Reproducing Analysis and Identification of Evil Twin Attack through Data Science Techniques Using AWID3 Dataset

Authors
Daniah Mohammed
Elizabeth Kaganovsky

April 10, 2024

Abstract

Despite the wealth of measures taken to safeguard the Wi-Fi connection process, evil twin attacks still proliferate and pose a serious threat to users seeking to join new networks. An evil twin attack consists of a malicious party creating a fake access point (AP) that mimics the credentials of the legitimate one, enticing those entering the coverage area to connect to it (in some variations of the attack, forcibly de-authenticating them from the legitimate network beforehand). The 2023 paper "Analysis and Identification of Evil Twin Attack through Data Science Techniques Using AWID3 Dataset" by da Silva et al. examines the potential of machine learning algorithms as a force against these attacks, being trained on the latest edition of the Aegean Wi-Fi Intrusion Dataset (AWID3)([1]) to detect abnormal and suspicious network data that likely signifies the presence of an evil twin. Algorithms ranged from 85.658% to 99.338% accuracy in identifying evil twin-related network traffic. This paper expanded on the work done by da Silva et al. by contributing an additional execution of this concept with a Support Vector Machine (SVM) and Neural Network (NN). Results indicate that SVM with a Radial Basis Function (RBF) kernel achieved an accuracy of 98.93% with a recall of 100%, while the Neural Network achieved an accuracy of 99.04% with a recall of 100%. These findings underscore the efficacy of machine learning algorithms in detecting evil twin attacks and highlight the potential of SVM RBF and NN as promising classifiers for intrusion detection systems.

1 Introduction

Equally as long as Wi-Fi has existed has existed attacks on Wi-Fi networks, perpetually evolving to try and counteract the increased security implemented with every update to the 802.11 IEEE Standard([1]). Despite the growing complexity and ever-diversifying cryptographic algorithms and multi-stage handshakes on connection for authentication and authorization providing more security than ever before, one nefarious attack persists and seemingly evades most methods of resistance–the evil twin. An evil twin entices users to connect to an access point (AP) hosted on the attacker's machine, which mimics the credentials of a specific AP on the network to appear permissible to the victim. Currently, beyond user awareness, it is typically the responsibility of the network administrator to monitor access points and attempt to thwart evil twins before they become a threat, which is a difficult, perpetual task and impractical for most use cases.

In recent years, machine learning has entered the discussion as a promising countermeasure against wireless attacks, particularly over Wi-Fi. Particularly of use is supervised machine learning, which is trained on labelled and organized datasets for maximum accuracy, a must-have in the realm of security. Following this, in an effort to combat evil twins and other Wi-Fi-based attacks, Chatzoglou et al. developed the Aegean Wi-Fi Intrusion Dataset (AWID3) which contains an extensive collection of traces of network traffic from both mundane and malicious interactions, primarily as training material for machine learning algorithms. The AWID3 features traces of evil twin attacks. Chatzoglou et al. first examined the use of data science techniques to detect suspicious traffic that suggested the presence of an evil twin([2]). The high effectiveness of the work suggested that machine learning algorithms could be extremely useful in detecting evil twins, and further algorithms were tested by da Silva et al.([3]) to similar conclusions.

2 Related Work

Few studies have explored the idea of using machine learning as a countermeasure against Wi-Fi attacks. However, there are many examples of works that explore methods of countering these intrusions that could eventually be integrated with machine learning to produce powerful tools for users to protect themselves.

Kilincer et al. (2020) examined multiple machine learning algorithms, including SVM algorithms such as SVM linear, SVM quadratic and SVM cubic. However, evaluation metrics differed from those used by da Silva et al., and the datasets used as training material did not feature significant traces utilizing WPA3 security([4]). The work of Rizzi et al.([5]) achieved similar results but utilized 156 features to classify suspicious communications from traces in the AWID dataset, a number which leads to greater accuracy in identification but would be impractical for use in user-end applications due to the amount of information requiring assessment by the algorithm.

3 Project Motivation

The purpose of this project was to expand upon the work done by da Silva et al.([3]) to further research into the promising possibility of using machine learning to combat evil twins and similar wireless threats. A thorough comparison of as many algorithms will add to the existing body of knowledge from which other engineers may build their own Intrusion Detection Systems (IDS).

4 Methods & Materials

Due to the nature of the change between the proposal and the final report, most tools intended for this project ended up not factoring significantly into the final product. Instead, a new selection of tools was created and utilized, with several drawn from those used by da Silva et al.([3]) to mimic the conditions of the initial implementation.

AWID3 dataset A database containing traces of legitimate authentication and authorization as well as various attacks performed in the IEEE 802.1X Extensible Authentication Protocol (EAP) environment, recorded by the University of the Aegean in Greece. The AWID3 includes traces of attacks performed on 802.11 standard networks, including WPA, WPA2 and WPA3. A selection of 16 relevant classification features for the identification of evil twin attacks were determined by Chatzoglou et al.([1]), which are utilized by da Silva et al.([3]) as well as in this paper.

Feature Extraction Previous research ([2]) has aimed to identify key features that could effectively detect Wi-Fi attacks based on dataset description and hypotheses developed with data exploration (which focus on understanding whether specific wireless network settings or configurations make a machine more prone to evil twin attacks,

and if encryption has an influence on said attacks). Features were reduced by removing constant and highly correlated columns and using a high Pearson correlation threshold of 0.950. The result was a selection of notably variable numerical and categorical features such as *'frame.len,' 'radiotap.dbm_antsignal,' 'wlan.duration,'* and assorted *'wlan.fc'* attributes.

Evil twin detection code by da Silva et al. Authors Leandro Marcos da Silva and Vinícius Moraes Andreghetti of the University of São Paulo, Brazil, generously provided their code used to analyze the AWID3 dataset. The code is written in the Python programming language and utilizes the Pandas, NumPy, Matplotlib, and scikit-learn libraries, and is considered to be utilized in this project under fair use. The code was modified significantly to facilitate use with Google Colab.

Google Colab A hosted Jupyter Notebook service that allows access to computing resources such as CPUs and Tensor Processing Units (TPUs) for machine learning purposes.

airgeddon An open-source wireless network auditing tool used by penetration testers capable of performing a variety of network attacks. This tool was utilized in the initial iteration of this project but was not used for any features in the final work.

Kali Linux An open-source, security-oriented Linux distribution favoured by security technicians due to the great variety of tools provided for penetration testing and network auditing. This tool was utilized in the initial iteration of this project but was not used for any features in the final work.

4.1 ML Algorithms Used

Each algorithm mentioned in Table 1 plays a crucial role in classifying evil twin attacks [6]:

- **Decision Tree:** A classification of multiple algorithms which partitions data into subsets determined by selected feature values.
- **Random Forest:** An "ensemble learning method" which constructs multiple decision trees.
- **Gaussian Naive Bayes:** A probabilistic classifier based on Bayes' theorem (used for determining conditional probability). GNB assumes that features are conditionally independent, given the class.
- **Multi-Layer Perceptron (MLP):** A type of feed-forward (guiding of future decisions as opposed to feed-back, evaluation of previous performance) neural network consisting of multiple layers of nodes (neurons), each connected to the nodes of the next layer. MLP is capable of learning complex relationships in datasets and is widely used for large-scale classification tasks.
- **Extreme Gradient Boosting (XGBoost):** A highly customizable, efficient and easily scalable implementation of gradient boosting machines. XGBoost builds multiple decision trees sequentially, where each subsequent tree corrects the errors made by the previous ones, aiming to improve performance with each iteration.
- **LightGBM:** A gradient-boosting framework that uses a tree-based learning algorithm. LightGBM is designed for efficient, distributed training on large-scale datasets, and provides both high accuracy with faster computation.
- **LightGBM (Optimized):** An optimized version of LightGBM, possibly with tuned hyperparameters (external configuration variables) or additional preprocessing steps, aimed at further improving the performance of the original algorithm.
- **SVM (Linear):** A comparatively simple SVM with a linear kernel. Most effective when the data is linearly separable and can handle large feature spaces efficiently, allowing for a linear decision boundary to separate classes.
- **SVM (RBF):** A more complex SVM with a Ravial Basis Function (RBF) kernel, capable of capturing complex, non-linear decision boundaries via mapping input data into a higher dimensional space. SVM RBF excels at classification tasks on data with non-linear relationships.
- **Neural Network:** A powerful machine learning algorithm, capable of learning even very intricate relationships between data that other algorithms may overlook. However, due to the multiple levels of processing, neural networks often require greater training time than other models.

4.2 Confusion Matrix

A confusion matrix is used to assess the performance of binary classification models, comparing

actual class labels administered by the curator of the dataset to those predicted by the algorithm. [6]:

- **True Positive (TP):** Instances that are correctly predicted as positive.
- **False Positive (FP):** Instances that are incorrectly predicted as positive (but are actually negative).
- **True Negative (TN):** Instances that are correctly predicted as negative.
- **False Negative (FN):** Instances that are incorrectly predicted as negative (but are actually positive).

4.3 Performance Metrics

Each performance metric in Table 1 provides valuable insights into the effectiveness of the trained model in classifying evil twin attacks [6]:

- **Accuracy:** Measures the overall correctness of a model's predictions, representing a ratio of correctly classified instances to the total number of actual instances.
- **F1-Score:** The harmonic mean of precision and recall.
- **Precision:** Measures the accuracy of positive predictions made by the model. Representative of the ratio of true positive predictions to the total number of positive predictions.
- **Recall:** Measures the ability of the model to correctly identify positive instances. Representative of true positive predictions to the total number of actual positive instances as a ratio.
- **AUC (Area Under the ROC Curve):** The ability of the model to distinguish between the different classes. AUC measures the area under the Receiver Operating Characteristic (ROC) curve. A higher AUC indicates better performance.
- **Training Time:** Time taken by the algorithm to train the model on the dataset (units may vary–in this paper, it is seconds).

5 Approach

Our initial idea was to perform an evil twin attack using tools such as Kali Linux and airgeddon, along with a WiFi adapter and router. However, due to technical challenges, including the lack of a suitable router for the attack, we shifted our focus to reproducing and expanding upon existing research.

During our exploration, we came across the paper titled *Analysis and Identification of Evil Twin Attack through Data Science Techniques Using AWID3 Dataset* by da Silva et al.([3]). This paper discusses the application of machine learning (ML) algorithms to detect evil twin attacks, utilizing the AWID3 dataset. Inspired by this work, we decided to dive deeper into this topic.

5.1 Implementation

In the technical aspect of our work, we utilized Google Colab and Python to run the code, leveraging the resources available in the GitHub repository related to the paper. We opted to use preprocessed features from the repository, as time constraints prevented us from extracting features directly from the AWID3 dataset. This decision also ensured consistency with the data used in the original study.

Finally, we collected performance metrics for both models trained - SVM with BFR kernel and Neural Network - and added them to the final report along with the corresponding code used.

Figure 1 shows the detection system overview.

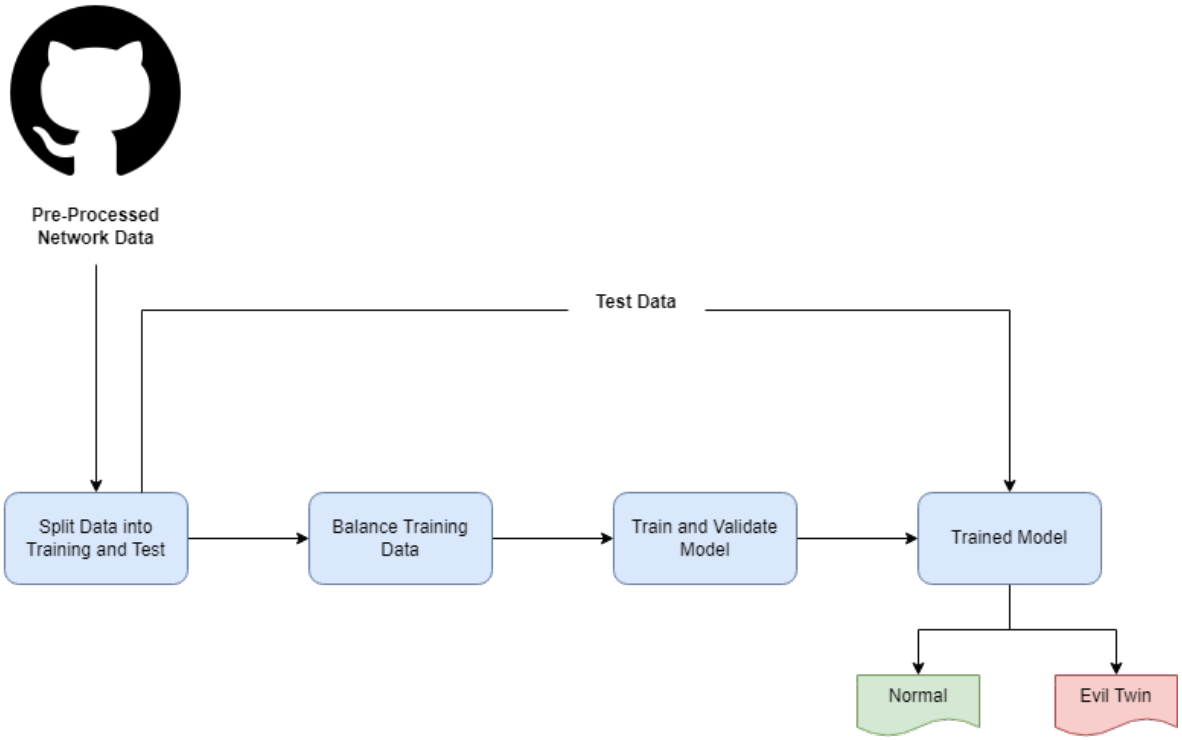

Figure 1: System overview for identifying evil twin attack

5.2 Challenges

Initially, we encountered challenges due to a lack of clarity regarding our proposed approach. However, after receiving feedback from our professor during the presentation, we gained a clearer understanding of the project requirements and objectives. This feedback guided us toward identifying a suitable paper for reproduction, aligning our efforts with the project goals.

Another challenge we faced was related to data preprocessing and feature extraction. Due to time constraints, we encountered difficulties in extracting features from the dataset and performing the necessary preprocessing steps. To mitigate this issue, we opted to utilize preprocessed data available from a GitHub repository associated with the chosen paper. This decision enabled us to focus on model implementation and evaluation, ensuring timely progress and alignment with project milestones.

6 Results and Discussion

Algorithm	Accuracy	F1-Score	Precision	Recall	AUC	Training Time
Decision Tree	0.9907	0.9908	0.9879	0.9937	0.9910	0.0420
Random Forest	**0.9934**	**0.9934**	**0.9900**	0.9968	0.9977	0.7802
Gaussian Naive Bayes	0.8566	0.8733	0.7826	0.9878	0.9885	0.0278
Multi-Layer Perceptron	0.9921	0.9921	0.9864	0.9979	0.9971	9.9356
XGBoost	0.9915	0.9916	0.9874	0.9958	0.9985	0.4921
LightGBM	0.9926	0.9926	0.9890	0.9963	0.9989	0.3206
LightGBM (Optimized)	0.9931	0.9932	0.9880	0.9984	**0.9994**	0.3055
SVM (Linear)	0.9877	0.9848	0.9766	0.9932	0.0000	**0.0180**
SVM (RBF)	0.9893	0.8027	0.6704	**1.0000**	0.9990	0.1718
Neural Network	0.9904	0.8185	0.6928	**1.0000**	0.9978	24.7670

Table 1: Performance Metrics

6.1 Results Analysis

From Table 1, it can be observed that each algorithm achieved different levels of performance across the various metrics. The best-performing algorithm varies depending on the specific metric considered. For example, Random Forest achieved the highest accuracy and F1-Score, while SVM with BFR Kernel achieved the highest precision. LightGBM (Optimized) demonstrated the highest AUC, indicating its ability to distinguish between positive and negative classes effectively. Moreover, SVM (RBF) and Neural Networks exhibited the highest recall, indicating their capability to correctly identify most of the actual positive instances. Overall, LightGBM (Optimized) performed the best across multiple metrics, making it a strong candidate for detecting evil twin attacks.

6.2 Neural Network (NN) Analysis

Neural Network (NN) is a powerful model known for its ability to learn complex patterns from data. In our analysis, the NN achieved an accuracy of 99.04%, indicating that it correctly classified the majority of instances. However, the F1-score of 81.85% suggests that the model's precision and recall are not as balanced. This imbalance could be due to a higher number of false positives or false negatives, which may require further investigation. Despite this, the high recall of 100% indicates that the NN effectively identified all actual positive instances, making it a reliable choice for detecting evil twin attacks. Figure 2 shows the Neural Network Classification Report.

6.3 Support Vector Machine with Radial Basis Function (SVM RBF) Analysis

SVM with Radial Basis Function (RBF) kernel is known for its ability to capture complex, non-linear decision boundaries. In our analysis, the SVM RBF achieved an accuracy of 98.93% and a high recall of 100%, indicating its effectiveness in correctly identifying actual positive instances. However, the F1-score of 80.27% suggests an imbalance between precision and recall, similar to the NN. Further investigation may be needed to understand the cause of this imbalance and improve the model's performance. Overall, the SVM RBF demonstrates strong potential for detecting evil twin attacks, especially in scenarios with non-linear relationships among features. Figure 3 shows the SVM RBF Classification Report.

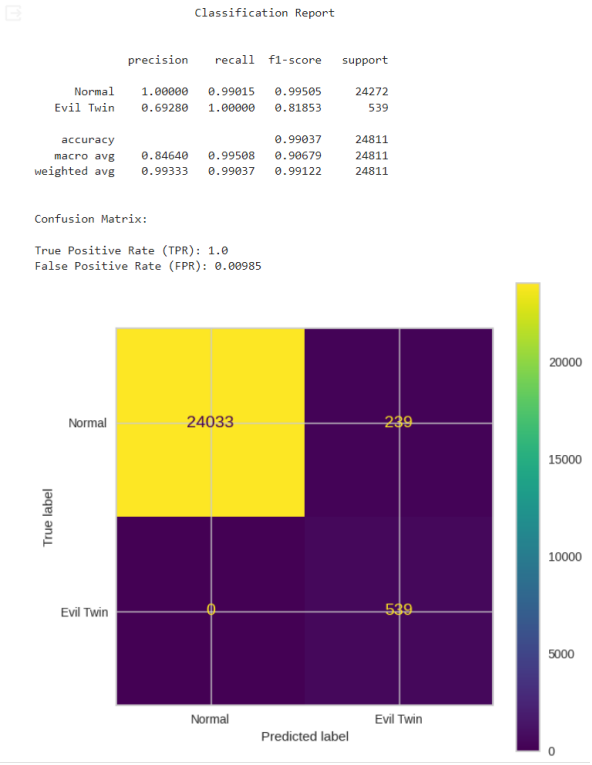

Figure 2: Neural Network Classification Report

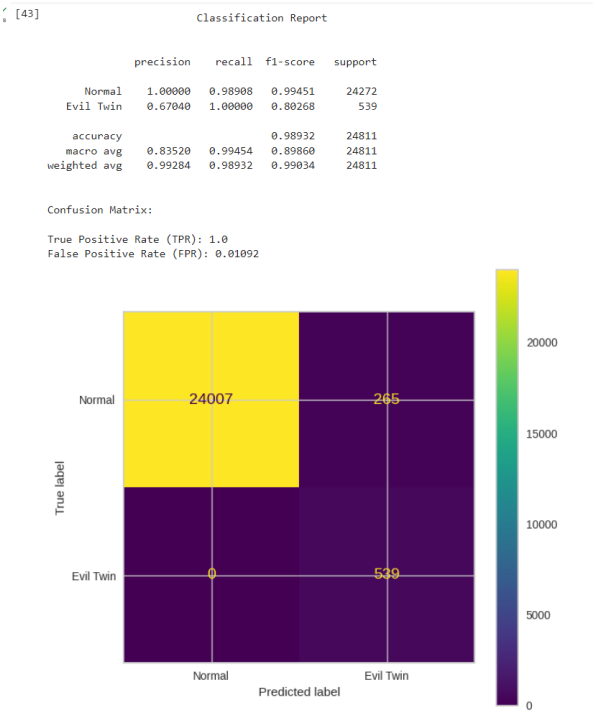

Figure 3: SVM RBF Classification Report

6.4 Choice of Neural Network and SVM RBF

Neural Network (NN) and Support Vector Machine with Radial Basis Function (SVM RBF) were chosen due to their inherent strengths in classification tasks:

- **Neural Network:** NNs are known for their ability to learn complex patterns from data, making them suitable for classification tasks involving high-dimensional input spaces. Due to the complexity and variability of Wi-Fi attacks, they appeared as a promising choice for an IDS algorithm.
- **SVM RBF:** SVM with RBF kernel is effective in capturing non-linear decision boundaries and is robust to overfitting. It performs well in scenarios where the data is not linearly separable. After examining the drawbacks of SVM with a linear kernel, it appeared that the features were not linear separable and therefore a more advanced algorithm was required.

Given the nature of evil twin detection, which may involve intricate patterns and non-linear relationships among features, NN and SVM RBF were considered appropriate choices for training the classification model.

7 Conclusion

This work continued the research done by da Silva et al.([3]) into the field of machine learning-based detection of evil twins, to expand the existing information available on file for future developers seeking to choose an appropriate algorithm to include in their Intrusion Detection Systems (IDS). The latest version of the Aegean Wi-Fi Intrusion Dataset, AWID3, was used in order to sample traces which utilized the WPA3 protocol for security, for a more robust and modern training set. After executing a similar experiment to that performed by da Silva et al. ([3]), the algorithms in question proved somewhat competitive and would make legitimate options for future IDS development. After our investigation, it was discovered that SVM RBF and Neural Network displayed the highest recall of all surveyed algorithms in both studies, with a 1.0 value, signifying the models could identify all suspicious data. However, both algorithms displayed low precision as they incorrectly marked significant amounts of otherwise mundane data as suspicious. Despite this, both still could be promising candidates for future IDS, as false positives are inconveniences and require connection re-establishment, while false negatives can let a malicious actor in.

8 Future Work

In addition to the findings presented in this report, there are several avenues for future exploration and improvement:

- **Deep Learning Algorithms:** Consider exploring the application of deep learning algorithms, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), which have shown promise in various security domains.
- **Real-Time Protection Application:** Develop and implement the algorithms studied here into a real-time protection application for end-users. This would involve optimizing the algorithms for efficiency and integrating them into existing security frameworks.
- **Enhanced Feature Selection:** Investigate the possibility of enhancing feature selection techniques to improve the accuracy and efficiency of evil twin detection algorithms. This could involve incorporating additional network traffic attributes or refining existing feature sets.

9 Acknowledgements

We would like to express our gratitude to Leandro Marcos da Silva and Vinícius Moraes Andreghetti of the University of São Paulo, Brazil, for generously providing their code and insights used in this study. Their contributions were instrumental in facilitating our research.

References

[1] E. Chatzoglou, G. Kambourakis, C. Kolias, and C. Smiliotopoulos, "Empirical evaluation of attacks against ieee 802.11 enterprise networks: The awid3 dataset," *IEEE Access 9*, 2021.

[2] E. Chatzoglou, G. Kambourakis, and C. Kolias, "Pick quality over quantity: Expert feature selection and data preprocessing for 802.11 intrusion detection systems," *IEEE Access 9*, 2021.

[3] L. M. da Silva, V. M. Andreghetti, R. A. F. Romero, and K. R. L. J. C. Branco, "Analysis and identification of evil twin attack through data science techniques using awid3 dataset," *MLMI: Machine Learning and Machine Intelligence*, 2024.

[4] I. F. Kilincer, F. Ertam, and A. Sengur, "Machine learning methods for cyber security intrusion detection: Datasets and comparative study," *Computer Networks*, vol. 188, p. 107840, 2020.

[5] A. Rizzi, G. Granato, and A. Baiocchi, "Frame-by-frame wi-fi attack detection algorithm with scalable and modular machine-learning design," *Applied Soft Computing*, vol. 91, p. 106188, 2020.

[6] A. Burkov, *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

10 Appendix

10.1 Contributions

Daniah Mohammed

- Implemented the technical aspect of the report.
- Chose and trained machine learning models.
- Wrote the following sections: Methods & Materials, Approach, Results and Discussion, and peer reviewed the other sections.
- Contributed to the analysis of results.
- Created visualizations for the results.

Elizabeth Kaganovsky

- Conducted literature review.
- Discussed and helped choose machine learning models.
- Sought out necessary resources for the experiment.
- Wrote sections of the Abstract, Introduction, Project Motivation, Methods & Materials, Future Work, and peer-reviewed other sections.
- Contributed to the analysis of results.

10.2 Code

Cloning the Repository and Installing Dependencies

```
# Clone the repo from the Analysis and Identification of Evil Twin
    Attack through Data Science Techniques Using AWID3 Dataset paper
!git clone https://github.com/silvamleandro/EvilTwin_detection.git

%cd /content/EvilTwin_detection/src

# Install important dependencies
!pip install pycaret
!pip install optuna
```

Imports and Function Definitions

```
# Imports
from imblearn.over_sampling import KMeansSMOTE
from pathlib import Path
from pycaret.classification import *
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, auc,
    classification_report, confusion_matrix, ConfusionMatrixDisplay,
    f1_score, precision_score, recall_score, roc_curve
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold
from time import time
from utils import split_X_y # utils.py
```

```
import lightgbm as lgb
import matplotlib.pyplot as plt
import numpy as np
import optuna
import pandas as pd
import pickle
import warnings
import xgboost as xgb
import pycaret
import time
```

Function Definitions Continued

```
# Ignore Pandas warnings
warnings.filterwarnings("ignore")
# Random state default
RANDOM_STATE = 42

# The functions used in the repo
def plot_roc_curve(model, fpr, tpr, roc_auc, k,
                   save_figure=False, reports_path=None, file_name=
                       None):
    # Config the plot
    plt.figure(figsize=(10,10))
    plt.grid()
    plt.plot([0, 1], [0, 1], ls='--')
    plt.ylim([-0.01, 1.01])
    plt.xlim([-0.01, 1.01])
    plt.xlabel('False Positive Rate', size=16)
    plt.ylabel('True Positive Rate', size=16)

    linspace = np.linspace(0, 1, 100)

    # Calculate the mean ROC
    int_tpr = [np.interp(linspace, fpr[i], tpr[i]) for i in range(k)]
    mean_tpr = np.mean(int_tpr, axis=0)

    # Plot all ROC curves for each fold and the average ROC curve
    for i in range(k):
        plt.plot(fpr[i], tpr[i], label=f'fold {i+1}, Area: {roc_auc
            [i]:.5f}')

    plt.plot(linspace, mean_tpr, label=f'Average Curve, Area: {auc(
        linspace, mean_tpr):.5f} (+/- {np.std(roc_auc)*2:.5f})')

    # Plot the graph
    plt.legend(prop={'size': 16})
    # Save the ROC curve
    plt.savefig(f'{reports_path}images/{file_name}-roc_curve.png')
    plt.show() # Show the plot



def plot_confusion_matrix(y_true, y_pred, class_names, normalized,
                          save_figure=False, reports_path=None,
                              file_name=None):
```

```
    # Get confusion matrix
    cm = confusion_matrix(y_true, y_pred, normalize=normalized)
    # Confusion matrix display
    cm_p = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels
        =class_names)
    fig, ax = plt.subplots(figsize=(7,7)) # Plot size
    plt.rcParams.update({'font.size': 14}) # Font size
    cm_p.plot(ax=ax) # Confusion matrix plot show

    if save_figure == True: # Save figure
        plt.savefig(f'{reports_path}images/{file_name}-cm.png')

    # Return confusion matrix
    return cm



def classification_with_report(model, X, y, k, class_names,
    save_report=False,
                                   reports_path=None, file_name=None,
                                       verbose=True):
    # Lists
    fpr, tpr, roc_auc = [], [], []
    original_label, predicted_label, predicted_proba = [], [], []

    # Stratified K-Fold cross-validation
    skf = StratifiedKFold(k)

    # Time counting
    start = time()

    # Train and test the model for each 'k' fold in all the data
    for train_index, test_index in skf.split(X,y):
        model.fit(X.iloc[train_index], y[train_index])
        y_pred_probability = model.predict_proba(X.iloc[test_index])

        # Predict to generate classification report
        y_pred = model.predict(X.iloc[test_index])
        predicted_label.extend(y_pred)
        original_label.extend(y[test_index])

        # Compute micro-average ROC curve and ROC area
        temp_fpr, temp_tpr, _ = roc_curve(y[test_index],
            y_pred_probability[:,1])
        fpr.append(temp_fpr)
        tpr.append(temp_tpr)
        roc_auc.append(auc(temp_fpr, temp_tpr))

    # Total time spent on training
    total_time = time() - start

    if verbose == True: # Show results
        # Results
        print('\t\t\tClassification Report\n\n')
        print(classification_report(original_label, predicted_label,
            target_names=class_names, digits=5))
```

```
        print('Accuracy:' + str(round(accuracy_score(original_label,
             predicted_label), 5)))
        print('F1-Score:' + str(round(f1_score(original_label,
            predicted_label), 5)))
        print('Precision:' + str(round(precision_score(
            original_label, predicted_label), 5)))
        print('Recall:' + str(round(recall_score(original_label,
            predicted_label), 5)))
        print('AUC:' + str(round(np.mean(roc_auc), 5)))
        print('Total Time:' + str(round(total_time, 5)) + 'seconds'
            )
        print('Confusion Matrix:\n')
        # Plot the ROC curve
        plot_roc_curve(model, fpr, tpr, roc_auc, k)
        # Plot the confusion matrix
        _ = plot_confusion_matrix(original_label, predicted_label,
            class_names, None)

    if save_report == True: # Save report
        # file_name (without extension)
        with open(reports_path + file_name + '.txt', 'a+') as f:
            f.write(type(model).__name__ + '\n')
            f.write(str(model))
            f.write('\n\n\n')
            f.write('\t\t\tClassification Report\n\n')
            f.write(classification_report(original_label,
                predicted_label, target_names=class_names, digits=5))
            f.write('\n\nAccuracy:' + str(round(accuracy_score(
                original_label, predicted_label), 5)))
            f.write('\n\nF1-Score:' + str(round(f1_score(
                original_label, predicted_label), 5)))
            f.write('\n\nPrecision:' + str(round(precision_score(
                original_label, predicted_label), 5)))
            f.write('\n\nRecall:' + str(round(recall_score(
                original_label, predicted_label), 5)))
            f.write('\nAUC:' + str(round(np.mean(roc_auc), 5)))
            f.write('\nTotal Time:' + str(round(total_time, 5)) + '
                seconds\n\n')
            # Save the ROC curve
            plot_roc_curve(model, fpr, tpr, roc_auc, k,
                           save_figure=True, reports_path=
                               reports_path, file_name=file_name)
            # Save the confusion matrix
            _ = plot_confusion_matrix(original_label, predicted_label
                , class_names, None,
                                      save_figure=True, reports_path=
                                          reports_path, file_name=
                                          file_name)

    return model # Return trained model



def report_model(model, X, y, class_names, normalized='true'):
    # Predict to generate report
    y_pred = model.predict(X)
```

```
    # Results
    print('\t\t\tClassification Report\n\n')
    print(classification_report(y, y_pred, target_names=class_names,
        digits=5))

    # Confusion matrix
    print('\nConfusion Matrix:\n')
    cm = plot_confusion_matrix(y, y_pred, class_names, normalized)
    # True Negative (TN)
    tn = cm[0][0]
    # False Negative (FN)
    fn = cm[1][0]
    # True Positive (TP)
    tp = cm[1][1]
    # False Positive (FP)
    fp = cm[0][1]
    # True Positive Rate (TRP)
    tpr = tp / (tp + fn)
    # False Positive Rate (FPR)
    fpr = fp / (fp + tn)
    print(f'True Positive Rate (TPR): {round(tpr, 5)}')
    print(f'False Positive Rate (FPR): {round(fpr, 5)}')
```

Data Loading and Preprocessing

```
# Accessing the pre-processed dataset
# General path
path = f'/content/EvilTwin_detection/'
# Pre-processed data path
preprocessed_data_path = f'{path}data/pre_processed/'
# Path to save the models
models_path = f'{path}/models/'
# Path to save results
reports_path = f'{path}/reports/'

# Load pre-processed training dataset
train_df = pd.read_csv(preprocessed_data_path + 'train_data.csv')
# Load pre-processed test dataset
test_df = pd.read_csv(preprocessed_data_path + 'test_data.csv')
# Split train data into X and y
X_train, y_train = split_X_y(train_df)
# Split test data into X and y
X_test, y_test = split_X_y(test_df)

# Balance train data with SMOTE
X_res, y_res = KMeansSMOTE(random_state=RANDOM_STATE).fit_resample(
    X_train, y_train)
```

Model Training and Evaluation

```
# Define the neural network model
nn_model = MLPClassifier(random_state=RANDOM_STATE)

# Train the model on the balanced training data
```

```
nn_model.fit(X_res, y_res)

# Predict labels for the test data
y_pred_nn = nn_model.predict(X_test)

report_model(nn_model, X_test, y_test, ['Normal', 'Evil-Twin'], None)

# Import the SVC class
from sklearn.svm import SVC

# Define the SVM model with RBF kernel
svm_rbf_model = SVC(kernel='rbf', random_state=RANDOM_STATE)

# Train the model on the balanced training data
svm_rbf_model.fit(X_res, y_res)

# Predict labels for the test data
y_pred_svm_rbf = svm_rbf_model.predict(X_test)

report_model(svm_rbf_model, X_test, y_test, ['Normal', 'Evil-Twin'],
    None)
```

Performance Metrics

```
import time
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score
    , recall_score, roc_auc_score
from sklearn.metrics import confusion_matrix, roc_curve, auc,
    classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np

# Define the train_model_and_get_metrics function
def train_model_and_get_metrics(model, X_train, y_train, X_test,
    y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    end_time = time.time()
    train_time = end_time - start_time

    y_pred = model.predict(X_test)

    metrics = {
        'Accuracy': accuracy_score(y_test, y_pred),
        'F1-Score': f1_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred)
    }

    if isinstance(model, SVC):
        decision_scores = model.decision_function(X_test)
        metrics['AUC'] = roc_auc_score(y_test, decision_scores)
```

```
    elif isinstance(model, MLPClassifier):
        y_pred_proba = model.predict_proba(X_test)[:, 1]
        metrics['AUC'] = roc_auc_score(y_test, y_pred_proba)

    return metrics, train_time

# Define models
models = {
    'SVM with RBF Kernel': SVC(kernel='rbf', random_state=
        RANDOM_STATE),
    'Neural Network': MLPClassifier(random_state=RANDOM_STATE)
}

# Train models and get metrics
metrics = {}
times = {}
for model_name, model in models.items():
    metrics[model_name], times[model_name] =
        train_model_and_get_metrics(model, X_res, y_res, X_test,
        y_test)

# Display metrics
for model_name, metric in metrics.items():
    print(f"{model_name}:")
    for metric_name, value in metric.items():
        print(f"{metric_name}: {value}")
    print(f"TT (Sec): {round(times[model_name], 4)}\n")
```