intrusion Detection System. After the completion of this project, we got an insight of how to collect data, pre-processing the data, analyses the data, cleaning the data and building a model.

In this study, we have used multiple machine learning models to predict the Intrusion Detection System. We have gone through the data analysis by performing feature engineering, finding the relation between features and label through visualizations. And got the important feature and we used these features to predict Intrusion Detection by building ML models. Performed hyper parameter tuning on the bestmodel and the best model's score increased Slightly and was giving accuracy score as 99.98% . We have also got good prediction results of Intrusion Detection.

Learning Outcomes of the Study in respect of DataScience :

While working on this project I learned many things about the features of Cyber Security and about the platforms and got the idea that how the machine learning models have helped to predict Intrusion Detection. I found that the project was quite interesting as the dataset contains several types of data. I used several types of plotting to visualize the relation between target and features. This graphical representation helped me to understand which features are important and how these features describe More on Cyber Security. Data cleaning was one of the important and crucial things in this project where I dealt with features having string values, features extraction andselection. Finally got decision Trees Classifier as best model.

The challenges I faced while working on this project was when I was Lack Of domain Knowledge, it took so much time to understand data. Finally, our aim was achieved by predicting the Intrusion Detection Model.

Limitations of this work and scope for future work

Limitations: The main limitation of this study is the lack of Domain Knowledge and Technical Terms that have been used. In the dataset our data is not properly distributed in some of the columns many of the values in the columns are having string values which I had taken care. Due to some reasons our models may not make the right patterns and the performance of the model also reduces. So that issues need to be taken care

****