XAI

📌 Overview of Explainable AI (XAI) Techniques

Explainable AI techniques are crucial for understanding how a machine learning model arrives at its predictions. In high-stakes areas like network intrusion detection (like the UGRansome dataset), explainability is critical for:

- **Model Validation**: Ensuring the model is making decisions for the right reasons.
- **Compliance**: Necessary in regulated industries where transparency is required.
- **Debugging & Trust**: Helps build trust in AI systems among stakeholders.

We'll use the UGRansome dataset to demonstrate each of these techniques.

🛠 Step 1: Setting Up the Environment

Prerequisite Libraries

Before we dive into XAI techniques, ensure you have the necessary packages installed:

```
pip install shap lime matplotlib seaborn pandas scikit-learn xgboost
```

Import Statements

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import shap
import lime
from lime.lime_tabular import LimeTabularExplainer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, roc_curve
```

```
from sklearn.inspection import plot_partial_dependence, permutation_importance
import xgboost as xgb
```

📂 Step 2: Loading and Preprocessing the UGRansome Dataset

```
# Load dataset
df = pd.read_csv('final(1).csv')

# Dropping unnecessary columns and handling missing values
df = df.drop(['Unnamed: 0'], axis=1, errors='ignore').dropna()

# Splitting features and target
X = df.drop('Target', axis=1)
y = df['Target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train a RandomForest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Step 3: Feature Importance Analysis

Feature Importance (Using RandomForest)

Understanding which features influence the model the most can help guide interpretability.

```
# Calculate feature importances
importances = model.feature_importances_
features = X.columns

# Plotting feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=features)
plt.title('Feature Importance')
plt.xlabel('Importance Score')
```

```
plt.ylabel('Features')
plt.show()
```

Interpretation

- **Feature Importance** highlights which features the model focuses on.
- Useful for reducing dimensionality by eliminating low-importance features.

Step 4: SHAP (SHapley Additive exPlanations)

Overview of SHAP

- **SHAP** values measure the impact of each feature on a single prediction.
- Provides both **global** (overall dataset) and **local** (individual predictions) explanations.
- Works well with complex models like RandomForest and XGBoost.

4.1 Global SHAP Analysis

```
# Initialize SHAP explainer
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)

# SHAP Summary Plot
shap.summary_plot(shap_values[1], X_test)
```

4.2 SHAP Force Plot for Individual Prediction

```
# Visualize SHAP values for a single prediction
shap.initjs()
index = 10  # Adjust the index for different samples
shap.force_plot(explainer.expected_value[1], shap_values[1][index],
X_test.iloc[index])
```

Interpretation of SHAP Plots

- **Summary Plot** shows the most influential features across all predictions.
- **Force Plot** explains how each feature contributed to a specific prediction.

Use Case

- Useful for identifying which factors led to a network intrusion prediction.

Step 5: LIME (Local Interpretable Model-Agnostic Explanations)

Overview of LIME

- LIME explains individual predictions by fitting a local, interpretable model around each prediction.
- Useful for **local interpretability** and answering questions like, "Why was this specific sample classified as malicious?"

5.1 LIME Analysis

```
# Initialize LIME explainer
explainer = LimeTabularExplainer(X_train.values,
                                 mode='classification',
                                 training_labels=y_train.values,
                                 feature_names=X_train.columns)

# Explain a single prediction
i = 15  # Adjust index to analyze different instances
exp = explainer.explain_instance(X_test.iloc[i].values, model.predict_proba)
exp.show_in_notebook()
```

Interpretation

- LIME explanations highlight the features that were most influential in classifying a specific instance.
- Particularly useful for debugging **false positives or negatives**.

Step 6: Partial Dependence Plots (PDP)

Overview of PDP

- Shows the relationship between a feature and the model’s predictions, averaging out the effects of all other features.

- Useful for understanding how changes in a feature affect the target prediction.

6.1 PDP Analysis

```
from sklearn.inspection import PartialDependenceDisplay

# Visualizing PDP for specific features
features_to_plot = [0, 1]  # Adjust indices to visualize different features
PartialDependenceDisplay.from_estimator(model, X_train,
features=features_to_plot, feature_names=X_train.columns)
plt.show()
```

Interpretation

- PDP helps you understand how **features affect predictions** in a non-linear model like RandomForest.
- Useful for assessing whether a feature has a positive or negative impact.

Step 7: Permutation Importance

Overview of Permutation Importance

- Measures the change in the model's performance when the values of a feature are shuffled.
- Helps to identify features that are most critical for the model's predictions.

7.1 Permutation Importance Analysis

```
result = permutation_importance(model, X_test, y_test, n_repeats=10,
random_state=42)
perm_importance = pd.Series(result.importances_mean, index=X_test.columns)

# Plotting Permutation Importance
plt.figure(figsize=(10, 6))
perm_importance.sort_values().plot.barh()
plt.title('Permutation Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```

Interpretation

- Useful for validating feature importance results obtained from the model's internal feature importance scores.

Step 8: Integrated Gradients (For Neural Networks)

Overview of Integrated Gradients

- A technique used for deep learning models to measure the importance of each feature by integrating gradients.
- It captures the relationship between input features and predictions.

```
import torch
from captum.attr import IntegratedGradients

# Assume we have a trained PyTorch model named 'nn_model'
ig = IntegratedGradients(nn_model)
attr = ig.attribute(torch.tensor(X_test.values, dtype=torch.float), target=1)
```

Interpretation

- **Integrated Gradients** are useful when you want to explain deep learning models used for tasks like anomaly detection.

Step 9: Putting It All Together

Here’s a complete workflow to integrate all the techniques:

```
# Load data
df = pd.read_csv('final(1).csv')
X = df.drop('Target', axis=1)
y = df['Target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train model
```

```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# SHAP Analysis
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test)

# LIME Analysis
explainer = LimeTabularExplainer(X_train.values, mode='classification',
training_labels=y_train.values, feature_names=X_train.columns)
exp = explainer.explain_instance(X_test.iloc[10].values, model.predict_proba)
exp.show_in_notebook()

# PDP Analysis
PartialDependenceDisplay.from_estimator(model, X_train, features=[0, 1],
feature_names=X_train.columns)
plt.show()

# Permutation Importance
result = permutation_importance(model, X_test, y_test, n_repeats=10,
random_state=42)
perm_importance = pd.Series(result.importances_mean, index=X_test.columns)
perm_importance.sort_values().plot.barh()
plt.show()
```

📚 Conclusion

- **SHAP** and **LIME** provide both global and local interpretability.
- **PDP** and **Permutation Importance** help with understanding the global impact of features.
- **Integrated Gradients** are ideal for deep learning models.

These techniques are invaluable for interpreting models, especially in domains like **network intrusion detection** where transparency and trust are crucial.

Let me know if you have any

specific questions or need further clarifications on any of the XAI methods!