RandomForest

June 17, 2025

[1]:
```python
import warnings
import os
import sys
import time
import joblib
import pandas as pd
import matplotlib.pyplot as plt
import shap

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler

warnings.filterwarnings('ignore')

try:
    from data_processing import DataProcessor
    from model_evaluation import evaluate_model
except ImportError:
    if '__file__' in globals():
        script_dir = os.path.dirname(os.path.abspath(__file__))
        if os.path.basename(script_dir) == 'src':
            project_root = os.path.dirname(script_dir)
        else:
            project_root = script_dir
    else:
        current_dir = os.getcwd()
        project_root = os.path.dirname(current_dir) if current_dir.
 ↪endswith('notebooks') else current_dir

    src_dir = os.path.join(project_root, 'src')
    if src_dir not in sys.path:
        sys.path.append(src_dir)

    from data_processing import DataProcessor
    from model_evaluation import evaluate_model
```

```python
if '__file__' in globals():
    script_dir = os.path.dirname(os.path.abspath(__file__))
    if os.path.basename(script_dir) == 'src':
        project_root = os.path.dirname(script_dir)
    else:
        project_root = script_dir
else:
    current_dir = os.getcwd()
    project_root = os.path.dirname(current_dir) if current_dir.
 ↪endswith('notebooks') else current_dir

data_dir = os.path.join(project_root, 'data')

cic_pkl_file_name = os.path.join(data_dir, "cic_dataframe.pkl")
cic_file_paths = [
    os.path.join(data_dir, f"CIC/nfstream/{day}-WorkingHours.
 ↪pcap_nfstream_labeled.csv")
    for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
]

tcpdump_pkl_file_name = os.path.join(data_dir, "tcpdump_dataframe.pkl")
tcpdump_file_paths = [
    os.path.join(data_dir, f"tcpdump/nfstream/{filename}_labeled.csv")
    for filename in [
        "normal_01",
        "normal_02",
        "normal_and_attack_01",
        "normal_and_attack_02",
        "normal_and_attack_03",
        "normal_and_attack_04",
        "normal_and_attack_05",
    ]
]

# Constants
NORMAL_LABEL = 1
ANOMALY_LABEL = -1

# Configuration
test_size = 0.2
random_state = 42
scaled = False
encode_categorical = True
shap_enabled = True
dev_mode = False
corr_threshold = 0.95
```

[2]:
```
# 1. Load and prepare the data
print("\nStep 1: Load and prepare the data")
if os.path.exists(cic_pkl_file_name):
    print(f"Loading dataframe from {cic_pkl_file_name}")
    dataframe = pd.read_pickle(cic_pkl_file_name)
else:
    print(f"Creating dataframe from pcap files and saving to␣
 ↪{cic_pkl_file_name}")
    dataframe = DataProcessor.get_dataframe(file_paths=cic_file_paths)
    dataframe.to_pickle(cic_pkl_file_name)

# Add new featueres:
print("Adding new features to the dataframe")
dataframe = DataProcessor.add_new_features(dataframe)

# Drop object columns and handle categorical data
print("Dropping object columns except for some categorical columns")
df_without_object, available_categorical = DataProcessor.drop_object_columns(
            dataframe, encode_categorical=encode_categorical
        )

# Split into features and labels
print("Splitting data into features (X) and labels (y)")
X, y = DataProcessor.split_to_X_y(df_without_object)

# Clean the data
print("Cleaning data")
DataProcessor.clean_data(X)

print(f"X.shape: {X.shape}")
print(f"y.shape: {y.shape}")

# Split the data into training and test sets
print(f"Splitting data into train and test sets with test_size={test_size}")
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_size, random_state=random_state, stratify=y
)

# Handle categorical encoding
print("Handling categorical encoding")
X_train, X_test, categorical_encoder = (
    DataProcessor.one_hot_encode_categorical(
        X_train, X_test, available_categorical, None
    )
)

print(f"X_train.shape: {X_train.shape}")
```

```python
print(f"X_test.shape: {X_test.shape}")

# Scaling the data
if scaled:
    print("Scaleing the data")
    scaler = MinMaxScaler()
    print("New MinMaxScaler instance created")
    X_train = pd.DataFrame(
        scaler.fit_transform(X_train),
        columns=X_train.columns,
        index=X_train.index,
    )
    X_test = pd.DataFrame(
        scaler.transform(X_test),
        columns=X_test.columns,
        index=X_test.index,
    )
else:
    scaler = None

# Label conversion
print("Converting labels: benign to 1 and anomalous to -1")
y_train = y_train.map(lambda x: 1 if x == "benign" else -1)
y_test = y_test.map(lambda x: 1 if x == "benign" else -1)

# Feature selection
features_to_drop = DataProcessor.get_features_to_drop()
print(f"Always drop id, src, timestamp...: {features_to_drop}")
X_train = X_train.drop(columns=features_to_drop)
X_test = X_test.drop(columns=features_to_drop)
print(f"Droped {len(features_to_drop)} features")

# Remove highly correlated features
print(f"Dropping highly correlated features with threshold={corr_threshold}")
X_train, dropped_corr = DataProcessor.
 ↪remove_highly_correlated_features(X_train, threshold=corr_threshold)
X_test = X_test.drop(columns=dropped_corr)
print(f"Droped {len(dropped_corr)} features: {dropped_corr}")

print(f"X_train.shape: {X_train.shape}")
print(f"y_train.shape: {y_train.shape}")
print(f"X_test.shape: {X_test.shape}")
print(f"y_test.shape: {y_test.shape}")

# reset index to ensure consistent indexing
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
```

```python
X_test = X_test.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

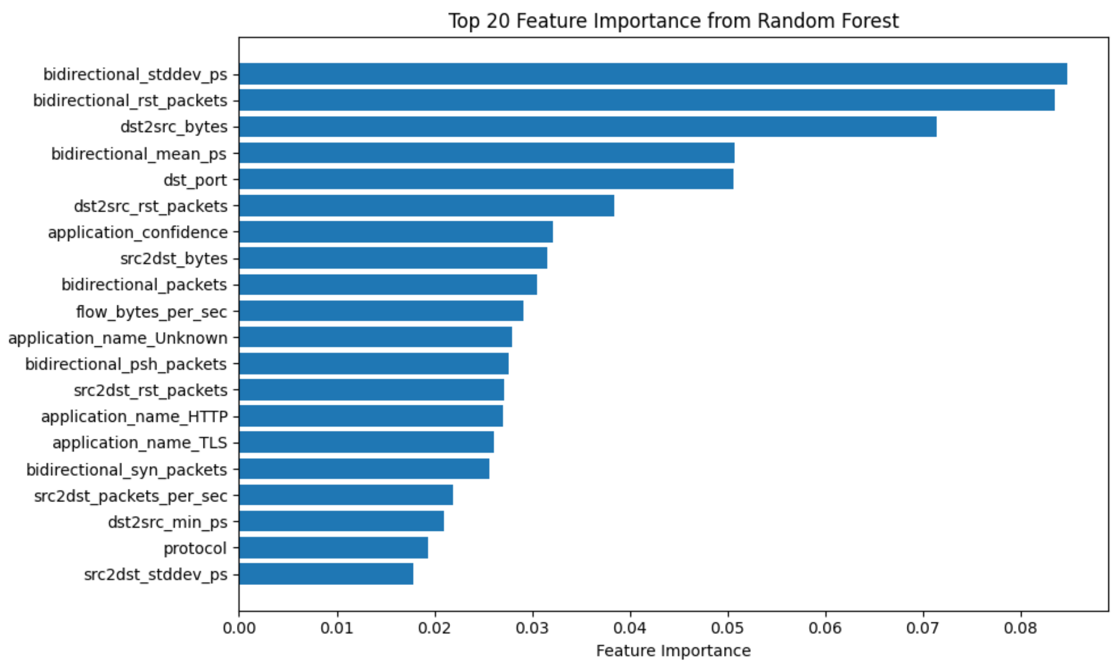
```
Step 1: Load and prepare the data
Loading dataframe from /Users/seu/Library/CloudStorage/Dropbox/Docs/Study/Weiter
bildung/DBS/Kurse/Projektarbeit/DS/repo/data/cic_dataframe.pkl
Adding new features to the dataframe
flow rate (bytes/sec and packets/sec)
packet rate (bytes/sec and packets/sec)
Down/Up Ratio
Dropping object columns except for some categorical columns
Retaining categorical features for encoding: ['application_name',
'application_category_name']
Number of columns before dropping object columns: 92
Dropped object columns (11): ['src_ip', 'src_mac', 'src_oui', 'dst_ip',
'dst_mac', 'dst_oui', 'requested_server_name', 'client_fingerprint',
'server_fingerprint', 'user_agent', 'content_type']
Number of columns after dropping object columns: 81
Splitting data into features (X) and labels (y)
Cleaning data
X.shape: (2916222, 80)
y.shape: (2916222,)
Splitting data into train and test sets with test_size=0.2
Handling categorical encoding
Processing categorical features: ['application_name',
'application_category_name']
Creating new OneHotEncoder
Added 261 one-hot encoded features
X_train.shape: (2332977, 339)
X_test.shape: (583245, 339)
Converting labels: benign to 1 and anomalous to -1
Always drop id, src, timestamp…: ['id', 'src_port',
'bidirectional_first_seen_ms', 'bidirectional_last_seen_ms',
'src2dst_first_seen_ms', 'src2dst_last_seen_ms', 'dst2src_first_seen_ms',
'dst2src_last_seen_ms']
Droped 8 features
Dropping highly correlated features with threshold=0.95
Droped 24 features: ['bidirectional_bytes', 'src2dst_duration_ms',
'src2dst_packets', 'dst2src_duration_ms', 'dst2src_packets',
'bidirectional_max_ps', 'src2dst_max_ps', 'dst2src_mean_ps',
'dst2src_stddev_ps', 'dst2src_max_ps', 'src2dst_max_piat_ms',
'bidirectional_ack_packets', 'src2dst_syn_packets', 'src2dst_cwr_packets',
'src2dst_ece_packets', 'src2dst_ack_packets', 'dst2src_ece_packets',
'dst2src_ack_packets', 'application_category_name_Download',
'application_category_name_Game', 'application_category_name_Mining',
'application_category_name_RPC', 'application_category_name_Shopping',
'application_category_name_Unspecified']
```

```
X_train.shape: (2332977, 307)
y_train.shape: (2332977,)
X_test.shape: (583245, 307)
y_test.shape: (583245,)
```

[3]:
```python
# 2. Feature Selection
print("\nStep 2: Feature Selection using Random Forest Classifier")

rfc_selector = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',
    random_state=random_state,
    n_jobs=-1
)

rfc_selector.fit(X_train, y_train)

# Feature Importance Extraction
feature_importance = rfc_selector.feature_importances_
feature_names = X_train.columns
# sort features by importance
importance_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importance
}).sort_values('importance', ascending=False)

print(f"Total features: {len(feature_names)}")
print(f"Random Forest with {rfc_selector.n_estimators} trees trained")
print("\nTop 20 most important features:")
print(importance_df.head(20))

# visualize feature importance
plt.figure(figsize=(10, 6))
top_features = importance_df.head(20)
plt.barh(range(len(top_features)), top_features['importance'])
plt.yticks(range(len(top_features)), top_features['feature'])
plt.xlabel('Feature Importance')
plt.title('Top 20 Feature Importance from Random Forest')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show(block=False)
```

```
Step 2: Feature Selection using Random Forest Classifier
Total features: 307
```

```
Random Forest with 100 trees trained

Top 20 most important features:
                      feature  importance
12      bidirectional_stddev_ps    0.084748
33    bidirectional_rst_packets    0.083522
9                 dst2src_bytes    0.071479
11        bidirectional_mean_ps    0.050756
1                      dst_port    0.050579
43          dst2src_rst_packets    0.038378
46       application_confidence    0.032124
8                 src2dst_bytes    0.031562
7         bidirectional_packets    0.030574
47           flow_bytes_per_sec    0.029129
281    application_name_Unknown    0.027939
32    bidirectional_psh_packets    0.027555
37          src2dst_rst_packets    0.027199
137       application_name_HTTP    0.027083
217        application_name_TLS    0.026072
28    bidirectional_syn_packets    0.025665
49       src2dst_packets_per_sec    0.021941
16               dst2src_min_ps    0.020947
2                      protocol    0.019319
15           src2dst_stddev_ps    0.017827
```



[4]:

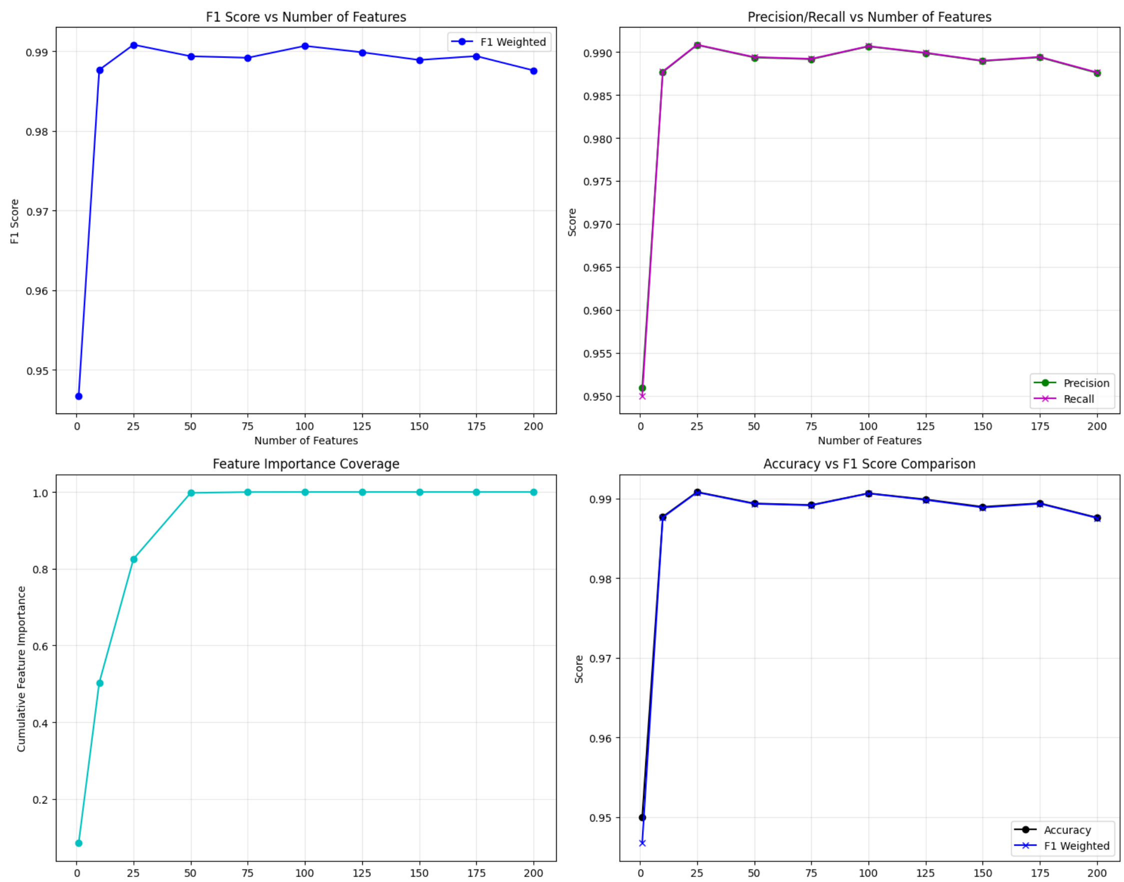
```
# 3. Compare number of features and cumulative importance
print("\nStep 3: Compare number of features and cumulative importance")
target_features_list = [1, 10, 25, 50, 75, 100, 125, 150, 175, 200]
results_comparison = []

for n_features in target_features_list:
    # select top n_features based on importance
    top_features_df = importance_df.head(n_features)
    top_features_list = top_features_df['feature'].tolist()

    X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(
        X_train, y_train,
        test_size=test_size,
        random_state=random_state,
        stratify=y_train
    )

    X_train_split_selected = X_train_split[top_features_list]
    X_val_split_selected = X_val_split[top_features_list]

    # print(f"X_train_split_selected.shape: {X_train_split_selected.shape}")
    # print(f"X_val_split_selected.shape: {X_val_split_selected.shape}")

    # Validate model for selected features
    test_rf = RandomForestClassifier(
        n_estimators=20,
        max_depth=5,
        min_samples_split=5,
        min_samples_leaf=2,
        max_features='sqrt',
        random_state=random_state,
    )

    test_rf.fit(X_train_split_selected, y_train_split)
    y_pred = test_rf.predict(X_val_split_selected)

    # metrics calculation
    accuracy = test_rf.score(X_val_split_selected, y_val_split)
    f1 = f1_score(y_val_split, y_pred, average='weighted')  # weighted average␣
↪for unbalanced classes
    precision = precision_score(y_val_split, y_pred, average='weighted')
    recall = recall_score(y_val_split, y_pred, average='weighted')
    print(f"Features: {n_features:2d}, F1(weighted): {f1:.4f}, "
          f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Recall:␣
↪{recall:.4f}")

    # cumulative importance of selected features
```

```
    cumulative_importance = top_features_df['importance'].sum()

    results_comparison.append({
        'n_features': n_features,
        'accuracy': accuracy,
        'f1_weighted': f1,
        'precision': precision,
        'recall': recall,
        'cumulative_importance': cumulative_importance
    })

# visualize results
results_df = pd.DataFrame(results_comparison)
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 12))

# n_features vs F1 Score
ax1.plot(results_df['n_features'], results_df['f1_weighted'], 'bo-', label='F1␣
↪Weighted')
ax1.set_xlabel('Number of Features')
ax1.set_ylabel('F1 Score')
ax1.set_title('F1 Score vs Number of Features')
ax1.legend()
ax1.grid(True, alpha=0.3)

# n_features vs Precision/Recall
ax2.plot(results_df['n_features'], results_df['precision'], 'go-',␣
↪label='Precision')
ax2.plot(results_df['n_features'], results_df['recall'], 'mx-', label='Recall')
ax2.set_xlabel('Number of Features')
ax2.set_ylabel('Score')
ax2.set_title('Precision/Recall vs Number of Features')
ax2.legend()
ax2.grid(True, alpha=0.3)

# n_features vs Cumulative Feature Importance
ax3.plot(results_df['n_features'], results_df['cumulative_importance'], 'co-',␣
↪label='Cumulative Importance')
ax3.set_xlabel('Number of Features')
ax3.set_ylabel('Cumulative Feature Importance')
ax3.set_title('Feature Importance Coverage')
ax3.grid(True, alpha=0.3)

# f1 Score vs Accuracy Comparison
ax4.plot(results_df['n_features'], results_df['accuracy'], 'ko-',␣
↪label='Accuracy')
ax4.plot(results_df['n_features'], results_df['f1_weighted'], 'bx-', label='F1␣
↪Weighted')
```

```
ax4.set_xlabel('Number of Features')
ax4.set_ylabel('Score')
ax4.set_title('Accuracy vs F1 Score Comparison')
ax4.legend()
ax4.grid(True, alpha=0.3)

plt.tight_layout()
plt.show(block=False)

# reusult summary
print("\n=== Feature Selection Results Summary (All Metrics) ===")
print(results_df.round(4))

# optimal feature selection based on F1 score
best_f1_idx = results_df['f1_weighted'].idxmax()
best_f1_result = results_df.iloc[best_f1_idx]

print("\n=== Optimal Feature Selection Results ===")
print(f"Best F1 (weighted): {best_f1_result['f1_weighted']:.4f} with␣
↪{best_f1_result['n_features']} features")
print(f"  - Accuracy: {best_f1_result['accuracy']:.4f}")
print(f"  - Precision: {best_f1_result['precision']:.4f}")
print(f"  - Recall: {best_f1_result['recall']:.4f}")
print(f"  - Feature importance coverage:␣
↪{best_f1_result['cumulative_importance']:.3f}")

print("\nOptimal Feature Selection based on F1 Score")
print("=== optimal features ===")
optimal_f1_features = int(best_f1_result['n_features'])
optimal_features_list = importance_df.head(optimal_f1_features)['feature'].
↪tolist()

X_train_optimal = X_train[optimal_features_list]
X_test_optimal = X_test[optimal_features_list]

print(f"optimal_features_list = {optimal_features_list}")
print(f"X_train_optimal.shape: {X_train_optimal.shape}")
```

```
Step 3: Compare number of features and cumulative importance
Features:  1, F1(weighted): 0.9467, Accuracy: 0.9500, Precision: 0.9509, Recall:
0.9500
Features: 10, F1(weighted): 0.9876, Accuracy: 0.9877, Precision: 0.9877, Recall:
0.9877
Features: 25, F1(weighted): 0.9908, Accuracy: 0.9908, Precision: 0.9908, Recall:
0.9908
Features: 50, F1(weighted): 0.9894, Accuracy: 0.9894, Precision: 0.9894, Recall:
```

```
0.9894
Features: 75, F1(weighted): 0.9892, Accuracy: 0.9892, Precision: 0.9892, Recall:
0.9892
Features: 100, F1(weighted): 0.9906, Accuracy: 0.9907, Precision: 0.9906,
Recall: 0.9907
Features: 125, F1(weighted): 0.9898, Accuracy: 0.9899, Precision: 0.9899,
Recall: 0.9899
Features: 150, F1(weighted): 0.9889, Accuracy: 0.9890, Precision: 0.9889,
Recall: 0.9890
Features: 175, F1(weighted): 0.9894, Accuracy: 0.9894, Precision: 0.9894,
Recall: 0.9894
Features: 200, F1(weighted): 0.9876, Accuracy: 0.9876, Precision: 0.9876,
Recall: 0.9876
```



=== Feature Selection Results Summary (All Metrics) ===

	n_features	accuracy	f1_weighted	precision	recall	cumulative_importance
0	1	0.9500	0.9467	0.9509	0.9500	0.0847
1	10	0.9877	0.9876	0.9877	0.9877	0.5029
2	25	0.9908	0.9908	0.9908	0.9908	0.8254

```
3          50    0.9894       0.9894     0.9894  0.9894                  0.9975
4          75    0.9892       0.9892     0.9892  0.9892                  0.9998
5         100    0.9907       0.9906     0.9906  0.9907                  1.0000
6         125    0.9899       0.9898     0.9899  0.9899                  1.0000
7         150    0.9890       0.9889     0.9889  0.9890                  1.0000
8         175    0.9894       0.9894     0.9894  0.9894                  1.0000
9         200    0.9876       0.9876     0.9876  0.9876                  1.0000

=== Optimal Feature Selection Results ===
Best F1 (weighted): 0.9908 with 25.0 features
  - Accuracy: 0.9908
  - Precision: 0.9908
  - Recall: 0.9908
  - Feature importance coverage: 0.825

Optimal Feature Selection based on F1 Score
=== optimal features ===
optimal_features_list = ['bidirectional_stddev_ps', 'bidirectional_rst_packets',
'dst2src_bytes', 'bidirectional_mean_ps', 'dst_port', 'dst2src_rst_packets',
'application_confidence', 'src2dst_bytes', 'bidirectional_packets',
'flow_bytes_per_sec', 'application_name_Unknown', 'bidirectional_psh_packets',
'src2dst_rst_packets', 'application_name_HTTP', 'application_name_TLS',
'bidirectional_syn_packets', 'src2dst_packets_per_sec', 'dst2src_min_ps',
'protocol', 'src2dst_stddev_ps', 'src2dst_mean_ps', 'flow_packets_psr_sec',
'bidirectional_fin_packets', 'dst2src_ackets_per_sec', 'src2dst_psh_packets']
X_train_optimal.shape: (2332977, 25)
```

[5]:
```python
# 4. Hyperparameter tuning
print("\nStep 4: Hyperparameter Tuning using Grid Search")

def tune_hyperparameters(X_train, y_train, dev_mode=False):
    if dev_mode:
        param_grid = {
            'n_estimators': [100],
            'max_depth': [10],
            'min_samples_split': [5],
            'min_samples_leaf': [2],
            'max_features': ['sqrt'],
            'random_state': [random_state]
        }
        cv = 2
    else:
        param_grid = {
            'n_estimators': [50, 100],
            'max_depth': [5, 7, 10],
            'min_samples_split': [2, 5],
            'min_samples_leaf': [1, 2],
```

```
            'max_features': ['sqrt', 'log2'],
            'random_state': [random_state]
        }
        cv = 3

    rf_tuner = RandomForestClassifier()
    grid_search = GridSearchCV(
        rf_tuner,
        param_grid,
        scoring='f1_weighted',
        cv=cv,
        verbose=2,
        n_jobs=-1
    )

    total_combinations = 1
    for param_values in param_grid.values():
        total_combinations *= len(param_values)

    print(f"Testing {total_combinations} parameter combinations...")
    start_time = time.time()

    grid_search.fit(X_train, y_train)

    end_time = time.time()
    print(f"Grid search completed in {end_time - start_time:.2f} seconds")
    print(f"Best parameters: {grid_search.best_params_}")
    print(f"Best CV score: {grid_search.best_score_:.4f}")

    return grid_search.best_estimator_

# Hyperparameter tuning
best_rf_model = tune_hyperparameters(X_train_optimal, y_train,␣
↪dev_mode=dev_mode)
```

```
Step 4: Hyperparameter Tuning using Grid Search
Testing 48 parameter combinations…
Fitting 3 folds for each of 48 candidates, totalling 144 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  47.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  47.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  47.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  47.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
```

```
min_samples_split=5, n_estimators=50, random_state=42; total time=  47.7s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  48.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  52.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  54.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  45.3s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  46.6s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  45.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  45.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.0min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.1min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  37.7s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  37.6s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  37.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  43.8s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  41.1s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  42.1s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
```

```
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.4min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  39.1s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.4min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  37.8s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.4min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  36.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  37.1s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.3min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  37.4s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.5min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  37.3s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.7min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.2min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  56.7s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  59.8s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.3min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.0min
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.0min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.0min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
```

```
min_samples_split=5, n_estimators=50, random_state=42; total time=  59.9s
[CV] END max_depth=5, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  57.0s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.1min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.0min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.2min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  52.4s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  54.0s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  53.3s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  46.5s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  46.6s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.3min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  56.3s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  48.1s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  49.6s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.6min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  48.8s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
```

```
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.0min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  50.5s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.7min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  51.2s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  50.1s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  50.5s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  52.3s
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  51.0s
[CV] END max_depth=7, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.7min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.7min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.7min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.3min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.3min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=7, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.4min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
```

```
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.3min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.6min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.6min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.4min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.5min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.4min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.8min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.5min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  59.2s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.3min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time=  54.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.1min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.7min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.7min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.7min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time=  59.0s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.0min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  56.5s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
```

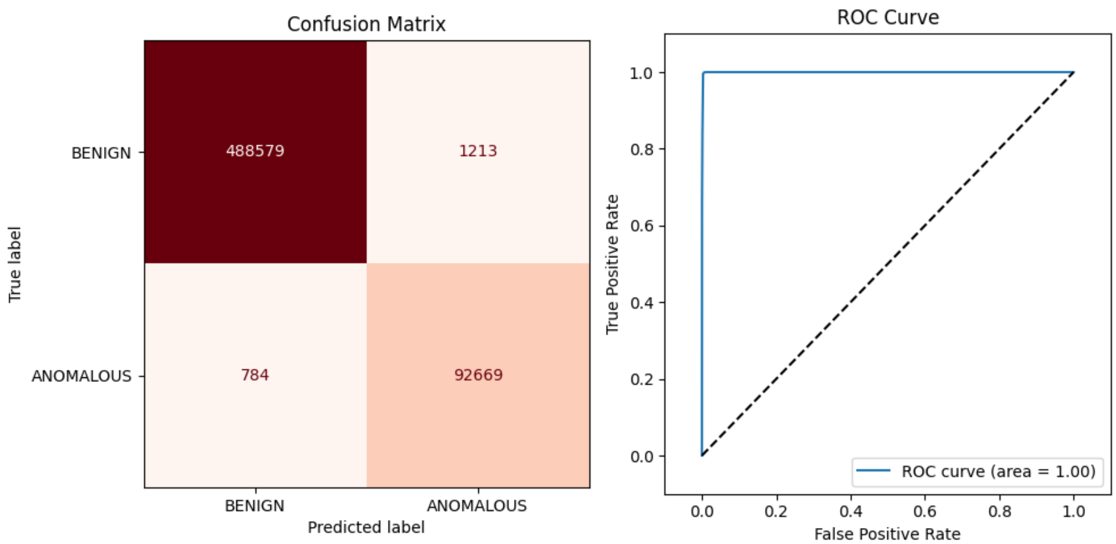
```
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.5min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time=  55.9s
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.5min
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.8min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.0min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=50, random_state=42; total time= 1.2min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=2, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  58.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time=  59.6s
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=50, random_state=42; total time= 1.1min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.3min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=1,
min_samples_split=5, n_estimators=100, random_state=42; total time= 2.4min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.8min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=2, n_estimators=100, random_state=42; total time= 1.9min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.7min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.6min
[CV] END max_depth=10, max_features=log2, min_samples_leaf=2,
min_samples_split=5, n_estimators=100, random_state=42; total time= 1.6min
Grid search completed in 1058.37 seconds
Best parameters: {'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf':
1, 'min_samples_split': 2, 'n_estimators': 100, 'random_state': 42}
Best CV score: 0.9965
```

[6]:
```python
# 5. Final Random Forest Classifier  model training and evaluation
print("\nStep 5: Final Random Forest Classifier model training and evaluation")
max_depth = best_rf_model.get_params()['max_depth']
```

```python
max_features = best_rf_model.get_params()['max_features']
min_samples_leaf = best_rf_model.get_params()['min_samples_leaf']
min_samples_split = best_rf_model.get_params()['min_samples_split']
n_estimators = best_rf_model.get_params()['n_estimators']
random_state = best_rf_model.get_params()['random_state']
print(f"Best Random Forest model with parameters: n_estimators={n_estimators}, "
      f"max_depth={max_depth}, min_samples_split={min_samples_split}, "
      f"min_samples_leaf={min_samples_leaf}, max_features={max_features}, "
      f"random_state={random_state}")

rf = RandomForestClassifier(
    n_estimators=n_estimators,
    max_depth=max_depth,
    min_samples_split=min_samples_split,
    min_samples_leaf=min_samples_leaf,
    max_features=max_features,
    random_state=random_state
)
# Train the model
print("Training the Random Forest model with optimal features...")
rf.fit(X_train_optimal, y_train)
print("Model trained successfully.")

evaluate_model(rf, X_test_optimal, y_test)
```

```
Step 5: Final Random Forest Classifier model training and evaluation
Best Random Forest model with parameters: n_estimators=100, max_depth=10,
min_samples_split=2, min_samples_leaf=1, max_features=log2, random_state=42
Training the Random Forest model with optimal features…
Model trained successfully.
using predict_proba for prediction scores

Test Data - BENIGN Count: 489792, Ratio: 83.98%
Test Data - ANOMALOUS Count: 93453, Ratio: 16.02%

Test result:
              precision    recall  f1-score   support

      BENIGN       1.00      1.00      1.00    489792
   ANOMALOUS       0.99      0.99      0.99     93453

    accuracy                           1.00    583245
   macro avg       0.99      0.99      0.99    583245
weighted avg       1.00      1.00      1.00    583245

[[488579   1213]
 [   784  92669]]
```

```
Precision: 0.987
Recall: 0.992
False Positive Rate: 0.002
False Negative Rate: 0.008
F1-Score: 0.989
Area under the curve: 1.000
```

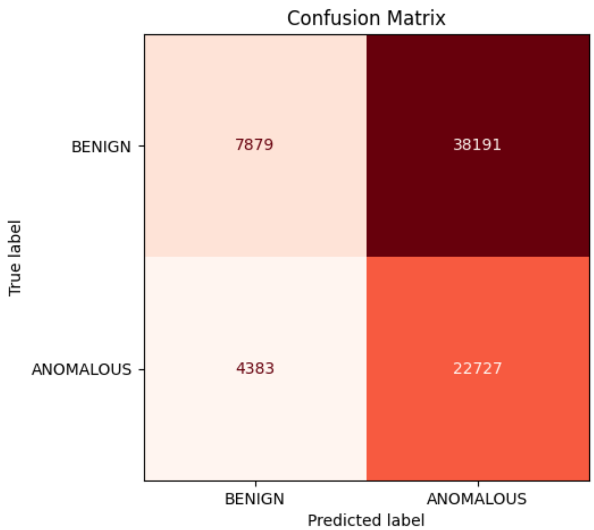


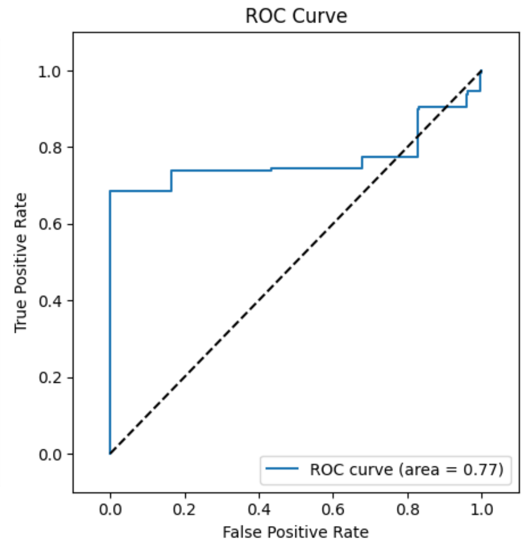
[7]:
```
# 6. Evaluate with tcpdump data
print("\nStep 6: Evaluate with tcpdump data")
if os.path.exists(tcpdump_pkl_file_name):
    print(f"Loading dataframe from {tcpdump_pkl_file_name}")
    tcpdump_dataframe = pd.read_pickle(tcpdump_pkl_file_name)
else:
    print(f"Creating dataframe from pcap files and saving to␣
 ↪{tcpdump_pkl_file_name}")
    tcpdump_dataframe = DataProcessor.
 ↪get_dataframe(file_paths=tcpdump_file_paths)
    tcpdump_dataframe.to_pickle(tcpdump_pkl_file_name)

# Add new featueres:
print("Adding new features to the dataframe...")
tcpdump_dataframe = DataProcessor.add_new_features(tcpdump_dataframe)

# Drop object columns and handle categorical data
print("Dropping object columns and handle encoding categorical data...")
tcpdump_df_without_object, available_categorical = DataProcessor.
 ↪drop_object_columns(
            tcpdump_dataframe, encode_categorical=encode_categorical
```

```python
        )
# Split into features and labels
print("Splitting data into features (X) and labels (y)...")
X_tcpdump, y_tcpdump = DataProcessor.split_to_X_y(tcpdump_df_without_object)

print("Cleaning data...")
DataProcessor.clean_data(X_tcpdump)

print(f"X_tcpdump.shape: {X_tcpdump.shape}")
print(f"y_tcpdump.shape: {y_tcpdump.shape}")

print("Handling categorical encoding...")
print(f"Available categorical features: {available_categorical}")
print(f"Use categorical_encoder: {categorical_encoder}")
X_tcpdump, _, categorical_encoder = (
    DataProcessor.one_hot_encode_categorical(
        X_tcpdump, None, available_categorical, categorical_encoder
    )
)
print(f"X_tcpdump.shape: {X_tcpdump.shape}")

if scaled and scaler is not None:
    print(f"Use MinMaxScaler instance: {scaler}")
    X_tcpdump = pd.DataFrame(
        scaler.transform(X_tcpdump),
        columns=X_tcpdump.columns,
        index=X_tcpdump.index,
    )

y_tcpdump = y_tcpdump.map(lambda x: 1 if x == "benign" else -1)

# Feature selection
print("Feature selection:")
X_tcpdump_optimal = X_tcpdump[optimal_features_list]

print(f"X_tcpdump_optimal.shape: {X_tcpdump_optimal.shape}")
print(f"y_tcpdump.shape: {y_tcpdump.shape}")

evaluate_model(rf, X_tcpdump_optimal, y_tcpdump, with_numpy=True)
```

```
Step 6: Evaluate with tcpdump data
Loading dataframe from /Users/seu/Library/CloudStorage/Dropbox/Docs/Study/Weiter
bildung/DBS/Kurse/Projektarbeit/DS/repo/data/tcpdump_dataframe.pkl
Adding new features to the dataframe…
flow rate (bytes/sec and packets/sec)
packet rate (bytes/sec and packets/sec)
Down/Up Ratio
```

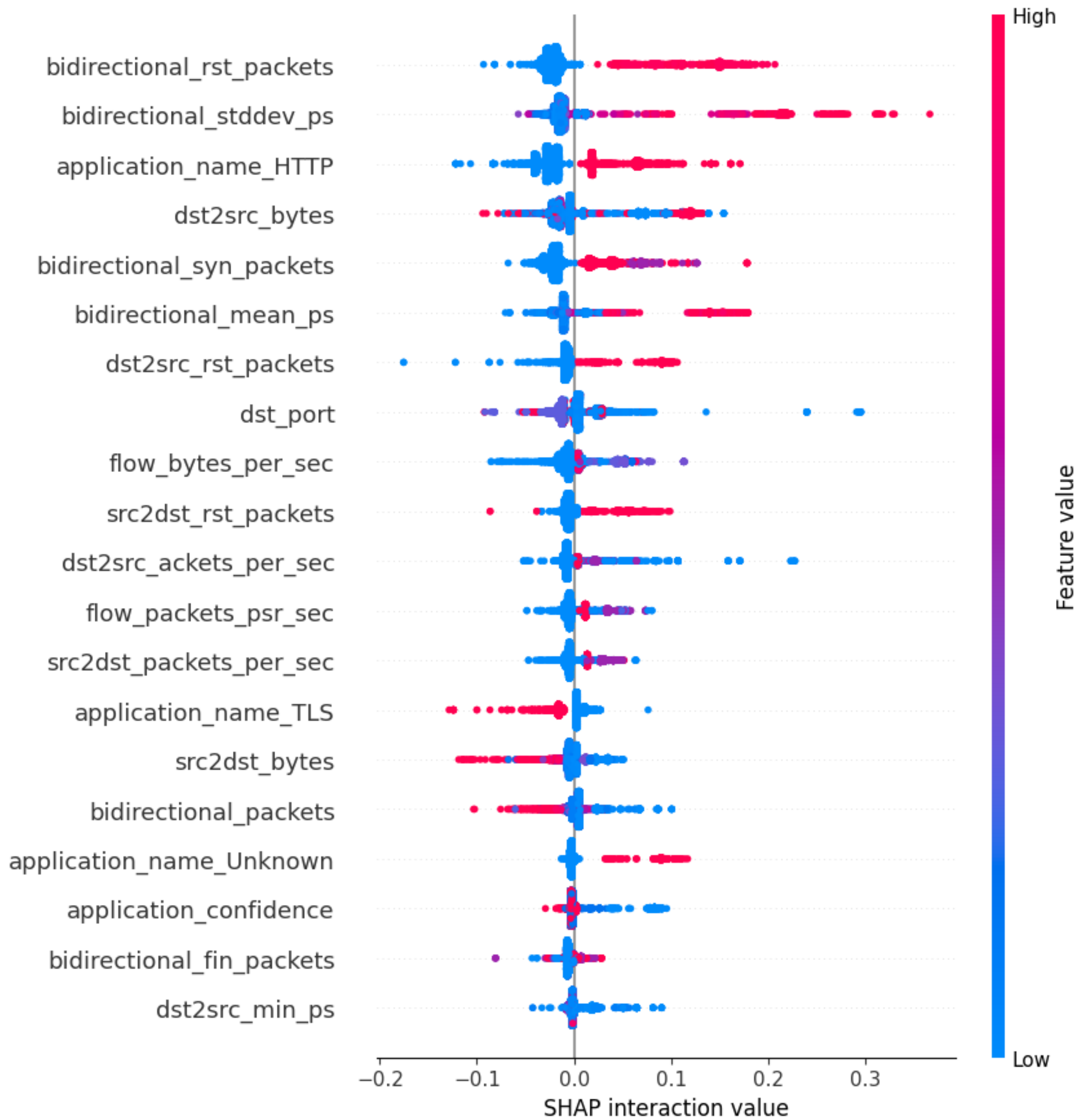
```
Dropping object columns and handle encoding categorical data…
Retaining categorical features for encoding: ['application_name',
'application_category_name']
Number of columns before dropping object columns: 92
Dropped object columns (11): ['src_ip', 'src_mac', 'src_oui', 'dst_ip',
'dst_mac', 'dst_oui', 'requested_server_name', 'client_fingerprint',
'server_fingerprint', 'user_agent', 'content_type']
Number of columns after dropping object columns: 81
Splitting data into features (X) and labels (y)…
Cleaning data…
X_tcpdump.shape: (73180, 80)
y_tcpdump.shape: (73180,)
Handling categorical encoding…
Available categorical features: ['application_name',
'application_category_name']
Use categorical_encoder: OneHotEncoder(drop='first', handle_unknown='ignore',
sparse_output=False)
Processing categorical features: ['application_name',
'application_category_name']
Using existing OneHotEncoder
Added 261 one-hot encoded features
X_tcpdump.shape: (73180, 339)
Feature selection:
X_tcpdump_optimal.shape: (73180, 25)
y_tcpdump.shape: (73180,)
using predict_proba for prediction scores

Test Data - BENIGN Count: 46070, Ratio: 62.95%
Test Data - ANOMALOUS Count: 27110, Ratio: 37.05%

Test result:
              precision    recall  f1-score   support

      BENIGN       0.64      0.17      0.27     46070
   ANOMALOUS       0.37      0.84      0.52     27110

    accuracy                           0.42     73180
   macro avg       0.51      0.50      0.39     73180
weighted avg       0.54      0.42      0.36     73180

[[ 7879 38191]
 [ 4383 22727]]
Precision: 0.373
Recall: 0.838
False Positive Rate: 0.829
False Negative Rate: 0.162
F1-Score: 0.516
Area under the curve: 0.768
```





[8]:
```
# 7. Interpretation with SHAP
shap_enabled = True
if shap_enabled:
    print("\nStep 7: Interpretation with SHAP")
    explainer = shap.TreeExplainer(rf)
    X_test_optimal_sampled = X_test_optimal.sample(n=10000,␣
↪random_state=random_state)
    shap_values = explainer.shap_values(X_test_optimal_sampled)

    plt.figure(figsize=(14, 8))

    if len(shap_values.shape) == 3:
        main_effects = shap_values[:, :, :-1].sum(axis=2)
        shap.summary_plot(main_effects, X_test_optimal_sampled,␣
↪feature_names=optimal_features_list, show=False)
    else:
        shap.summary_plot(shap_values, X_test_optimal_sampled,␣
↪feature_names=optimal_features_list, show=False)

    plt.tight_layout()
    plt.subplots_adjust(left=0.15, right=0.85, top=0.95, bottom=0.1)
    plt.xlabel('SHAP interaction value', fontsize=12)
    plt.show()
```

```
Step 7: Interpretation with SHAP
```



[9]:

```python
# 8. Feature Importance
print("\nStep 8: Feature Importance")
with pd.option_context('display.max_rows', None):
    print(importance_df.head(len(optimal_features_list)))
```

```
Step 8: Feature Importance
                      feature  importance
12    bidirectional_stddev_ps    0.084748
33  bidirectional_rst_packets    0.083522
9               dst2src_bytes    0.071479
11      bidirectional_mean_ps    0.050756
```

```
1                         dst_port    0.050579
43             dst2src_rst_packets    0.038378
46          application_confidence    0.032124
8                    src2dst_bytes    0.031562
7            bidirectional_packets    0.030574
47               flow_bytes_per_sec    0.029129
281       application_name_Unknown    0.027939
32       bidirectional_psh_packets    0.027555
37             src2dst_rst_packets    0.027199
137          application_name_HTTP    0.027083
217           application_name_TLS    0.026072
28       bidirectional_syn_packets    0.025665
49         src2dst_packets_per_sec    0.021941
16                  dst2src_min_ps    0.020947
2                         protocol    0.019319
15               src2dst_stddev_ps    0.017827
14                 src2dst_mean_ps    0.016880
48            flow_packets_psr_sec    0.016514
34       bidirectional_fin_packets    0.016263
50          dst2src_ackets_per_sec    0.016198
36             src2dst_psh_packets    0.015190
```

[10]:

```
model_dir = os.path.join(project_root, 'models', 'rf')
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
    print(f"Created model directory: {model_dir}")

# save the model
model_file_name = os.path.join(model_dir, "model.pkl")
print(f"Saving the model to {model_file_name}")
joblib.dump(rf, model_file_name)
print("Model saved successfully.")

# save the encoder
encoder_file_name = os.path.join(model_dir, "encoder.pkl")
print(f"Saving the encoder to {encoder_file_name}")
joblib.dump(categorical_encoder, encoder_file_name)
print("Encoder saved successfully.")

# save the importance_df
importance_file_name = os.path.join(model_dir, "importance_df.pkl")
print(f"Saving the importance DataFrame to {importance_file_name}")
importance_df.to_pickle(importance_file_name)
print("Importance DataFrame saved successfully.")

# save the optimal_features_list
```

```python
optimal_features_file_name = os.path.join(model_dir, "optimal_features_list.
 ↪pkl")
print(f"Saving the optimal features list to {optimal_features_file_name}")
joblib.dump(optimal_features_list, optimal_features_file_name)
print("Optimal features list saved successfully.")
```

```
Created model directory: /Users/seu/Library/CloudStorage/Dropbox/Docs/Study/Weit
erbildung/DBS/Kurse/Projektarbeit/DS/repo/models/rf
Saving the model to /Users/seu/Library/CloudStorage/Dropbox/Docs/Study/Weiterbil
dung/DBS/Kurse/Projektarbeit/DS/repo/models/rf/model.pkl
Model saved successfully.
Saving the encoder to /Users/seu/Library/CloudStorage/Dropbox/Docs/Study/Weiterb
ildung/DBS/Kurse/Projektarbeit/DS/repo/models/rf/encoder.pkl
Encoder saved successfully.
Saving the importance DataFrame to /Users/seu/Library/CloudStorage/Dropbox/Docs/
Study/Weiterbildung/DBS/Kurse/Projektarbeit/DS/repo/models/rf/importance_df.pkl
Importance DataFrame saved successfully.
Saving the optimal features list to /Users/seu/Library/CloudStorage/Dropbox/Docs
/Study/Weiterbildung/DBS/Kurse/Projektarbeit/DS/repo/models/rf/optimal_features_
list.pkl
Optimal features list saved successfully.
```