```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
import xgboost as xgb
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix, classification_report, f1_score

df1 = pd.read_csv("Low_samples_Aug_RF.csv")
df2 = pd.read_csv("Mid_samples_RF.csv")
df3 = pd.read_csv("High_samples_Aug_RF.csv")
df = pd.concat([df1, df2, df3], ignore_index=True)

test_df = pd.read_csv("./Test_Balanced_RF.csv")

df.columns = df.columns.str.strip()
test_df.columns = test_df.columns.str.strip()

features = ['Total Length of Fwd Packets', 'Total Length of Bwd Packets',
            'Fwd Packet Length Max', 'Fwd Packet Length Mean', 'Bwd Packet Length Max',
            'Bwd Packet Length Min', 'Bwd Packet Length Mean', 'Bwd Packet Length Std',
            'Flow Packets/s', 'Flow IAT Max', 'Fwd IAT Total', 'Fwd IAT Mean',
            'Fwd IAT Std', 'Fwd IAT Max', 'Fwd Header Length', 'Max Packet Length',
            'Packet Length Mean', 'Packet Length Std', 'Packet Length Variance',
            'PSH Flag Count', 'Average Packet Size', 'Avg Fwd Segment Size',
            'Avg Bwd Segment Size', 'Fwd Header Length.1', 'Subflow Fwd Bytes',
            'Subflow Bwd Bytes', 'Init_Win_bytes_forward', 'Init_Win_bytes_backward']

def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(axis=1)
    return df[indices_to_keep].astype(np.float64)

X = df[features]
y = df["Label"]
X = clean_dataset(X)
y = y[X.index]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

le = LabelEncoder()
y_encoded = le.fit_transform(y)

random_forest = RandomForestClassifier(random_state=42)
xgboost_classifier = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
adaboost_classifier = AdaBoostClassifier(random_state=42)
classifiers = [random_forest, xgboost_classifier, adaboost_classifier]
```

```
/tmp/ipykernel_1492856/986576310.py:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df.dropna(inplace=True)
```

```
n_splits = 5
skf = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42)

best_ensemble_f1_score = 0
best_meta_model = None
total_ensemble_f1_score = 0  # To accumulate F1-scores

for train_index, val_index in skf.split(X_scaled, y_encoded):
    X_train, X_val = X_scaled[train_index], X_scaled[val_index]
    y_train, y_val = y_encoded[train_index], y_encoded[val_index]

    base_model_predictions = []
```

```
    for classifier in classifiers:
        classifier.fit(X_train, y_train)
        predictions = classifier.predict(X_val)
        base_model_predictions.append(predictions)

    meta_feature_matrix = np.column_stack(base_model_predictions)

    meta_model = RandomForestClassifier(random_state=42)
    meta_model.fit(meta_feature_matrix, y_val)

    base_model_test_predictions = []

    for classifier in classifiers:
        predictions = classifier.predict(X_scaled)
        base_model_test_predictions.append(predictions)

    meta_feature_matrix_test = np.column_stack(base_model_test_predictions)
    final_predictions = meta_model.predict(meta_feature_matrix_test)

    ensemble_f1_score = f1_score(y_encoded, final_predictions, average='macro')

    total_ensemble_f1_score += ensemble_f1_score  # Accumulate F1-scores

    if ensemble_f1_score > best_ensemble_f1_score:
        best_ensemble_f1_score = ensemble_f1_score
        best_meta_model = meta_model

    print("Ensemble F1-score for fold:", ensemble_f1_score)

print("Best Ensemble F1-score:", best_ensemble_f1_score)

# Calculate and print the average F1-score
avg_ensemble_f1_score = total_ensemble_f1_score / n_splits
print("Average Ensemble F1-score:", avg_ensemble_f1_score)
```

```
Ensemble F1-score for fold: 0.9760517985938606
Ensemble F1-score for fold: 0.9836886951581716
Ensemble F1-score for fold: 0.9738723828552652
Ensemble F1-score for fold: 0.9794918642434244
Ensemble F1-score for fold: 0.9757456779399812
Best Ensemble F1-score: 0.9836886951581716
Average Ensemble F1-score: 0.9777700837581407
```

```
best_meta_model_predictions_val = best_meta_model.predict(meta_feature_matrix)

# Inverse transform to get original class labels for validation set
y_val_actual = le.inverse_transform(y_val)
best_meta_model_predictions_val_actual = le.inverse_transform(best_meta_model_predictions_val)

# Calculate and print the ensemble F1-score on the validation set
ensemble_f1_score_val = f1_score(y_val_actual, best_meta_model_predictions_val_actual, average='macro')
print("Ensemble F1-score on validation set:", ensemble_f1_score_val)
```

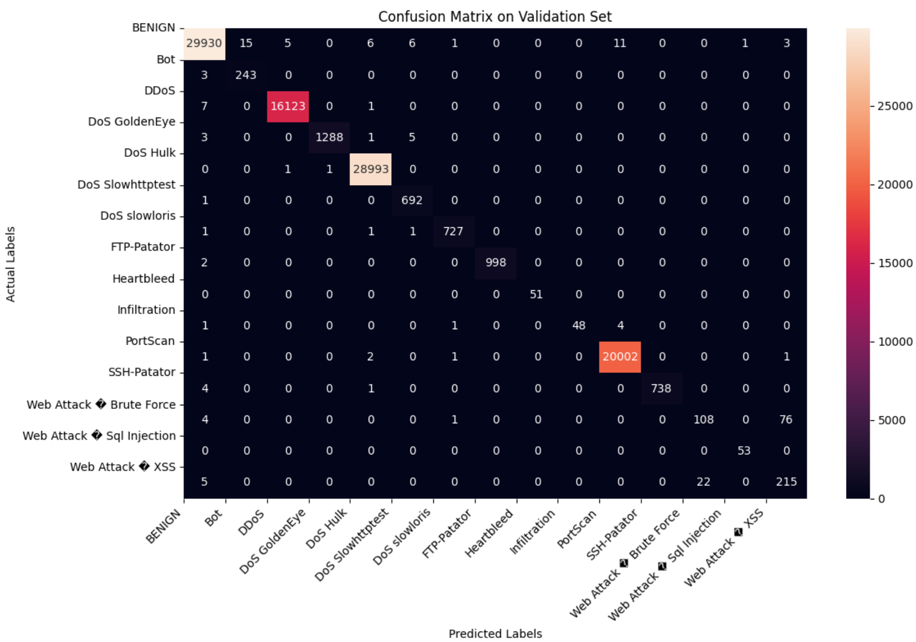
```
Ensemble F1-score on validation set: 0.9566084378224333
```

```
# Calculate and print the classification report on the validation set
print("Classification Report on validation set:")
print(classification_report(y_val_actual, best_meta_model_predictions_val_actual))
```

```
Classification Report on validation set:
                          precision    recall  f1-score   support

                  BENIGN       1.00      1.00      1.00     29978
                     Bot       0.94      0.99      0.96       246
                    DDoS       1.00      1.00      1.00     16131
           DoS GoldenEye       1.00      0.99      1.00      1297
                DoS Hulk       1.00      1.00      1.00     28995
        DoS Slowhttptest       0.98      1.00      0.99       693
           DoS slowloris       0.99      1.00      1.00       730
             FTP-Patator       1.00      1.00      1.00      1000
              Heartbleed       1.00      1.00      1.00        51
            Infiltration       1.00      0.89      0.94        54
                PortScan       1.00      1.00      1.00     20007
             SSH-Patator       1.00      0.99      1.00       743
```

```
  Web Attack � Brute Force       0.83      0.57      0.68       189
Web Attack � Sql Injection       0.98      1.00      0.99        53
          Web Attack � XSS       0.73      0.89      0.80       242

                  accuracy                           1.00    100409
                 macro avg       0.96      0.95      0.96    100409
              weighted avg       1.00      1.00      1.00    100409
```

```
# Calculate and print the confusion matrix on the validation set
cm_val = confusion_matrix(y_val_actual, best_meta_model_predictions_val_actual)
plt.figure(figsize=(12, 8))  # Adjust the figure size
sns.heatmap(cm_val, annot=True, fmt='.0f')
plt.title("Confusion Matrix on Validation Set")
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.xticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=45, ha='right')  # Rotate and adjust alignment
plt.yticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=0, va='center')  # Adjust alignment
plt.tight_layout()  # Improve layout
plt.show()
```



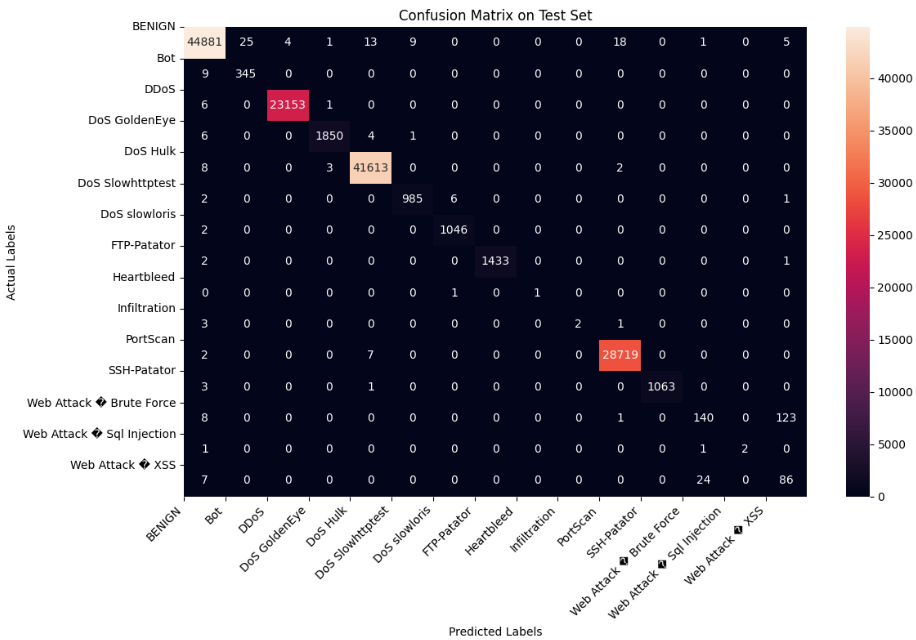
```
base_model_test_predictions = []

for classifier in classifiers:
    predictions = classifier.predict(X_test_scaled)
    base_model_test_predictions.append(predictions)

meta_feature_matrix_test = np.column_stack(base_model_test_predictions)
final_predictions_test = best_meta_model.predict(meta_feature_matrix_test)

# Calculate and print the ensemble F1-score on the test set
ensemble_f1_score_test = f1_score(y_test_actual, final_predictions_test_actual, average='macro')
print("Ensemble F1-score on test set:", ensemble_f1_score_test)
```

```
Ensemble F1-score on test set: 0.861223826909521
```

```
# Inverse transform to get original class labels for the test set
final_predictions_test_actual = le.inverse_transform(final_predictions_test)

# Calculate and print the classification report on the test set
print("Classification Report on test set:")
print(classification_report(y_test_actual, final_predictions_test_actual))
```

```
Classification Report on test set:
                            precision    recall  f1-score   support

                    BENIGN       1.00      1.00      1.00     44957
                       Bot       0.93      0.97      0.95       354
                      DDoS       1.00      1.00      1.00     23160
             DoS GoldenEye       1.00      0.99      1.00      1861
                  DoS Hulk       1.00      1.00      1.00     41626
          DoS Slowhttptest       0.99      0.99      0.99       994
             DoS slowloris       0.99      1.00      1.00      1048
               FTP-Patator       1.00      1.00      1.00      1436
                Heartbleed       1.00      0.50      0.67         2
              Infiltration       1.00      0.33      0.50         6
                  PortScan       1.00      1.00      1.00     28728
               SSH-Patator       1.00      1.00      1.00      1067
  Web Attack � Brute Force       0.84      0.51      0.64       272
Web Attack � Sql Injection       1.00      0.50      0.67         4
          Web Attack � XSS       0.40      0.74      0.52       117

                  accuracy                           1.00    145632
                 macro avg       0.94      0.84      0.86    145632
              weighted avg       1.00      1.00      1.00    145632
```

```
# Calculate and print the confusion matrix on the test set
cm_test = confusion_matrix(y_test_actual, final_predictions_test_actual)
plt.figure(figsize=(12, 8))  # Adjust the figure size
sns.heatmap(cm_test, annot=True, fmt='.0f')
plt.title("Confusion Matrix on Test Set")
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.xticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=45, ha='right')
plt.yticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=0, va='center')
plt.tight_layout()
plt.show()
```




```
# Saving the model file


# import pickle
# model_filename = "ULAK_Ensemble.pkl"
# with open(model_filename, 'wb') as model_file:
#     pickle.dump(best_meta_model, model_file)
# print("Best meta-model saved as", model_filename)
```

```
     Best meta-model saved as ULAK_Ensemble.pkl
```

```
import pickle

# Save the trained models, scaler, and encoder to a pickle file
trained_models = {
    'random_forest': random_forest,
    'xgboost_classifier': xgboost_classifier,
    'adaboost_classifier': adaboost_classifier,
    'meta_model': best_meta_model,
    'label_encoder': le,
    'scaler': scaler
}

with open('trained_models.pkl', 'wb') as file:
    pickle.dump(trained_models, file)
```

Test your custom file here using the trained model.

Just change the file name and path of the model and testing file

```
model_path= "trained_models.pkl"
data_path = "Test.csv"
```

```
import pickle
import pandas as pd
import numpy as np
from sklearn.metrics import f1_score

# Load the saved models and preprocessing objects from pickle file
with open(model_path, 'rb') as file:
    trained_models = pickle.load(file)
```

```
# User provides the path to the test data CSV file

test_df = pd.read_csv(data_path)
test_df.columns = test_df.columns.str.strip()

# List of features used during training
test_features = ['Total Length of Fwd Packets', 'Total Length of Bwd Packets',
            'Fwd Packet Length Max', 'Fwd Packet Length Mean', 'Bwd Packet Length Max',
            'Bwd Packet Length Min', 'Bwd Packet Length Mean', 'Bwd Packet Length Std',
            'Flow Packets/s', 'Flow IAT Max', 'Fwd IAT Total', 'Fwd IAT Mean',
            'Fwd IAT Std', 'Fwd IAT Max', 'Fwd Header Length', 'Max Packet Length',
            'Packet Length Mean', 'Packet Length Std', 'Packet Length Variance',
            'PSH Flag Count', 'Average Packet Size', 'Avg Fwd Segment Size',
            'Avg Bwd Segment Size', 'Fwd Header Length.1', 'Subflow Fwd Bytes',
            'Subflow Bwd Bytes', 'Init_Win_bytes_forward', 'Init_Win_bytes_backward']

# Select features for testing
X_test = test_df[test_features]
Y_test= test_df["Label"]
```

```
# Clean and scale the test data using the saved scaler
def clean_dataset(df):
    assert isinstance(df, pd.DataFrame), "df needs to be a pd.DataFrame"
    df.dropna(inplace=True)
    indices_to_keep = ~df.isin([np.nan, np.inf, -np.inf]).any(axis=1)
    return df[indices_to_keep].astype(np.float64)

X_test = clean_dataset(X_test)

Y_test = Y_test[X_test.index]

#Y_test = clean_dataset(Y_test)
X_test_scaled = trained_models['scaler'].transform(X_test)

# Predict using the loaded models
random_forest = trained_models['random_forest']
xgboost_classifier = trained_models['xgboost_classifier']
adaboost_classifier = trained_models['adaboost_classifier']
meta_model = trained_models['meta_model']
label_encoder = trained_models['label_encoder']

# Generate predictions from the base models on the test data
rf_predictions = random_forest.predict(X_test_scaled)
xgb_predictions = xgboost_classifier.predict(X_test_scaled)
ada_predictions = adaboost_classifier.predict(X_test_scaled)

# Combine the predictions with the original features to create the new feature matrix for the meta-model
base_model_predictions = np.column_stack((rf_predictions, xgb_predictions, ada_predictions))

# Predict using the meta-model
final_predictions = meta_model.predict(base_model_predictions)

# Inverse transform to get original labels
final_predictions_actual = label_encoder.inverse_transform(final_predictions)

# Print the predicted labels for the user to see
print("Predicted labels:", final_predictions_actual)
```

```
from sklearn.metrics import classification_report

# Load the true labels for the test data
true_labels = test_df["Label"]

# Calculate and print the classification report
classification_rep = classification_report(Y_test, final_predictions_actual)
print("Classification Report:\n", classification_rep)
```

```
Classification Report:
                              precision    recall  f1-score   support

                     BENIGN       1.00      1.00      1.00     44957
```

```
                         Bot       0.93      0.97      0.95       354
                        DDoS       1.00      1.00      1.00     23160
               DoS GoldenEye       1.00      0.99      1.00      1861
                    DoS Hulk       1.00      1.00      1.00     41626
            DoS Slowhttptest       0.99      0.99      0.99       994
               DoS slowloris       0.99      1.00      1.00      1048
                 FTP-Patator       1.00      1.00      1.00      1436
                  Heartbleed       1.00      0.50      0.67         2
                Infiltration       1.00      0.33      0.50         6
                    PortScan       1.00      1.00      1.00     28728
                 SSH-Patator       1.00      1.00      1.00      1067
    Web Attack � Brute Force       0.84      0.51      0.64       272
  Web Attack � Sql Injection       1.00      0.50      0.67         4
            Web Attack � XSS       0.40      0.74      0.52       117

                    accuracy                           1.00    145632
                   macro avg       0.94      0.84      0.86    145632
                weighted avg       1.00      1.00      1.00    145632
```

```
# Calculate and print the confusion matrix

cm_test = confusion_matrix(Y_test, final_predictions_test_actual)
plt.figure(figsize=(12, 8))  # Adjust the figure size
sns.heatmap(cm_test, annot=True, fmt='.0f')
plt.title("Confusion Matrix on Test Set")
plt.xlabel("Predicted Labels")
plt.ylabel("Actual Labels")
plt.xticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=45, ha='right')
plt.yticks(ticks=np.arange(len(class_names)), labels=class_names, rotation=0, va='center')
plt.tight_layout()
plt.show()
```