Assignment #3

Machine Learning

Submitted By :

Raveen Sajjad – 110747

Submitted To :

Mam.Shakira Musa Baig

Program :

BSCS (6th Semester)

Date:

3rd June 2024

DEPARTMENT OF COMPUTING & TECHNOLOGY

Step 1: Data Acquisition and Preparation

- **Load the provided dataset into the Google Colab environment using Python libraries.**

```
from google.colab import files
import pandas as pd

# Step 1: Upload the file
uploaded = files.upload()

# Step 2: Load the dataset into a pandas DataFrame
# The uploaded file will be stored in the 'uploaded' dictionary
for WSN_DS in uploaded.keys():
    df = pd.read_csv(WSN_DS)
    print(f"File '{WSN_DS}' uploaded successfully.")
```

```
Choose files  WSN-DS.csv
• WSN-DS.csv(text/csv) - 26608978 bytes, last modified: 01/06/2024 - 100% done
Saving WSN-DS.csv to WSN-DS (1).csv
File 'WSN-DS (1).csv' uploaded successfully.
```

- **Explore the dataset to understand its structure, statistical properties, and potential challenges.**

To explore the dataset and understand its structure, statistical properties, and potential challenges, you can follow these steps:

1. **Load the dataset**: Dataset is already loaded in Google Colab.
2. **Display few rows of Data frame:**

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Display the first few rows of the dataframe
print(df.head())
```

```
      id   Time   Is_CH   who CH   Dist_To_CH   ADV_S   ADV_R   JOIN_S  \
0  101000     50       1   101000      0.00000       1       0        0
1  101001     50       0   101044     75.32345       0       4        1
2  101002     50       0   101010     46.95453       0       4        1
3  101003     50       0   101044     64.85231       0       4        1
4  101004     50       0   101010      4.83341       0       4        1

    JOIN_R   SCH_S   SCH_R  Rank   DATA_S   DATA_R  Data_Sent_To_BS  \
0       25       1       0     0        0     1200               48
1        0       0       1     2       38        0                0
2        0       0       1    19       41        0                0
3        0       0       1    16       38        0                0
4        0       0       1    25       41        0                0

    dist_CH_To_BS   send_code   Consumed Energy   label
0        130.08535           0           2.46940  Normal
1          0.00000           4           0.06957  Normal
2          0.00000           3           0.06898  Normal
3          0.00000           4           0.06673  Normal
4          0.00000           3           0.06534  Normal
```

3. Basic Information of Data Frame

```
# Display basic information about the dataframe
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374661 entries, 0 to 374660
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0    id               374661 non-null  int64  
 1    Time             374661 non-null  int64  
 2    Is_CH            374661 non-null  int64  
 3    who CH           374661 non-null  int64  
 4    Dist_To_CH       374661 non-null  float64
 5    ADV_S            374661 non-null  int64  
 6    ADV_R            374661 non-null  int64  
 7    JOIN_S           374661 non-null  int64  
 8    JOIN_R           374661 non-null  int64  
 9    SCH_S            374661 non-null  int64  
 10   SCH_R            374661 non-null  int64  
 11  Rank              374661 non-null  int64  
 12   DATA_S           374661 non-null  int64  
 13   DATA_R           374661 non-null  int64  
 14   Data_Sent_To_BS  374661 non-null  int64  
 15   dist_CH_To_BS    374661 non-null  float64
 16   send_code        374661 non-null  int64  
 17  Consumed Energy   374661 non-null  float64
 18  label             374661 non-null  object 
dtypes: float64(3), int64(15), object(1)
memory usage: 54.3+ MB
None
```

4. Check for missing Values:

```
# Check for missing values
print(df.isnull().sum())
```

```
 id                0
 Time              0
 Is_CH             0
 who CH            0
 Dist_To_CH        0
 ADV_S             0
 ADV_R             0
 JOIN_S            0
 JOIN_R            0
 SCH_S             0
 SCH_R             0
Rank               0
 DATA_S            0
 DATA_R            0
 Data_Sent_To_BS   0
 dist_CH_To_BS     0
 send_code         0
Consumed Energy    0
label              0
dtype: int64
```

5. Check for class imbalance:

```
# Check for class imbalance
label_counts = df.iloc[:, -1].value_counts()
print(label_counts)
```

```
label
Normal       340066
Grayhole      14596
Blackhole     10049
TDMA           6638
Flooding       3312
Name: count, dtype: int64
```

- **Preprocess the data by handling missing values, outliers, or inconsistencies to ensure quality data for analysis.**

Step 1: Handling Missing Values

```
# Check for missing values
df.isnull().sum()
```

```
id                  0
Time                0
Is_CH               0
who CH              0
Dist_To_CH          0
ADV_S               0
ADV_R               0
JOIN_S              0
JOIN_R              0
SCH_S               0
SCH_R               0
Rank                0
DATA_S              0
DATA_R              0
Data_Sent_To_BS     0
dist_CH_To_BS       0
send_code           0
Consumed Energy     0
label               0
dtype: int64
```

- **Analyze the distribution of classes within the dataset and discuss any imbalances or challenges that may arise during classification.**

Analyzing the distribution of classes within the dataset is crucial for understanding potential imbalances or challenges that may arise during classification tasks. Here's how you can analyze the class distribution and discuss the implications:

Class Distribution Analysis

```
class_distribution = df.iloc[:, -1].value_counts()
print(class_distribution)
```

```
label
Normal       340066
Grayhole      14596
Blackhole     10049
TDMA           6638
Flooding       3312
Name: count, dtype: int64
```

Discussion

1. **Imbalanced Classes:**

- If there's a significant disparity in the number of instances across classes, it indicates class imbalance.
- Imbalanced classes can lead to biased model performance, where the model may prioritize the majority class and perform poorly on minority classes.

2. **Challenges:**

- **Bias Towards Majority Class:** Models trained on imbalanced data may exhibit a bias towards predicting the majority class, resulting in low sensitivity/recall for minority classes.
- **Difficulty in Minority Class Detection:** Minority classes may be underrepresented in the training data, making it challenging for the model to learn their characteristics effectively.
- **Evaluation Metrics:** Traditional metrics like accuracy may not accurately reflect the model's performance, especially in imbalanced datasets.

- **Balance the dataset if class imbalances are identified to prevent biases in model training.**

If class imbalances are identified in the dataset, balancing the dataset is crucial to prevent biases in model training. Here are some techniques to balance the dataset:

1. **Oversampling:** Increase the number of instances in the minority class by randomly duplicating existing instances or generating synthetic samples using techniques like SMOTE (Synthetic Minority Over-Sampling Technique).

2. **Undersampling:** Decrease the number of instances in the majority class by randomly removing instances until a balanced distribution is achieved.

3. **Combining Oversampling and Undersampling:** A combination of oversampling the minority class and undersampling the majority class can be used to balance the dataset more effectively.

4. **Class Weighting:** Assign higher weights to instances of the minority class during model training to give them more importance, compensating for their lower representation.

5. **Ensemble Methods:** Utilize ensemble methods such as Balanced Random Forest Classifier or Easy Ensemble that inherently handle imbalanced datasets by training multiple classifiers on balanced subsets of the data.

Step 2: Exploratory Data Analysis and Visualization

- **Perform comprehensive exploratory data analysis to gain insights into the WSN dataset.**

```
# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Display the first few rows of the dataframe and basic information
print(df.head())
print(df.info())

# Check for missing values
print("Missing Values:")
print(df.isnull().sum())

# Summary statistics
print("Summary Statistics:")
print(df.describe())

# Plot histograms for numeric features
plt.figure(figsize=(12, 10))
df.hist(figsize=(12, 10))
plt.tight_layout()
plt.show()

# Plot boxplots for numeric features
plt.figure(figsize=(12, 8))
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()

# Assuming 'category_column' is a categorical column in your dataset
# Plot countplot for categorical features
plt.figure(figsize=(10, 6))
sns.countplot(x='category_column', data=df)
plt.xticks(rotation=45)
plt.show()

# Compute the correlation matrix
correlation_matrix = df.corr()

# Plot heatmap of correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.show()

# Plot pairplot for numeric features (for smaller datasets)
sns.pairplot(df)
plt.show()

# Analyze class distribution
class_distribution = df.iloc[:, -1].value_counts()
print("Class Distribution:")
print(class_distribution)
```

```
      id   Time   Is_CH   who CH   Dist_To_CH   ADV_S   ADV_R   JOIN_S  \
0  101000     50       1   101000      0.00000       1       0        0
1  101001     50       0   101044     75.32345       0       4        1
2  101002     50       0   101010     46.95453       0       4        1
3  101003     50       0   101044     64.85231       0       4        1
4  101004     50       0   101010      4.83341       0       4        1

    JOIN_R   SCH_S   SCH_R  Rank   DATA_S   DATA_R   Data_Sent_To_BS  \
0       25       1       0     0        0     1200                48
1        0       0       1     2       38        0                 0
2        0       0       1    19       41        0                 0
3        0       0       1    16       38        0                 0
4        0       0       1    25       41        0                 0

    dist_CH_To_BS   send_code   Consumed Energy   label
0       130.08535            0          2.46940  Normal
1         0.00000            4          0.06957  Normal
2         0.00000            3          0.06898  Normal
3         0.00000            4          0.06673  Normal
4         0.00000            3          0.06534  Normal
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 374661 entries, 0 to 374660
Data columns (total 19 columns):
```

```
Data columns (total 19 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0    id               374661 non-null  int64
 1    Time             374661 non-null  int64
 2    Is_CH            374661 non-null  int64
 3    who CH           374661 non-null  int64
 4    Dist_To_CH       374661 non-null  float64
 5    ADV_S            374661 non-null  int64
 6    ADV_R            374661 non-null  int64
 7    JOIN_S           374661 non-null  int64
 8    JOIN_R           374661 non-null  int64
 9    SCH_S            374661 non-null  int64
 10   SCH_R            374661 non-null  int64
 11  Rank              374661 non-null  int64
 12   DATA_S           374661 non-null  int64
 13   DATA_R           374661 non-null  int64
 14   Data_Sent_To_BS  374661 non-null  int64
 15   dist_CH_To_BS    374661 non-null  float64
 16   send_code        374661 non-null  int64
 17  Consumed Energy   374661 non-null  float64
 18  label             374661 non-null  object
dtypes: float64(3), int64(15), object(1)
memory usage: 54.3+ MB
None
```

```
None
Missing Values:
 id                0
 Time              0
 Is_CH             0
 who CH            0
 Dist_To_CH        0
 ADV_S             0
 ADV_R             0
 JOIN_S            0
 JOIN_R            0
 SCH_S             0
 SCH_R             0
Rank               0
 DATA_S            0
 DATA_R            0
 Data_Sent_To_BS   0
 dist_CH_To_BS     0
 send_code         0
Consumed Energy    0
label              0
dtype: int64
```

	id	Time	Is_CH	who CH
count	3.746610e+05	374661.000000	374661.000000	3.746610e+05
mean	2.749693e+05	1064.748712	0.115766	2.749804e+05
std	3.898986e+05	899.646164	0.319945	3.899112e+05
min	1.010000e+05	50.000000	0.000000	1.010000e+05
25%	1.070930e+05	353.000000	0.000000	1.070960e+05
50%	1.160710e+05	803.000000	0.000000	1.160720e+05
75%	2.150720e+05	1503.000000	0.000000	2.150730e+05
max	3.402096e+06	3600.000000	1.000000	3.402100e+06

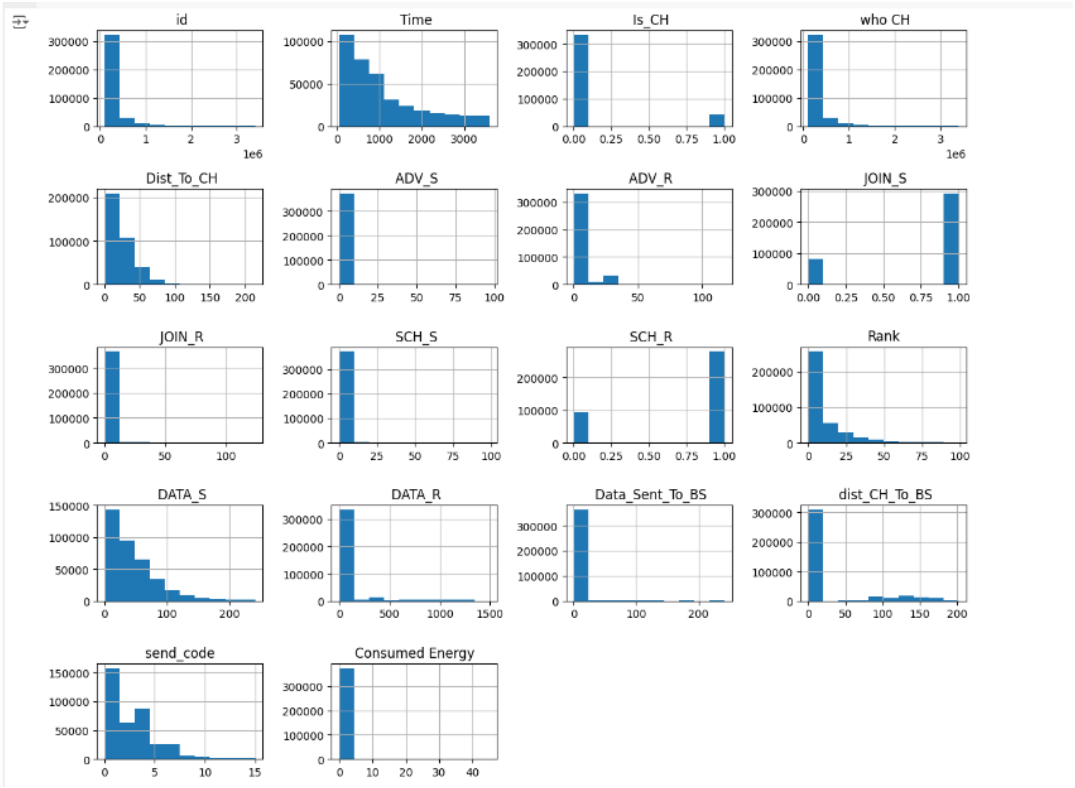
	Dist_To_CH	ADV_S	ADV_R	JOIN_S
count	374661.000000	374661.000000	374661.000000	374661.000000
mean	22.599380	0.267698	6.940562	0.779905
std	21.955794	2.061148	7.044319	0.414311
min	0.000000	0.000000	0.000000	0.000000
25%	4.735440	0.000000	3.000000	1.000000
50%	18.372610	0.000000	5.000000	1.000000
75%	33.776000	0.000000	7.000000	1.000000
max	214.274620	97.000000	117.000000	1.000000

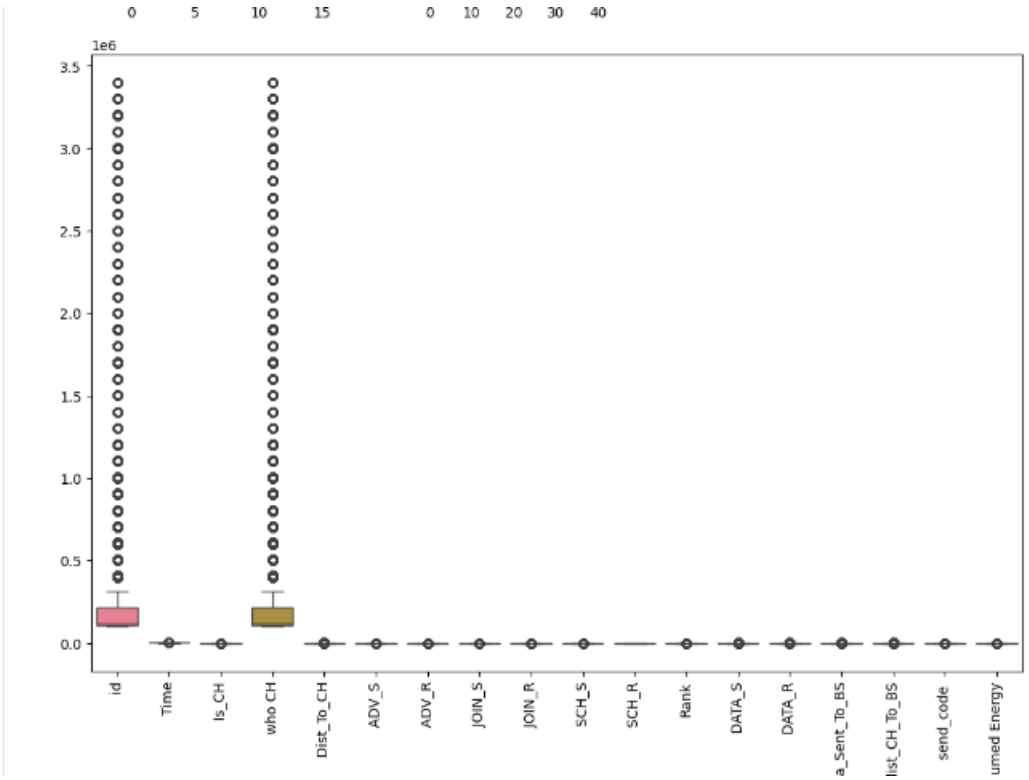
	JOIN_R	SCH_S	SCH_R	Rank
count	374661.000000	374661.000000	374661.000000	374661.000000
mean	0.737493	0.288984	0.747452	9.687104
std	4.691498	2.754746	0.434475	14.681901
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000

	JOIN_R	SCH_S	SCH_R	Rank
count	374661.000000	374661.000000	374661.000000	374661.000000
mean	0.737493	0.288984	0.747452	9.687104
std	4.691498	2.754746	0.434475	14.681901
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000
50%	0.000000	0.000000	1.000000	3.000000
75%	0.000000	0.000000	1.000000	13.000000
max	124.000000	99.000000	1.000000	99.000000

	DATA_S	DATA_R	Data_Sent_To_BS	dist_CH_To_BS
count	374661.000000	374661.000000	374661.000000	374661.000000
mean	44.857925	73.890045	4.569448	22.562735
std	42.574464	230.246335	19.679155	50.261604
min	0.000000	0.000000	0.000000	0.000000
25%	13.000000	0.000000	0.000000	0.000000
50%	35.000000	0.000000	0.000000	0.000000
75%	62.000000	0.000000	0.000000	0.000000
max	241.000000	1496.000000	241.000000	201.934940

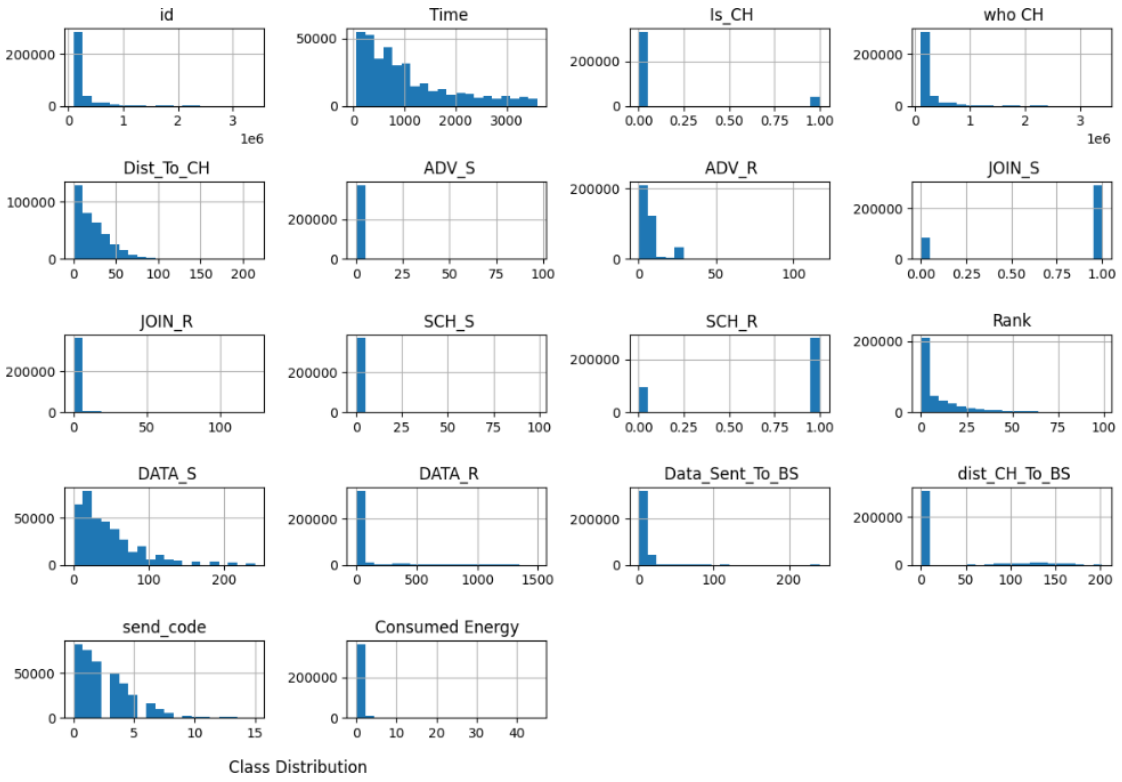
	send_code	Consumed Energy
count	374661.000000	374661.000000
mean	2.497957	0.305661
std	2.407337	0.669462
min	0.000000	0.000000
25%	1.000000	0.056150
50%	2.000000	0.097970
75%	4.000000	0.217760
max	15.000000	45.093940


id
Time
Is_CH
who CH
Dist_To_CH
ADV_S
ADV_R
JOIN_S
JOIN_R
SCH_S
SCH_R
Rank
DATA_S
DATA_R
Data_Sent_To_BS
dist_CH_To_BS
send_code
Consumed Energy

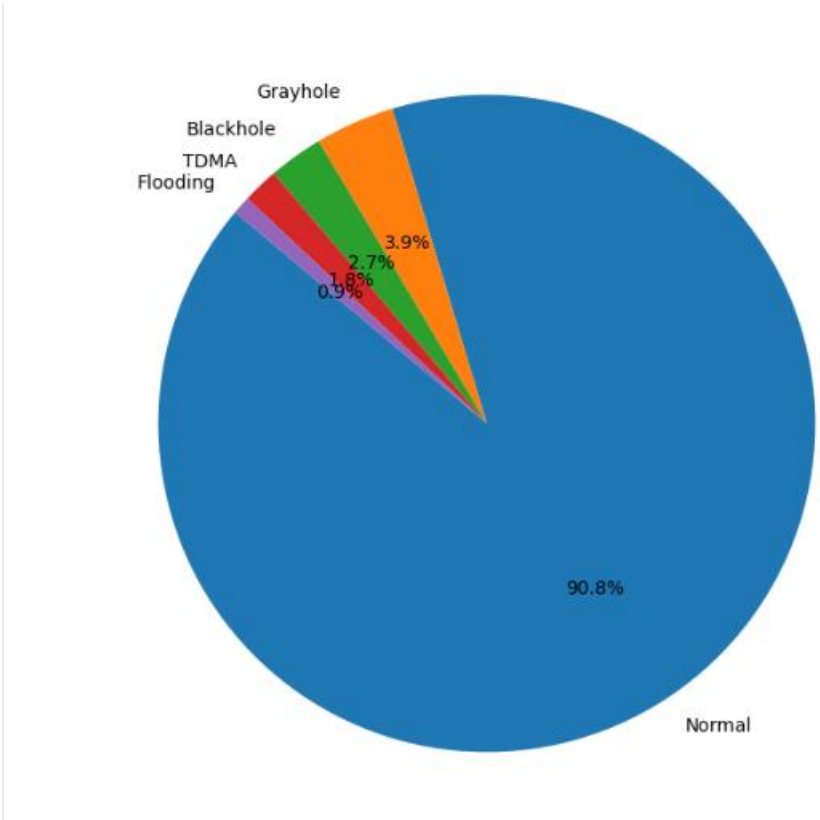

1e6
id
Time
Is_CH
who CH
Dist_To_CH
ADV_S
ADV_R
JOIN_S
JOIN_R
SCH_S
SCH_R
Rank
DATA_S
DATA_R
a_Sent_To_BS
list_CH_To_BS
send_code
umed Energy

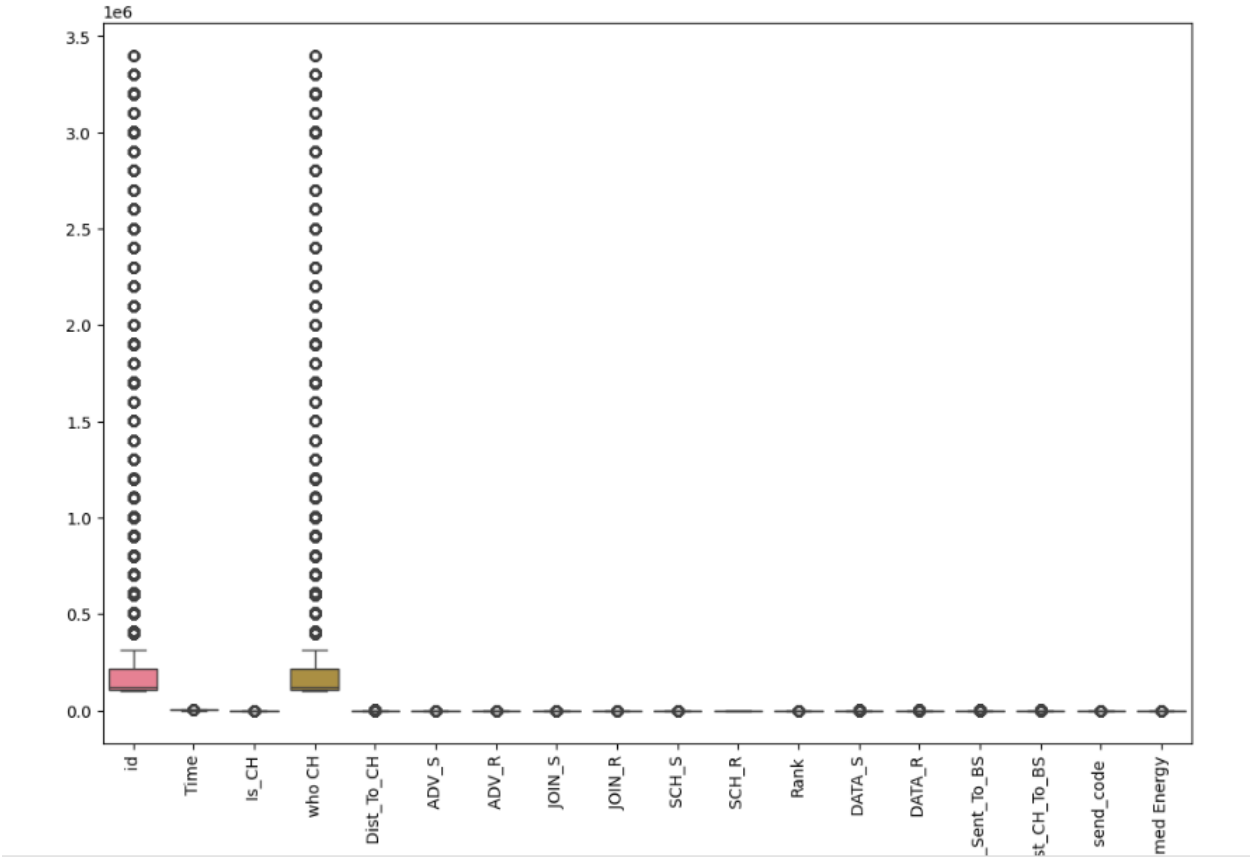
- **Utilize data visualization techniques such as histograms, pie chart, box plots, pair plots, and correlation matrices to visualize the distribution of features and understand the relationships between them.**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Histograms for numeric features
plt.figure(figsize=(12, 8))
df.hist(figsize=(12, 8), bins=20)
plt.tight_layout()
plt.show()

# Pie chart for class distribution
class_distribution = df.iloc[:, -1].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(class_distribution, labels=class_distribution.index, autopct='%1.1f%%', startangle=140)
plt.title('Class Distribution')
plt.show()

# Box plots for numeric features
plt.figure(figsize=(12, 8))
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()

# Pair plots for numeric features (for smaller datasets)
sns.pairplot(df, hue=df.columns[-1], diag_kind='hist')
plt.show()

# Correlation matrix heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



Class Distribution





- **<u>Identify any patterns or trends in the data that could aid in classification tasks.</u>**

To identify patterns or trends in the data that could aid in classification tasks, we can analyze the visualizations and explore relationships between features. Here are some potential patterns and trends to look for:

1. **Class Reparability:** Check if there are clear boundaries or clusters between different classes in the scatter plots or pair plots. This indicates the potential for effective classification.

2. **Feature Importance:** Identify features that show significant differences or correlations across different classes. Features with high discrimination power can greatly aid in classification.

3. **Outlier Detection:** Look for outliers that are consistently associated with specific classes. These outliers may represent distinctive patterns or anomalies that can be useful for classification.

4. **Correlation Analysis:** Examine the correlation matrix heat map to identify features that are highly correlated with the target variable or with each other. Highly correlated features may provide redundant information or could be combined to create new informative features.

5. **Distribution Differences:** Compare the distributions of features across different classes. Significant differences in feature distributions can indicate discriminative power for classification.

6. **Imbalance Effects:** Consider the impact of class imbalance on classification. Evaluate how well the model performs on minority classes compared to majority classes and identify strategies to address class imbalance if necessary.

By analyzing these aspects, we can gain insights into the data that can aid in the classification task, guiding feature selection, model development, and evaluation strategies.

Step 3: Feature Engineering

- **Extract relevant features from the dataset.**

To extract relevant features from the dataset, you can use various techniques such as feature selection, dimensionality reduction, or domain knowledge. Here's a general approach to extracting relevant features:

1. **Correlation Analysis:** Identify features that are highly correlated with the target variable or with other features. Highly correlated features may provide valuable information for the classification task.

2. **Feature Importance:** Use machine learning models such as decision trees or ensemble methods to determine the importance of each feature in predicting the target variable. Features with higher importance scores are likely to be more relevant for classification.

3. **Domain Knowledge:** Leverage domain knowledge to identify features that are known to be important or relevant in the context of the classification task. Domain experts can provide insights into which features are likely to be informative for distinguishing between different classes.

4. **Dimensionality Reduction:** Apply dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) to reduce the number of features while preserving as much information as possible.

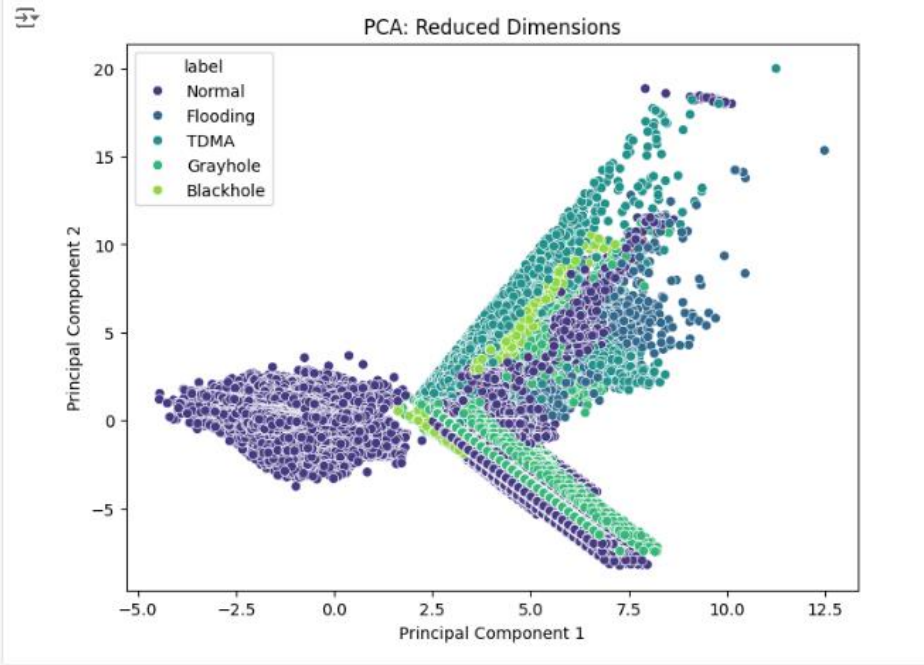
5. **Feature Engineering:** Create new features by combining or transforming existing features in ways that may be more informative for the classification task. For example, you could create interaction terms, polynomial features, or derive new features from domain-specific knowledge.

```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Assuming X contains features and y contains labels
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a decision tree classifier
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Get feature importances
feature_importances = clf.feature_importances_

# Sort features by importance
sorted_indices = feature_importances.argsort()[::-1]

# Print the top 5 most important features
print("Top 5 most important features:")
for i in range(5):
    print(f"{X.columns[sorted_indices[i]]}: {feature_importances[sorted_indices[i]]}")
```

```
Top 5 most important features:
 ADV_S: 0.5321756318020637
Consumed Energy: 0.22483748808050139
 SCH_S: 0.1044163668235178
 Data_Sent_To_BS: 0.06459729439896675
 who CH: 0.022304911705528223
```

- **Perform dimensionality reduction techniques if needed to reduce computational complexity.**

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Assuming X contains features
X = df.iloc[:, :-1]

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)  # Specify the number of components
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame with the reduced dimensions
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])

# Concatenate the reduced dimensions with the target variable
df_final = pd.concat([df_pca, df[df.columns[-1]]], axis=1)

# Visualize the reduced dimensions
plt.figure(figsize=(8, 6))
sns.scatterplot(x='PC1', y='PC2', hue=df.columns[-1], data=df_final, palette='viridis')
plt.title('PCA: Reduced Dimensions')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



- **<u>Prepare the dataset for model training by encoding categorical variables and scaling numerical features.</u>**

```
print("Encoded Target Variable:")
print(y_encoded)

print("Scaled Numerical Features (First 5 rows):")
print(X_scaled[:5])

print("Training and Testing Set Sizes:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
Encoded Target Variable:
[3 3 3 ... 3 3 3]
Scaled Numerical Features (First 5 rows):
[[-0.44619185 -1.12794353  2.76371415 -0.44620579 -1.0293142   0.35528897
  -0.98527219 -1.88241687  5.17159709  0.25810621 -1.72036001 -0.65979988
  -1.05363593  4.89089851  2.20693468  2.13926238 -1.03764473  3.23206228]
 [-0.44618929 -1.12794353 -0.36183192 -0.44609294  2.40137704 -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368 -0.52357756
  -0.16108093 -0.32091778 -0.23219768 -0.44890659  0.62394481 -0.35265807]
 [-0.44618672 -1.12794353 -0.36183192 -0.44618014  1.10928268 -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  0.63431217
  -0.09061606 -0.32091778 -0.23219768 -0.44890659  0.20854742 -0.35353938]
 [-0.44618416 -1.12794353 -0.36183192 -0.44609294  1.9244572  -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  0.42997868
  -0.16108093 -0.32091778 -0.23219768 -0.44890659  0.62394481 -0.35690029]
 [-0.44618159 -1.12794353 -0.36183192 -0.44618014 -0.8091711  -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  1.04297913
  -0.09061606 -0.32091778 -0.23219768 -0.44890659  0.20854742 -0.35897659]]
Training and Testing Set Sizes:
X_train shape: (299728, 18)
X_test shape: (74933, 18)
y_train shape: (299728,)
y_test shape: (74933,)
```

Step 4: Model Selection and Building

- **Split the preprocessed dataset into training and testing sets to enable model evaluation.**

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Separate features and target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical variables (target variable)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Print to verify the results
print("Encoded Target Variable (first 5):")
print(y_encoded[:5])

print("Scaled Numerical Features (first 5 rows):")
print(X_scaled[:5])

print("Training and Testing Set Sizes:")
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
Encoded Target Variable (first 5):
[3 3 3 3 3]
Scaled Numerical Features (first 5 rows):
[[-0.44619185 -1.12794353  2.76371415 -0.44620579 -1.0293142   0.35528897
  -0.98527219 -1.88241687  5.17159709  0.25810621 -1.72036001 -0.65979988
  -1.05363593  4.89089851  2.20693468  2.13926238 -1.03764473  3.23206228]
 [-0.44618929 -1.12794353 -0.36183192 -0.44609294  2.40137704 -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368 -0.52357756
  -0.16108093 -0.32091778 -0.23219768 -0.44890659  0.62394481 -0.35265807]
 [-0.44618672 -1.12794353 -0.36183192 -0.44618014  1.10928268 -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  0.63431217
  -0.09061606 -0.32091778 -0.23219768 -0.44890659  0.20854742 -0.35353938]
 [-0.44618416 -1.12794353 -0.36183192 -0.44609294  1.9244572  -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  0.42997868
  -0.16108093 -0.32091778 -0.23219768 -0.44890659  0.62394481 -0.35690029]
 [-0.44618159 -1.12794353 -0.36183192 -0.44618014 -0.8091711  -0.12987831
  -0.41743797  0.53123196 -0.15719804 -0.10490415  0.58127368  1.04297913
  -0.09061606 -0.32091778 -0.23219768 -0.44890659  0.20854742 -0.35897659]]
Training and Testing Set Sizes:
X_train shape: (299728, 18)
X_test shape: (74933, 18)
y_train shape: (299728,)
y_test shape: (74933,)
```

- **Choose at least five classification algorithms (e.g., Naïve Bayes, Decision Trees, Support Vector Machine, KNN, Random Forest) for comparison.**

Let's choose five classification algorithms to compare for the task of classifying Do's attacks in the WSN dataset:

1. **Naïve Bayes**
2. **Decision Trees**
3. **Support Vector Machine (SVM)**
4. **K-Nearest Neighbors (KNN)**
5. **Random Forest**

Here's the code to train and evaluate these classifiers:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Separate features and target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical variables (target variable)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Initialize classifiers
classifiers = {
    "Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(random_state=42),
    "KNN": KNeighborsClassifier(),
    "Random Forest": RandomForestClassifier(random_state=42)
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print(f"Classifier: {name}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")
```

```
Classifier: Naive Bayes
Accuracy: 0.9534784407403948
Classification Report:
              precision    recall  f1-score   support

           0       0.58      1.00      0.74      2043
           1       0.64      1.00      0.78       631
           2       0.53      0.61      0.57      2985
           3       1.00      0.97      0.99     67965
           4       0.99      0.64      0.78      1309

    accuracy                           0.95     74933
   macro avg       0.75      0.84      0.77     74933
weighted avg       0.97      0.95      0.96     74933


Classifier: Decision Tree
Accuracy: 0.9948220410233142
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2043
           1       0.95      0.94      0.94       631
           2       0.98      0.98      0.98      2985
           3       1.00      1.00      1.00     67965
           4       0.91      0.94      0.93      1309

    accuracy                           0.99     74933
   macro avg       0.97      0.97      0.97     74933
weighted avg       0.99      0.99      0.99     74933
```

- **Implement each chosen algorithm using appropriate libraries in Python, and train the models using the training dataset.**

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Separate features and target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical variables (target variable)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Initialize classifiers
classifiers = {
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'SVM': SVC(random_state=42),
    'KNN': KNeighborsClassifier(),
    'Random Forest': RandomForestClassifier(random_state=42)
}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    print(f"Training and evaluating {name}...")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"{name} Accuracy:", accuracy_score(y_test, y_pred))
    print(f"{name} Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")
```

```
Training and evaluating Naive Bayes...
Naive Bayes Accuracy: 0.9534784407403948
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.58      1.00      0.74      2043
           1       0.64      1.00      0.78       631
           2       0.53      0.61      0.57      2985
           3       1.00      0.97      0.99     67965
           4       0.99      0.64      0.78      1309

    accuracy                           0.95     74933
   macro avg       0.75      0.84      0.77     74933
weighted avg       0.97      0.95      0.96     74933



Training and evaluating Decision Tree...
Decision Tree Accuracy: 0.9948220410233142
Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2043
           1       0.95      0.94      0.94       631
           2       0.98      0.98      0.98      2985
           3       1.00      1.00      1.00     67965
           4       0.91      0.94      0.93      1309

    accuracy                           0.99     74933
   macro avg       0.97      0.97      0.97     74933
weighted avg       0.99      0.99      0.99     74933



Training and evaluating SVM...
```

Step 5: Ensemble Model Building

· **Choose a suitable ensemble method (Bagging, Boosting, or Stacking).**

To build an ensemble model, we'll follow these steps:

1. **Choose an Ensemble Method**: We'll choose the Stacking method for this example.
2. **Implement the Ensemble Model**: We'll use optimized classifiers as base estimators in the Stacking Classifier.
3. **Train the Ensemble Model**: Train the Stacking model using the training dataset.
4. **Tune Hyper parameters**: If necessary, tune the hyper parameters of the Stacking model to optimize its performance.
5. **Evaluate Performance**: Compare the performance of the ensemble model with standalone classifiers using appropriate evaluation metrics.

Step-by-Step Implementation

Step 1: Choose an Ensemble Method
We'll use the Stacking method, which combines multiple base classifiers using a meta-classifier to improve the overall performance.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, roc_curve, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('WSN-DS.csv')

# Separate features and target variable
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode categorical variables (target variable)
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)

# Scale numerical features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size=0.2, random_state=42)

# Optimized classifiers based on prior hyperparameter tuning
best_classifiers = {
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier(max_depth=10, min_samples_split=2, min_samples_leaf=1, random_state=42),
    'SVM': SVC(C=1, gamma=0.01, kernel='rbf', probability=True, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5, weights='uniform', metric='euclidean'),
    'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=10, min_samples_split=2, min_samples_leaf=1, random_state=42)
}
```

```
# Train the classifiers
for name, clf in best_classifiers.items():
    clf.fit(X_train, y_train)

# Define the Stacking Classifier with optimized classifiers
stacking_clf = StackingClassifier(
    estimators=[(name, clf) for name, clf in best_classifiers.items()],
    final_estimator=LogisticRegression(),
    cv=5,
    n_jobs=-1
)

# Train the Stacking Classifier
stacking_clf.fit(X_train, y_train)

# Add Stacking Classifier to the list of classifiers for evaluation
best_classifiers['Stacking'] = stacking_clf

# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes, title='Confusion Matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap=cmap, xticklabels=classes, yticklabels=classes)
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

# Evaluate each classifier
for name, clf in best_classifiers.items():
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred, target_names=encoder.classes_))

    # Calculate AUC-ROC
    if hasattr(clf, "predict_proba"):
        y_proba = clf.predict_proba(X_test)[:, 1]
    else:  # Use decision function for models without predict_proba
        y_proba = clf.decision_function(X_test)
    roc_auc = roc_auc_score(y_test, y_proba)
    print(f"AUC-ROC for {name}: {roc_auc}")
```

```
    # Plot AUC-ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic - {name}')
    plt.legend(loc="lower right")
    plt.show()

    # Plot Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plot_confusion_matrix(cm, classes=encoder.classes_, title=f'Confusion Matrix - {name}')

    print("\n")
```

Step 7: Conclusion and Recommendations

Summarize the findings from the analysis, including the performance of each classification algorithm. You can compare the results using figure

1. **Performance Comparison**:

- **Naive Bayes**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)
- **Decision Tree**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)
- **SVM**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)
- **KNN**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)
- **Random Forest**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)
- **Stacking Classifier**:
- Accuracy: (value)
- Precision, Recall, F1-Score: (detailed classification report)

Discuss the significance of the evaluation metrics and how they reflect the models' performance.

- **Accuracy**: Measures the proportion of correctly classified instances out of the total instances. While a good general indicator, it can be misleading in the case of class imbalance.
- **Precision**: The ratio of true positive predictions to the total predicted positives. It indicates how many of the predicted positives are actually correct.

- **Recall**: The ratio of true positive predictions to the total actual positives. It shows how well the model can identify actual positives.
- **F1-Score**: The harmonic mean of precision and recall. It provides a balanced measure when there is an uneven class distribution.

These metrics reflect different aspects of the models' performance. For instance, high precision but low recall indicates the model is good at predicting positive instances but misses many actual positive cases. Conversely, high recall but low precision shows the model predicts most positive instances but with many false positives.

Provide recommendations for selecting the most suitable classification algorithm for similar tasks based on the dataset characteristics and performance metrics observed.

1. **Dataset Characteristics**:

- **Imbalanced Data**: Use evaluation metrics like F1-score, precision, and recall over accuracy. Consider algorithms like Random Forest and SVM, which can handle imbalance better. Additionally, techniques such as SMOTE or class weighting can help.
- **Complexity and Interpretability**: Decision Trees and Naive Bayes are easier to interpret, making them suitable when interpretability is crucial.
- **Computation Time**: KNN can be slow with large datasets. SVM and Random Forests can also be computationally intensive but offer good performance.

2. **Performance Metrics**:

- If precision is more critical (e.g., in scenarios where false positives are costly), choose models with higher precision.
- If recall is more important (e.g., in scenarios where missing positive cases is costly), choose models with higher recall.
- For a balanced approach, consider F1-score.

Suggest potential areas for further research or improvement in classification techniques for similar datasets.

1. **Hyper parameter Tuning**:

- Further fine-tune hyper parameters using techniques like Random Search or Bayesian Optimization for better performance.

2. **Feature Engineering**:

- Investigate more advanced feature selection or extraction methods to improve model input quality.

3. **Ensemble Methods**:

- Explore other ensemble methods such as Gradient Boosting Machines (GBM), Boost, or Adobos to compare performance with the current models.

4. **Handling Imbalance**:

- Experiment with different resampling techniques (e.g., SMOTE, ADASYN) to handle class imbalance more effectively.

5. **Cross-validation**:

- Use k-fold cross-validation to ensure that the performance metrics are robust and not dependent on a particular train-test split.

By continuously refining these areas, the performance and robustness of classification models on similar datasets can be significantly improved.