ABUAD Journal of Engineering Research and Development (AJERD)
ISSN (online): 2645-2685; ISSN (print): 2756-6811

Volume 8, Issue 2

Supplementary Data

Edit View Run Kernel Settings Help

Code

```
# ## Explainable Intrusion Detection System (X-IDS)
# ### Integrating ML with SHAP/LIME for Transparent Cybersecurity

import pandas as pd
import numpy as np
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import mutual_info_classif
import shap
import lime
import lime.lime_tabular
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
import time
import json

# Configuration
pd.set_option('display.max_columns', None)
np.random.seed(42)
tf.random.set_seed(42)
```

Edit View Run Kernel Settings Help

Code

```python

# --- Data Preparation ---
def load_and_preprocess_data(dataset_name='NSL-KDD'):
    """Load and preprocess dataset with feature selection"""
    if dataset_name == 'NSL-KDD':
        # Load NSL-KDD dataset (replace with actual path)
        df = pd.read_csv('KDDTrain+.txt', header=None)
        cols = [
            'duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes',
            'land', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins',
            'logged_in', 'num_compromised', 'root_shell', 'su_attempted',
            'num_root', 'num_file_creations', 'num_shells', 'num_access_files',
            'num_outbound_cmds', 'is_host_login', 'is_guest_login', 'count',
            'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate',
            'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
            'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count',
            'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
            'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
            'dst_host_serror_rate', 'dst_host_srv_serror_rate',
            'dst_host_rerror_rate', 'dst_host_srv_rerror_rate', 'label'
        ]
        df.columns = cols

        # Convert labels to binary (normal = 0, attack = 1)
        df['label'] = df['label'].apply(lambda x: 0 if x == 'normal' else 1)

        # Select relevant features based on mutual information
        X = df.drop('label', axis=1)
        y = df['label']
        cat_cols = ['protocol_type', 'service', 'flag']
        num_cols = [col for col in X.columns if col not in cat_cols]

```

```
    else:  # CICIDS2017
        # Load sample of CICIDS2017 (replace with actual path)
        df = pd.read_csv('CICIDS2017_sample.csv')
        df['label'] = df['Label'].apply(lambda x: 0 if 'BENIGN' in x else 1)
        df = df.drop('Label', axis=1)

        # Feature selection
        X = df.drop('label', axis=1)
        y = df['label']
        cat_cols = []
        num_cols = X.columns.tolist()

    # Preprocessing pipeline
    # One-hot encoding for categorical features
    if cat_cols:
        ohe = OneHotEncoder(handle_unknown='ignore', sparse=False)
        X_cat = pd.DataFrame(ohe.fit_transform(X[cat_cols]))
        X_cat.columns = ohe.get_feature_names_out(cat_cols)
        X = pd.concat([X.drop(cat_cols, axis=1), X_cat], axis=1)

    # Normalization
    scaler = MinMaxScaler()
    X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

    # Feature selection using mutual information
    mi_scores = mutual_info_classif(X, y)
    mi_df = pd.DataFrame({'Feature': X.columns, 'MI_Score': mi_scores})
    top_features = mi_df.sort_values('MI_Score', ascending=False).head(20)['Feature'].tolist()
    X = X[top_features]

    return X, y, top_features
```

Edit View Run Kernel Settings Help

Code

```python

# --- Model Training ---
def train_models(X_train, y_train):
    """Train and tune multiple ML models"""
    models = {}

    # Random Forest
    rf_params = {'n_estimators': [100, 300, 500], 'max_depth': [5, 10, 15]}
    rf = GridSearchCV(RandomForestClassifier(), rf_params, cv=3, scoring='f1')
    rf.fit(X_train, y_train)
    models['Random Forest'] = rf.best_estimator_

    # SVM
    svm_params = {'C': [0.1, 1, 10], 'gamma': [0.01, 0.1, 1]}
    svm = GridSearchCV(SVC(probability=True), svm_params, cv=3, scoring='f1')
    svm.fit(X_train, y_train)
    models['SVM'] = svm.best_estimator_

    # Decision Tree
    dt_params = {'max_depth': [5, 10, 15], 'criterion': ['gini', 'entropy']}
    dt = GridSearchCV(DecisionTreeClassifier(), dt_params, cv=3, scoring='f1')
    dt.fit(X_train, y_train)
    models['Decision Tree'] = dt.best_estimator_

    # Logistic Regression
    lr = LogisticRegression(max_iter=1000, C=0.1)
    lr.fit(X_train, y_train)
    models['Logistic Regression'] = lr

    # Deep Neural Network
    model = Sequential([
        Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=20, batch_size=64, validation_split=0.1, verbose=0)
    models['DNN'] = model

    return models

```

jupyter XAI Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code

```python

# --- XAI Functions ---
def generate_shap_explanations(model, X_sample, model_type='tree'):
    """Generate SHAP explanations for a model"""
    if model_type == 'tree':
        explainer = shap.TreeExplainer(model)
    elif model_type == 'dnn':
        explainer = shap.DeepExplainer(model, X_sample.values)
    else:
        explainer = shap.KernelExplainer(model.predict_proba, X_sample)

    shap_values = explainer.shap_values(X_sample)
    return explainer, shap_values

def generate_lime_explanation(model, X_train, instance, feature_names):
    """Generate LIME explanation for a single instance"""
    explainer = lime.lime_tabular.LimeTabularExplainer(
        training_data=X_train.values,
        feature_names=feature_names,
        class_names=['Normal', 'Attack'],
        mode='classification'
    )
    exp = explainer.explain_instance(
        data_row=instance.values,
        predict_fn=model.predict_proba,
        num_features=10
    )
    return exp
```

jupyter XAI Last Checkpoint: 1 hour ago

e Edit View Run Kernel Settings Help

Code JupyterLab

```python

# --- Dashboard Interface ---
def main():
    st.set_page_config(layout="wide")
    st.title("🔍 Explainable Intrusion Detection System (X-IDS)")
    st.markdown("""
    Machine Learning-powered IDS with SHAP & LIME explanations for transparent threat detection
    """)

    # Sidebar configuration
    st.sidebar.header("Configuration")
    dataset = st.sidebar.selectbox("Dataset", ["NSL-KDD", "CICIDS2017"])
    model_type = st.sidebar.selectbox("Model", ["Random Forest", "SVM", "Decision Tree", "Logistic Regression", "DNN"])

    # Load data
    with st.spinner("Loading and preprocessing data..."):
        X, y, feature_names = load_and_preprocess_data(dataset)
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

        # Handle imbalance with SMOTE
        sm = SMOTE(random_state=42)
        X_train, y_train = sm.fit_resample(X_train, y_train)

    # Train models
    with st.spinner(f"Training {model_type} model..."):
        if 'models' not in st.session_state:
            st.session_state.models = train_models(X_train, y_train)
        model = st.session_state.models[model_type]

        # Make predictions
        if model_type == 'DNN':
            y_pred = (model.predict(X_test) > 0.5).astype(int)
        else:
            y_pred = model.predict(X_test)

        # Calculate metrics
        accuracy = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        roc_auc = roc_auc_score(y_test, y_pred)

    # Display performance metrics
    st.subheader("Model Performance")
    col1, col2, col3 = st.columns(3)
    col1.metric("Accuracy", f"{accuracy:.4f}")
    col2.metric("F1 Score", f"{f1:.4f}")
    col3.metric("ROC AUC", f"{roc_auc:.4f}")
```

```python

# Explanation section
st.subheader("Explainability Dashboard")
explanation_type = st.radio("Explanation Type", ["Global (SHAP)", "Local (SHAP)", "Local (LIME)"])

if explanation_type == "Global (SHAP)":
    st.info("Global Feature Importance using SHAP")
    with st.spinner("Generating global explanations..."):
        # Sample data for faster computation
        X_sample = X_train.sample(100, random_state=42)

        # Generate SHAP values
        model_for_shap = model
        shap_model_type = 'tree' if model_type != 'DNN' else 'dnn'
        explainer, shap_values = generate_shap_explanations(
            model_for_shap,
            X_sample,
            model_type=shap_model_type
        )

        # Plot summary
        fig, ax = plt.subplots()
        shap.summary_plot(shap_values, X_sample, plot_type="bar", show=False)
        st.pyplot(fig)

        # Feature importance dataframe
        shap_df = pd.DataFrame({
            'Feature': feature_names,
            'Importance': np.abs(shap_values).mean(0)
        }).sort_values('Importance', ascending=False).head(10)
```

jupyter XAI Last Checkpoint: 1 hour ago

Edit View Run Kernel Settings Help

Code

```python

    elif explanation_type.startswith("Local (SHAP)"):
        st.info("Local Explanation for Individual Prediction using SHAP")
        instance_idx = st.slider("Select instance", 0, len(X_test)-1, 50)
        instance = X_test.iloc[[instance_idx]]
        actual_label = "Attack" if y_test.iloc[instance_idx] == 1 else "Normal"
        pred_label = "Attack" if y_pred[instance_idx] == 1 else "Normal"

        st.write(f"**Actual:** {actual_label} | **Predicted:** {pred_label}")

        with st.spinner("Generating local SHAP explanation..."):
            # Generate SHAP values
            model_for_shap = model
            shap_model_type = 'tree' if model_type != 'DNN' else 'dnn'
            explainer, shap_values = generate_shap_explanations(
                model_for_shap,
                instance,
                model_type=shap_model_type
            )

            # Force plot
            st.subheader("SHAP Force Plot")
            fig, ax = plt.subplots()
            shap.force_plot(
                explainer.expected_value,
                shap_values[0],
                instance,
                matplotlib=True,
                show=False
            )
            st.pyplot(fig)

            # Waterfall plot
            st.subheader("Feature Contribution")
            fig, ax = plt.subplots()
            shap.plots.waterfall(shap_values[0], max_display=10, show=False)
            st.pyplot(fig)
```

```python

    else:  # LIME explanation
        st.info("Local Explanation for Individual Prediction using LIME")
        instance_idx = st.slider("Select instance", 0, len(X_test)-1, 100)
        instance = X_test.iloc[[instance_idx]]
        actual_label = "Attack" if y_test.iloc[instance_idx] == 1 else "Normal"
        pred_label = "Attack" if y_pred[instance_idx] == 1 else "Normal"

        st.write(f"**Actual:** {actual_label} | **Predicted:** {pred_label}")

        with st.spinner("Generating LIME explanation..."):
            # Generate LIME explanation
            exp = generate_lime_explanation(
                model,
                X_train,
                instance.iloc[0],
                feature_names
            )

            # Show explanation
            st.subheader("LIME Explanation")
            st.write(exp.as_list())

            # Visualize
            fig = exp.as_pyplot_figure()
            st.pyplot(fig)

            # Show raw features
            st.subheader("Instance Features")
            st.dataframe(instance)

if __name__ == "__main__":
    main()
```