ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÌNH VÀ TRUYỀN THÔNG

Trần Phúc Hoàng Duy – 21522011

Lý Thế Nguyên – 21522389

Lê Triệu Vỉ - 21522785

Lê Xuân Sơn - 21521386

BÁO CÁO CUỐI KỲ

Môn học: Hệ thống tìm kiếm, phát hiện và ngăn ngừa xâm nhập

Lớp: NT204.O21.ATCL

XAI for intrusion detection system: comparing explanations based on global and local scope

GVHD: Đỗ Hoàng Hiển

TP. HỒ CHÍ MINH, 2024

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Problem

In our increasingly interconnected world, cybersecurity remains paramount. As digital threats evolve in complexity, advanced Intrusion Detection Systems (IDS) are crucial for safeguarding our networks and systems. However, these systems often rely on intricate machine learning models, making it challenging to comprehend and explain their decision-making processes. Fortunately, eXplainable Artificial Intelligence (XAI) has emerged as a powerful ally in the field of intrusion detection.

1.2 Goal

The primary objective of this project is to enhance the interpretability of machine learning models used in IDS through the application of Explainable Artificial Intelligence (XAI) methods. This involves:

- Evaluating and comparing the effectiveness of various XAI methods in explaining the decisions made by machine learning models.
- Providing insights into the internal workings of these models to help cybersecurity experts understand and trust their predictions.
- Specifically focusing on three types of models: Random Forest, eXtreme Gradient Boosting (XGBoost), and Light Gradient Boosting Machine (LightGBM).

1.3 Research methods

We use XAI methods, including Permutation Importance, SHapley Additive exPlanation (SHAP), Local Interpretable Model-Agnostic Explanation (LIME) to explain the decisions of machine learning models

CHAPTER 2. Theoretical basis

2.1 Theory

To understand the application of XAI methods in IDS, it is essential to comprehend the underlying principles of these methods:

- Permutation Importance (PI): This method assesses the importance of features by evaluating the change in model performance when the values of a feature are randomly shuffled. It provides a global view of feature importance by showing how each feature contributes to the model's overall performance.
- SHapley Additive exPlanation (SHAP): SHAP values are derived from cooperative game theory and provide a unified measure of feature importance. SHAP values quantify the contribution of each feature to a particular prediction by considering all possible feature combinations, ensuring both global and local interpretability.
- Local Interpretable Model-Agnostic Explanation (LIME): LIME generates local explanations by approximating the black-box model with an interpretable surrogate model around the vicinity of the prediction. It helps in understanding individual predictions by highlighting which features contributed most to a specific decision

CHAPTER 3. System design analysis

3.1 Overall architecture

To obtain the global explanation, we have used Permutation Importance and SHAP explanation algorithms, and for local explanations, we have used LIME, covering both global and local scope of explanation to IDSs on Random Forest, eXtreme Gradient Boosting and Light Gradient Boosting machine learning models.

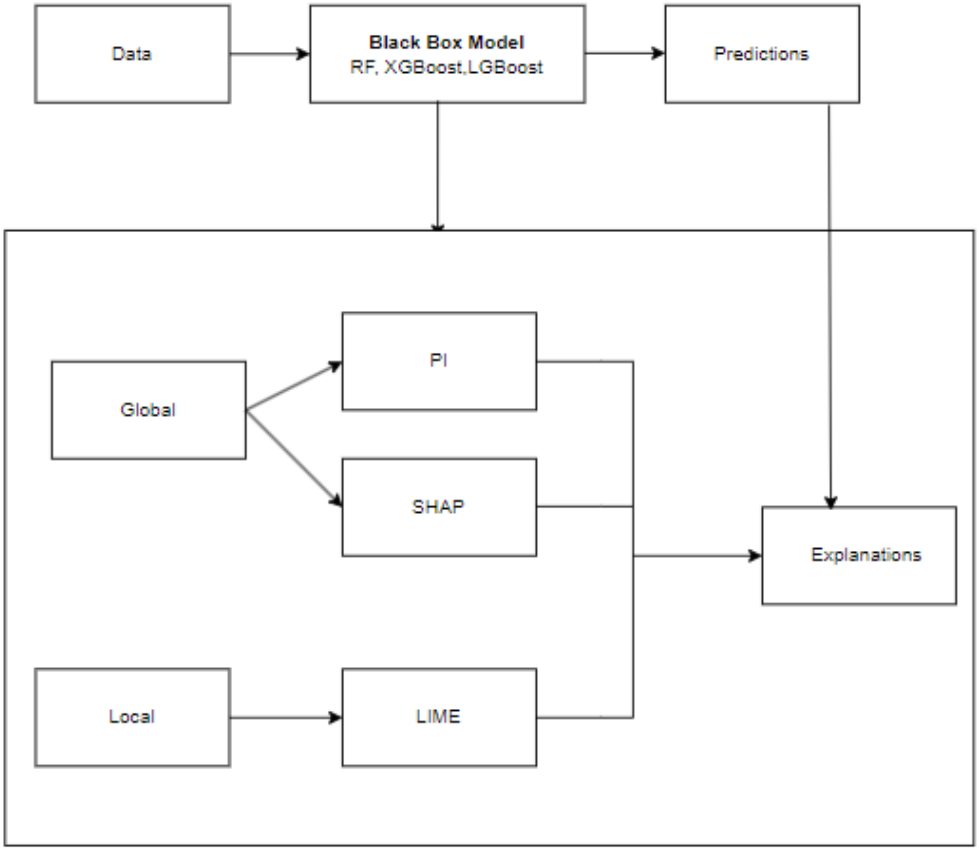

Figure 1 System overview

3.1.1 Permutation importance (PI)

Algorithm 1: Compute Permutation Importance of feature k

Require: Input: m - Trained Model, y - Target Vector, X - Feature Matrix

Ensure: Estimate the original model error $Ori = E(y, m(X))$

a. Generate feature matrix X^{perm} by permuting feature K in the data X. This breaks the association between feature k and true outcome y.
b. Estimate error $Perm = E(Y, m(X^{perm})$ based on the predictions of the permuted data.
c. Calculate permutation feature importance $PI^k = Perm/Ori$.Alternatively, the difference can be used: $PI^k = Perm - Ori$. We are using $PI^k = Perm/Ori$

Sort features by descending PI

It is based on the concept that randomly shuffling a single column should cause a loss of accurate predictions since the resulting data no longer corresponds to the original observed target value. Model accuracy especially suffers if we shuffle a column that the model depended on heavily for predictions. For stability of the results, any number from 50 to 100 permutations is recommended.

3.1.2 SHapley Additive exPlanations (SHAP)

SHAP covers both global and local scope of explanation based on Shapley value, a method in coalitional game theory. SHAP needs to satisfy the Shapely properties, namely the symmetry, wherein features that contribute equally must get the same value, dummy wherein the features that do not contribute to prediction should get zero value, and additivity wherein the features' contribution must add up to the difference of prediction and average.

Algorithm 2: Compute Shapley value of feature j [30]

Result: Φ_j, Shapley value for the value of the j-th feature
Input: Number of iterations K, instance of interest x, feature index j, number of features p,data matrix X, and machine learning model g;
for $k=1,...K$ **do**

end
Draw random instance t from the data matrix X;
Choose a random permutation λ of the feature values;
Order instance x :

$x_\lambda = (x_{(1)}, \ldots, x_{(j)}, \ldots, x_{(p)})$;
▷ Order the feature values of instance x according to the random permutation λ
Order instance t :

$t_\lambda = (t_{(1)}, \ldots, t_{(j)}, \ldots, t_{(p)})$;
▷ Order the feature values of random instance t according to the random permutation λ
Construct two new instances:

With feature j:

$x_{+j} = (x_{(1)}, \ldots, x_{(j-1)}, x_{(j)}, t_{(j+1)}, \ldots, t_{(p)})$;
▷ New instance formed which contains the feature values of x upto jth feature and the remaining feature values from instance t, while keeping the order of permutation
Without feature j:

$x_{-j} = (x_{(1)}, \ldots, x_{(j-1)}, t_{(j)}, t_{(j+1)}, \ldots, t_{(p)})$;
▷ New instance formed which contains the feature values of x upto $(j-1)$th feature and from jth feature copy the values from instance t, while keeping the order of permutation
Compute marginal contribution
$\Phi_j^k = \hat{g}(x_{+j}) - \hat{g}(x_{-j})$;
▷ $\hat{g}(.)$ is the output of machine learning model

Compute Shapley value as the average:
$\Phi_j(x) = \frac{1}{K} \sum_1^K \Phi_j^k$

First, select an instance of interest x of which explanation need to be generated, a feature j and the number of iterations needed K. For each iteration, a random instance t is selected from the data and a random order of features is generated.

Two new feature vectors are created by combining values of the instance of interest x and the random instance t. The instance x+ j is the instance of interest, but all feature values in the order after the jth feature are replaced by feature values from the sample t. The

instance $x- j$ is same as $x+ j$, but feature j and other features in order are replaced by the value for feature j and the values of other features in order from the sample t. The difference in the prediction from the black box is computed and all the differences are averaged. This process is repeated for each and every feature, to find out its Shapley value.

3.1.3 Local interpretable model-agnostic explanations(LIME)

LIME uses a surrogate model to provide local explanations. LIME explains a model by inspecting the changes that happen to the predictions when the model is given with variations of data. It generates a new dataset having perturbed samples and the corresponding predictions of the black box model. This perturbed dataset is used for training the interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest.

CHAPTER 4: System implementation

The list of libraries used in ML algorithm and XAI methods are included here:

***Machine Learning Algorithm*:**

• Random Forest classifier - **Scikit learn**

• XGBoost - **Scikit learn**

• Light Gradient Boosting - **Scikit learn**

***XAI methods*:**

• Permutation Importance - **eli5 library**

• SHapley Additive exPlanations - **shap python package**

• Local Interpretable Model Agnostic Explanations – **lime python package**

CHAPTER 5. Implementation

5.1 Experiential Scenario:

The primary focus of this work is to build a classifier for network intrusion detection that can not only give a good prediction performance but also augment the classifier model with an explainer that can explain the predictions of the classifier. We are using the RF model, XGBoost and LightGBM models on the Kaggle IDS dataset for binary classification.

5.2 Implementation:

5.2.1 Prepare data:

- First, we check the missing data and duplicate in dataset.

Missing Data

```
[9]  1 total = train.shape[0]
     2 missing_columns = [col for col in train.columns if train[col].isnull().sum() > 0]
     3 for col in missing_columns:
     4     null_count = train[col].isnull().sum()
     5     per = (null_count/total) * 100
     6     print(f"{col}: {null_count} ({round(per, 3)}%)")
```

No missing values

Duplicates

```
[10]  1 print(f"Number of duplicate rows: {train.duplicated().sum()}")
```

```
Number of duplicate rows: 0
```

Great! No duplicates

- Then, encode the label and select 15 features from dataset.

Label Encoding

```
[11] 1 def le(df):
     2     for col in df.columns:
     3         if df[col].dtype == 'object':
     4             label_encoder = LabelEncoder()
     5             df[col] = label_encoder.fit_transform(df[col])
     6
     7 le(train)
     8 le(test)
```

```
[12] 1 train.drop(['num_outbound_cmds'], axis=1, inplace=True)
     2 test.drop(['num_outbound_cmds'], axis=1, inplace=True)
     3 train.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	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_h
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	0.03	0.17	0.00	
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	0.60	0.88	0.00	
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	0.05	0.00	0.00	
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	0.00	0.03	0.04	
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	0.00	0.00	0.00	

5 rows × 41 columns

Feature selection

```
[13] 1 X_train = train.drop(['class'], axis=1)
     2 Y_train = train['class']
```

```
[14] 1 rfc = RandomForestClassifier()
     2
     3 rfe = RFE(rfc, n_features_to_select=15)
     4 rfe = rfe.fit(X_train, Y_train)
     5
     6 feature_map = [(i, v) for i, v in itertools.zip_longest(rfe.get_support(), X_train.columns)]
     7 selected_features = [v for i, v in feature_map if i==True]
     8
     9 selected_features
```

```
['protocol_type',
 'service',
 'flag',
 'src_bytes',
 'dst_bytes',
 'logged_in',
 'count',
 'srv_count',
 'same_srv_rate',
 'diff_srv_rate',
 'dst_host_srv_count',
 'dst_host_same_srv_rate',
 'dst_host_diff_srv_rate',
 'dst_host_same_src_port_rate',
 'dst_host_srv_serror_rate']
```

```
[15] 1 X_train = X_train[selected_features]
```

- Split and scale data

Split and scale data

```
[ ] 1 scale = StandardScaler()
    2 X_train = scale.fit_transform(X_train)
    3 test = scale.fit_transform(test)
```

```
[17] 1 x_train, x_test, y_train, y_test = train_test_split(X_train, Y_train, train_size=0.70, random_state=2)
```

5.2.2 Random forest model:

- Train random forest model

```
[25] 1 rf = RandomForestClassifier(max_features = study_rf.best_trial.params['rf_max_features'], max_depth = study_rf.best_trial.params['rf_max_depth'], n_estimators = study_rf.best_trial.params['r
     2 rf.fit(X_train, y_train)
     3
     4 rf_train, rf_test = rf.score(x_train, y_train), rf.score(x_test, y_test)
     5
     6 print(f"Train Score: {rf_train}")
     7 print(f"Test Score: {rf_test}")
```

```
Train Score: 0.9998865827378927
Test Score: 0.9965599364911352
```

- Apply XAI to explain the result of model:

Permutation importance (PI)

```
[28]  1 eli5.show_weights(regressor, feature_names = selected_features)
```

SHapley Additive exPlanations (SHAP)

```
[30]  1 # Create object that can calculate shap values
      2 explainer = shap.TreeExplainer(regressor)
      3 # Calculate Shap values
      4 shap_values = explainer.shap_values(x_test)
```

```
[31]  1 shap.summary_plot(shap_values, x_test,feature_names=selected_features,class_names=['0','1'],plot_type="bar")
```

Local interpretable model-agnostic explanations (LIME)

```
1 # LIME has one explainer for all the models
2 # explainer = lime.lime_tabular.LimeTabularExplainer(X_train, feature_names=X.columns.values.tolist(),class_names=['normal','anomaly'], verbose=True, mode='regressor')
3 lime_explainer = lime.lime_tabular.LimeTabularExplainer(X_train,
4                                                         mode = 'classification',
5                                                         feature_names = selected_features,
6                                                         class_names = ['normal', 'anomaly']
7                                                         )
8 predict_rfc_prob = lambda x: rf.predict_proba(x).astype(float)
9
10 _lime = lime_explainer.explain_instance(x_test[10],
11                                         predict_rfc_prob,
12                                         num_features = 15)
13 _lime.show_in_notebook()
```

5.2.3 XGBoost Gradient Boosting Model:

- Train model:

```
1 xgb_model = XGBClassifier(objective="binary:logistic", random_state=42)
2 xgb_model.fit(x_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

```
[34]  1 xgb_train, xgb_test = xgb_model.score(x_train , y_train), xgb_model.score(x_test , y_test)
      2
      3 print(f"Training Score: {xgb_train}")
      4 print(f"Test Score: {xgb_test}")
```

```
Training Score: 1.0
Test Score: 0.9965599364911352
```

- Apply XAI to model:

Permutation importance (PI)

```
[35]  1 eli5.show_weights(xgb_model, feature_names = selected_features)
```

SHapley Additive exPlanations (SHAP)

```
[36]  1 # Create object that can calculate shap values
      2 explainer = shap.TreeExplainer(xgb_model)
      3 # Calculate Shap values
      4 shap_values = explainer.shap_values(x_test)
```

```
[37]  1 shap.summary_plot(shap_values, x_test,feature_names=selected_features,class_names=['0','1'],plot_type="bar")
```

Local interpretable model-agnostic explanations (LIME)

```
[38]  1 # LIME has one explainer for all the models
      2 # explainer = lime.lime_tabular.LimeTabularExplainer(X_train, feature_names=X.columns.values.tolist(),class_names=['normal','anomaly'], verbose=True, mode='regressor')
      3 lime_explainer = lime.lime_tabular.LimeTabularExplainer(X_train,
      4                                                         mode = 'classification',
      5                                                         feature_names = selected_features,
      6                                                         class_names = ['normal', 'anomaly']
      7                                                         )
      8 predict_rfc_prob = lambda x: xgb_model.predict_proba(x).astype(float)
      9
     10 person_1_lime = lime_explainer.explain_instance(x_test[10],
     11                                                 predict_rfc_prob,
     12                                                 num_features = 15)
     13 person_1_lime.show_in_notebook()
```

5.2.4 Light Gradient Boosting Model:

- Train model:

```
[39]  1 lgb_model = LGBMClassifier(random_state=42)
      2 lgb_model.fit(x_train, y_train)
```

```
[LightGBM] [Info] Number of positive: 9389, number of negative: 8245
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001888 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1808
[LightGBM] [Info] Number of data points in the train set: 17634, number of used features: 15
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.532437 -> initscore=0.129932
[LightGBM] [Info] Start training from score 0.129932
LGBMClassifier
LGBMClassifier(random_state=42)
```

```
[40]  1 lgb_train, lgb_test = lgb_model.score(x_train , y_train), lgb_model.score(x_test , y_test)
      2
      3 print(f"Training Score: {lgb_train}")
      4 print(f"Test Score: {lgb_test}")
```

```
Training Score: 1.0
Test Score: 0.9969568668960043
```

- Apply XAI to model:

Permutation importance (PI)

```
[41]  1 eli5.show_weights(lgb_model, feature_names = selected_features)
```

SHapley Additive exPlanations (SHAP)

```
[42]  1 # Create object that can calculate shap values
      2 explainer = shap.TreeExplainer(lgb_model)
      3 # Calculate Shap values
      4 shap_values = explainer.shap_values(x_test)
```

```
[43]  1 shap.summary_plot(shap_values, x_test,feature_names=selected_features,class_names=['0','1'],plot_type="bar")
```

Local interpretable model-agnostic explanations (LIME)

```
[44]  1  # LIME has one explainer for all the models
      2 # explainer = lime.lime_tabular.LimeTabularExplainer(X_train, feature_names=X.columns.values.tolist(),class_names=['normal','anomaly'], verbose=True, mode='regressor')
      3 lime_explainer = lime.lime_tabular.LimeTabularExplainer(x_test,
      4                                                         mode = 'classification',
      5                                                         feature_names = selected_features,
      6                                                         class_names = ['normal', 'anomaly']
      7                                                         )
      8 predict_rfc_prob = lambda x: lgb_model.predict_proba(x).astype(float)
      9
     10 person_1_lime = lime_explainer.explain_instance(x_test[10],
     11                                                 predict_rfc_prob,
     12                                                 num_features = 15)
     13 person_1_lime.show_in_notebook()
```

5.3 Kết quả:

5.3.1 Random Forest model:

Weight	Feature
0.7679 ± 0.0245	src_bytes
0.0634 ± 0.0184	protocol_type
0.0511 ± 0.0137	dst_host_srv_count
0.0253 ± 0.0088	hot
0.0244 ± 0.0121	dst_bytes
0.0144 ± 0.0158	dst_host_same_src_port_rate
0.0118 ± 0.0176	flag
0.0104 ± 0.0091	service
0.0089 ± 0.0061	dst_host_srv_diff_host_rate
0.0077 ± 0.0075	dst_host_same_srv_rate
0.0053 ± 0.0126	count
0.0043 ± 0.0052	dst_host_diff_srv_rate
0.0037 ± 0.0117	srv_count
0.0009 ± 0.0025	diff_srv_rate
0.0005 ± 0.0013	same_srv_rate

Figure 2 Permutation importance score using RF model on Kaggle Binary classification dataset

This result shows the top 15 important features obtained by the permutation importance method on RF Model using Kaggle dataset. The top three features are *src_bytes*, *protocol_type* and *dst_host_srv_count* as shown in the 2nd column of the table.

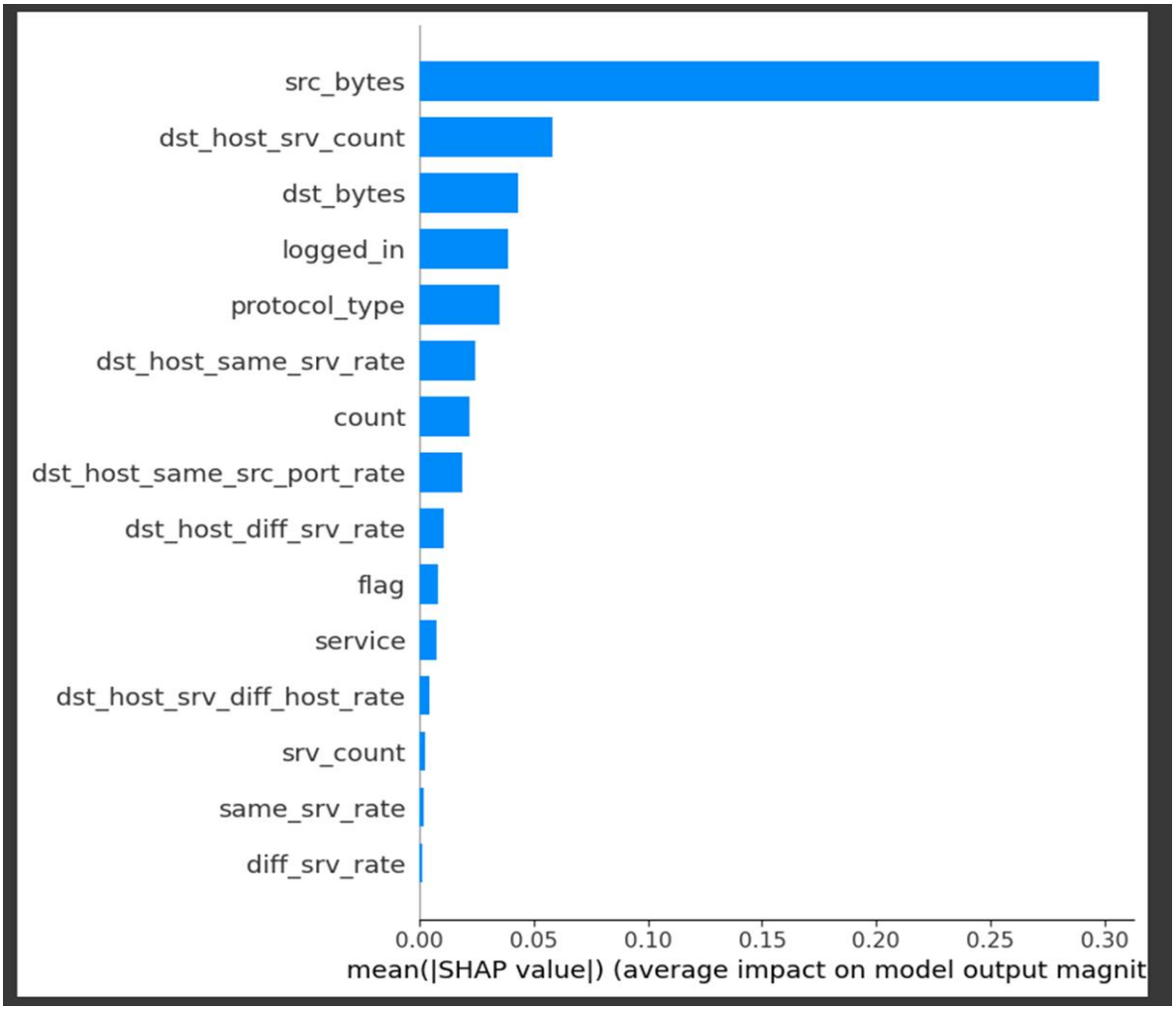

Figure 3 SHAP Summary plot using RF model on NSL-KDD binary-class classification dataset

The result shows that SHAP summary plots on Kaggle binary classification dataset. The top 3 features using RF Model are *src_bytes*, *dst_host_srv_count* and *dst_bytes*.

Figure 4 LIME Explanation for correctly classified normal instance using RF model on Kaggle classification dataset

The picture show that:

- *dst_host_srv, dst_host_diff_srv_rate, protocol_type, service, dst_host_srv_diff_host_rate, dst_host_same_src_port_rate and same_srv_rate contributes to normal prediction.*
- *hot, flag, srv_count, dst_host_same_srv_rate, dst_bytes, count, diff_srv_rate, src_bytes contributes to anomaly detection.*

5.3.2 XGBoost Gradient Boosting Model:

Weight	Feature
0.4236	src_bytes
0.1825	protocol_type
0.1102	hot
0.0580	count
0.0387	dst_host_srv_count
0.0380	dst_host_same_src_port_rate
0.0377	dst_bytes
0.0207	flag
0.0180	dst_host_srv_diff_host_rate
0.0172	diff_srv_rate
0.0162	service
0.0138	srv_count
0.0111	dst_host_diff_srv_rate
0.0089	dst_host_same_srv_rate
0.0054	same_srv_rate

Figure 5 Permutation importance score using XGBoost on Kaggle Binary classification dataset

This result shows the top 15 important features obtained by the permutation importance method on XGBoost Model using Kaggle dataset. The top three features are *src_bytes*, *protocol_type* and *hot* as shown in the 2nd column of the table.

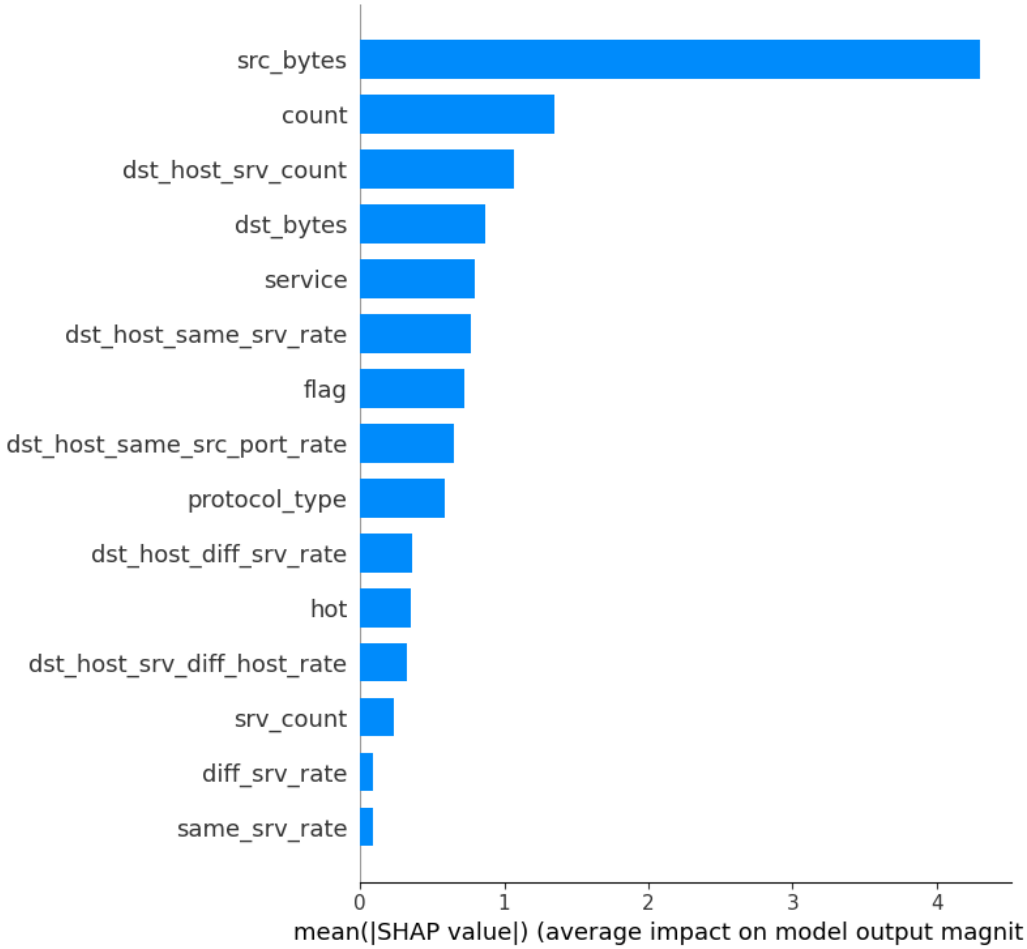

Figure 6 SHAP Summary plot using XGBoost model on NSL-KDD binary-class classification dataset

The result shows that SHAP summary plots on Kaggle binary classification dataset. The top 3 features using XGBoost Model are *src_bytes*, *count* and *dst_host_srv_count*.

Figure 7 LIME Explanation for correctly classified normal instance using XGBoost model on Kaggle classification dataset

The picture show that:

- *protocol_type, src_bytes, dst_host_srv_count, same_srv_rate, dst_host_diff_srv_rate, dst_host_srv_diff_host_rate contributes to normal prediction.*
- *hot, flag, count, srv_count, dst_bytes, dst_host_name_srv_rate, dst_host_name_src_port_rate contributes to anomaly detection.*

5.3.3 Light Gradient Boosting Model:

Weight	Feature
0.7491	src_bytes
0.0624	protocol_type
0.0423	dst_host_srv_count
0.0262	hot
0.0253	dst_bytes
0.0231	dst_host_same_src_port_rate
0.0174	count
0.0110	flag
0.0110	service
0.0093	dst_host_same_srv_rate
0.0086	dst_host_srv_diff_host_rate
0.0070	dst_host_diff_srv_rate
0.0038	srv_count
0.0028	diff_srv_rate
0.0007	same_srv_rate

Figure 8 Permutation importance score using Light Gradient Boosting model on Kaggle Binary classification dataset

This result shows the top 15 important features obtained by the permutation importance method on LightGB Model using Kaggle dataset. The top three features are *src_bytes*, *protocol_type* and *dst_host_srv_count* as shown in the 2nd column of the table.

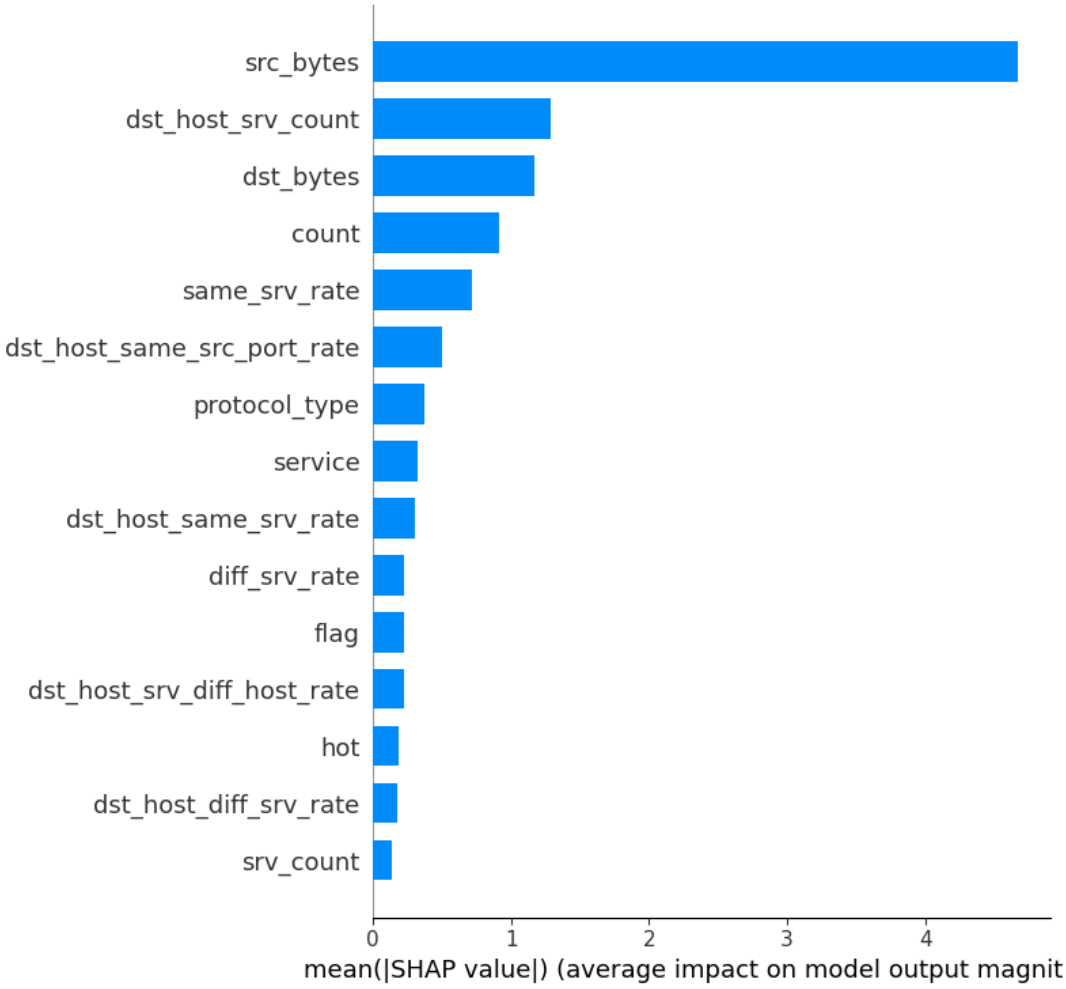

Figure 9 SHAP Summary plot using Light Gradient Boosting model on NSL-KDD binary-class classification dataset

The result shows that SHAP summary plots on Kaggle binary classification dataset. The top 3 features using LightGB Model are *src_bytes*, *dst_host_srv_count* and *dst_bytes*.

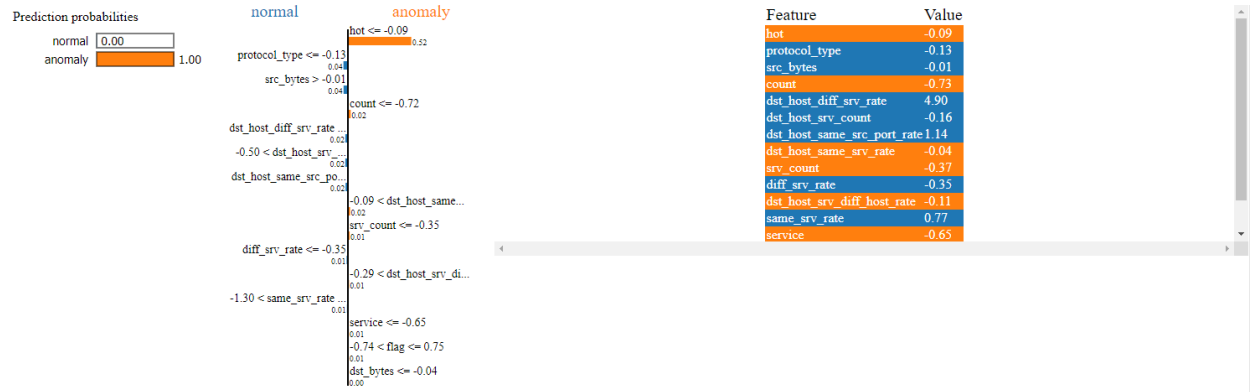

Figure 10 LIME Explanation for correctly classified normal instance using Light Gradient Boosting model on Kaggle classification dataset

The picture show that:

- *hot, count, dst_host_same_srv_rate, srv_count, dst_host_srv_diff_host_rate, service contributes to normal prediction.*
- *protocol_type, src_bytes, dst_host_diff_srv_rate, dst_host_srv_count, dst_host_same_src_port_rate, diff_srv_rate, same_srv_rate contributes to anomaly detection.*

CHAPTER 6. Conclusions and Future Works

6.1 Conclusions

Summary of the Problem and Methodology:

The primary objective of this research was to enhance the interpretability of machine learning models used in Intrusion Detection Systems (IDS) through the application of Explainable Artificial Intelligence (XAI) methods. IDS are crucial for protecting computer networks from various cyber threats, but the complexity and opacity of machine learning models can make it difficult to understand and trust their predictions. This project aimed to address this issue by evaluating and comparing several XAI methods on tree-based machine learning models, specifically Random Forest, eXtreme Gradient Boosting (XGBoost), and Light Gradient Boosting Machine (LightGBM).

To achieve this, we followed a structured research methodology:

- Data Collection and Preprocessing: We collected and preprocessed two datasets: a public network dataset from Kaggle and the NSL-KDD dataset. Preprocessing included handling missing values, encoding categorical variables, normalizing numerical features, and selecting relevant features.
- Model Training: We trained three machine learning models (Random Forest, XGBoost, and LightGBM) on the preprocessed datasets. Hyperparameter tuning was performed to optimize the models' performance, and cross-validation was used to ensure robustness.
- Application of XAI Methods: We implemented and applied four XAI methods—Permutation Importance, SHapley Additive exPlanation (SHAP), Local Interpretable Model-Agnostic Explanation (LIME), and Contextual Importance and Utility (CIU)—to explain the models' decisions.

Achieved Results:

- Permutation Importance (PI): PI provided a straightforward way to identify the most influential features by measuring the impact on model performance when features were randomly shuffled. However, PI sometimes lacked granularity in explaining individual predictions.
- SHapley Additive exPlanation (SHAP): SHAP values offered detailed and consistent explanations based on game theory, attributing each feature's

contribution to the overall prediction. SHAP was particularly useful for understanding complex interactions between features.

- Local Interpretable Model-Agnostic Explanation (LIME): LIME generated local explanations by approximating the black-box model with an interpretable model in the vicinity of the prediction. This method was effective in providing insights into specific predictions, but its explanations could vary with different local approximations.

Through these methods, we gained a comprehensive understanding of the decision-making processes of the machine learning models. Each XAI method demonstrated unique strengths and limitations:

- **Strengths**: SHAP provided highly detailed and consistent explanations, making it suitable for understanding complex interactions between features. LIME was effective for local, instance-based explanations, which were useful for specific case analyses.
- **Limitations**: PI, while easy to implement, sometimes lacked the depth required for detailed interpretability and did not consider feature interactions. LIME's reliance on local approximations could lead to inconsistencies in explanations.

6.2 Future Work

- Enhancements: Future work could focus on integrating these XAI methods into real-time IDS to provide immediate and actionable explanations for detected threats.
- Exploring Other Models and Datasets: Expanding the research to include other types of machine learning models and larger, more diverse datasets could provide further insights.
- User-Centric Explanations: Developing user-friendly interfaces and visualization tools to present the explanations in a more accessible manner for cybersecurity professionals.
- Advanced XAI Techniques: Exploring new and advanced XAI techniques that offer even deeper insights into model behavior and decision-making processes.