OC-SVM

June 19, 2025

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

from sklearn.model_selection import train_test_split, ParameterGrid
from sklearn.svm import OneClassSVM
from sklearn.metrics import f1_score, precision_score, recall_score,␣
↪roc_auc_score
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)
```

```python
    from data_processing import DataProcessor
    from model_evaluation import evaluate_model

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
cache_size = 2000
scaled = True
```

```
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 {os.path.basename(cic_pkl_file_name)}")
    dataframe = pd.read_pickle(cic_pkl_file_name)
else:
    print(f"Creating dataframe from pcap files and saving to {os.path.
 ↪basename(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}")
print(f"X_test.shape: {X_test.shape}")

# Scaling the data
if scaled:
    print("Scaling the data")
    scaler = MinMaxScaler()
    print("New MinMaxScaler instance is 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}")
```

```python
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)
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 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)
Scaling the data
New MinMaxScaler instance is created
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]:
```
# 2. Load Feature Importance from Random Forest
print("\nStep 2: Load Feature Importance from Random Forest")
feature_importance_file = os.path.join(project_root, "models", "rf",␣
↪"importance_df.pkl")
if os.path.exists(feature_importance_file):
    print(f"Loading feature importance from {os.path.
↪basename(feature_importance_file)}")
    feature_importance_df = pd.read_pickle(feature_importance_file)
else:
    print(f"Feature importance file not found: {os.path.
↪basename(feature_importance_file)}")
    exit(1)
feature_importance_df = feature_importance_df.sort_values(by='importance',␣
↪ascending=False)
print("Feature importance loaded and sorted by importance")
```

```
Step 2: Load Feature Importance from Random Forest
Loading feature importance from importance_df.pkl
Feature importance loaded and sorted by importance
```

[4]:
```
# 3. Performance validation with different feature counts
print("\nStep 3: Performance evaluation with different feature counts")
if dev_mode:
    target_features_list = [5, 10]
else:
    target_features_list = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65,␣
↪70, 75, 80, 85, 90, 95, 100]
results_comparison = []

for n_features in target_features_list:
    print(f"\nValidating with {n_features} features")

    # select top n_features based on F-scores
    top_features = feature_importance_df.head(n_features)['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]
    X_val_split_selected = X_val_split[top_features]

    X_train_split_selected_normal = X_train_split_selected[y_train_split ==␣
↪NORMAL_LABEL]

    print(f"Training samples (normal): {len(X_train_split_selected_normal)}")
    print(f"Validate samples: {len(X_val_split_selected)}")

    # Reduce number of training samples
    max_samples = 10000
    X_train_sample = X_train_split_selected_normal.sample(n=max_samples,␣
↪random_state=random_state) if len(X_train_split_selected_normal) >␣
↪max_samples else X_train_split_selected_normal

    print(f"Training samples used: {len(X_train_sample)}")

    # Train OneClass SVM
    try:
        oc_svm = OneClassSVM(
            nu=0.1,
            kernel='rbf',
            gamma='scale'
        )

        oc_svm.fit(X_train_sample)

        y_pred_val = oc_svm.predict(X_val_split_selected)

        y_true_binary = (y_val_split != NORMAL_LABEL).astype(int)  # normal: 0,␣
↪anomaly: 1
        y_pred_binary = (y_pred_val != NORMAL_LABEL).astype(int)   # normal: 0,␣
↪anomaly: 1

        # Metrics calculation
        precision = precision_score(y_true_binary, y_pred_binary,␣
↪zero_division=0)
        recall = recall_score(y_true_binary, y_pred_binary, zero_division=0)
        f1 = f1_score(y_true_binary, y_pred_binary, zero_division=0)

        # Normal and anomaly detection rates
```

```
        normal_total = np.sum(y_true_binary == 0)
        anomaly_total = np.sum(y_true_binary == 1)

        if normal_total > 0:
            normal_detection_rate = np.sum((y_true_binary == 0) &␣
↪(y_pred_binary == 0)) / normal_total
        else:
            normal_detection_rate = 0.0

        if anomaly_total > 0:
            anomaly_detection_rate = np.sum((y_true_binary == 1) &␣
↪(y_pred_binary == 1)) / anomaly_total
        else:
            anomaly_detection_rate = 0.0

        # outlier fraction
        outlier_fraction = np.sum(y_pred_val == -1) / len(y_pred_val)

        results_comparison.append({
            'n_features': n_features,
            'precision': precision,
            'recall': recall,
            'f1_score': f1,
            'normal_detection_rate': normal_detection_rate,
            'anomaly_detection_rate': anomaly_detection_rate,
            'outlier_fraction': outlier_fraction,
            'training_samples': len(X_train_sample)
        })

        print(f"  F1: {f1:.4f}, Normal Det.: {normal_detection_rate:.4f}, "
              f"Anomaly Det.: {anomaly_detection_rate:.4f}, Outlier:␣
↪{outlier_fraction:.4f}")

    except Exception as e:
        print(f"  Error: {e}")
        results_comparison.append({
            'n_features': n_features,
            'precision': 0.0, 'recall': 0.0, 'f1_score': 0.0,
            'normal_detection_rate': 0.0, 'anomaly_detection_rate': 0.0,
            'outlier_fraction': 0.0, 'training_samples': 0
        })

print(f"\nCompleted evaluation for {len(results_comparison)} feature␣
↪configurations.")

# isualization and analysis
print("Results visualization")
```

```
results_df = pd.DataFrame(results_comparison)
print("\nResults Summary:")
print(results_df.round(4))

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))

# F1-Score
ax1.plot(results_df['n_features'], results_df['f1_score'], 'bo-', linewidth=2,␣
↪markersize=8)
ax1.set_xlabel('Number of Features')
ax1.set_ylabel('F1 Score')
ax1.set_title('OneClass SVM: F1 Score vs Number of Features')
ax1.grid(True, alpha=0.3)
ax1.set_ylim(0, 1)

# Detection Rates
ax2.plot(results_df['n_features'], results_df['normal_detection_rate'], 'go-',
         label='Normal Detection Rate', linewidth=2, markersize=8)
ax2.plot(results_df['n_features'], results_df['anomaly_detection_rate'], 'ro-',
         label='Anomaly Detection Rate', linewidth=2, markersize=8)
ax2.set_xlabel('Number of Features')
ax2.set_ylabel('Detection Rate')
ax2.set_title('OneClass SVM: Detection Rates vs Number of Features')
ax2.legend()
ax2.grid(True, alpha=0.3)
ax2.set_ylim(0, 1)

# Precision vs Recall
ax3.plot(results_df['n_features'], results_df['precision'], 'mo-',
         label='Precision', linewidth=2, markersize=8)
ax3.plot(results_df['n_features'], results_df['recall'], 'co-',
         label='Recall', linewidth=2, markersize=8)
ax3.set_xlabel('Number of Features')
ax3.set_ylabel('Score')
ax3.set_title('OneClass SVM: Precision/Recall vs Number of Features')
ax3.legend()
ax3.grid(True, alpha=0.3)
ax3.set_ylim(0, 1)

# Outlier Fraction
ax4.plot(results_df['n_features'], results_df['outlier_fraction'], 'ko-',
         linewidth=2, markersize=8)
ax4.set_xlabel('Number of Features')
ax4.set_ylabel('Outlier Fraction')
ax4.set_title('OneClass SVM: Outlier Fraction vs Number of Features')
ax4.grid(True, alpha=0.3)
```

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

print("Optimal feature selection")

# Calculate balanced score
results_df['balanced_score'] = (
    results_df['f1_score'] +
    results_df['normal_detection_rate'] +
    results_df['anomaly_detection_rate']
) / 3

# NaN values handling
valid_results = results_df[results_df['balanced_score'] > 0]

if len(valid_results) > 0:
    best_idx = valid_results['balanced_score'].idxmax()
    best_result = valid_results.loc[best_idx]

    optimal_n_features = int(best_result['n_features'])
    optimal_features_list = feature_importance_df.
 ↪head(optimal_n_features)['feature'].tolist()

    print(f"Optimal number of features: {optimal_n_features}")
    print(f"F1 Score: {best_result['f1_score']:.4f}")
    print(f"Normal Detection Rate: {best_result['normal_detection_rate']:.4f}")
    print(f"Anomaly Detection Rate: {best_result['anomaly_detection_rate']:.
 ↪4f}")
    print(f"Balanced Score: {best_result['balanced_score']:.4f}")
    print(f"Outlier Fraction: {best_result['outlier_fraction']:.4f}")

    print("\nSelected features for OneClass SVM:")
    for i, (_, row) in enumerate(feature_importance_df.head(optimal_n_features).
 ↪iterrows(), 1):
        print(f"{i:2d}. {row['feature']:<35} {row['importance']:.4f}")

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

print(f"\nX_train_optimal.shape: {X_train_optimal.shape}")
print(f"X_test_optimal.shape: {X_test_optimal.shape}")
```

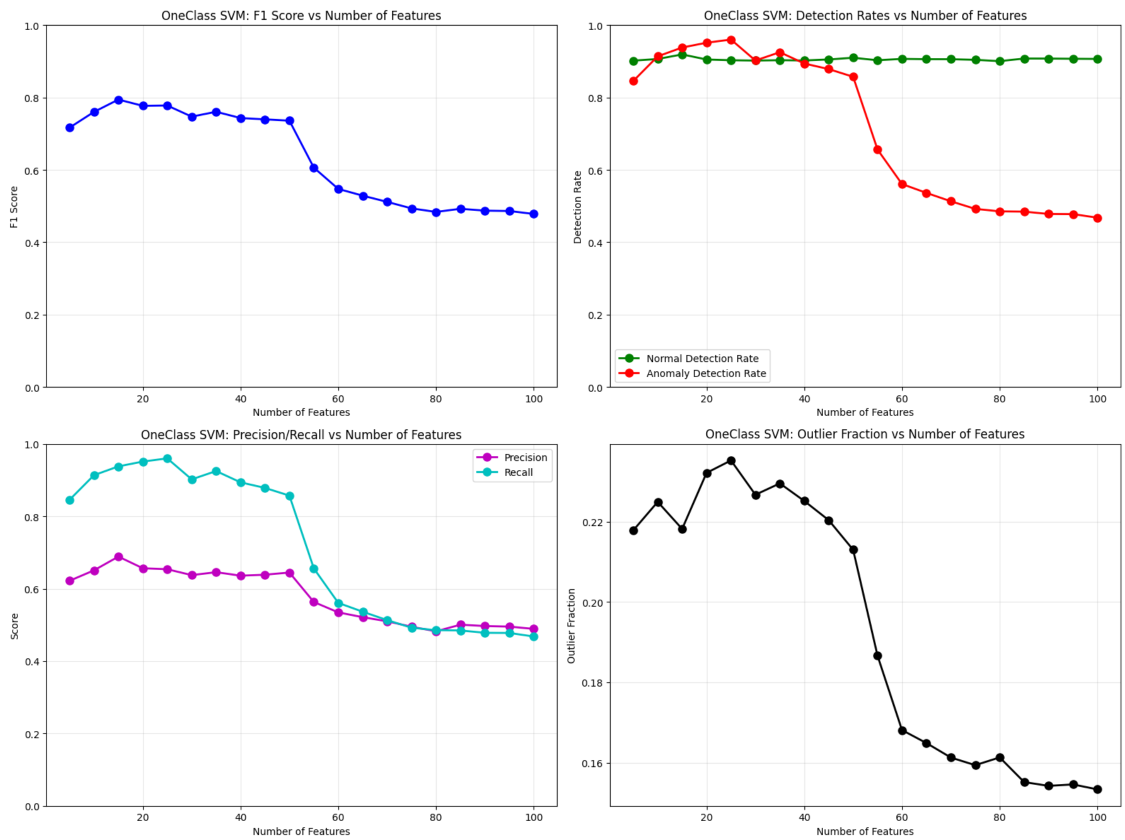
```
Step 3: Performance evaluation with different feature counts

Validating with 5 features
```

```
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7168, Normal Det.: 0.9019, Anomaly Det.: 0.8457, Outlier: 0.2179

Validating with 10 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7602, Normal Det.: 0.9065, Anomaly Det.: 0.9138, Outlier: 0.2250

Validating with 15 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7944, Normal Det.: 0.9191, Anomaly Det.: 0.9381, Outlier: 0.2182

Validating with 20 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7770, Normal Det.: 0.9050, Anomaly Det.: 0.9514, Outlier: 0.2322

Validating with 25 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7779, Normal Det.: 0.9030, Anomaly Det.: 0.9602, Outlier: 0.2353

Validating with 30 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7472, Normal Det.: 0.9021, Anomaly Det.: 0.9022, Outlier: 0.2267

Validating with 35 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7606, Normal Det.: 0.9032, Anomaly Det.: 0.9252, Outlier: 0.2296

Validating with 40 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7433, Normal Det.: 0.9024, Anomaly Det.: 0.8940, Outlier: 0.2252

Validating with 45 features
```

```
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7397, Normal Det.: 0.9052, Anomaly Det.: 0.8786, Outlier: 0.2204

Validating with 50 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.7360, Normal Det.: 0.9099, Anomaly Det.: 0.8573, Outlier: 0.2130

Validating with 55 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.6061, Normal Det.: 0.9030, Anomaly Det.: 0.6560, Outlier: 0.1866

Validating with 60 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.5474, Normal Det.: 0.9068, Anomaly Det.: 0.5609, Outlier: 0.1681

Validating with 65 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.5288, Normal Det.: 0.9060, Anomaly Det.: 0.5364, Outlier: 0.1649

Validating with 70 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.5116, Normal Det.: 0.9059, Anomaly Det.: 0.5133, Outlier: 0.1613

Validating with 75 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4936, Normal Det.: 0.9042, Anomaly Det.: 0.4923, Outlier: 0.1594

Validating with 80 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4838, Normal Det.: 0.9006, Anomaly Det.: 0.4854, Outlier: 0.1613

Validating with 85 features
```

```
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4925, Normal Det.: 0.9077, Anomaly Det.: 0.4847, Outlier: 0.1551

Validating with 90 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4874, Normal Det.: 0.9076, Anomaly Det.: 0.4782, Outlier: 0.1542

Validating with 95 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4864, Normal Det.: 0.9071, Anomaly Det.: 0.4778, Outlier: 0.1546

Validating with 100 features
Training samples (normal): 1567331
Validate samples: 466596
Training samples used: 10000
  F1: 0.4783, Normal Det.: 0.9067, Anomaly Det.: 0.4679, Outlier: 0.1533

Completed evaluation for 20 feature configurations.
Results visualization

Results Summary:
    n_features  precision  recall  f1_score  normal_detection_rate  \
0            5     0.6220  0.8457    0.7168                 0.9019
1           10     0.6508  0.9138    0.7602                 0.9065
2           15     0.6888  0.9381    0.7944                 0.9191
3           20     0.6566  0.9514    0.7770                 0.9050
4           25     0.6539  0.9602    0.7779                 0.9030
5           30     0.6376  0.9022    0.7472                 0.9021
6           35     0.6458  0.9252    0.7606                 0.9032
7           40     0.6361  0.8940    0.7433                 0.9024
8           45     0.6387  0.8786    0.7397                 0.9052
9           50     0.6448  0.8573    0.7360                 0.9099
10          55     0.5633  0.6560    0.6061                 0.9030
11          60     0.5346  0.5609    0.5474                 0.9068
12          65     0.5214  0.5364    0.5288                 0.9060
13          70     0.5100  0.5133    0.5116                 0.9059
14          75     0.4950  0.4923    0.4936                 0.9042
15          80     0.4823  0.4854    0.4838                 0.9006
16          85     0.5006  0.4847    0.4925                 0.9077
17          90     0.4969  0.4782    0.4874                 0.9076
18          95     0.4953  0.4778    0.4864                 0.9071
19         100     0.4890  0.4679    0.4783                 0.9067
```

	anomaly_detection_rate	outlier_fraction	training_samples
0	0.8457	0.2179	10000
1	0.9138	0.2250	10000
2	0.9381	0.2182	10000
3	0.9514	0.2322	10000
4	0.9602	0.2353	10000
5	0.9022	0.2267	10000
6	0.9252	0.2296	10000
7	0.8940	0.2252	10000
8	0.8786	0.2204	10000
9	0.8573	0.2130	10000
10	0.6560	0.1866	10000
11	0.5609	0.1681	10000
12	0.5364	0.1649	10000
13	0.5133	0.1613	10000
14	0.4923	0.1594	10000
15	0.4854	0.1613	10000
16	0.4847	0.1551	10000
17	0.4782	0.1542	10000
18	0.4778	0.1546	10000
19	0.4679	0.1533	10000



```
Optimal feature selection
Optimal number of features: 15
F1 Score: 0.7944
Normal Detection Rate: 0.9191
Anomaly Detection Rate: 0.9381
Balanced Score: 0.8839
Outlier Fraction: 0.2182

Selected features for OneClass SVM:
 1. bidirectional_stddev_ps           0.0847
 2. bidirectional_rst_packets         0.0835
 3. dst2src_bytes                     0.0715
 4. bidirectional_mean_ps             0.0508
 5. dst_port                          0.0506
 6. dst2src_rst_packets               0.0384
 7. application_confidence            0.0321
 8. src2dst_bytes                     0.0316
 9. bidirectional_packets             0.0306
10. flow_bytes_per_sec                0.0291
11. application_name_Unknown          0.0279
12. bidirectional_psh_packets         0.0276
13. src2dst_rst_packets               0.0272
14. application_name_HTTP             0.0271
15. application_name_TLS              0.0261

X_train_optimal.shape: (2332977, 15)
X_test_optimal.shape: (583245, 15)
```

[5]:

```python
# 4. Hyperparameter tuning for OneClassSVM
print("\nStep 4: Hyperparameter tuning for OneClass SVM")
# Hyperparameter grid for OneClassSVM
if dev_mode:
    param_grid = {
        'kernel': ['rbf'],
        'nu': [0.05],
        'gamma': ['scale']
    }
else:
    param_grid = {
        'kernel': ['linear', 'rbf'],
        'nu': [0.01, 0.05, 0.1, 0.15, 0.2],
        'gamma': ['scale', 0.001, 0.01, 0.1, 1]
    }
best_score = -np.inf
best_params = None
best_model = None
```

```
total_combinations = sum(1 for _ in ParameterGrid(param_grid))

with tqdm(total=total_combinations, desc="Hyperparameter Tuning", file=sys.
 ↪stdout) as pbar:
    for i, params in enumerate(ParameterGrid(param_grid), 1):
        model = OneClassSVM(**params)
        # split X_train_optimal
        X_train_split, X_val_split, y_train_split, y_val_split =␣
 ↪train_test_split(
            X_train_optimal, y_train,
            test_size=test_size,
            random_state=random_state,
            stratify=y_train
        )

        sample_size = 10000
        X_train_split_benign = X_train_split[y_train_split == NORMAL_LABEL]
        X_train_split_benign = X_train_split_benign.sample(n=sample_size,␣
 ↪random_state=random_state) if len(X_train_split_benign) > sample_size else␣
 ↪X_train_split_benign

        model.fit(X_train_split_benign)
        y_pred = model.predict(X_val_split)
        score = f1_score(y_val_split, y_pred, pos_label=-1)

        if score > best_score:
            best_score = score
            best_params = params
            best_model = model

        # update the progress bar description
        pbar.set_description(f"F1: {score:.4f} | Best: {best_score:.4f}")
        # write the current params and score to the progress bar
        pbar.write(f"[{i}/{total_combinations}] Params: {params}, F1 Score:␣
 ↪{score:.4f}")
        pbar.update(1)

print("Best params:", best_params)
print("Best F1 score:", best_score)
```

```
Step 4: Hyperparameter tuning for OneClass SVM
[1/50] Params: {'gamma': 'scale', 'kernel': 'linear', 'nu': 0.01}, F1 Score:
0.2463
[2/50] Params: {'gamma': 'scale', 'kernel': 'linear', 'nu': 0.05}, F1 Score:
0.4258
```

```
[3/50] Params: {'gamma': 'scale', 'kernel': 'linear', 'nu': 0.1}, F1 Score:
0.3617
[4/50] Params: {'gamma': 'scale', 'kernel': 'linear', 'nu': 0.15}, F1 Score:
0.2987
[5/50] Params: {'gamma': 'scale', 'kernel': 'linear', 'nu': 0.2}, F1 Score:
0.2943
[6/50] Params: {'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.01}, F1 Score: 0.3490
[7/50] Params: {'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.05}, F1 Score: 0.8088
[8/50] Params: {'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.1}, F1 Score: 0.7944
[9/50] Params: {'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.15}, F1 Score: 0.7269
[10/50] Params: {'gamma': 'scale', 'kernel': 'rbf', 'nu': 0.2}, F1 Score: 0.5989
[11/50] Params: {'gamma': 0.001, 'kernel': 'linear', 'nu': 0.01}, F1 Score:
0.2463
[12/50] Params: {'gamma': 0.001, 'kernel': 'linear', 'nu': 0.05}, F1 Score:
0.4258
[13/50] Params: {'gamma': 0.001, 'kernel': 'linear', 'nu': 0.1}, F1 Score:
0.3617
[14/50] Params: {'gamma': 0.001, 'kernel': 'linear', 'nu': 0.15}, F1 Score:
0.2987
[15/50] Params: {'gamma': 0.001, 'kernel': 'linear', 'nu': 0.2}, F1 Score:
0.2943
[16/50] Params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.01}, F1 Score: 0.6748
[17/50] Params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.05}, F1 Score: 0.8318
[18/50] Params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.1}, F1 Score: 0.5960
[19/50] Params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.15}, F1 Score: 0.7009
[20/50] Params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.2}, F1 Score: 0.5431
[21/50] Params: {'gamma': 0.01, 'kernel': 'linear', 'nu': 0.01}, F1 Score:
0.2463
[22/50] Params: {'gamma': 0.01, 'kernel': 'linear', 'nu': 0.05}, F1 Score:
0.4258
[23/50] Params: {'gamma': 0.01, 'kernel': 'linear', 'nu': 0.1}, F1 Score: 0.3617
[24/50] Params: {'gamma': 0.01, 'kernel': 'linear', 'nu': 0.15}, F1 Score:
0.2987
[25/50] Params: {'gamma': 0.01, 'kernel': 'linear', 'nu': 0.2}, F1 Score: 0.2943
[26/50] Params: {'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.01}, F1 Score: 0.6716
[27/50] Params: {'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.05}, F1 Score: 0.8017
[28/50] Params: {'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.1}, F1 Score: 0.6626
[29/50] Params: {'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.15}, F1 Score: 0.6451
[30/50] Params: {'gamma': 0.01, 'kernel': 'rbf', 'nu': 0.2}, F1 Score: 0.6172
[31/50] Params: {'gamma': 0.1, 'kernel': 'linear', 'nu': 0.01}, F1 Score: 0.2463
[32/50] Params: {'gamma': 0.1, 'kernel': 'linear', 'nu': 0.05}, F1 Score: 0.4258
[33/50] Params: {'gamma': 0.1, 'kernel': 'linear', 'nu': 0.1}, F1 Score: 0.3617
[34/50] Params: {'gamma': 0.1, 'kernel': 'linear', 'nu': 0.15}, F1 Score: 0.2987
[35/50] Params: {'gamma': 0.1, 'kernel': 'linear', 'nu': 0.2}, F1 Score: 0.2943
[36/50] Params: {'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.01}, F1 Score: 0.6158
[37/50] Params: {'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.05}, F1 Score: 0.8009
[38/50] Params: {'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.1}, F1 Score: 0.7754
[39/50] Params: {'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.15}, F1 Score: 0.6856
```

```
[40/50] Params: {'gamma': 0.1, 'kernel': 'rbf', 'nu': 0.2}, F1 Score: 0.6521
[41/50] Params: {'gamma': 1, 'kernel': 'linear', 'nu': 0.01}, F1 Score: 0.2463
[42/50] Params: {'gamma': 1, 'kernel': 'linear', 'nu': 0.05}, F1 Score: 0.4258
[43/50] Params: {'gamma': 1, 'kernel': 'linear', 'nu': 0.1}, F1 Score: 0.3617
[44/50] Params: {'gamma': 1, 'kernel': 'linear', 'nu': 0.15}, F1 Score: 0.2987
[45/50] Params: {'gamma': 1, 'kernel': 'linear', 'nu': 0.2}, F1 Score: 0.2943
[46/50] Params: {'gamma': 1, 'kernel': 'rbf', 'nu': 0.01}, F1 Score: 0.3445
[47/50] Params: {'gamma': 1, 'kernel': 'rbf', 'nu': 0.05}, F1 Score: 0.8140
[48/50] Params: {'gamma': 1, 'kernel': 'rbf', 'nu': 0.1}, F1 Score: 0.7831
[49/50] Params: {'gamma': 1, 'kernel': 'rbf', 'nu': 0.15}, F1 Score: 0.7339
[50/50] Params: {'gamma': 1, 'kernel': 'rbf', 'nu': 0.2}, F1 Score: 0.5633
F1: 0.5633 | Best: 0.8318: 100%|      | 50/50 [06:13<00:00,  7.47s/it]
Best params: {'gamma': 0.001, 'kernel': 'rbf', 'nu': 0.05}
Best F1 score: 0.8318200847768067
```

[6]:

```
# 5. Find the best optimal sample size
print("\nStep 5: Find the best optimal sample size")

nu = float(best_params['nu'])
kernel = best_params['kernel']
gamma = best_params['gamma']

if dev_mode:
    sample_sizes = [100, 500, 1000, 5000]
else:
    sample_sizes = [100, 1_000, 5_000, 10_000, 20_000, 50_000, 100_000, 200_000]

results = []

with tqdm(total=len(sample_sizes), desc="Sample Size Tuning", file=sys.stdout)␣
↪as pbar:
    for i, n_samples in enumerate(sample_sizes, 1):
        # split X_train_optimal
        X_train_split, X_val_split, y_train_split, y_val_split =␣
↪train_test_split(
            X_train_optimal, y_train,
            test_size=test_size,
            random_state=random_state,
            stratify=y_train
        )

        X_train_split_normal = X_train_split[y_train_split == NORMAL_LABEL]
        # randomly take samples of n_samples
        X_train_split_normal_sampled = X_train_split_normal.
↪sample(n=min(n_samples, len(X_train_split_normal)),␣
↪random_state=random_state)
```

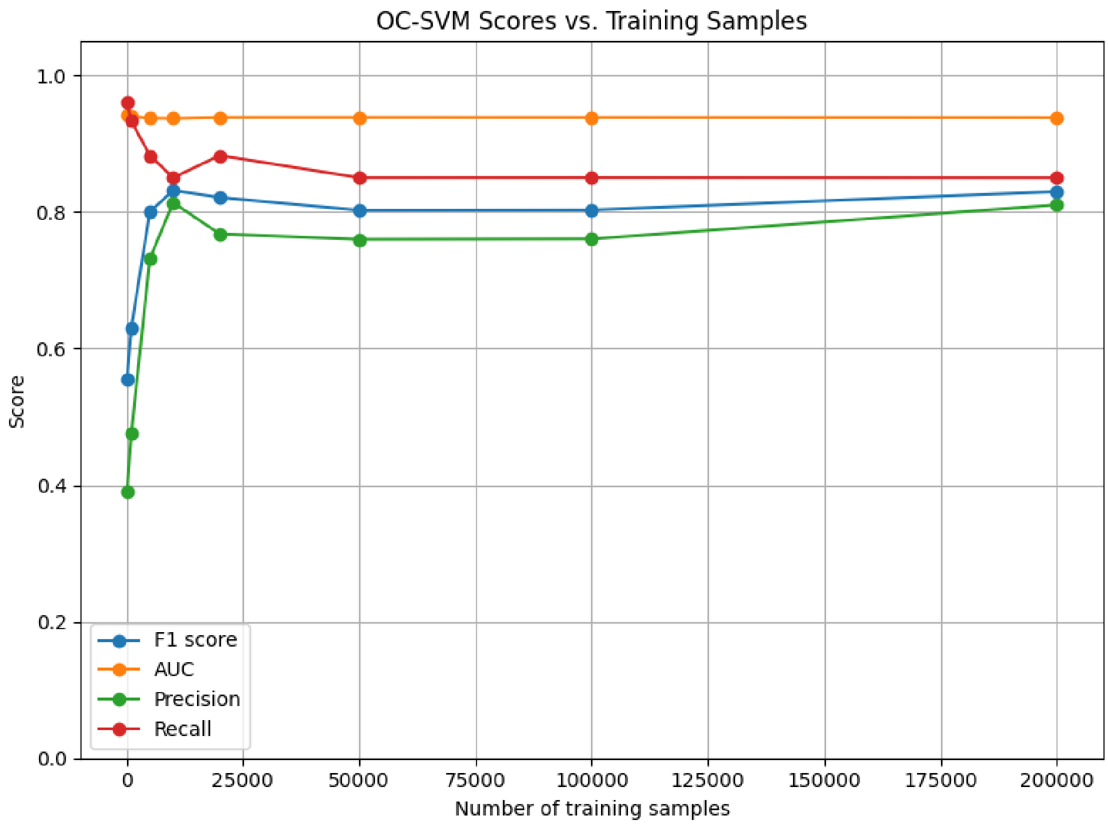
```python
        # OneClassSVM instance
        ocsvm = OneClassSVM(kernel=kernel, gamma=gamma, nu=nu)

        # train
        t0 = time.time()
        ocsvm.fit(X_train_split_normal_sampled)
        train_time = time.time() - t0

        # test
        t0 = time.time()
        y_pred = ocsvm.predict(X_val_split)
        test_time = time.time() - t0

        # OneClassSVM predict: normal=1, anomaly=-1
        # metric for anomaly
        y_val_split_bin = (y_val_split != NORMAL_LABEL).astype(int) # convert to binary: normal=0, anomaly=1
        y_pred_label = (y_pred != NORMAL_LABEL).astype(int) # convert to binary: normal=0, anomaly=1

        # calculate metrics
        f1 = f1_score(y_val_split_bin, y_pred_label)
        y_score = ocsvm.decision_function(X_val_split)
        auc = roc_auc_score(y_val_split_bin, -y_score)
        precision = precision_score(y_val_split_bin, y_pred_label, zero_division=0)
        recall = recall_score(y_val_split_bin, y_pred_label, zero_division=0)
        # Calculate false positive rate (FPR) and false negative rate (FNR)
        # FPR: proportion of normal samples incorrectly classified as anomaly
        # FNR: proportion of anomaly samples incorrectly classified as normal
        fp = np.sum((y_val_split_bin == 0) & (y_pred_label == 1))
        tn = np.sum((y_val_split_bin == 0) & (y_pred_label == 0))
        fn = np.sum((y_val_split_bin == 1) & (y_pred_label == 0))
        tp = np.sum((y_val_split_bin == 1) & (y_pred_label == 1))
        fpr = fp / (fp + tn) if (fp + tn) > 0 else 0.0
        fnr = fn / (fn + tp) if (fn + tp) > 0 else 0.0
        results.append([n_samples, train_time, test_time, f1, auc, precision, recall])

        # update the progress bar description
        pbar.set_description(f"n_samles: {n_samples}")
        # write the current params and score to the progress bar
        pbar.write(f"[{i}/{len(sample_sizes)}] n_samples: {n_samples}")
        pbar.write(f"  Training (sec): {train_time:.1f}")
        pbar.write(f"  Predict (sec): {test_time:.1f}")
        pbar.write(f"  Precision: {precision:.3f}")
        pbar.write(f"  Recall: {recall:.3f}")
```

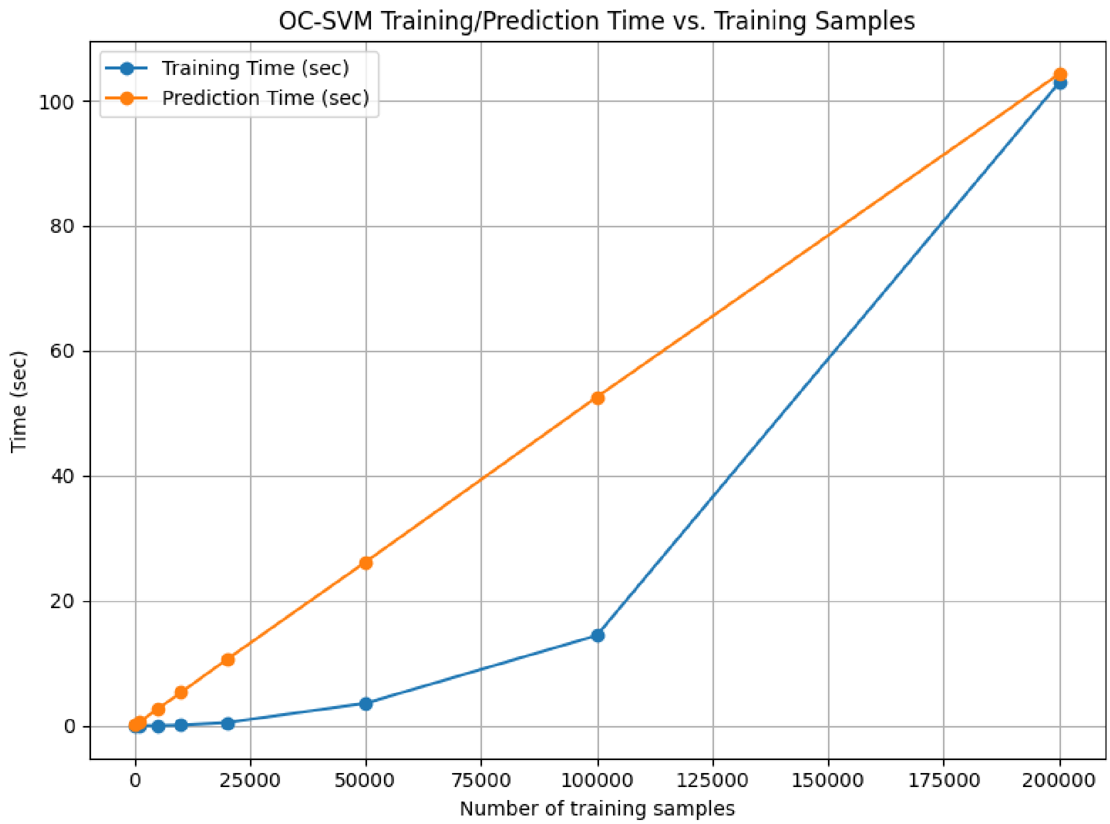
```
        pbar.write(f"  False Positive Rate: {fpr:.4f}")
        pbar.write(f"  False Negative Rate: {fnr:.4f}")
        pbar.write(f"  F1: {f1:.2f}")
        pbar.write(f"  AUC: {auc:.3f}")
        pbar.update(1)

# DataFrame
df_results = pd.DataFrame(
    results,
    columns=["Samples", "Training (sec)", "Prediction (sec)", "F1", "AUC",␣
 ↪"Precision", "Recall"]
)
print(df_results)

# plot results
plt.figure(figsize=(8, 6))
plt.plot(df_results["Samples"], df_results["F1"], marker='o', label="F1 score")
plt.plot(df_results["Samples"], df_results["AUC"], marker='o', label="AUC")
plt.plot(df_results["Samples"], df_results["Precision"], marker='o',␣
 ↪label="Precision")
plt.plot(df_results["Samples"], df_results["Recall"], marker='o',␣
 ↪label="Recall")
plt.xlabel("Number of training samples")
plt.ylabel("Score")
plt.ylim(0, 1.05)
plt.title("OC-SVM Scores vs. Training Samples")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show(block=False)

plt.figure(figsize=(8, 6))
plt.plot(df_results["Samples"], df_results["Training (sec)"], marker='o',␣
 ↪label="Training Time (sec)")
plt.plot(df_results["Samples"], df_results["Prediction (sec)"], marker='o',␣
 ↪label="Prediction Time (sec)")
plt.xlabel("Number of training samples")
plt.ylabel("Time (sec)")
plt.title("OC-SVM Training/Prediction Time vs. Training Samples")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show(block=False)
```

```
Step 5: Find the best optimal sample size
[1/8] n_samples: 100
```

```
  Training (sec): 0.0
  Predict (sec): 0.1
  Precision: 0.391
  Recall: 0.960
  False Positive Rate: 0.2850
  False Negative Rate: 0.0395
  F1: 0.56
  AUC: 0.942
[2/8] n_samples: 1000
  Training (sec): 0.0
  Predict (sec): 0.6
  Precision: 0.477
  Recall: 0.933
  False Positive Rate: 0.1954
  False Negative Rate: 0.0669
  F1: 0.63
  AUC: 0.940
[3/8] n_samples: 5000
  Training (sec): 0.0
  Predict (sec): 2.7
  Precision: 0.732
  Recall: 0.883
  False Positive Rate: 0.0616
  False Negative Rate: 0.1170
  F1: 0.80
  AUC: 0.938
[4/8] n_samples: 10000
  Training (sec): 0.1
  Predict (sec): 5.3
  Precision: 0.814
  Recall: 0.850
  False Positive Rate: 0.0371
  False Negative Rate: 0.1496
  F1: 0.83
  AUC: 0.937
[5/8] n_samples: 20000
  Training (sec): 0.5
  Predict (sec): 10.6
  Precision: 0.768
  Recall: 0.883
  False Positive Rate: 0.0509
  False Negative Rate: 0.1171
  F1: 0.82
  AUC: 0.939
[6/8] n_samples: 50000
  Training (sec): 3.6
  Predict (sec): 26.2
  Precision: 0.760
```

```
  Recall: 0.850
  False Positive Rate: 0.0512
  False Negative Rate: 0.1495
  F1: 0.80
  AUC: 0.939
[7/8] n_samples: 100000
  Training (sec): 14.4
  Predict (sec): 52.6
  Precision: 0.761
  Recall: 0.851
  False Positive Rate: 0.0510
  False Negative Rate: 0.1494
  F1: 0.80
  AUC: 0.939
[8/8] n_samples: 200000
  Training (sec): 102.9
  Predict (sec): 104.3
  Precision: 0.811
  Recall: 0.850
  False Positive Rate: 0.0379
  False Negative Rate: 0.1496
  F1: 0.83
  AUC: 0.939
n_samles: 200000: 100%|      | 8/8 [08:50<00:00, 66.26s/it]
   Samples  Training (sec)  Prediction (sec)        F1       AUC  Precision  \
0      100        0.000580          0.085931  0.556162  0.942042   0.391399
1     1000        0.001595          0.554827  0.631088  0.940229   0.476767
2     5000        0.025738          2.680942  0.800536  0.937704   0.732158
3    10000        0.095851          5.311382  0.831820  0.937261   0.813993
4    20000        0.498666         10.625688  0.821475  0.938792   0.768035
5    50000        3.596228         26.214029  0.802775  0.938817   0.760122
6   100000       14.439900         52.582480  0.803140  0.938713   0.760712
7   200000      102.871995        104.266991  0.830032  0.938514   0.810587

     Recall
0  0.960488
1  0.933121
2  0.883002
3  0.850445
4  0.882908
5  0.850499
6  0.850579
7  0.850432
```

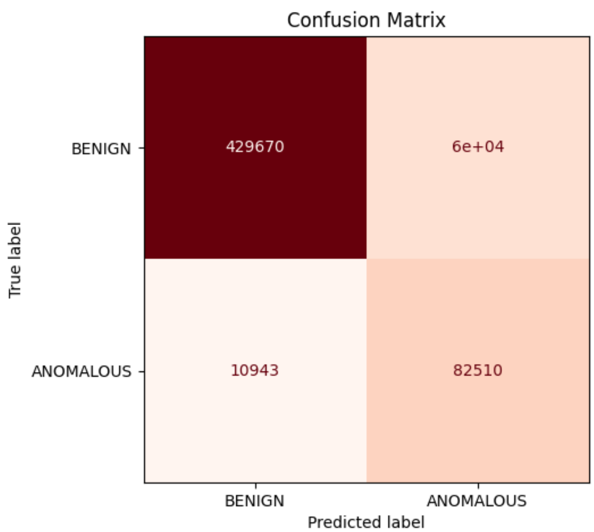

OC-SVM Scores vs. Training Samples
Score
Number of training samples
0.0
0.2
0.4
0.6
0.8
1.0
0
25000
50000
75000
100000
125000
150000
175000
200000
F1 score
AUC
Precision
Recall

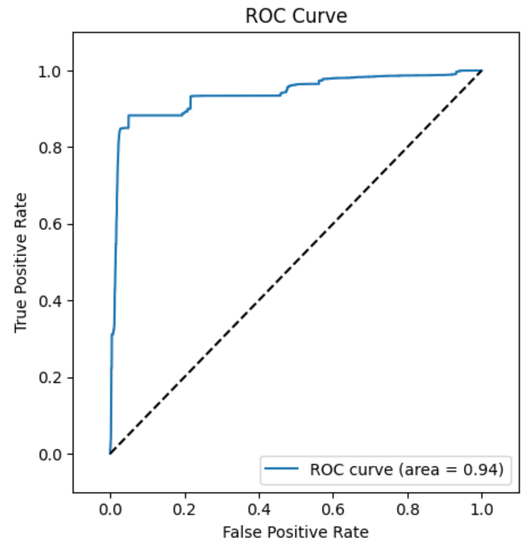


[7]:
```
# 6. Final One-Class SVM model training and evaluation
print("\nStep 6: Final One-Class SVM model training and evaluation")

sample_size = 10_000
print(f"Using sample size: {sample_size}")

nu = float(best_params['nu'])
kernel = best_params['kernel']
gamma = best_params['gamma']
print(f"Using nu={nu}, kernel={kernel}, gamma={gamma} for final One-Class SVM␣
↪model.")

final_ocsvm = OneClassSVM(kernel=kernel, nu=nu, gamma=gamma,␣
↪cache_size=cache_size)
print("Training One-Class SVM model on BENIGN data...")
X_train_benign = X_train_optimal[y_train == NORMAL_LABEL]

X_train_benign = X_train_benign.sample(n=sample_size,␣
↪random_state=random_state) if len(X_train_benign) > sample_size else␣
↪X_train_benign
```

```python
print(f"Sampled X_train_benign shape: {X_train_benign.shape}")
final_ocsvm.fit(X_train_benign)
print("One-Class SVM training complete.")

evaluate_model(final_ocsvm, X_test_optimal, y_test, with_numpy=False)
```

```
Step 6: Final One-Class SVM model training and evaluation
Using sample size: 10000
Using nu=0.05, kernel=rbf, gamma=0.001 for final One-Class SVM model.
Training One-Class SVM model on BENIGN data…
Sampled X_train_benign shape: (10000, 15)
One-Class SVM training complete.
using decision_function 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       0.98      0.88      0.92    489792
   ANOMALOUS       0.58      0.88      0.70     93453

    accuracy                           0.88    583245
   macro avg       0.78      0.88      0.81    583245
weighted avg       0.91      0.88      0.89    583245

[[429670  60122]
 [ 10943  82510]]
Precision: 0.578
Recall: 0.883
False Positive Rate: 0.123
False Negative Rate: 0.117
F1-Score: 0.699
Area under the curve: 0.938
```





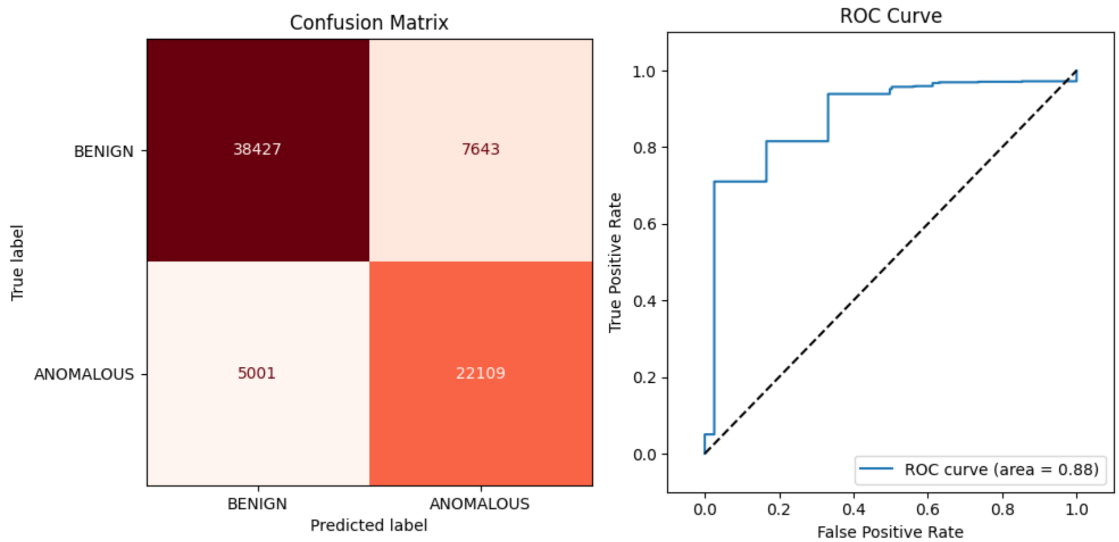
[8]:

```
# 7. Evaluate with tcpdump data
print("\nStep 7: Evaluate with tcpdump data")
if os.path.exists(tcpdump_pkl_file_name):
    print(f"Loading dataframe from {os.path.basename(tcpdump_pkl_file_name)}")
    tcpdump_dataframe = pd.read_pickle(tcpdump_pkl_file_name)
else:
    print(f"Creating dataframe from pcap files and saving to {os.path.
 ↪basename(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
        )
# 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)
```

```python
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(final_ocsvm, X_tcpdump_optimal, y_tcpdump, with_numpy=False)
```

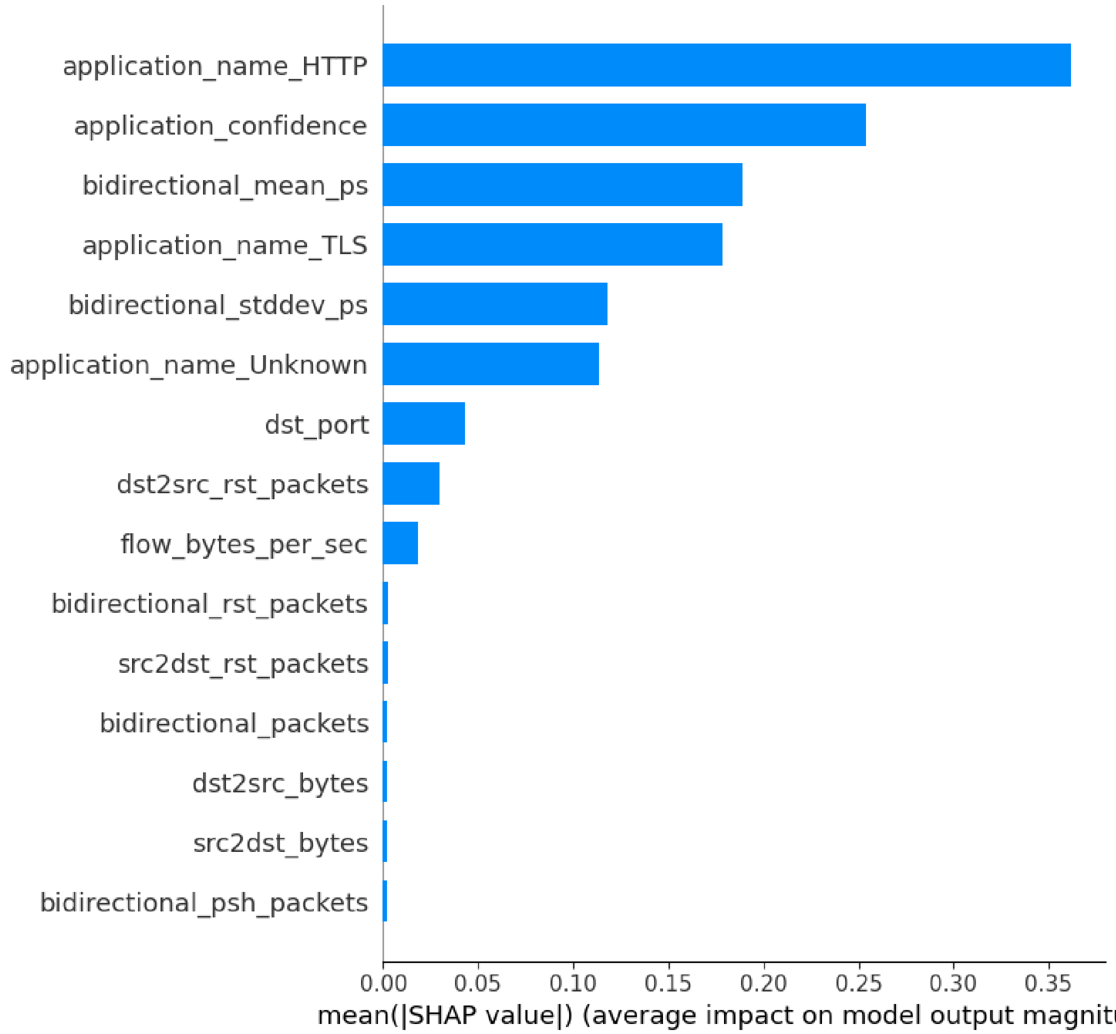
```
Step 7: Evaluate with tcpdump data
Loading dataframe from tcpdump_dataframe.pkl
Adding new features to the dataframe…
flow rate (bytes/sec and packets/sec)
packet rate (bytes/sec and packets/sec)
Down/Up Ratio
Dropping object columns and handle encoding categorical data…
Retaining categorical features 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)
Use MinMaxScaler instance: MinMaxScaler()
Feature selection:
X_tcpdump_optimal.shape: (73180, 15)
y_tcpdump.shape: (73180,)
using decision_function 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       0.88      0.83      0.86     46070
   ANOMALOUS       0.74      0.82      0.78     27110

    accuracy                           0.83     73180
   macro avg       0.81      0.82      0.82     73180
weighted avg       0.83      0.83      0.83     73180

[[38427  7643]
 [ 5001 22109]]
Precision: 0.743
Recall: 0.816
False Positive Rate: 0.166
False Negative Rate: 0.184
F1-Score: 0.778
Area under the curve: 0.878
```

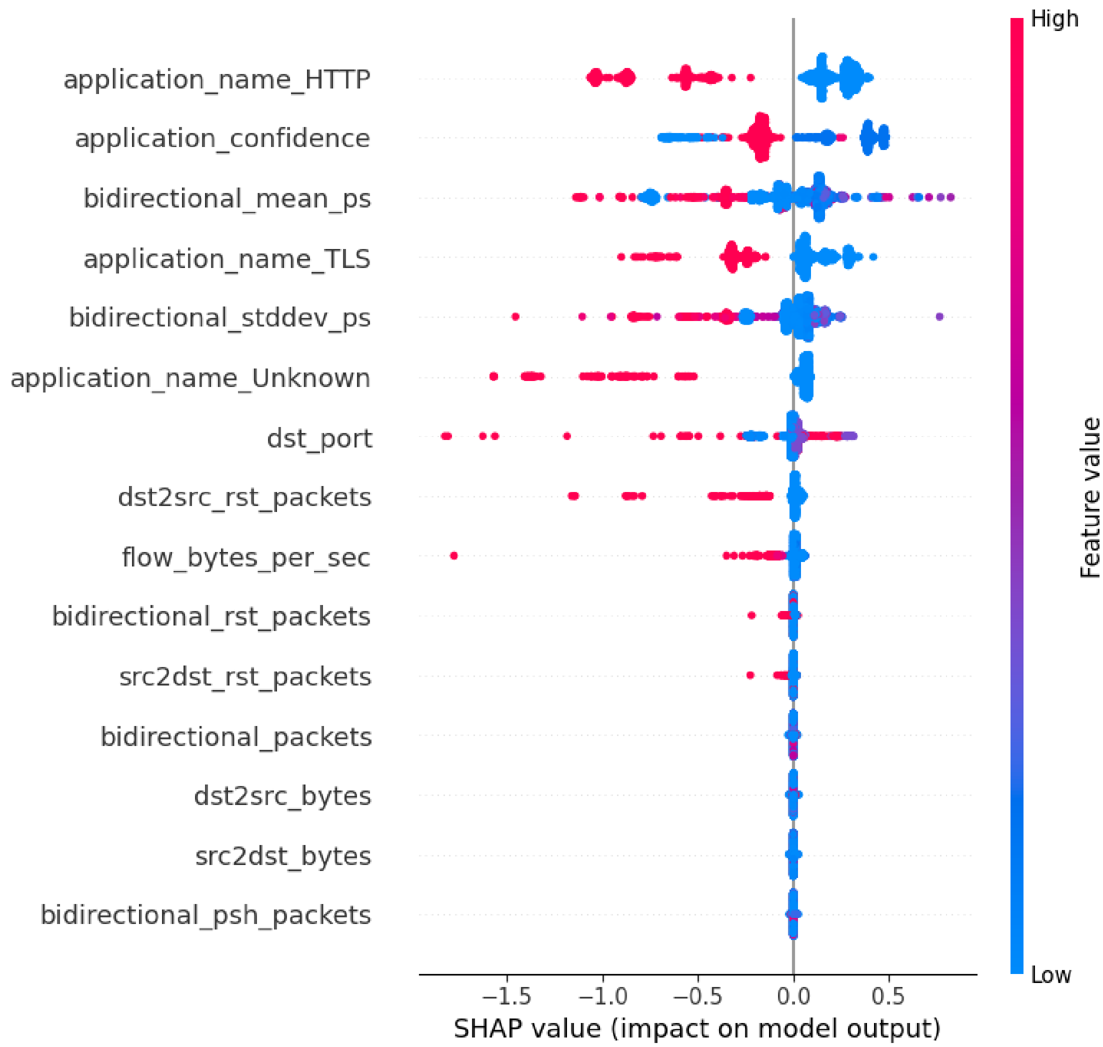


[9]:
```
# 8. Interpretation with SHAP
if shap_enabled:
    print("\nStep 8: Interpretation with SHAP")
    background_data_summary = shap.kmeans(X_train_optimal, 100)
    explainer = shap.KernelExplainer(final_ocsvm.predict,␣
↪background_data_summary)
    X_test_optimal_sampled = X_test_optimal.sample(n=2000,␣
↪random_state=random_state)
    shap_values = explainer.shap_values(X_test_optimal_sampled)
    shap.summary_plot(shap_values, X_test_optimal_sampled,␣
↪feature_names=optimal_features_list, plot_type="bar", max_display=30)
    shap.summary_plot(shap_values, X_test_optimal_sampled,␣
↪feature_names=optimal_features_list, max_display=30)
```

```
Step 8: Interpretation with SHAP

  0%|          | 0/2000 [00:00<?, ?it/s]
```



application_name_HTTP
application_confidence
bidirectional_mean_ps
application_name_TLS
bidirectional_stddev_ps
application_name_Unknown
dst_port
dst2src_rst_packets
flow_bytes_per_sec
bidirectional_rst_packets
src2dst_rst_packets
bidirectional_packets
dst2src_bytes
src2dst_bytes
bidirectional_psh_packets
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
mean(|SHAP value|) (average impact on model output magnitu



[10]:
```
model_dir = os.path.join(project_root, 'models', 'ocsvm')
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
    print("Created model directory: models/ocsvm")

# save the model
model_file_name = os.path.join(model_dir, "model.pkl")
print(f"Saving the model to {os.path.basename(model_file_name)}")
joblib.dump(final_ocsvm, 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 {os.path.basename(encoder_file_name)}")
```

```python
joblib.dump(categorical_encoder, encoder_file_name)
print("Encoder saved successfully.")

# save the scaler
scaler_file_name = os.path.join(model_dir, "scaler.pkl")
print(f"Saving the scaler to {os.path.basename(scaler_file_name)}")
joblib.dump(scaler, scaler_file_name)
print("Scaler saved successfully.")

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

```
Created model directory: models/ocsvm
Saving the model to model.pkl
Model saved successfully.
Saving the encoder to encoder.pkl
Encoder saved successfully.
Saving the scaler to scaler.pkl
Scaler saved successfully.
Saving the optimal features list to optimal_features_list.pkl
Optimal features list saved successfully.
```