STRIDE BASED CYBER THREAT INTELLIGENCE MODEL USING MACHINE LEARNING

Report submitted to SASTRA Deemed to be University as per the requirement of the course

CSE300 / INT300 / ICT300 – MINI PROJECT

Submitted by

SHREYAS RAVISHANKAR
[124157052, B.Tech. Computer Science and Engineering
(with Spl. in Cyber Security and Blockchain Technology)]

VISHAL KIRTHIC N
[124157074, B.Tech. Computer Science and Engineering
(with Spl. in Cyber Security and Blockchain Technology)]

MAY 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613401.

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613401.

<u>BONAFIDE CERTIFICATE</u>

This is to certify that the report titled **'STRIDE Based Cyber Threat Intelligence Model using Machine Learning'** submitted as a requirement for the course CSE300 / INT300 / ICT300 – Mini Project for B.Tech. is a Bonafide record of the work done by **Mr. Shreyas Ravishankar [124157052, B.Tech. Computer Science and Engineering (with Spl. In Cyber Security and Blockchain Technology)]** and **Mr. Vishal Kirthic N [124157074, B.Tech. Computer Science and Engineering (with Spl, in Cyber Security and Blockchain Technology)]** during the academic year 2022-2023, in the School of Computing, under my supervision.

Signature of the Project Supervisor :
Name with Affiliation :
Date :

Mini Project Viva-voce held on ____________________

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

எந்நன்றி கொன்றார்க்கும் உய்வுண்டாம் உய்வில்லை
செய்ந்நன்றி கொன்ற மகற்கு. | குறள் – 110

We would like to thank our Honourable Chancellor, **Prof. R. Sethuraman,** for providing us with the opportunity and the necessary infrastructure for carrying out the project as a part of our Undergraduate studies.

We are forever indebted to our Vice-Chancellor, **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean, Planning & Development, for their encouragement and strategic support at every step in this lovely journey called College.

We would also like to extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for their continued support in giving various ideas to carefully sculpt the project into becoming a reality.

Our heartfelt thanks to **Dr. A. Umamakeswari,** Dean, School of Computing, **Dr. S. Gopalakrishnan,** Associate Dean, Dept. of Computer Applications, **Dr. B. Santhi,** Associate Dean, Research and **Dr. A. Muthaiah,** Associate Dean, Dept. of Information Technology and Information Communication Technology.

No number of words/adjectives can express how grateful we are for our driving force and guiding light, **Dr. V. S. Shankar Sriram,** Associate Dean, Dept. of Computer Science and Engineering, whose valuable inputs, deep insights and invaluable suggestions have helped us making progress throughout this project, especially in times when we were stuck on a problem and found it difficult to find a breakthrough.

We would also like to extend our gratitude to our Project Review Panel Members, **Dr. N. Sasikala Devi** and **Dr. P. Shanthi** for making us think out-of-the-box to make the code more efficient and giving insights on how to improve the project during every meeting that happened. At the long run, they have proven to be the game-changers in the overall project.

We would like to thank all the Teaching and the Non-Teaching faculties of the School of Computing who have directly or indirectly helped us in completing the project.

Last, but certainly not the least, we would like to gratefully acknowledge our friends and family for all their encouragement, motivation and emotional support for being with us all along in all the highs and lows for us to successfully complete the project.

Thanking the God almighty and everyone for believing in us and providing an opportunity to showcase our skills throughout the project.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

➢ DTC – Decision Tree Classifier	➢ SVM – Support Vector Machine
➢ GNB – Gaussian Naïve Bayes	➢ SML – Supervised Machine Learning
➢ KNN – K-Nearest Neighbours	➢ CV – Cross Validation
➢ LogReg – Logistic Regression	➢ XSS – Cross Site Scripting
➢ MLP – Multi Layer Perceptron	➢ SQL – Structured Query Language
➢ RFC – Random Forest Classifier	➢ KDD – Knowledge Discovery Databases

NOTATIONS

- $\hat{y}$ – Output of the Sigmoid Function
- $\sigma(x)$ – The Sigmoid Function
- $P(t = c|x)$ – Probability Function
- μ_c – Class Specific Mean Vector*
- Σ_c – Class Specific Covariance Matrix*

** The exact formula under which the notation is used is explained in Chapter 3.*

ABSTRACT

As technology advances, the risks of encountering cyber-attacks of various scales become increasingly prevalent. Despite a sharp-focus on end-user convenience, many organizations often fail to prioritize system security, which in turn leads to various cyber-attacks that have seemingly spiked in the recent times owing to the COVID-19 Pandemic.

To address these shortcomings, the technique of Advanced Cyber Threat Detection and Prevention Mechanisms [ACTDP] based on the concept of Defensive Security Measures and Blue Teams have been developed in the recent times. However, current approaches have been limited in their ability to effectively model and detect Cyber Threats. Traditional Defensive Mechanisms such as Intrusion Detection Systems [IDS] and Intrusion Prevention Systems [IPS] have been able to detect threats and subsequently any targeted attacks, but they are always prone to False Alarms [FAs] which can result [in Sensitive Cases] in Real Threats getting unnoticed.

Therefore, we as a part of our Mini Project are proposing a Machine Learning [ML] Based Cyber Threat Intelligence [CTI] Model using the STRIDE Framework. STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. The ML Model is trained on Large Datasets [preferably the NSL-KDD Dataset (or) equals] to more accurately identify threats and minimize FAs.

The Use of Machine Learning results in improved Accuracy Levels when compared to its Traditional Counterparts. This lightweight and effective approach helps Cyber Security Teams efficiently mitigate the seemingly most dangerous threats that has the potential to cause catastrophic loss of information to many high-profile Organizations.

Keywords:
Cyber-Attacks, Machine Learning, STRIDE, Intelligent Cyber Threat Modeling, Cyber Security.

TABLE OF CONTENTS

CHAPTER 1
KNOWING STRIDE

What is STRIDE?

STRIDE is one of the most sought out threat modelling frameworks by today's enterprises which helps to identify potential security loopholes in a system. It was first introduced by Microsoft (by Praerit Garg and Loren Kohnfelder) in the early 2000s and since then, it has been widely used in the field of IT Security. It provides the mnemonic for the security threats in these 6 categories: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privileges. Considering each of these vectors, the STRIDE model provides a way to systematically analyse each one of them, identify and map these potential vulnerabilities. With this model, security professionals can address these vulnerabilities and take steps to mitigate them.

Why did we choose STRIDE:

This model has a structured framework to identify threats in the system where all possible attack vectors are considered. It has been redesigned, refined and developed over the years and has been well-established by so many experts in the field of cyber security.

It is also easy to understand and implement in the modern and/or legacy systems and can have a common language for addressing the security threats. It is also flexible and can be applied to a wide range of systems which also includes software applications, networks and hardware devices which makes it a versatile tool for security professionals.

With all these benefits and more, the conclusion is that the STRIDE model is a powerful tool for threat modelling which can help organizations identify and mitigate potential security risks and considering its simplicity, flexibility and effectiveness, this has made it one of the most popular and widely used threat modelling frameworks by many enterprises.

The 6 Attack Vectors:

Spoofing:

Spoofing is a type of security threat where the attacker attempts to successfully penetrate the security of the system to gain unauthorized access by impersonating themselves as a legitimate user/employee. This could be the result of credentials stolen through social engineering or gaining credentials from a possible breach etc.

These types of attacks are very difficult to detect as this process involves trying to access as someone who is already inside the system which makes it a very dangerous threat.

To handle this type of threat, security professionals often employ measures such as multi-factor authentication, encryption, definitive access controls etc. With these measures, this will ensure only users who are actually legitimate can access the system and ensure they don't misuse the system's resources and successfully conducting spoofing on the system becomes difficult.

Tampering:

Tampering is a type of security threat where an attacker attempts to modify the data of a system or modify the rules of a system in order to gain unauthorized access or cause harm to the system. There are various ways to call tampering such as altering the contents of a database, modifying the rules of the firewall, modify the source code, access controls etc.

The goal of this attack vector is to bypass security controls and gain access to sensitive data. To prevent these types of attacks, security professionals employ measures such as secure coding practices, monitoring the system logs for suspicious activity, encryption and hashing etc. Tampering can undermine the security and integrity of a system and this is extremely dangerous.

Repudiation:

Repudiation is a type of security threat where an attacker is able to deny that he/she is the person who has performed this particular action. This is the result of modifying the system's logs and/or other records which make it difficult to determine who is the actual attacker and make it appear as though they did not carry out that particular activity.

These kinds of attacks can be an issue in situations of legal or financial consequences and to prevent such attacks, security professionals employ techniques such as digital signatures, audit trails etc. This will help ensure that all actions are properly recorded and tracked and there would be no way for the user to deny that they have conducted this action which can help identify malicious users in the system.

Information Disclosure:

Information Disclosure is a type of attack vector where the attacker is able to gain access to sensitive/confidential information of a system without any form of authorization. This is the result of the data not being properly secured when it is transmitted over an unsecured network or proper security controls/methods are not enforced on the system carrying this information rendering the system vulnerable.

This type of attack can come in various forms such as possible breaches, social engineering attacks, phishing etc. Sometimes, in order to gain such information attackers can enforce SQL Injection, XSS Injection or try specific scripts which can leak information due to unsecure source code.

The impact can be severe as the attacker can gain information that can be used against the employees or users of the system or in worst case scenario, the attacker can use it against the system itself. To prevent these attacks, security professionals enforce techniques such as encryption, access controls, data masking, secure coding etc. These help ensure that sensitive information is protected and makes it difficult for attackers to gain access to these. More importantly, training and awareness programs can help people to recognize and avoid social engineering attacks and phishing attacks.

Denial of Service (DOS):

Denial of Service is a type of security attack where the attacker attempts to disrupt a system or disable a network by overwhelming it with traffic or other forms of malicious activity. The main goal is to stop service or in other words, make it unavailable to legitimate users to and from the system and in certain cases, this can be very dangerous either reputation wise or financial wise. This can include flooding a network's traffic, exploiting vulnerability in a software or hardware or launching a distributed attack using multiple compromised system (DDOS) etc.

To prevent this attack, security professionals often use techniques such as load balancers, limiters, IDS and IPS systems. These measures help identify, prevent and block those malicious users before they can overwhelm the network.

Elevation of Privileges:

Elevation of Privileges is a type of security threat where an attacker is able to gain access to data or resources that they are not supposed to. This means they were able to increase the permission levels of what they can access in the system. This can occur when there are vulnerabilities in the system if exploited, can help the attacker to elevate his privileges to the system (Bypass security controls).

This form of attack can take place in many forms such as exploiting certain vulnerabilities in operating systems or applications, using social engineering techniques, manipulating system settings etc. An attacker who is able to gain administrative level privileges can possibly install virus or malware, or steal sensitive information and this makes it a very dangerous attack vector.

To prevent this kind of an attack, security professionals use techniques such as access controls, separation of responsibilities etc. Ensuring users have access that they only require and no extra permissions given to them would better mitigate this kind of attack or reduce the attack surface for this kind of an attack.

CHAPTER 2
THE NSL-KDD DATASET

The KDD99 dataset is a benchmark dataset that is mainly used for Intrusion detection systems (IDS) that was developed by researchers at the University of California, Irvine in 1999. This dataset consists of network traffic data that was captured in the simulated environment and it contains attack as well as normal packets.

This dataset is mainly used in the development, training and evaluation of Intrusion Detection Systems and it is used to compare the effectiveness of different algorithms and other techniques as this dataset is the very benchmark for any studies in this or related field.

This dataset contains over 4.9 million records which are broadly divided into these 5 categories of attacks: Denial of Service (DoS) , Probe, Remote to Local (U2R), User to Root (U2R) and Normal. Despite being old, this dataset is still used in research and development of Intrusion and Detection Systems due to its size, diversity and it also includes both known and unknown attack vectors. It Is a valuable resource for researchers and practitioners in the field of cybersecurity for the past 2 decades and one of the reasons why we have decided to work with the KDD99 dataset for our project.

The 41 features of the dataset:

duration: Length of time in seconds for connecting to the target

protocol_type: Protocol used for connecting to the target. Can be TCP, UDP, SMTP or ICMP

service: Service with which we are interacting with the target

flag: Status of connection, S0 means failed, SF means success, REJ means rejected

src_bytes: the number of bytes sent from the source target to the destination

dst_bytes: the number of bytes sent from the destination to the source target

land: a binary flag indicating whether the connection is to/from the same host/port. If the result is 1, then they are from the same host/port and if 0, they are not and are from different host/port

wrong_fragment: Number of wrong fragments received. This means the packets are malformed or not formed properly. Can be malicious

urgent: Number of urgent packets received
hot: Number of hot indicators which can mean failed login attempts or malicious activity from our case

num_failed_logins: Failed login attempts

logged_in: Binary flag where if result is 1, it means successful login of user and 0 is opposite (user didn't login)

num_compromised: Number of packets which are sent for compromising the target

root_shell: Binary flag indicating that if result is 1, root shell is obtained by the attacker and 0 means nt obtained

su_attempted: Same as above for gaining superuser access

num_root: Number of root accesses

num_file_creations: The number of file creation operations *num_shells:* Number of prompts calling the shell. (cmd, terminalk etc)

num_access_files: Number of accesses to sensitive files
num_outbound_cmds: Number of outbound commands in ftp session

is_host_login: 1 means packet belongs to hot list and 0 means no

is_guest_login: guest login. Same as above
count: Number of connections made to the same destination within the 2 seconds time frame

srv_count: Number of connections to the same service within the 2 seconds time frame

srv_serror_rate: Percentage of connections which has SYN error

rerror_rate: Percentage of packets which have REJ errors

srv_rerror_rate: REJ errors to the same service

same_srv_rate: Percentage of connections to same service

diff_srv_rate: Percentage of connections to different service

srv_diff_host_rate: Percentage of connections to different hosts

dst_host_count: Number of connections to the same destination within the 2 seconds time frame

dst_host_srv_count: Number of connections to the same service within the 2 seconds time frame

dst_host_same_srv_rate: Percentage of connections to the same service on the same destination. Time not necessarily mentioned

dst_host_diff_srv_rate: Percentage of connections to different services on the accessed by the same destination in their current connection

dst_host_same_src_port_rate: Same destination same source port as current connection

dst_host_srv_diff_host_rate: Same service on a different destination host

dst_host_serror_rate: Connections which have SYN errors to the same destination host

dst_host_srv_serror_rate: SYN error to same service on the same destination host

dst_host_rerror_rate: REJ errors to the same destination host, if any

dst_host_srv_rerror_rate: REJ errors to the same service on the same destination host, if any

labels: Labels on each connection indicating if packet is normal or which attack vector is involved. For the KDD99 dataset, it has various attack vectors whereas, for our Training and Testing dataset, either the packets are normal or the attack vectors are only associated to the STRIDE vector model.

CHAPTER 3
COMPARITIVE STUDY ON THE EFFICIENCIES OF DIFFERENT SUPERVISED ML ALGORITHMS TO TRAIN THE STRIDE BASED CYBER THREAT INTELLIGENCE MODEL

We chose the following 7 Supervised Machine Learning algorithms to train our STRIDE Dataset and perform the comparative study. A brief description about the algorithms have also been given:

Random Forest:

The Random Forest Algorithm [RFA] is one among the most popularly used SML Algorithm when training models based on the properties of classification or regression. The Algorithm takes the training dataset as the input, constructs 'n' number of Decision Trees and identifies the majority vote which would be the output of the training model. The vote is usually based on the input hyperparameters given and are famously known to handle both continuous and categorical decisional class variables. They follow the ensemble model of training, meaning that they create multiple different models during training and all these models together decide on the predictions made when the model is used for predicting the output and hence, the RFA Algorithm is usually found to be performing better when compared to the other SML Algorithms.

Logistic Regression:

Logistic Regression is generally used for Multiclass Training Models. It is a probabilistic model, meaning that the algorithm computes the probability which lies between 0 and 1 (inclusive of both) and the value obtained would decide on the outcome from the data. The probability function for Logistic Regression is known as the “Sigmoid” function, which is as follows:

$$\hat{y} = \sigma(\theta^t.x)$$
$$where\ \sigma(\theta^t.x) = \frac{1}{1 + e^{-\theta^t.x}}$$

Decision Tree Classifier:

The Decision Tree Algorithm [DTC] follows a systematic tree structutre that contains the following components: i) A Root Node, ii) Branches, iii) Internal Nodes and iv) Leaf Nodes. The features of the input Dataset are represented by the internal nodes, while the branches represent the rules followed by the DTC to train the model. The Output of the model is represented by the leaf nodes. Predictions are made based on 2 algorithms that the model deploys to predict - the Information Gain Algorithm (or) The Gini Coefficient Algorithm - which are also the hyperparameters of the model - and have shown to provide exceptional results in input cases that well fit the structure of the Decision Tree.

Gaussian Naïve Bayes:

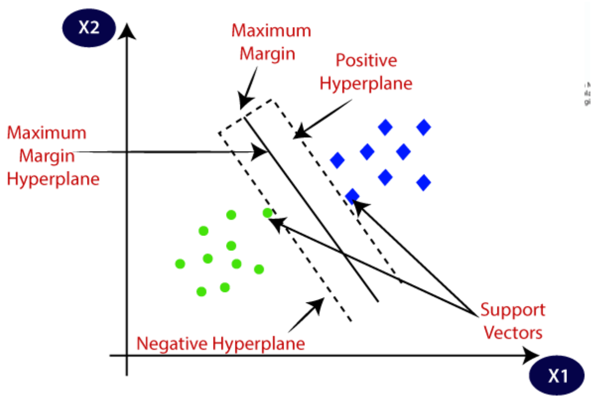
The Gaussian Naive Bayes Algorithm [GNB] predicts the output variable based on the condition that each conditional feature of the input dataset can independently define a function to identify the outcome. The final prediction is based on the combination of all the predictor functions taken as parameters for the probability function and the final classification is assigned to the group with the highest probability. The Bayes Theorem, which is the core concept of this algorithm, is as follows:

$$P(t=c|x,\mu_c,\Sigma_c) = \frac{P(x|t=c,\mu_{c,}\Sigma_c)P(t=c)}{\sum_k^{K=1} P(x|t=k,\mu_k,\Sigma_c)P(t=k)}$$

where:
x = Input Variables
t = Target Variables
μ = Class-Specific Mean Vector
Σ = Class-Specific Covariance Matrix
LHS of the Equation = Class Posterior
Nr. of the RHS = Conditional Class Density

Support Vector Machines:

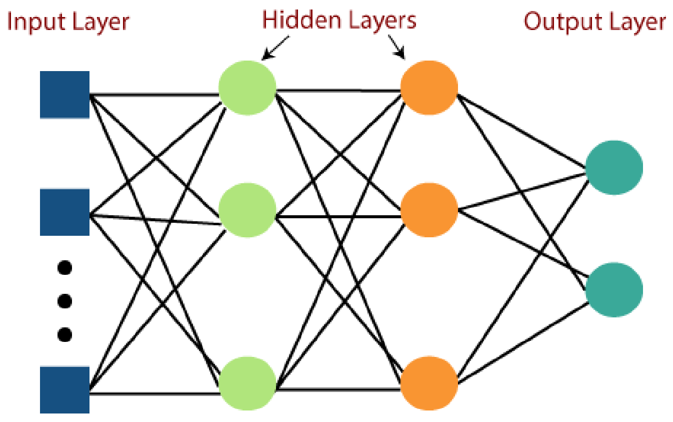
The Main aim for Support Vector Machines [SVM] is to identify the best boundary/line that can contain all the n-dimensional parameters into defined classes so that if any new point is to be added in the future, it can be done so based on the similarity it shares with the already defined classes. Technically known as the 'hyperplane', the best line/boundary is identified as follows:



The extreme points/vectors are usually used in identifying the best hyperplane. The algorithm is named Support Vector Machines because these extreme points/cases of the input are technically known as 'Support Vectors' when training the model.

Multilayer Perceptron [MLP] Algorithm:

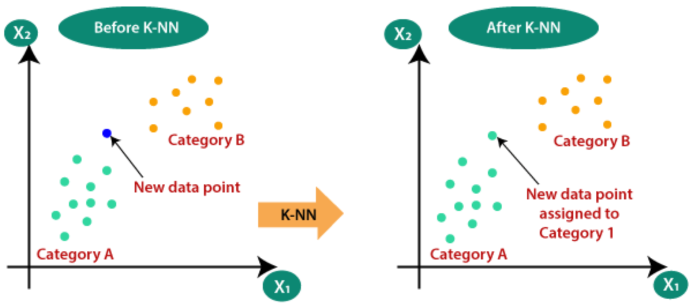
The MLP Algorithm is a class of continuous feed-forward Artificial Neural Networks [ANN]. The output of the model is defined on the basis of three layers the algorithm creates - The Input Layer, The Hidden Layer and the Output Layer. The core concept of the MLP Algorithm is backpropagation - a reverse mode of automatic differentiation which are the hyperparameters tuned for the mode. The following diagram shows a simple structure of the MLP Algorithm:



MLPs are very efficient in case of Binary Class Models, but it doesn't mean that they can't handle Multiclass Models. It's just that the execution time for identifying the hyperparameters and training the model might take quite the bit of time.

K-Nearest Neighbours (KNN):

The KNN Algorithm is based on the concept of clustering, which identifies the similarities between the input data and places the new data (the test data) based on its similarity between the new and the existing clusters. This algorithm is known for the concept of 'lazy-learning', which means that it does not immediately learn from the training data and instead stores and performs the prediction during the time of classification. The following image gives a structural flow of the KNN Algorithm:



So, now that all our SML algorithms have been explained, we now will go through the 3 Cross Validation Techniques that we used on all the algorithms to decide on the best CV value to tune the hyperparameters:

K-Fold Cross Validation Algorithm:

The k-Fold CV Technique is a technique that gives us an understanding on how efficient/skilful the ML Algorithm can train the model under the given circumstances (hyperparameters). It is normally deployed on a small sample of unseen data and provides us with an estimate as to how will the model perform while making predictions based on the input test data. The higher the value of CV, more efficient and accurate the model is in identifying the True and the Predicted Labels of the Training model.

Stratified K-Fold Cross Validation:

Stratified k-fold cross-validation is the same as the normal k-fold cross-validation technique, but instead of Random Sampling (as in the case of normal k-fold), it uses the concept of Stratified Sampling. The folds are made by implementing the concept called "Percentage Class Preservation" under each class and therefore, in cases where normal k-fold have proven to provide poor accuracy scores (datasets with highly imbalanced data), the Stratified k-fold technique tend to provide better scores.

Shuffle Split Cross Validation Algorithm:

The Shuffle Split CV Algorithm uses the concept of "Repeated Random Subsampling Validation" to identify the scores for the given input. The number of iterations is decided upon analysis of the input and the results are then averaged over the splits. This algorithm is usually not preferred for imbalanced sets because of its Randomised Subsampling nature so the Stratified counterpart of this algorithm can be used in such situations.

Exploratory Data Analysis [EDA], Encoding, StandardScaling and train_test_split was performed on the STRIDE Dataset (refer Source Code for the same) and the above 3 Cross Validation techniques were deployed to identify the Mean Accurate Scores for each training algorithm. The following results were obtained:

SML Algorithm	**k-Fold**	**Stratified k-Fold**	**Shuffle Split**
Decision Tree Classifier	0.97785	0.97757	0.97523
Gaussian Naïve Bayes	-	-	-
K-Nearest Neighbours	0.97185	0.97214	0.97047
Logistic Regression	0.94828	0.94614	0.94458
Multilayer Perceptron	0.97500	0.97700	0.97489
Random Forest Classifier	0.98400	0.98400	0.98277
Support Vector Machine	0.95014	0.94957	0.94718

Table 3.1: Mean Accurate Scores of the different CV Techniques on the 7 Algorithms chosen

Accordingly, we used the best Cross Validation Technique (highlighted in yellow) for tuning the hyperparameters using the GridSearchCV() function and used those hyperparameters to train the models. The Scores (in %) [Train-Valid Score / Train-Test Score] are as follows:

	DTC	**GNB**	**KNN**	**LogReg**	**MLP**	**RFC**	**SVM**
Accuracy Scores	97/97	84/83	97/97	97/96	97/97	98/98	96/96

Table 3.2: Accuracy Scores of the 7 SML Algorithms after training and testing the STRIDE Dataset

Source Code with Code Output

STRIDE Based Cyber Threat Intelligence Model using Machine Learning

```
[1]: # Importing all the necessary Modules
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

```
[2]: # Loading the Dataset from Storage
     STRIDE = pd.read_csv("STRIDE.csv")
```

```
[3]: # Displaying the Dimensionality of the Dataset
     STRIDE.shape
```

```
[3]: (7000, 43)
```

```
[4]: # Checking the Categorical Unique Values of the Decisional Attribute
     STRIDE['label'].unique()
```

```
[4]: array(['Information Disclosure', 'Repudiation', 'Normal', 'Spoofing',
            'Escalation of Privileges', 'DoS', 'Tampering'], dtype=object)
```

```
[5]: # Since there was an unnecessary Column in the Dataset, removing that column
     STRIDE = STRIDE.drop(columns='Unnamed: 0')
```

```
[6]: # Displaying the Dimensionality of the Dataset
     STRIDE.shape
```

```
[6]: (7000, 42)
```

```
[7]: # Changing the Decisional Labels to Numerical Equivalents
     # 0 -> Spoofing      3 -> Info Disclosure
     # 1 -> Tampering     4 -> DoS
     # 2 -> Repudiation   5 -> Escalation of Privileges   6 -> Normal
```

```
[8]: STRIDE['label'] = STRIDE['label'].map({'Spoofing': 0, 'Tampering': 1,␣
       ↪'Repudiation': 2,
                                            'Information Disclosure':3, 'DoS':4,␣
       ↪'Escalation of Privileges':5, 'Normal':6})
```

[9]:
```
# Checking the Numerical Unique Values of the Decisional Attribute
STRIDE['label'].unique()
```

[9]:
```
array([3, 2, 6, 0, 5, 4, 1], dtype=int64)
```

[10]:
```
# Splitting the Dataset based on its Conditional & Decisional Attributes
X = STRIDE.iloc[:,:-1]
y = STRIDE.iloc[:,-1]
```

[11]:
```
# Performing Standard Scaling on the Input Dataset
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(X)
print(X)
```

```
[[-0.13281325  0.31209268 -0.77010519 … -0.12551435 -0.34682057
  -0.33486729]
 [-0.13281325  0.31209268  0.65016483 … -0.12551435  3.04967565
   3.04037357]
 [-0.13281325  0.31209268  1.32648389 … -0.12551435 -0.34682057
  -0.33486729]
 …
 [-0.13281325  0.31209268  0.98832436 … -0.12551435 -0.34682057
  -0.33486729]
 [-0.03262808  0.31209268  1.73227533 … -0.12551435 -0.34682057
  -0.33486729]
 [-0.06195057  0.31209268  1.73227533 … -0.12551435 -0.34682057
  -0.33486729]]
```

[12]:
```
# Splitting the Dataset into Train, Test and Validation Datasets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33,␣
 ↪random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,␣
 ↪test_size=0.33, random_state=42)
```

[13]:
```
# Algorithm 1: Decision Tree Classifier
```

[14]:
```
# Creating a DecisionTreeClassifier for identifying the Hyperparamters
from sklearn.tree import DecisionTreeClassifier

# Create a DecisionTreeClassifier object
dtc = DecisionTreeClassifier(random_state=42)
```

[15]:
```
# Now, we are going to perform the following 3 Cross-Validation Technqiues for␣
 ↪all the Algorithms:
# 1. k-Fold Cross Validation
# 2. Stratified Cross Validation
# 3. ShuffleSplit Cross Validation
```

[16]:
```python
from sklearn.model_selection import cross_val_score, KFold

kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(dtc, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
Mean accuracy: 0.9778571428571429
```

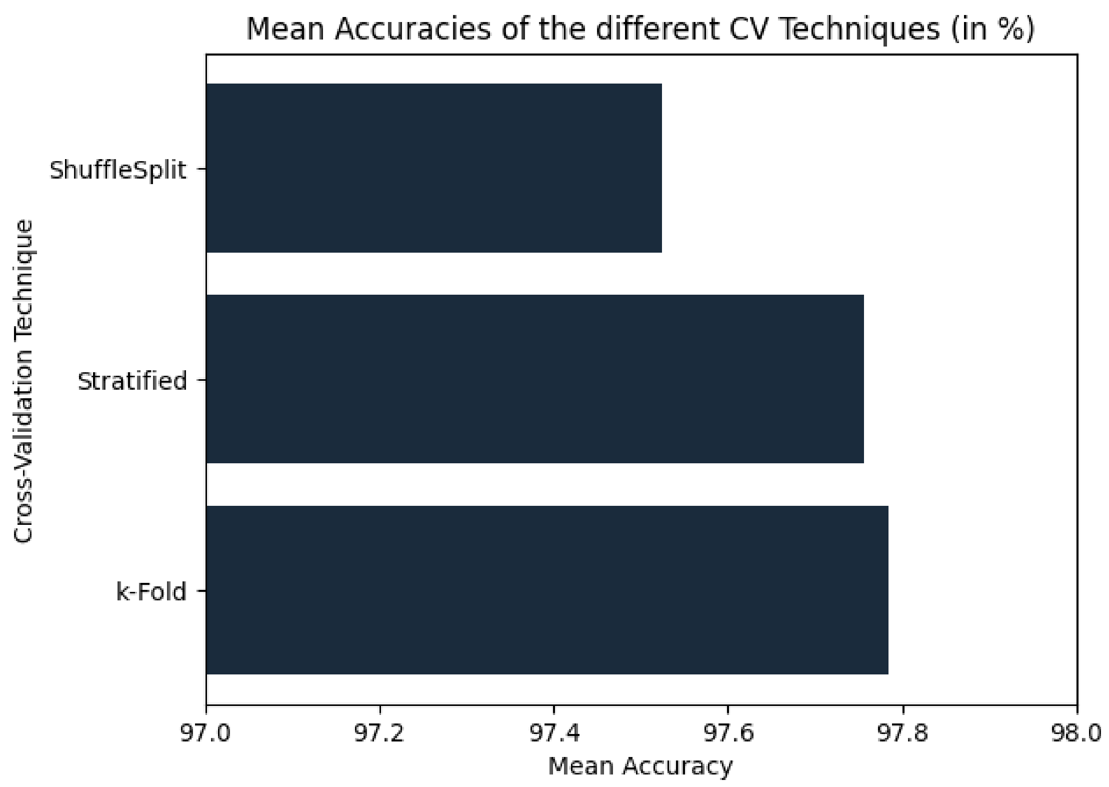
[17]:
```python
from sklearn.model_selection import StratifiedKFold

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(dtc, X, y, cv=skf)
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
Mean accuracy: 0.9775714285714285
```

[18]:
```python
from sklearn.model_selection import ShuffleSplit

ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(dtc, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

```
Mean accuracy: 0.9752380952380953
```

[19]:
```python
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(97,98)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[20]:
```
# From the above Bar Chart, we know that the best Cross Validation
# Technique is k-fold. We use that to identify the optimal HyperParameters.
```

[21]:
```
# Defining the Hyperparameter Space for our Dataset with the Identified CVScore
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters grid
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'criterion': ['gini', 'entropy']
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(estimator=dtc, param_grid=param_grid, cv=kf)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 15,
'min_samples_split': 2}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.972630704889978
```

[22]:
```
# Creating a DecisionTreeClassifier with the Best Hyperparameters
best_params = grid_search.best_params_
dtc = DecisionTreeClassifier(**best_params, random_state=42)
```

[23]:
```
# Training the Dataset
from sklearn.metrics import classification_report

dtc.fit(X_train, y_train)
y_vpred = dtc.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

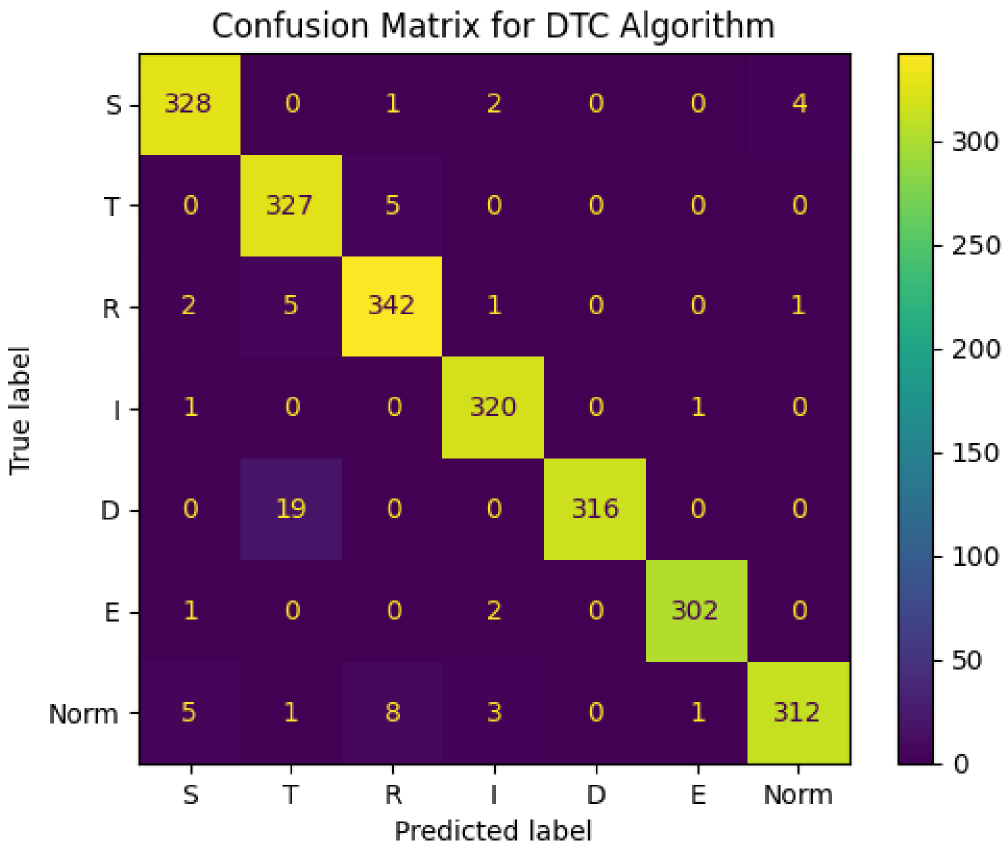
```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       1.00      0.98      0.99       213
           1       0.93      0.99      0.95       213
           2       0.96      0.96      0.96       213
           3       0.98      0.97      0.98       237
           4       1.00      0.97      0.98       201
           5       0.97      0.99      0.98       222
           6       0.98      0.96      0.97       249

    accuracy                           0.97      1548
   macro avg       0.97      0.97      0.97      1548
weighted avg       0.97      0.97      0.97      1548
```

[24]:
```
y_tpred = dtc.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.97      0.98      0.98       335
           1       0.93      0.98      0.96       332
           2       0.96      0.97      0.97       351
           3       0.98      0.99      0.98       322
           4       1.00      0.94      0.97       335
           5       0.99      0.99      0.99       305
           6       0.98      0.95      0.96       330
```

```
    accuracy                           0.97      2310
   macro avg       0.97      0.97      0.97      2310
weighted avg       0.97      0.97      0.97      2310
```

[25]:
```
# Plotting the Confusion Matrix for the above Data
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for DTC Algorithm')
plt.show()
```



[26]:
```
# Algorithm 2: Gaussian Naive Bayes
```

[27]:
```
# Create the Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
```

```
gnb = GaussianNB(priors=None)
```

[28]:
```
# Training the Dataset
gnb.fit(X_train, y_train)
y_vpred = gnb.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

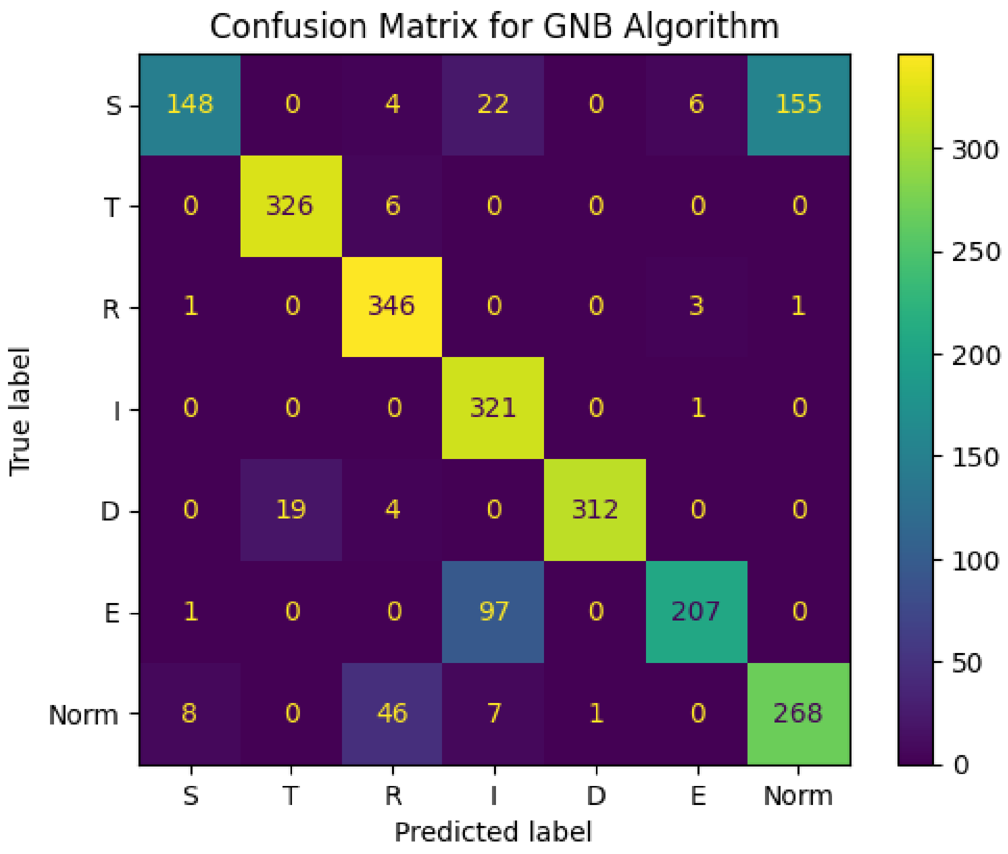
```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.93      0.41      0.57       213
           1       0.96      0.99      0.97       213
           2       0.82      1.00      0.90       213
           3       0.77      0.97      0.86       237
           4       1.00      0.96      0.98       201
           5       0.93      0.77      0.84       222
           6       0.65      0.78      0.71       249

    accuracy                           0.84      1548
   macro avg       0.87      0.84      0.83      1548
weighted avg       0.86      0.84      0.83      1548
```

[29]:
```
y_tpred = gnb.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.94      0.44      0.60       335
           1       0.94      0.98      0.96       332
           2       0.85      0.99      0.91       351
           3       0.72      1.00      0.83       322
           4       1.00      0.93      0.96       335
           5       0.95      0.68      0.79       305
           6       0.63      0.81      0.71       330

    accuracy                           0.83      2310
   macro avg       0.86      0.83      0.83      2310
weighted avg       0.86      0.83      0.83      2310
```

[30]:
```
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
```

```
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for GNB Algorithm')
plt.show()
```



[31]:
```
# Algorithm 3: K-Nearest Neighbours
```

[32]:
```
# Create a KNN classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

[33]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(knn, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
Mean accuracy: 0.9718571428571428
```

[34]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(knn, X, y, cv=skf)
```

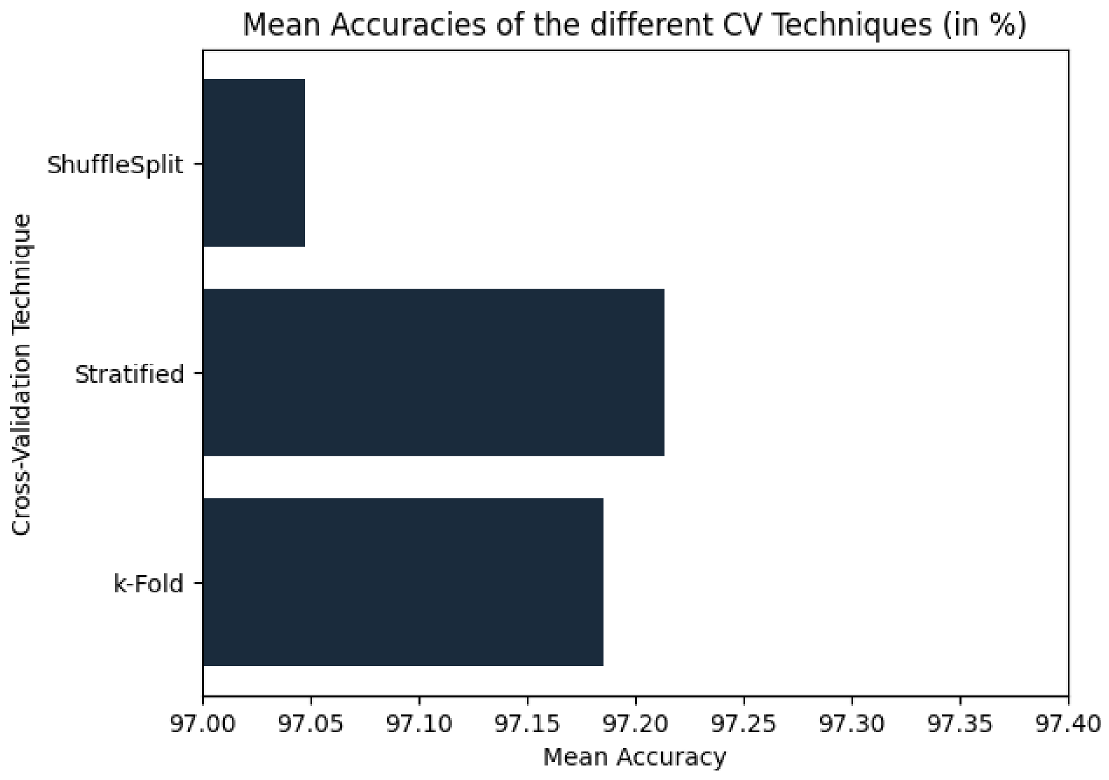
```
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
Mean accuracy: 0.9721428571428572
```

[35]:
```
ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(knn, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

```
Mean accuracy: 0.9704761904761906
```

[36]:
```
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(97.0,97.4)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[37]:
```
# From the above Bar Chart, we know that the best Cross Validation
# Technique is Stratified k-Fold. We use that to identify the optimal␣
 ↪HyperParameters.
```

[38]:
```
# Define the hyperparameters to tune
param_grid = {
    'n_neighbors': [2, 3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=skf)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
Best Hyperparameters: {'n_neighbors': 2, 'p': 1, 'weights': 'distance'}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.9751779692768828
```

[39]:
```
# Creating a KNNClassifier with the Best Hyperparameters
best_params = grid_search.best_params_
knn = KNeighborsClassifier(**best_params)
```

[40]:
```
# Training the Dataset
knn.fit(X_train, y_train)
y_vpred = knn.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

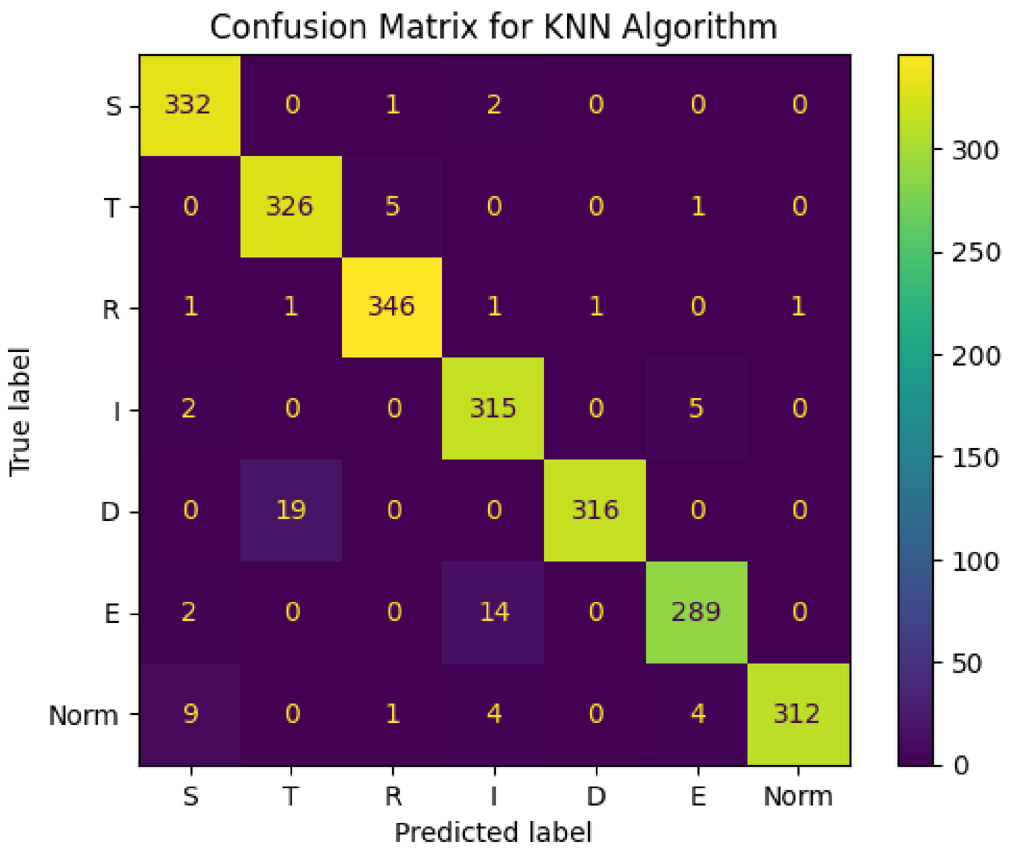
```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.98      0.98      0.98       213
           1       0.95      0.99      0.97       213
           2       0.97      0.98      0.98       213
           3       0.96      0.96      0.96       237
           4       1.00      0.97      0.98       201
           5       0.94      0.96      0.95       222
           6       0.99      0.97      0.98       249

    accuracy                           0.97      1548
   macro avg       0.97      0.97      0.97      1548
```

```
weighted avg       0.97      0.97      0.97      1548
```

[41]:
```
y_tpred = knn.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.96      0.99      0.98       335
           1       0.94      0.98      0.96       332
           2       0.98      0.99      0.98       351
           3       0.94      0.98      0.96       322
           4       1.00      0.94      0.97       335
           5       0.97      0.95      0.96       305
           6       1.00      0.95      0.97       330

    accuracy                           0.97      2310
   macro avg       0.97      0.97      0.97      2310
weighted avg       0.97      0.97      0.97      2310
```

[42]:
```
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for KNN Algorithm')
plt.show()
```



[43]:
```
# Algorithm 4: Logistic Regression
```

[44]:
```
# Creating a LogisticRegressor for identifying the Hyperparamters
from sklearn.linear_model import LogisticRegression

# Create the logistic regression model
logreg = LogisticRegression(max_iter=1000, random_state=42)
```

[45]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(logreg, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
Mean accuracy: 0.9482857142857142
```

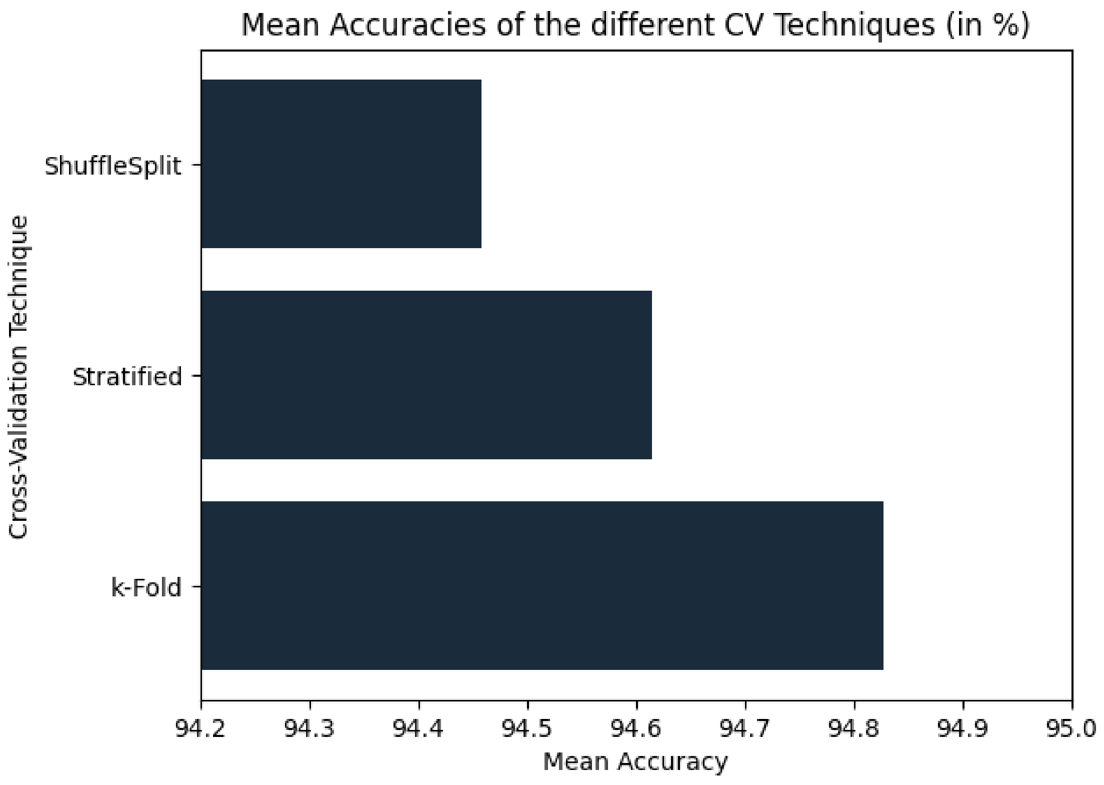
[46]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(logreg, X, y, cv=skf)
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
Mean accuracy: 0.9461428571428572
```

[47]:
```python
ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(logreg, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

```
Mean accuracy: 0.9445887445887446
```

[48]:
```python
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(94.2,95.0)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[49]:
```python
# From the above Bar Chart, we know that the best Cross Validation
# Technique is kFold. We use that to identify the optimal HyperParameters.
```

```
[50]: # Defining the Hyperparameter Space for our Dataset with the Identified CVScore
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l2'],
    'solver': ['newton-cg', 'sag', 'saga', 'lbfgs']
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(logreg, param_grid, cv=kf)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
```

```
regression
  n_iter_i = _check_optimize_result(

Best Hyperparameters: {'C': 100, 'penalty': 'l2', 'solver': 'newton-cg'}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.9605363887679361
```

[51]:
```
# Creating a LogisticRegressor with the Best Hyperparameters
best_params = grid_search.best_params_
logreg = LogisticRegression(**best_params, random_state=42)
```

[52]:
```
# Training the Dataset
logreg.fit(X_train, y_train)
y_vpred = logreg.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

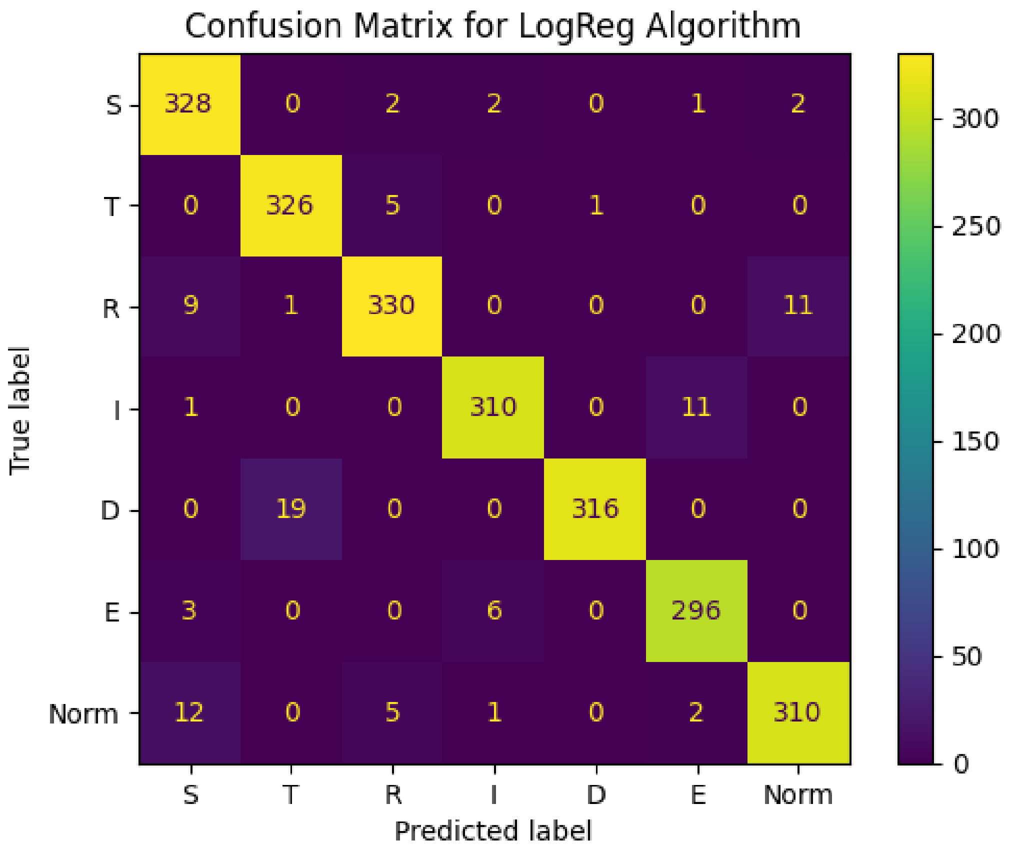
```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.95      0.97      0.96       213
           1       0.97      0.98      0.97       213
           2       0.97      0.97      0.97       213
           3       0.97      0.96      0.97       237
           4       0.98      0.97      0.97       201
           5       0.95      0.98      0.97       222
           6       0.97      0.93      0.95       249

    accuracy                           0.97      1548
   macro avg       0.97      0.97      0.97      1548
weighted avg       0.97      0.97      0.97      1548
```

[53]:
```
y_tpred = logreg.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.93      0.98      0.95       335
           1       0.94      0.98      0.96       332
           2       0.96      0.94      0.95       351
           3       0.97      0.96      0.97       322
           4       1.00      0.94      0.97       335
           5       0.95      0.97      0.96       305
           6       0.96      0.94      0.95       330
```

```
    accuracy                           0.96      2310
   macro avg       0.96      0.96      0.96      2310
weighted avg       0.96      0.96      0.96      2310
```

[54]:
```python
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for LogReg Algorithm')
plt.show()
```



[55]:
```python
# Algorithm 5: Multilayer Perceptron Algorithm
```

[56]:
```python
# Creating a MultiLayer Perceptron Model for identifying the Hyperparamters
from sklearn.neural_network import MLPClassifier

# Define the MLP classifier
mlp = MLPClassifier(random_state=42)
```

[57]:
```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(mlp, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
Mean accuracy: 0.975
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[58]:
```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(mlp, X, y, cv=skf)
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
Mean accuracy: 0.977
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[59]:
```
ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(mlp, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

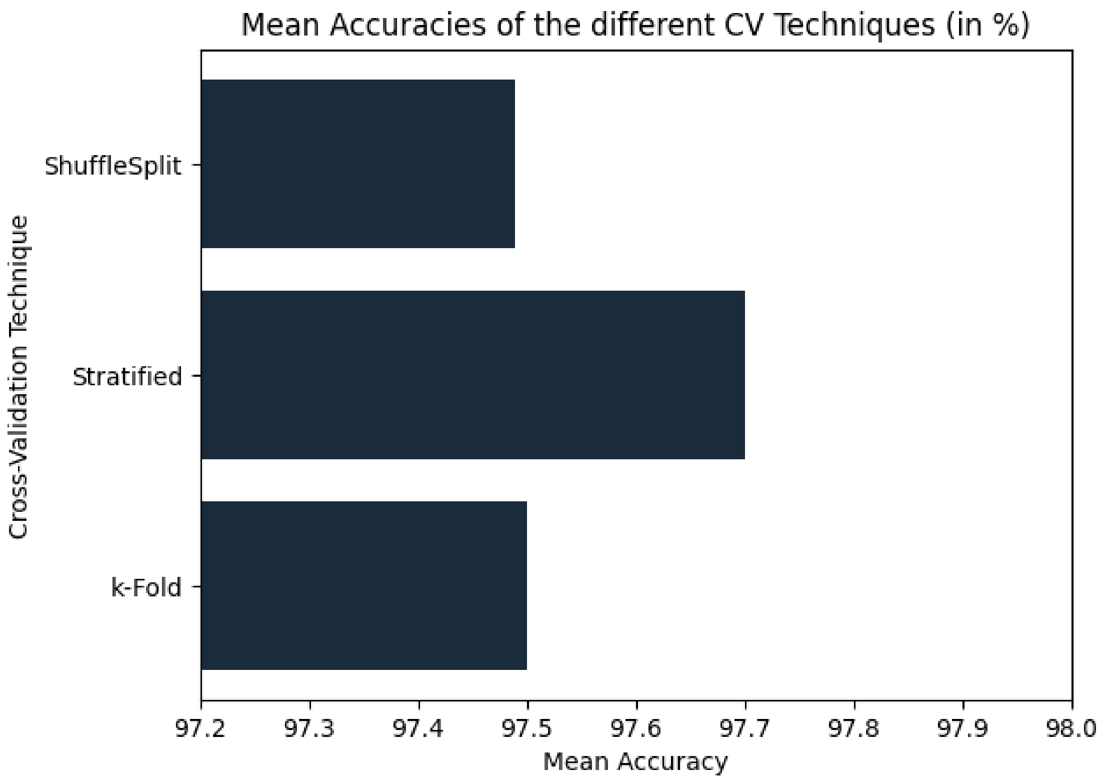
```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
Mean accuracy: 0.9748917748917749
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:684:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
```

```
the optimization hasn't converged yet.
  warnings.warn(
```

[60]:
```
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(97.2,98)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[61]:
```
# From the above Bar Chart, we know that the best Cross Validation
# Technique is Stratified kFold. We use that to identify the optimal␣
↪HyperParameters.
```

[62]:
```
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'hidden_layer_sizes': [(50,), (100,)],
    'activation': ['logistic', 'tanh'],
    'solver': ['adam', 'lbfgs'],
    'max_iter': [100, 200],
    'learning_rate': ['constant', 'adaptive']
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(mlp, param_grid, cv=skf, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:\n", grid_search.best_params_, sep="")
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
Best Hyperparameters:
{'activation': 'logistic', 'hidden_layer_sizes': (100,), 'learning_rate':
'constant', 'max_iter': 200, 'solver': 'lbfgs'}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.9716757971909715
```

```
C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[63]:
```
# Creating a MLPClassifier with the Best Hyperparameters
best_params = grid_search.best_params_
mlp = MLPClassifier(**best_params, random_state=42)
```

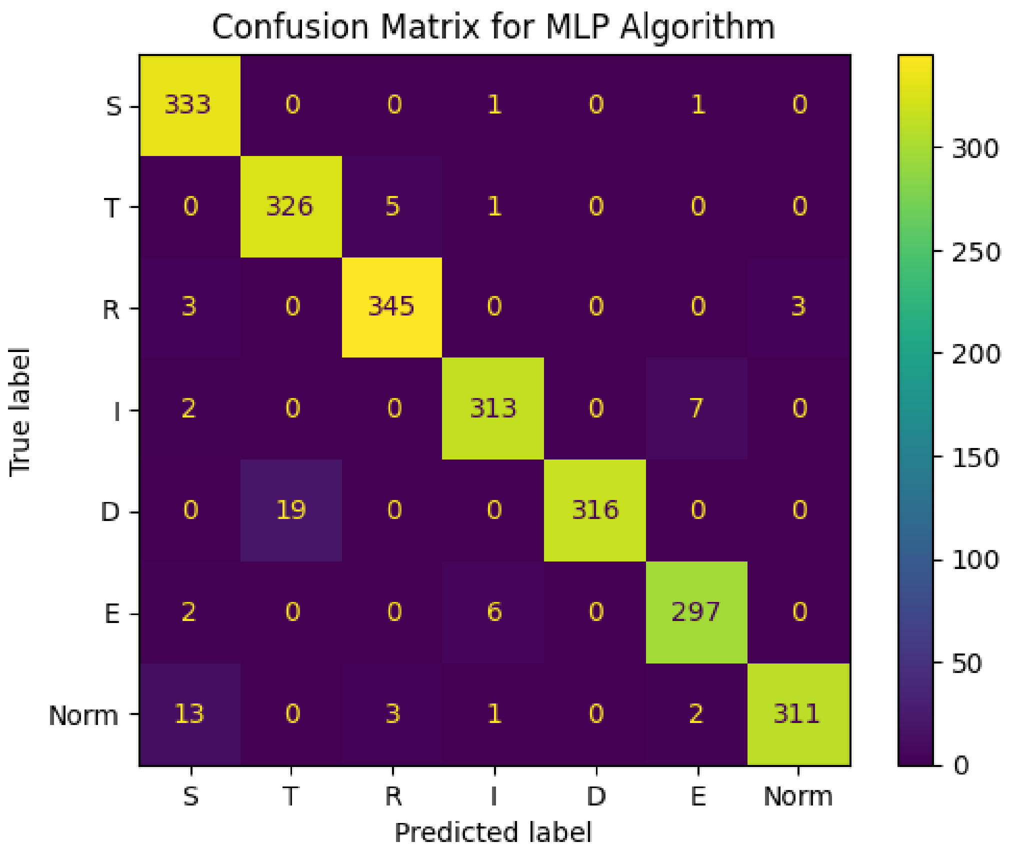
[64]:
```
# Training the Dataset
mlp.fit(X_train, y_train)
y_vpred = mlp.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.95      0.98      0.97       213
           1       0.96      0.99      0.97       213
           2       0.99      0.98      0.98       213
           3       0.97      0.95      0.96       237
           4       0.98      0.97      0.97       201
           5       0.94      0.98      0.96       222
           6       0.99      0.94      0.96       249

    accuracy                           0.97      1548
   macro avg       0.97      0.97      0.97      1548
weighted avg       0.97      0.97      0.97      1548

C:\Users\visha\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:541:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

[65]:
```python
y_tpred = mlp.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.94      0.99      0.97       335
           1       0.94      0.98      0.96       332
           2       0.98      0.98      0.98       351
           3       0.97      0.97      0.97       322
           4       1.00      0.94      0.97       335
           5       0.97      0.97      0.97       305
           6       0.99      0.94      0.97       330

    accuracy                           0.97      2310
   macro avg       0.97      0.97      0.97      2310
weighted avg       0.97      0.97      0.97      2310
```

[66]:
```python
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for MLP Algorithm')
plt.show()
```



[67]:
```
# Algortihm 6: Random Forest Classifier
```

[68]:
```
# Creating a RandomForestClassifier for identifying the Hyperparamters
from sklearn.ensemble import RandomForestClassifier
RFC = RandomForestClassifier(random_state=42)
```

[69]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(RFC, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
Mean accuracy: 0.984
```

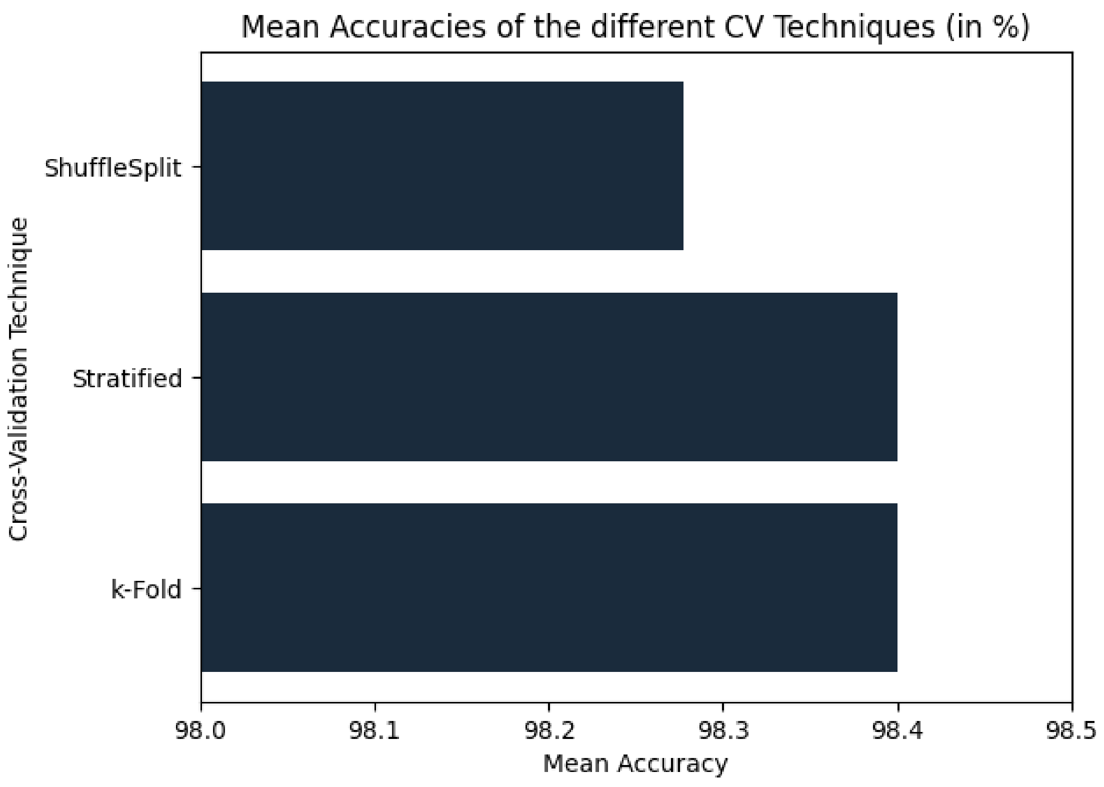
[70]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(RFC, X, y, cv=skf)
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
Mean accuracy: 0.984
```

[71]:
```
ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(RFC, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

```
Mean accuracy: 0.9827705627705627
```

[72]:
```
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(98,98.5)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[73]:
```
# From the above Bar Chart, we know that the best Cross Validation
# Technique is k-Fold. We use that to identify the optimal HyperParameters.
```

[74]:
```
# Defining the Hyperparameter Space for our Dataset with the Identified CVScore
param_grid = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_depth': [5, 10, 15, 20, 25],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4] }

# Performing Grid Search to find the best Hyperparameters
grid_search = GridSearchCV(estimator=RFC, param_grid=param_grid, cv=kf)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
Best Hyperparameters: {'max_depth': 15, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 500}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.9789970937591768
```

[75]:
```
# Creating a RandomForestClassifier with the Best Hyperparameters
best_params = grid_search.best_params_
RFC = RandomForestClassifier(**best_params, random_state=42)
```

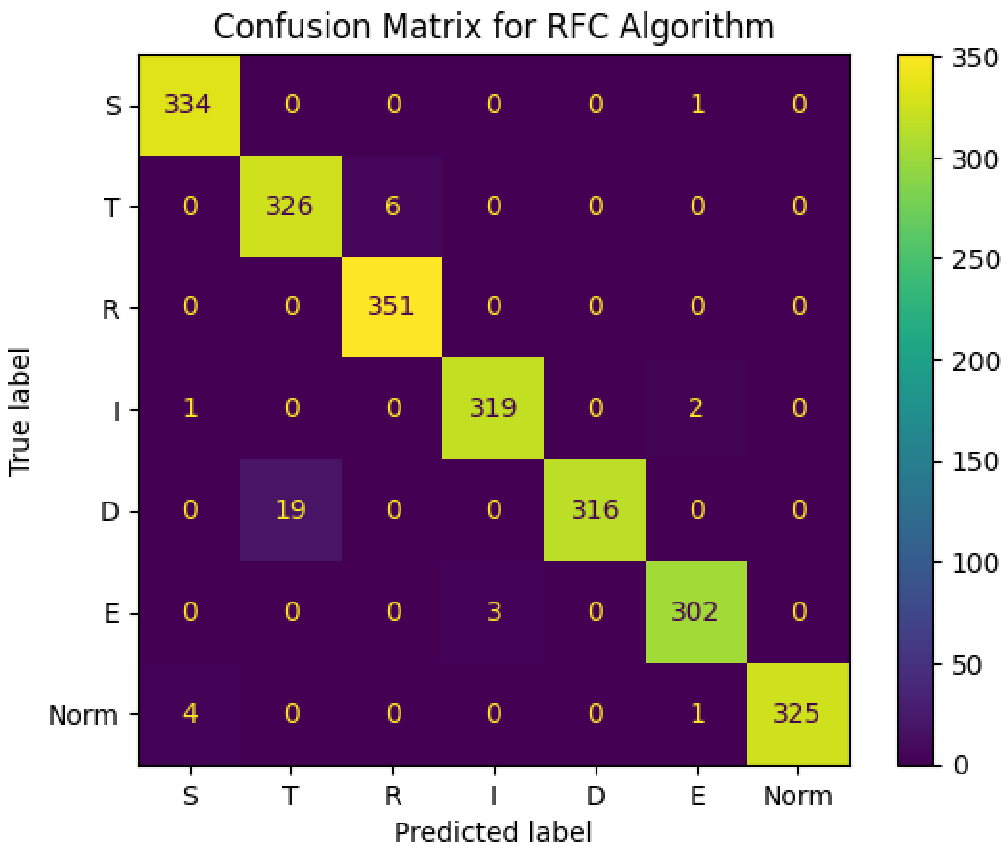
[76]:
```
RFC.fit(X_train, y_train)
y_vpred = RFC.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.99      0.98      0.99       213
           1       0.97      0.99      0.98       213
           2       0.98      1.00      0.99       213
           3       0.99      0.96      0.97       237
           4       1.00      0.97      0.98       201
           5       0.96      0.99      0.98       222
           6       0.99      0.99      0.99       249

    accuracy                           0.98      1548
   macro avg       0.98      0.98      0.98      1548
weighted avg       0.98      0.98      0.98      1548
```

[77]:
```
y_tpred = RFC.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.99      1.00      0.99       335
           1       0.94      0.98      0.96       332
           2       0.98      1.00      0.99       351
           3       0.99      0.99      0.99       322
           4       1.00      0.94      0.97       335
           5       0.99      0.99      0.99       305
           6       1.00      0.98      0.99       330

    accuracy                           0.98      2310
   macro avg       0.98      0.98      0.98      2310
weighted avg       0.98      0.98      0.98      2310
```

[78]:
```
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for RFC Algorithm')
plt.show()
```



[79]:
```
# Algorithm 7: Support Vector Machine
```

[80]:
```
# Create an SVM classifier
from sklearn.svm import SVC
svm = SVC(random_state=42)
```

[81]:
```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(svm, X, y, cv=kf)
kfscore = scores.mean()
print("Mean accuracy:", kfscore)
```

```
Mean accuracy: 0.9501428571428571
```

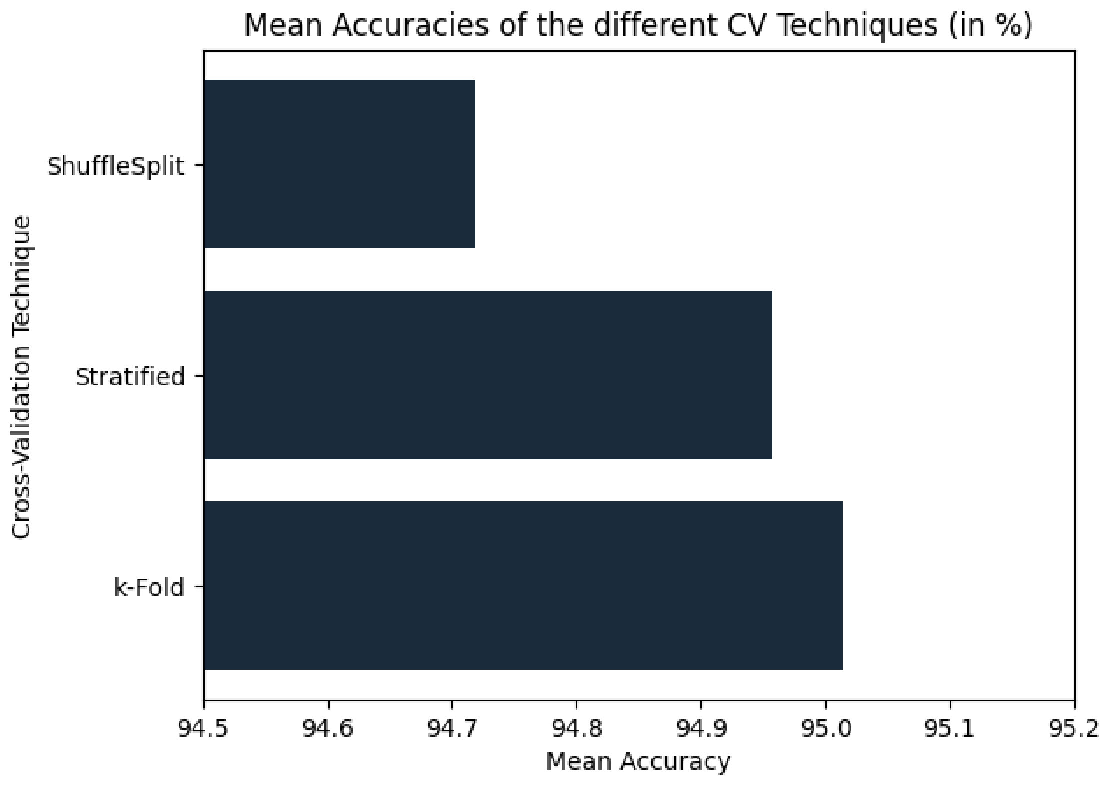
[82]:
```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(svm, X, y, cv=skf)
skfscore = scores.mean()
print("Mean accuracy:", skfscore)
```

```
Mean accuracy: 0.9495714285714285
```

[83]:
```
ss = ShuffleSplit(n_splits=5, test_size=0.33, random_state=42)
scores = cross_val_score(svm, X, y, cv=ss)
ssscore = scores.mean()
print("Mean accuracy:", ssscore)
```

```
Mean accuracy: 0.9471861471861471
```

[84]:
```
# Plotting a Bar Chart for the Mean Accuracies

Names = ['k-Fold', 'Stratified', 'ShuffleSplit']
Values = [kfscore*100, skfscore*100, ssscore*100]
plt.title(f"Mean Accuracies of the different CV Techniques (in %)")
plt.xlabel(f"Mean Accuracy")
plt.ylabel(f"Cross-Validation Technique")
plt.xlim(94.5,95.2)
plt.barh(Names, Values, color="#1a2b3c")
plt.show()
```



[85]:
```
# From the above Bar Chart, we know that the best Cross Validation
# Technique is kFold. We use that to identify the optimal HyperParameters.
```

[86]:
```
# Defining the Hyperparameter Space for our Dataset with the Identified CVScore
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['sigmoid', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(svm, param_grid, cv=kf)
grid_search.fit(X_train, y_train)

# Printing the best Hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Number of Cross Val Score:",  grid_search.n_splits_)
print("Best Mean Cross-Validation Score:", grid_search.best_score_)
```

```
Best Hyperparameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Best Number of Cross Val Score: 5
Best Mean Cross-Validation Score: 0.9573567385294623
```

[87]:
```
# Creating a SVM with the Best Hyperparameters
best_params = grid_search.best_params_
svm = SVC(**best_params, random_state=42)
```

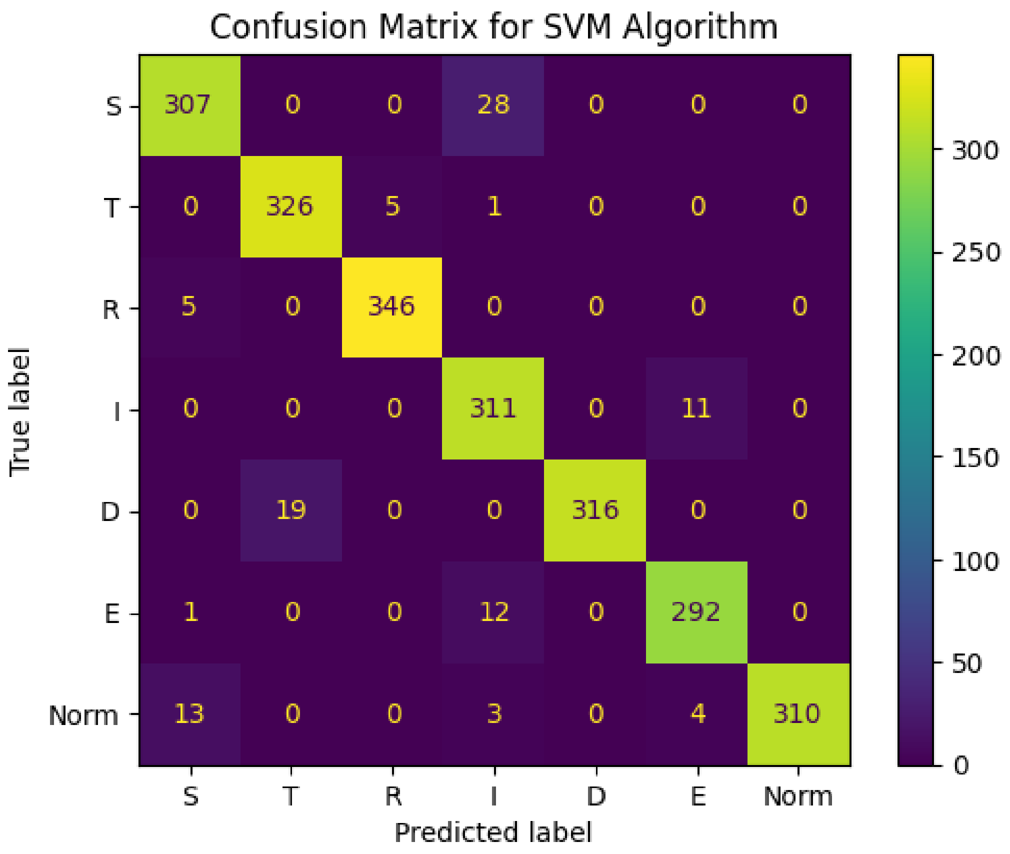
[88]:
```
svm.fit(X_train, y_train)
y_vpred = svm.predict(X_valid)
print("Classification Report between Train & Validation Sets:\n")
print(classification_report(y_valid, y_vpred))
```

```
Classification Report between Train & Validation Sets:

              precision    recall  f1-score   support

           0       0.96      0.90      0.93       213
           1       0.97      0.99      0.98       213
           2       0.98      0.99      0.98       213
           3       0.89      0.96      0.92       237
           4       1.00      0.97      0.98       201
           5       0.95      0.97      0.96       222
           6       1.00      0.96      0.98       249

    accuracy                           0.96      1548
   macro avg       0.96      0.96      0.96      1548
weighted avg       0.96      0.96      0.96      1548
```

[89]:
```
y_tpred = svm.predict(X_test)
print("Classification Report between Train & Test Sets:\n")
print(classification_report(y_test, y_tpred))
```

```
Classification Report between Train & Test Sets:

              precision    recall  f1-score   support

           0       0.94      0.92      0.93       335
           1       0.94      0.98      0.96       332
           2       0.99      0.99      0.99       351
           3       0.88      0.97      0.92       322
           4       1.00      0.94      0.97       335
           5       0.95      0.96      0.95       305
           6       1.00      0.94      0.97       330

    accuracy                           0.96      2310
   macro avg       0.96      0.96      0.96      2310
weighted avg       0.96      0.96      0.96      2310
```

[90]:
```
# Plotting the Confusion Matrix for the above Data
cnf_matrix = confusion_matrix(y_test, y_tpred)
labels = ['S', 'T', 'R', 'I', 'D', 'E', 'Norm']
ConfusionMatrixDisplay(cnf_matrix, display_labels = labels).plot()
plt.title('Confusion Matrix for SVM Algorithm')
plt.show()
```



	S	T	R	I	D	E	Norm
S	307	0	0	28	0	0	0
T	0	326	5	1	0	0	0
R	5	0	346	0	0	0	0
I	0	0	0	311	0	11	0
D	0	19	0	0	316	0	0
E	1	0	0	12	0	292	0
Norm	13	0	0	3	0	4	310

CONCLUSION

Based on multiple rounds of hyperparameter tunings, repeated improvisation of our Dataset, training the models and inferring the Confusion Matrix visualised, we found out that the **Random Forest Classifier** algorithm performs the best when compared to the others (highlighted in yellow in the table given before the Source Code Part). With an accuracy score of 98%, the Cyber Threat Intelligence Model on STRIDE would provide us the best results if trained using the Random Forest Algorithm. Even though the algorithm takes quite the bit of computational time for execution, the end-result, which is the accuracy obtained from training the model, effectively trades off for the same.

FUTURE PLANS

Throughout the course of this project, we used a customised dataset that we obtained from the standardised NSL-KDD Dataset. So, moving forward, we would like to perform the following as a continuation of our current findings:

- Simulate all the 6 attack vectors of STRIDE;
- Intercept them through Wireshark Packet Analyser Tool ;
- Perform Feature Extraction with adherence to the NSL-KDD Dataset;
- Use that dataset to test the Model.

In that way, we could get a more "real-life" interpretation on how the Model fares when deployed as a complete Standalone Threat Intelligent Model [STIM] in network architectures all across the world. Also, if the above-mentioned proposed idea is executed successfully, the next step would be coming up with the prototype for the STIM System that can effectively detect and warn the organisations on potential attacks so that effective countermeasures can be deployed immediately and action can be taken to minimise / mitigate potential data / financial loses the organisation might incur from the same.

REFERENCES

The following Resource Websites of interest were used along the course of the project:

https://scikit-learn.org/
[Documentation for the Models used for the algorithm were accessed here so that we can tune the hyperparameters accordingly to get maximum accuracy, were obtained here.]

https://pandas.pydata.org/docs/
[Documentation pertaining to Exploratory Data Analysis to be performed on the NSL-KDD Dataset were obtained here to create our customised STRIDE Dataset.]

https://matplotlib.org/stable/index.html
[Documentation for plotting Confusion Matrices and Bar Charts were obtained here.]

https://www.ibm.com/topics/machine-learning
[We learnt a lot about the Supervised Machine Learning Algorithms from here.]

https://www.analyticsvidhya.com/
[We learnt about the Different Cross Validation Techniques available from here.]

https://medium.com/@randylaosat/a-beginners-guide-to-machine-learning-dfadc19f6caf
[The Summary for this report was prepared based on inputs from this website.]

https://www.unb.ca/cic/datasets/nsl.html
[The NSL-KDD Dataset was downloaded from this website.]

https://towardsdatascience.com/a-deeper-dive-into-the-nsl-kdd-data-set-15c753364657
[We learnt about the different features and significance of the NSL-KDD Dataset from here.

Base Paper References:

1) Hernan, S.; Lambert, S.; Ostwald, T.; Shostack, A. Threat Modeling—Uncover Security Design Flaws Using the STRIDE Approach.

2) MSDN Mag. 2006. Available online: https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncoversecurity-design-flaws-using-the-stride-approach (accessed on 26 July 2022).

3) Wang, Q.; Guo,W.; Zhang, K.; Ororbia, A.G.; Xing, X.; Liu, X.; Giles, C.L. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1145–1153.

4) Rosenberg, I.; Shabtai, A.; Elovici, Y.; Rokach, L. Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain. ACM Comput. Surv. 2021, 54, 1–36.

APPENDIX

<u>Base Paper:</u>

Mauri, L. and Damiani, E., 2022. Modeling Threats to AI-ML Systems Using STRIDE. Sensors, 22(17), p.6662. [Site: https://www.mdpi.com/1424-8220/22/17/6662]

<u>With references from the following Paper:</u>

Mauri, L. and Damiani, E., 2021, July. Stride-AI: An approach to identifying vulnerabilities of machine learning assets. In 2021 IEEE International Conference on Cyber Security and Resilience (CSR) (pp. 147-154). IEEE. [Site: https://ieeexplore.ieee.org/document/9527917]