S223090226

SHREYAS VIVEK

SIT719 5.1 D

SECTION 1: DECLARE THE MODULES

```
import os
from collections import defaultdict
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, precision_score, recall_score, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import time
import warnings
warnings.filterwarnings('ignore')
```

SECTION 2: Data import and preprocess

Run this but dont worry if it does not make any sense Jump to SECTION 3 that is related to your HD task.

```
!pip install wget
import wget
```

```
link_to_data = 'https://raw.githubusercontent.com/shreyas-vivek/SIT719-5.1D/main/NSL_KDD_Dataset/training_attack_types.txt?raw=true'
DataSet = wget.download(link_to_data)
```

```
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9655 sha256=102ebf644c635ca071d4da3899f2c8d6c21f9b05fd38ae23d66133d1fc9740bd
  Stored in directory: /root/.cache/pip/wheels/8b/f1/7f/5c94f0a7a505ca1c81cd1d9208ae2064675d97582078e6c769
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

```
DataSet
```

```
'training_attack_types.txt'
```

```
header_names = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromis


# Differentiating between nominal, binary, and numeric features

# root_shell is marked as a continuous feature in the kddcup.names
# file, but it is supposed to be a binary feature according to the
# dataset documentation

# training_attack_types.txt maps each of the 22 different attacks to 1 of 4 categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types

col_names = np.array(header_names)

nominal_idx = [1, 2, 3]
binary_idx = [6, 11, 13, 14, 20, 21]
numeric_idx = list(set(range(41)).difference(nominal_idx).difference(binary_idx))

nominal_cols = col_names[nominal_idx].tolist()
binary_cols = col_names[binary_idx].tolist()
numeric_cols = col_names[numeric_idx].tolist()
```

```
# training_attack_types.txt maps each of the 22 different attacks to 1 of 4 categories
# file obtained from http://kdd.ics.uci.edu/databases/kddcup99/training_attack_types

category = defaultdict(list)
category['benign'].append('normal')

with open(DataSet, 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)
```

```
attack_mapping = dict((v,k) for k in category for v in category[k])
```

```
attack_mapping
```

```
{'normal': 'benign',
 'apache2': 'dos',
 'back': 'dos',
 'mailbomb': 'dos',
 'processtable': 'dos',
 'snmpgetattack': 'dos',
 'teardrop': 'dos',
 'smurf': 'dos',
 'land': 'dos',
 'neptune': 'dos',
 'pod': 'dos',
 'udpstorm': 'dos',
 'ps': 'u2r',
 'buffer_overflow': 'u2r',
 'perl': 'u2r',
 'rootkit': 'u2r',
 'loadmodule': 'u2r',
 'xterm': 'u2r',
 'sqlattack': 'u2r',
 'httptunnel': 'u2r',
 'ftp_write': 'r2l',
 'guess_passwd': 'r2l',
 'snmpguess': 'r2l',
 'imap': 'r2l',
 'spy': 'r2l',
 'warezclient': 'r2l',
 'warezmaster': 'r2l',
 'multihop': 'r2l',
 'phf': 'r2l',
 'named': 'r2l',
 'sendmail': 'r2l',
 'xlock': 'r2l',
 'xsnoop': 'r2l',
 'worm': 'probe',
 'nmap': 'probe',
 'ipsweep': 'probe',
 'portsweep': 'probe',
 'satan': 'probe',
 'mscan': 'probe',
 'saint': 'probe'}
```

```
#Processing Training Data

train_file='https://raw.githubusercontent.com/shreyas-vivek/SIT719-5.1D/main/NSL_KDD_Dataset/KDDTrain%2B.txt'



train_df = pd.read_csv(train_file, names=header_names)
```

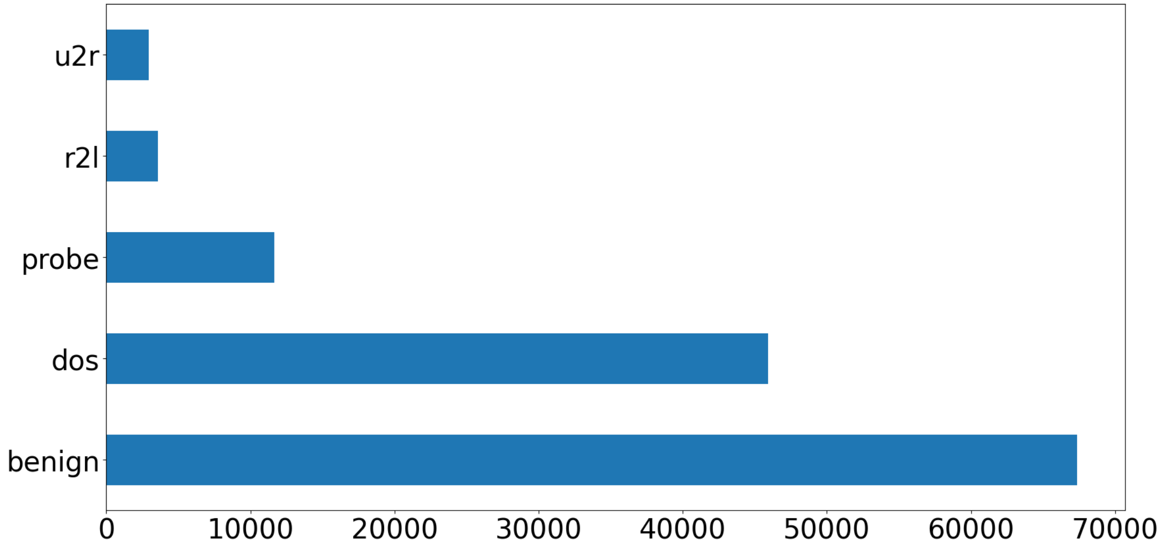
```
train_df['attack_category'] = train_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])

train_df.drop(['success_pred'], axis=1, inplace=True)
```

```
#Processing test Data
test_file='https://raw.githubusercontent.com/shreyas-vivek/SIT719-5.1D/main/NSL_KDD_Dataset/KDDTest%2B.txt'
test_df = pd.read_csv(test_file, names=header_names)
test_df['attack_category'] = test_df['attack_type'] \
                                .map(lambda x: attack_mapping[x])
test_df.drop(['success_pred'], axis=1, inplace=True)
```

```
train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()

test_attack_types = test_df['attack_type'].value_counts()
test_attack_cats = test_df['attack_category'].value_counts()

train_attack_types.plot(kind='barh', figsize=(20,10), fontsize=20)

train_attack_cats.plot(kind='barh', figsize=(20,10), fontsize=30)

train_df[binary_cols].describe().transpose()
train_df.groupby(['su_attempted']).size()
train_df['su_attempted'].replace(2, 0, inplace=True)
test_df['su_attempted'].replace(2, 0, inplace=True)
train_df.groupby(['su_attempted']).size()
train_df.groupby(['num_outbound_cmds']).size()

#Now, that's not a very useful feature - let's drop it from the dataset

train_df.drop('num_outbound_cmds', axis = 1, inplace=True)
test_df.drop('num_outbound_cmds', axis = 1, inplace=True)
numeric_cols.remove('num_outbound_cmds')



#Data Preparation

train_Y = train_df['attack_category']
train_x_raw = train_df.drop(['attack_category','attack_type'], axis=1)
test_Y = test_df['attack_category']
test_x_raw = test_df.drop(['attack_category','attack_type'], axis=1)
```

```
combined_df_raw = pd.concat([train_x_raw, test_x_raw])
combined_df = pd.get_dummies(combined_df_raw, columns=nominal_cols, drop_first=True)

train_x = combined_df[:len(train_x_raw)]
test_x = combined_df[len(train_x_raw):]

# Store dummy variable feature names
dummy_variables = list(set(train_x)-set(combined_df_raw))

#execute the commands in console
train_x.describe()
train_x['duration'].describe()
# Experimenting with StandardScaler on the single 'duration' feature
from sklearn.preprocessing import StandardScaler

durations = train_x['duration'].values.reshape(-1, 1)
standard_scaler = StandardScaler().fit(durations)
scaled_durations = standard_scaler.transform(durations)
pd.Series(scaled_durations.flatten()).describe()

# Experimenting with MinMaxScaler on the single 'duration' feature
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(durations)
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_durations.flatten()).describe()

# Experimenting with RobustScaler on the single 'duration' feature
from sklearn.preprocessing import RobustScaler

min_max_scaler = RobustScaler().fit(durations)
robust_scaled_durations = min_max_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Experimenting with MaxAbsScaler on the single 'duration' feature
from sklearn.preprocessing import MaxAbsScaler

max_Abs_scaler = MaxAbsScaler().fit(durations)
robust_scaled_durations = max_Abs_scaler.transform(durations)
pd.Series(robust_scaled_durations.flatten()).describe()

# Let's proceed with StandardScaler- Apply to all the numeric columns

standard_scaler = StandardScaler().fit(train_x[numeric_cols])

train_x[numeric_cols] = \
    standard_scaler.transform(train_x[numeric_cols])

test_x[numeric_cols] = \
```

```
    standard_scaler.transform(test_x[numeric_cols])

train_x.describe()



train_Y_bin = train_Y.apply(lambda x: 0 if x is 'benign' else 1)
test_Y_bin = test_Y.apply(lambda x: 0 if x is 'benign' else 1)
```



SECTION 3: Multi class classification

This is the section where you have to add other algorithms, tune algorithms and visualize to compare and analyze algorithms

C

```
# 5-class classification version
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, zero_one_loss

classifier = DecisionTreeClassifier(random_state=17)
classifier.fit(train_x, train_Y)

pred_y = classifier.predict(test_x)

results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)

print(results)
print(error)
```

```
[[9365   56  289    1    0]
 [1541 5998   97    0    0]
 [ 677  220 1526    0    0]
 [2278    1   14  277    4]
 [ 175    0    5    5   15]]
0.2378903477643719
```

```
# ... (Previous code here)

# Define parameter grids for hyperparameter tuning (you can modify these)
param_grids = {
    "Decision Tree": {"max_depth": [None, 10, 20, 30]},
    "Random Forest": {"n_estimators": [10, 50, 100]},
    "SVC": {"C": [0.1, 1, 10], "kernel": ["linear", "rbf"]},
    "KNN": {"n_neighbors": [3, 5, 7]},
    "Logistic Regression": {"C": [0.1, 1, 10], "penalty": ["l1", "l2"]},
}

# Initialize an empty list to store results
results = []
```

```
# Classifier 1: Decision Tree
```

```
classifier_name = "Decision Tree"
param_grid = param_grids[classifier_name]
classifier = classifiers[classifier_name]
result = perform_classification(classifier_name, classifier, param_grid)
results.append(result)
```

```
# Classifier 2: Random Forest
classifier_name = "Random Forest"
param_grid = param_grids[classifier_name]
classifier = classifiers[classifier_name]
result = perform_classification(classifier_name, classifier, param_grid)
results.append(result)
```

```
# Classifier 3: Support Vector Classifier (SVC)
classifier_name = "SVC"
param_grid = param_grids[classifier_name]
classifier = classifiers[classifier_name]
result = perform_classification(classifier_name, classifier, param_grid)
results.append(result)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-28-6535eb368279> in <cell line: 5>()
      3 param_grid = param_grids[classifier_name]
      4 classifier = classifiers[classifier_name]
----> 5 result = perform_classification(classifier_name, classifier, param_grid)
      6 results.append(result)

                         7 frames

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
   1705                 (self._jobs[0].get_status(
   1706                     timeout=self.timeout) == TASK_PENDING)):
-> 1707                 time.sleep(0.01)
   1708                 continue
   1709

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
print("hello")
```

```
hello
```

Double-click (or enter) to edit

```
# Classifier 4: K-Nearest Neighbors (KNN)
classifier_name = "KNN"
param_grid = param_grids[classifier_name]
classifier = classifiers[classifier_name]
result = perform_classification(classifier_name, classifier, param_grid)
results.append(result)
```

```
# Classifier 5: Logistic Regression
classifier_name = "Logistic Regression"
param_grid = param_grids[classifier_name]
classifier = classifiers[classifier_name]
result = perform_classification(classifier_name, classifier, param_grid)
results.append(result)
```

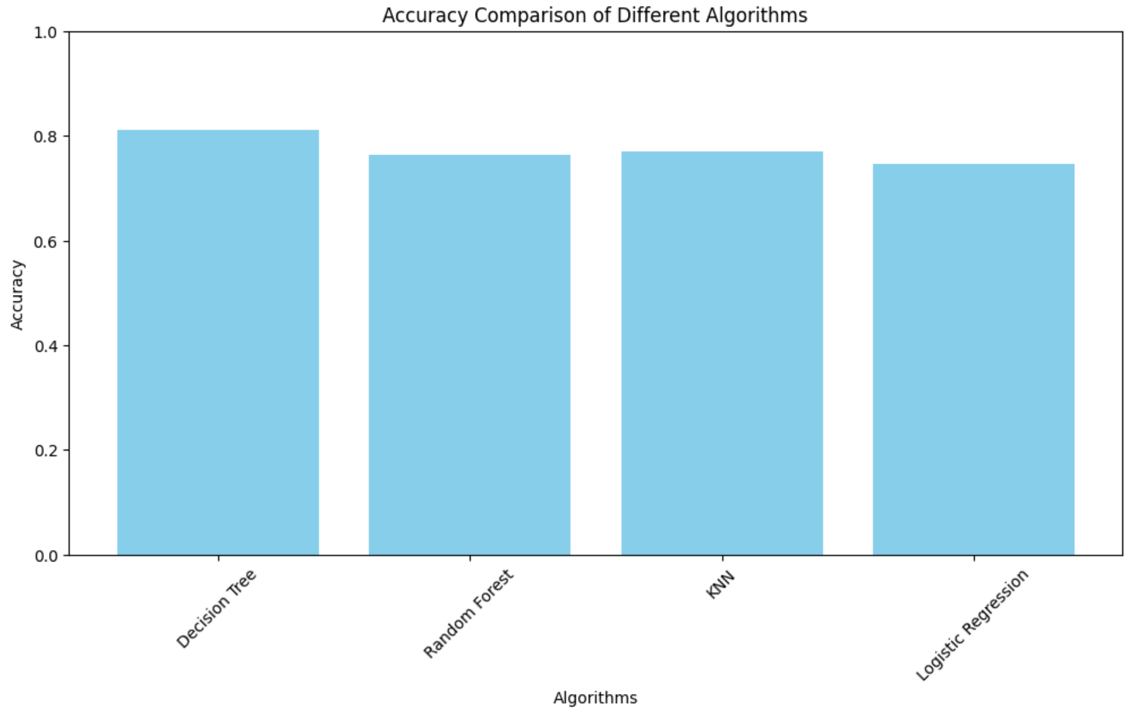
```
# Define label names
label_names = ['normal', 'dos', 'probe', 'r2l', 'u2r']
```

```
# Define functions for visualization and plotting
def visualize_accuracy(results):
    plt.figure(figsize=(12, 6))
    plt.bar([result["Classifier"] for result in results], [result["Accuracy"] for result in results], color='skyblue')
    plt.xlabel("Algorithms")
    plt.ylabel("Accuracy")
    plt.title("Accuracy Comparison of Different Algorithms")
    plt.ylim([0, 1])
    plt.xticks(rotation=45)
    plt.show()
```

```
def plot_confusion_matrices(results):
    for result in results:
        classifier_name = result["Classifier"]
        confusion_matrix = result["Confusion Matrix"]

        plt.figure(figsize=(8, 6))
```

```
        ConfusionMatrixDisplay(confusion_matrix, display_labels=label_names).plot(cmap='Blues', xticks_rotation='horizontal')
        plt.title(f"Confusion Matrix - {classifier_name}")
        plt.show()
```

```
# Call the visualization and plotting functions
visualize_accuracy(results)
plot_confusion_matrices(results)
```



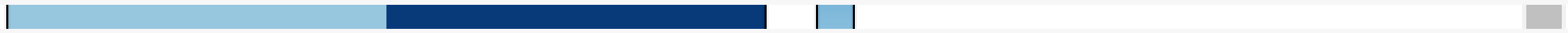

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-2ea78a9baa3a> in <cell line: 3>()
      1 # Call the visualization and plotting functions
      2 visualize_accuracy(results)
----> 3 plot_confusion_matrices(results)

8 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/axis.py in set_ticklabels(self, labels, minor,

   1968             if len(locator.locs) != len(labels) and len(labels) != 0:
```

C

```
ValueError: The number of FixedLocator locations (2), usually from a call to set_ticks, does not
```

```
# ... (Previous code here)

# **Section 3: Benchmark Classification Algorithms**
from sklearn.model_selection import GridSearchCV

# Initialize the classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(),
```

```
    "Random Forest": RandomForestClassifier(),
    "SVC": SVC(),
    "KNN": KNeighborsClassifier(),
    "Logistic Regression": LogisticRegression(),
}
```



```
# Define parameter grids for hyperparameter tuning (you can modify these)
param_grids = {
    "Decision Tree": {"max_depth": [None, 10, 20, 30]},
    "Random Forest": {"n_estimators": [10, 50, 100]},
    "SVC": {"C": [0.1, 1, 10], "kernel": ["linear", "rbf"]},
    "KNN": {"n_neighbors": [3, 5, 7]},
    "Logistic Regression": {"C": [0.1, 1, 10], "penalty": ["l1", "l2"]},
}
```

```
# Lists to store results
results = {}
confusion_matrices = {}
```

```
# Iterate through classifiers
for classifier_name, classifier in classifiers.items():
    # Perform hyperparameter tuning using GridSearchCV
    param_grid = param_grids.get(classifier_name, {})  # Get the parameter grid for this classifier
    grid_search = GridSearchCV(classifier, param_grid, cv=5, n_jobs=-1, scoring="accuracy")

    # Fit the model to training data
    grid_search.fit(train_x, train_Y_bin)

    # Get the best model
    best_classifier = grid_search.best_estimator_

    # Make predictions on the test data
    predictions = best_classifier.predict(test_x)

    # Calculate performance metrics
    accuracy = accuracy_score(test_Y_bin, predictions)
    precision = precision_score(test_Y_bin, predictions)
    recall = recall_score(test_Y_bin, predictions)
    f1 = f1_score(test_Y_bin, predictions)

    # Calculate the confusion matrix
    confusion = confusion_matrix(test_Y_bin, predictions)
```

```
    # Store results
    results[classifier_name] = {"Accuracy": accuracy, "Precision": precision, "Recall": recall, "F1": f1}
    confusion_matrices[classifier_name] = confusion
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-15-fd5c599f42fa> in <cell line: 2>()
      6
      7     # Fit the model to training data
----> 8     grid_search.fit(train_x, train_Y_bin)
      9
     10     # Get the best model

6 frames

/usr/local/lib/python3.10/dist-packages/joblib/parallel.py in _retrieve(self)
   1705                 (self._jobs[0].get_status(
   1706                     timeout=self.timeout) == TASK_PENDING)):
-> 1707                 time.sleep(0.01)
   1708                 continue
   1709

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
# Define label names
label_names = ['normal', 'dos', 'probe', 'r2l', 'u2r']

# Visualize and compare the accuracy of different algorithms
plt.figure(figsize=(12, 6))
plt.bar(results.keys(), [result["Accuracy"] for result in results.values()], color='skyblue')
plt.xlabel("Algorithms")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison of Different Algorithms")
plt.ylim([0, 1])
plt.xticks(rotation=45)
plt.show()
```

```
# Plot the confusion matrix for each scenario
for classifier_name, confusion_matrix in confusion_matrices.items():
    plt.figure(figsize=(8, 6))
    ConfusionMatrixDisplay(confusion_matrix, display_labels=label_names).plot(cmap='Blues', xticks_rotation='horizontal')
    plt.title(f"Confusion Matrix - {classifier_name}")
    plt.show()
```