National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cybersecurity

Tarankaarthikeshwar Srinivasan
Student ID: 23267941

School of Computing
National College of Ireland

Supervisor: Michael Pantridge

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Tarankaarthikeshwar Srinivasan
Student ID:	23267941
Programme:	MSc in Cybersecurity **Year:** 2025
Module:	MSc Research Project
Lecturer:	Michael Pantridge
Submission Due Date:	15/09/2025
Project Title:	Novel AI-Driven Approach to Cloud Intrusion Detection Using Boruta-PSO Feature Selection
Word Count:	1357 **Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Tarankaarthikeshwar Srinivasan

Date: 15/09/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Tarankaarthikeshwar Srinivasan
Student ID: 23267941

1 Introduction

This configurational manual provides details on a Novel AI-Driven Approach to cloud intrusion detection using Boruta and PSO feature selection, the hardware and software requirements to recreate this project in any environment. This configurational manual will provide detailed information about the project code, the feature selection algorithms used, the models trained and the use of SMOTE to optimally solve the problem of class imbalance.

Project overview: Today, cloud computing is a critical part of the modern digital infrastructure, and it is frequently targeted in cyber-attacks. These environments need intrusion detection systems (IDS) with highly effective security control measures. This paper introduces a scalable and reliable IDS framework with a high-level feature engineering and data reduction approach to improve detection scores. The CICIDS 2017 dataset, with attack labels more specific and detail oriented, is re-labeled into general categories so to avoid complexity. A hybrid feature selection using Boruta and Particle Swarm Optimization (PSO) is employed to keep the top features only. Class Imbalance is handled via Synthetic Minority Over-Sampling Technique (SMOTE). In the end, a staged trained approach is adopted to make this updated model available for training multiple types of models such as Random Forest, LightGBM, Extra Tree, CatBoost, XGBoost.

2 Requirements

Hardware Requirements:

- Processor (Minimum): i5 to i7 (or equivalent).
- Memory (RAM): 8 to 16 GB
- Disk Space: 5 GB.

Software Requirements:

- Programming Language: Python 3.11
- Core-Python Modules:
    - **Scikit-learn** – machine learning model development and evaluation
    - **Pandas** – data handling and preprocessing
    - **Matplotlib** – data visualization and plotting
    - **Seaborn** – exploratory data visualization and analysis
- Additional Python Modules:

- **Boruta:** wrapper-based feature selection is implemented using random forest to select features.

```
In [ ]: !pip install boruta

Collecting boruta
  Downloading Boruta-0.4.3-py3-none-any.whl.metadata (8.8 kB)
Requirement already satisfied: numpy>=1.10.4 in /usr/local/lib/python3.11/dist-packages (from boruta) (2.0.2)
Requirement already satisfied: scikit-learn>=0.17.1 in /usr/local/lib/python3.11/dist-packages (from boruta) (1.6.1)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.11/dist-packages (from boruta) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.17.1->boruta) (1.
5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=0.17.1->boru
ta) (3.6.0)
Downloading Boruta-0.4.3-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.9/57.9 kB 2.0 MB/s eta 0:00:00
Installing collected packages: boruta
Successfully installed boruta-0.4.3
```

- **Particle Swarm Optimization (PSO):** used as the second factor for hybrid-feature selection, optimized the Boruta selected features on classification accuracy or F1-score.

```
In [ ]: !pip install pyswarms

Collecting pyswarms
  Downloading pyswarms-1.3.0-py2.py3-none-any.whl.metadata (33 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from pyswarms) (1.16.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from pyswarms) (2.0.2)
Requirement already satisfied: matplotlib>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from pyswarms) (3.10.0)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages (from pyswarms) (25.3.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from pyswarms) (4.67.1)
Requirement already satisfied: future in /usr/local/lib/python3.11/dist-packages (from pyswarms) (1.0.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages (from pyswarms) (6.0.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms) (0.1
2.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms)
(4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms)
(1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms) (2
5.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarms)
(3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=1.3.1->pyswarm
s) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib>=1.3.
1->pyswarms) (1.17.0)
Downloading pyswarms-1.3.0-py2.py3-none-any.whl (104 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.1/104.1 kB 3.6 MB/s eta 0:00:00
Installing collected packages: pyswarms
Successfully installed pyswarms-1.3.0
```

- **Synthetic Minority Over-sampling Technique (SMOTE: imblearn):** used to tackle the class imbalance issue by generating synthetic data points.

```
In [ ]: !pip install imblearn

Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (from imblearn) (0.13.0)
Requirement already satisfied: numpy<3,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblearn)
(2.0.2)
Requirement already satisfied: scipy<2,>=1.10.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblearn)
(1.16.0)
Requirement already satisfied: scikit-learn<2,>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblea
rn) (1.6.1)
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblea
rn) (0.1.3)
Requirement already satisfied: joblib<2,>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblearn)
(1.5.1)
Requirement already satisfied: threadpoolctl<4,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imble
arn) (3.6.0)
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

3 Project Implementation

3.1 Handling Dataset

Dataset Description: In this study, we have used the widely used dataset CICIDS 2017 as a benchmark for evaluating intrusion detection systems. It has benign network of traffic logs and adversarial logs of multiple attack types such as Brute Force, DoS/DDoS, Web Attacks, Infiltration, Botnet and Port Scans. Every record consists of more than 80 numerical and categorical features which summarize flow statistics, packet counts, timing attributes, as well the information on protocols. Popular or unique, more general attack group names are used to simplify analysis over specific detailed attacks. The preprocessing includes dropping missing values, label encoding for categorical features and Min-Max Normalization of numerical features. Using the Synthetic Minority Over-Sampling Technique (SMOTE) to address class imbalance, which create synthetic samples for each minority attack.

In [ ]:
```
#to unzip the dataset file

!unzip CICIDS2017.zip
```

```
Archive:  CICIDS2017.zip
  inflating: Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv
  inflating: Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv
  inflating: Friday-WorkingHours-Morning.pcap_ISCX.csv
  inflating: Monday-WorkingHours.pcap_ISCX.csv
  inflating: Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv
  inflating: Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv
  inflating: Tuesday-WorkingHours.pcap_ISCX.csv
  inflating: Wednesday-workingHours.pcap_ISCX.csv
```

In [ ]:
```
# extracting csv to dataframe

# with multiple file in multiple dataframes
data1 = pd.read_csv("Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv")
data2 = pd.read_csv("Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv")
data3 = pd.read_csv("Friday-WorkingHours-Morning.pcap_ISCX.csv")
data4 = pd.read_csv("Monday-WorkingHours.pcap_ISCX.csv")
data5 = pd.read_csv("Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv")
data6 = pd.read_csv("Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv")
data7 = pd.read_csv("Tuesday-WorkingHours.pcap_ISCX.csv")
data8 = pd.read_csv("Wednesday-workingHours.pcap_ISCX.csv")

#concatenating all dataframes to single dataframe

df = pd.concat([data1,data2])

df = pd.concat([df,data3])

df = pd.concat([df,data4])

df = pd.concat([df,data5])

df = pd.concat([df,data6])

df = pd.concat([df,data7])

df = pd.concat([df,data8])
```

There are eight CSV files within the CICIDS 2017 dataset, we are reading each CSV file and loading it into separate pandas data-frame. Then they were concatenating to stack the data-frames row-wise and gradually merging all the data-frames into a single data-frame.

In [ ]:
```
# Loading Dataset

Data=pd.read_csv("CICIDS2017.csv")
Data.head()
```

Out[4]:

	Destination Port	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	...	min_seg_size_forward	Active Mean	Active Std	Active Max	Active Min	Idl Mea
0	54865	3	2	0	12	0	6	6	6.0	0.0	...	20	0.0	0.0	0	0	0.
1	55054	109	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	0	0.
2	55055	52	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	0	0.
3	46236	34	1	1	6	6	6	6	6.0	0.0	...	20	0.0	0.0	0	0	0.
4	54863	3	2	0	12	0	6	6	6.0	0.0	...	20	0.0	0.0	0	0	0.

5 rows × 79 columns

In [ ]:
```
#checking duplicates

Data.duplicated().sum()
```

Out[4]:
```
np.int64(308381)
```

In [ ]:
```
# Removing duplicates

Data = Data.drop_duplicates()
Data.duplicated().sum()
```

Out[5]:
```
np.int64(0)
```

In [ ]:
```
# Finding null values, printing if count is >0
null_counts = Data.isnull().sum()
for column, count in null_counts.items():
  if count > 0:
    print(f"Column '{column}': {count} null values")
```

```
Column 'Flow Bytes/s': 353 null values
```

Removing any null values, duplicates within the combined dataset.

In [ ]:
```
#null values present in 'Flow Bytes/s'

#using fill na with mean to fill tha null values

Data.fillna({'Flow Bytes/s': Data['Flow Bytes/s'].mean()}, inplace=True)
Data['Flow Bytes/s'].isnull().sum()
```

Out[6]:
```
np.int64(0)
```

In [ ]:
```
#Types of labels
Data[' Label'].unique()
```

Out[14]:
```
array(['BENIGN', 'DDoS', 'PortScan', 'Bot', 'Infiltration',
       'Web Attack \ufffd Brute Force', 'Web Attack \ufffd XSS',
       'Web Attack \ufffd Sql Injection', 'FTP-Patator', 'SSH-Patator',
       'DoS slowloris', 'DoS Slowhttptest', 'DoS Hulk', 'DoS GoldenEye',
       'Heartbleed'], dtype=object)
```

In [ ]:
```
Data[' Label'].value_counts()
```

In [ ]:
```
Data[' Label'].value_counts()
```

Out[15]:

Label	count
BENIGN	2096484
DoS Hulk	172849
DDoS	128016
PortScan	90819
DoS GoldenEye	10286
FTP-Patator	5933
DoS slowloris	5385
DoS Slowhttptest	5228
SSH-Patator	3219
Bot	1953
Web Attack \ufffd Brute Force	1470
Web Attack \ufffd XSS	652
Infiltration	36
Web Attack \ufffd Sql Injection	21
Heartbleed	11

dtype: int64

There are 15 unique types of attacks within the dataset, some of the attack-types are very similar to each other, thus, we will be combining them together and reducing the label count to 7 including benign activity.

Lets reduce the labels:

Normal: 'BENIGN': 'Normal'

Botnet: 'Bot': 'Botnet'

Brute Force: 'FTP-Patator': 'Brute Force' 'SSH-Patator': 'Brute Force'

DoS/DDoS: 'DoS Hulk': 'DoS/DDoS' 'DoS GoldenEye': 'DoS/DDoS' 'DoS slowloris': 'DoS/DDoS' 'DoS Slowhttptest': 'DoS/DDoS' 'DDoS': 'DoS/DDoS' 'Heartbleed': 'DoS/DDoS'

Infiltration: 'Infiltration': 'Infiltration'

Port Scan: 'PortScan': 'Port Scan'

Web Attack: 'Web Attack – XSS': 'Web Attack' 'Web Attack – Sql Injection': 'Web Attack' 'Web Attack – Brute Force': 'Web Attack'

This grouping simplifies the labels while maintaining their core meanings, making it easier to categorize and understand the different types of threats and tools.

In [ ]:
```
#Labels updation
labels = {'BENIGN': 'Normal', 'Bot': 'Botnet', 'FTP-Patator': 'Brute Force', 'SSH-Patator': 'Brute Force', 'DoS Hulk': 'DoS/DDoS'
          'DoS GoldenEye': 'DoS/DDoS','DoS slowloris': 'DoS/DDoS', 'DoS Slowhttptest': 'DoS/DDoS', 'DDoS': 'DoS/DDoS',
          'Heartbleed': 'DoS/DDoS', 'Infiltration': 'Infiltration', 'PortScan': 'Port Scan','Web Attack \ufffd Brute Force': 'Web Att
          'Web Attack \ufffd XSS': 'Web Attack', 'Web Attack \ufffd Sql Injection': 'Web Attack' }
```

In [ ]:
```
Data[' Label'] = Data[' Label'].map(labels)
```

In [ ]:
```
Data[' Label'].unique()
```

Out[9]:
```
array(['Normal', 'DoS/DDoS', 'Port Scan', 'Botnet', 'Infiltration',
       'Web Attack', 'Brute Force'], dtype=object)
```

In [ ]:
```
Data[' Label'].value_counts()
```

Out[10]:

Label	count
Normal	2096484
DoS/DDoS	321775
Port Scan	90819
Brute Force	9152
Web Attack	2143
Botnet	1953
Infiltration	36

dtype: int64

In [ ]:
```
Data.shape
```

Out[20]:
```
(2522362, 79)
```

In [ ]:
```
# saving labeled save as csv file

Data.to_csv('/content/drive/MyDrive/CICIDS2017_processed.csv', index=False)
```

The above are the 7 final labels Normal, DoS/DDoS, Port Scan, Brute Force, Web Attack, Botnet, and Infiltration. The final dataset is saved upon which feature-selection would be applied.

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

print(Data[' Label'].unique())
Data[' Label'] = le.fit_transform(Data[' Label'])
print(Data[' Label'].unique())
```

```
['Normal' 'DoS/DDoS' 'Port Scan' 'Botnet' 'Infiltration' 'Web Attack'
 'Brute Force']
[4 2 5 0 3 6 1]
```

In [ ]:
```
Data[' Label'].value_counts()
```

Out[6]:

Label	count
4	2096484
2	321775
5	90819
1	9152
6	2143
0	1953
3	36

dtype: int64

Assigning a numeric value to the labels for the machine learning model, as ML models cannot process characters.

In [ ]:
```
# taking 10000 samples from classes which have more than 10000 also shuffling while extraction
# since some of the classes has minimal records

#4→2096484
#2→321775
#5→90819

from sklearn.utils import resample

# Separating majority and minority classes

#labels with 10000+
label_2 = Data[Data[' Label'] == 2]
label_4 = Data[Data[' Label'] == 4]
label_5 = Data[Data[' Label'] == 5]

#labels with less than 10000
label_0 = Data[Data[' Label'] == 0]
label_1 = Data[Data[' Label'] == 1]
label_3 = Data[Data[' Label'] == 3]
label_6 = Data[Data[' Label'] == 6]

# Downsampling majority class
label_2_sam_red = resample(label_2,
                                replace=False,    # sample without replacement
                                n_samples=10000,     # to match minority class
                                random_state=123) # reproducible results

label_4_sam_red = resample(label_4,
                                replace=False,    # sample without replacement
                                n_samples=10000,     # to match minority class
                                random_state=123)

label_5_sam_red = resample(label_5,
                                replace=False,    # sample without replacement
                                n_samples=10000,     # to match minority class
                                random_state=123)

# Combine minority class with downsampled majority class
df_balanced = pd.concat([label_0, label_1, label_3, label_6, label_2_sam_red, label_4_sam_red, label_5_sam_red])

# Display new class counts
df_balanced[' Label'].value_counts()
```

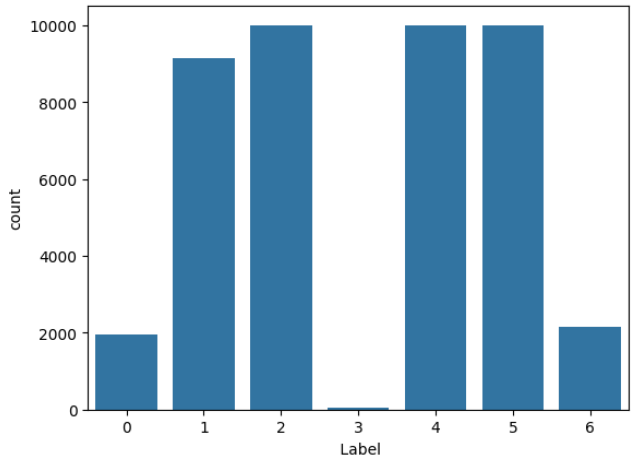
The above code is a balancing step to ensure the models trained are not biased. First we have identified classes with more than 10000 samples (Normal, DoS, Port Scan) and classes with less than 10000 samples (Brute Force, Web Attack, Botnet, Infiltration). Then we randomly down sample the classes and minority classes are retained in full to avoid data loss, finally the balanced subset is concatenated into a single data-frame, now the data distribution is more uniform.

3.2 Class Balancing and Feature selection

In [ ]:
```
#loading features and target

X=df_balanced.drop([' Label'],axis=1).values
y=df_balanced[' Label'].values
```

In [ ]:
```
import seaborn as sns
sns.countplot(x=' Label', data=df_balanced)
plt.show()
```



The above bar graph clearly shows the imbalance in the dataset, if the models are trained with this dataset they will be biased with the label which has the greatest number of samples, thus SMOTE is applied here to balance the dataset.

In [ ]:
```
# after facing the above problem
# ValueError: Input X contains infinity or a value too large for dtype('float64').

#checking for infinite values , replacing with nan and fill them
import pandas as pd
import numpy as np
# Check for infinite values in 'x'
inf_rows = np.isinf(X).any(axis=1)

# Replace infinite values with NaN
X[inf_rows] = np.nan

# Impute NaN values using a suitable method -> mean
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# again smote method
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=888)
X, y = smote.fit_resample(X, y)
pd.Series(y).value_counts()
```

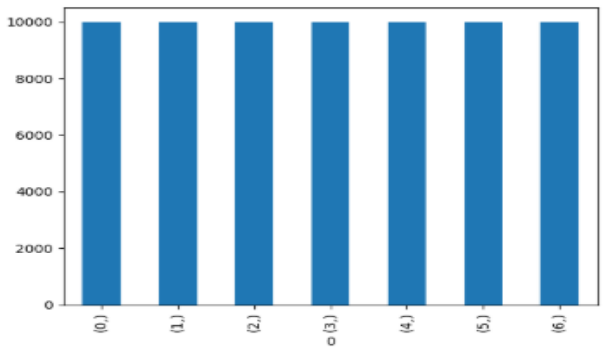
Out[14]:

	count
0	10000
1	10000
3	10000
6	10000
2	10000
4	10000
5	10000

dtype: int64

In [ ]:
```
pd.DataFrame(y).value_counts().plot(kind='bar')
```

Out[15]:
```
<Axes: xlabel='0'>
```



In the above code snippet, the infinite values are replaced with NaN, then the NaN values are replaced with the mean of their respective columns. After which SMOTE is applied which generates synthetic samples for minority classes by interpolating between samples. The application of SMOTE can be clearly seen in the bar graph showcasing balanced classes.

In [ ]:
```
# X, y to df

df = pd.DataFrame(X, columns=Data.columns.drop(' Label'))
df['Label'] = y
```

In [ ]:
```
# taking minimal samples for feature selection purpose

import pandas as pd
def undersample_to_desired(df, target_column, desired_count):
    undersampled_df = pd.DataFrame()
    for class_label in df[target_column].unique():
        class_df = df[df[target_column] == class_label]
        if len(class_df) > desired_count:
            class_undersampled = resample(class_df,
                                          replace=False,
                                          n_samples=desired_count,
                                          random_state=123) # reproducible results
        else:
            class_undersampled = class_df
        undersampled_df = pd.concat([undersampled_df, class_undersampled])
    return undersampled_df


desired_samples_per_class = 500
df_undersampled_500 = undersample_to_desired(df, 'Label', desired_samples_per_class)

# Display new class counts
df_undersampled_500['Label'].value_counts()
```

Out[17]:

Label	count
0	500
1	500
3	500
6	500
2	500
4	500
5	500

dtype: int64

After applying SMOTE we will be selecting 500 samples out of the 10000 samples upon which further feature selection will be applied to select the optimal features.

In [ ]:
```python
# Boruta Feature Selection
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy

# Boruta with Random Forest
rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
boruta = BorutaPy(estimator=rf, n_estimators='auto', max_iter=20, verbose=1, random_state=42)
boruta.fit(X.values, y)

# Selected features
boruta_selected = X.columns[boruta.support_].tolist()
X_boruta = X[boruta_selected]
print("✅ Boruta-selected features:", boruta_selected)
```

```
Iteration: 1 / 20
Iteration: 2 / 20
Iteration: 3 / 20
Iteration: 4 / 20
Iteration: 5 / 20
Iteration: 6 / 20
Iteration: 7 / 20
Iteration: 8 / 20
Iteration: 9 / 20
Iteration: 10 / 20
Iteration: 11 / 20
Iteration: 12 / 20
Iteration: 13 / 20
Iteration: 14 / 20
Iteration: 15 / 20
Iteration: 16 / 20
Iteration: 17 / 20
Iteration: 18 / 20
Iteration: 19 / 20


BorutaPy finished running.

Iteration:      20 / 20
Confirmed:      63
Tentative:      2
Rejected:       13
✅ Boruta-selected features: [' Destination Port', ' Flow Duration', ' Total Fwd Packets', ' Total Backward Packets', 'Total Le
ngth of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Min', ' Fwd Packet Length M
ean', ' Fwd Packet Length Std', 'Bwd Packet Length Max', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Leng
th Std', 'Flow Bytes/s', ' Flow Packets/s', ' Flow IAT Mean', ' Flow IAT Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Tota
l', ' Fwd IAT Mean', ' Fwd IAT Std', ' Fwd IAT Max', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IA
T Max', ' Bwd IAT Min', 'Fwd PSH Flags', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' Min P
acket Length', ' Max Packet Length', ' Packet Length Mean', ' Packet Length Std', ' Packet Length Variance', ' SYN Flag Count',
' PSH Flag Count', ' ACK Flag Count', ' URG Flag Count', ' Down/Up Ratio', ' Average Packet Size', ' Avg Fwd Segment Size', ' A
vg Bwd Segment Size', ' Fwd Header Length.1', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', ' Subflow Bwd Packets', ' Subflow Bw
d Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' act_data_pkt_fwd', ' min_seg_size_forward', 'Active Mean', '
Active Max', ' Active Min', 'Idle Mean', ' Idle Max', ' Idle Min']
```

BORUTA feature selection is applied to the selected 500 samples of data for each class. Random forest classifier is used as a base estimator for BORUTA, and the tree depth for this classifier is set to 5 to focus on main features that matter. The algorithm itself chooses the optimal number of trees and is iterated 20 times, each time a features importance is compared with random shadow features to decide its relevance. Finally, after 20 iteration we are left with 63 confirmed feature out of 80 features from the original dataset.

```
In [ ]: # PSO Feature Selection on Boruta Output
import pyswarms as ps
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# PSO fitness function
def pso_fitness(mask):
    mask = np.array(mask, dtype=bool)
    if np.count_nonzero(mask) == 0:
        return 1.0  # penalize empty sets
    X_sub = X_boruta.loc[:, mask]
    clf = RandomForestClassifier(n_estimators=100, random_state=42)
    score = cross_val_score(clf, X_sub, y, cv=3, scoring='accuracy').mean()
    return 1 - score  # PSO minimizes error

def fitness_function(particles):
    return [pso_fitness(p > 0.5) for p in particles]

# Running PSO
dimensions = X_boruta.shape[1]
options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9, 'k': 10, 'p': 2} # Added 'k' and 'p'
optimizer = ps.discrete.BinaryPSO(n_particles=10, dimensions=dimensions, options=options)
best_cost, best_position = optimizer.optimize(fitness_function, iters=10)

# Final features
final_features = X_boruta.columns[best_position.astype(bool)].tolist()
X_final = X_boruta[final_features]
print("✅ Final Boruta + PSO features:", final_features)

2025-07-25 11:15:55,656 - pyswarms.discrete.binary - INFO - Optimize for 10 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9, 'k': 10, 'p': 2}
pyswarms.discrete.binary: 100%|██████████|10/10, best_cost=0.012
2025-07-25 11:19:25,150 - pyswarms.discrete.binary - INFO - Optimization finished | best cost: 0.011998777119793691, best pos:
[1 0 0 0 1 1 1 0 1 0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0
 0 0 0 0 0 1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 0]

✅ Final Boruta + PSO features: [' Destination Port', 'Total Length of Fwd Packets', ' Total Length of Bwd Packets', ' Fwd Packet Length Max', ' Fwd Packet Length Mean', ' Bwd Packet Length Min', ' Bwd Packet Length Mean', ' Bwd Packet Length Std', ' Flow IAT Max', ' Flow IAT Min', 'Fwd IAT Total', ' Fwd IAT Std', ' Fwd IAT Min', 'Bwd IAT Total', ' Bwd IAT Mean', ' Bwd IAT Std', ' Bwd IAT Max', ' Fwd Header Length', ' Bwd Header Length', 'Fwd Packets/s', ' Bwd Packets/s', ' ACK Flag Count', ' URG Flag Count', ' Average Packet Size', ' Avg Bwd Segment Size', 'Subflow Fwd Packets', ' Subflow Fwd Bytes', 'Init_Win_bytes_forward', ' Init_Win_bytes_backward', ' min_seg_size_forward', 'Active Mean', ' Active Min', ' Idle Max']
```

Here after the application of BORUTA, PSO is applied on BORUTRA selected features. The fitness function of PSO uses a Random forest classifier, first each particle's binary vector is converted to Boolean mask for selected features and then the classifier with 3-fold cross validation is used to compute mean accuracy. The swarm size is set to 10, dimensions equal to the number of Boruta-selected features (63) and finally, the PSO is run for 10 iterations to search for optimal features.

```
In [ ]: final_features

Out[23]: [' Destination Port',
          'Total Length of Fwd Packets',
          ' Total Length of Bwd Packets',
          ' Fwd Packet Length Max',
          ' Fwd Packet Length Mean',
          ' Bwd Packet Length Min',
          ' Bwd Packet Length Mean',
          ' Bwd Packet Length Std',
          ' Flow IAT Max',
          ' Flow IAT Min',
          'Fwd IAT Total',
          ' Fwd IAT Std',
          ' Fwd IAT Min',
          'Bwd IAT Total',
          ' Bwd IAT Mean',
          ' Bwd IAT Std',
          ' Bwd IAT Max',
          ' Fwd Header Length',
          ' Bwd Header Length',
          'Fwd Packets/s',
          ' Bwd Packets/s',
          ' ACK Flag Count',
          ' URG Flag Count',
          ' Average Packet Size',
          ' Avg Bwd Segment Size',
          'Subflow Fwd Packets',
          ' Subflow Fwd Bytes',
          'Init_Win_bytes_forward',
          ' Init_Win_bytes_backward',
          ' min_seg_size_forward',
          'Active Mean',
          ' Active Min',
          ' Idle Max']
```

These are the final selected features upon which all the models were trained, 33 features. 80 (original) -> [BORUTA] 63 features -> [PSO] 33 features.

3.3 Model Training

In [10]:
```python
# extratrees

# Simulating multiple n_estimators to plot as line graph
n_estimators_list = [10, 30, 50, 70, 100]
train_acc_list, test_acc_list = [], []
train_loss_list, test_loss_list = [], []

for n in n_estimators_list:
    et_model = ExtraTreesClassifier(
        n_estimators=n,
        max_depth=None,
        min_samples_split=5,
        min_samples_leaf=2,
        max_features='sqrt',
        bootstrap=True,
        random_state=42,
        n_jobs=-1
    )

    et_model.fit(X_train, Y_train)

    y_pred_train = et_model.predict(X_train)
    y_proba_train = et_model.predict_proba(X_train)
    y_pred_et = et_model.predict(X_test)
    y_proba_et = et_model.predict_proba(X_test)

    train_acc_list.append(accuracy_score(Y_train, y_pred_train))
    test_acc_list.append(accuracy_score(Y_test, y_pred_et))
    train_loss_list.append(log_loss(Y_train, y_proba_train))
    test_loss_list.append(log_loss(Y_test, y_proba_et))
```

In [9]:
```python
# randomforest

n_estimators_list = [10, 30, 50, 70, 100]
train_acc_list = []
test_acc_list = []
train_loss_list = []
test_loss_list = []

for n in n_estimators_list:
    rf_model = RandomForestClassifier(n_estimators=n, random_state=42, max_depth=None)
    rf_model.fit(X_train, Y_train)

    y_pred_train = rf_model.predict(X_train)
    y_proba_train = rf_model.predict_proba(X_train)
    y_pred_rf = rf_model.predict(X_test)
    y_proba_rf = rf_model.predict_proba(X_test)

    train_acc_list.append(accuracy_score(Y_train, y_pred_train))
    test_acc_list.append(accuracy_score(Y_test, y_pred_rf))
    train_loss_list.append(log_loss(Y_train, y_proba_train))
    test_loss_list.append(log_loss(Y_test, y_proba_rf))
```

In [8]:
```python
# catboost

# Initializing model
model = CatBoostClassifier(
    iterations=100,
    learning_rate=0.5,
    depth=3,
    loss_function='MultiClass',
    eval_metric='MultiClass',
    verbose=0
)

# Fitting with eval_set to track metrics
model.fit(X_train, Y_train, eval_set=(X_test, Y_test), plot=False)

# Accuracy
preds_class = model.predict(X_test)
accuracy_cb = accuracy_score(Y_test, preds_class)
accuracy_scores.append(accuracy_cb)
print("Accuracy: %.2f%%" % (accuracy_cb * 100.0))
```

In [7]:
```
# xg boost

# Creating DMatrix
dtrain = xgb.DMatrix(X_train, label=Y_train)
dtest = xgb.DMatrix(X_test, label=Y_test)

# Parameters
param = {
    'max_depth': 2,
    'eta': 0.3,
    'objective': 'multi:softprob',  # for probability output
    'num_class': 7,
    'eval_metric': ['mlogloss', 'merror']
}
num_round = 100

# Training with eval result tracking
evals_result = {}
bst = xgb.train(
    param,
    dtrain,
    num_round,
    evals=[(dtrain, 'train'), (dtest, 'eval')],
    evals_result=evals_result,
    verbose_eval=False
)

# Predicting labels and probabilities
y_pred = np.argmax(bst.predict(dtest), axis=1)
y_proba = bst.predict(dtest)
```

In [11]:
```
# LGBM

# Creating LightGBM datasets
train_data = lgb.Dataset(X_train, label=Y_train)
test_data = lgb.Dataset(X_test, label=Y_test, reference=train_data)

# Defining parameters
params = {
    'objective': 'multiclass',
    'metric': ['multi_logloss', 'multi_error'],
    'num_class': 7,
    'boosting_type': 'gbdt',
    'num_leaves': 50,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

# Training model and capture evaluation results
evals_result = {}

bst = lgb.train(
    params,
    train_data,
    num_boost_round=100,
    valid_sets=[train_data, test_data],
    valid_names=['train', 'valid'],
    callbacks=[
        lgb.early_stopping(10),
        lgb.record_evaluation(evals_result)
    ]
)

# Predicting
y_pred = bst.predict(X_test, num_iteration=bst.best_iteration)
y_pred_class = y_pred.argmax(axis=1)
```

5 models were trained Random Forest, CatBoost, LightGBM, XGBoost, Extra Trees. Out of these 5 models LightGBM achieved the highest accuracy of 99.84%.

References

Jaiswal, SK & Gupta, RK 2022, ‘Evolutionary feature selection using PSO for anomaly detection’, IEEE.

Kennedy, J & Eberhart, R 1995, ‘Particle swarm optimization’, IEEE, pp. 1942–1948.

Moustafa, AA, Slay, S & Abeshu, M 2024, ‘Hybrid feature selection for network intrusion detection systems using Mutual Information and Boruta’, Journal of Big Data, 1, vol. 11, pp. 1–19.