Project Report

Microsoft Cybersecurity Incidents Classification with Machine Learning

By

Karthikeyan M

1. Introduction

This project aims to develop a machine learning model to assist Security Operation Centers (SOCs) by automating the triage process of cybersecurity incidents. The model is trained to classify incidents into True Positive (TP), Benign Positive (BP), or False Positive (FP) using the comprehensive GUIDE dataset. The ultimate goal is to provide SOC analysts with precise, context-rich recommendations that enhance the security posture of enterprise environments.

2. Problem Statement

In SOC environments, efficiently triaging incidents is critical. With a vast number of alerts generated daily, manually categorizing incidents is time-consuming and prone to errors. The machine learning model developed in this project will predict the triage grade of incidents based on historical data and customer responses, ensuring that real threats are promptly addressed.

3. Workflow Overview

The project follows a structured workflow to ensure a comprehensive approach to model development:

3.1 Basic Overview

The initial step involves understanding the dataset structure and defining the project scope, including the evaluation metrics like macro-F1 score, precision, and recall, which are crucial for assessing model performance.

3.2 Data Exploration

3.2.1 Objective

To gain insights into the dataset's structure, identify patterns, correlations, and any anomalies that might affect the model's performance.

3.2.2 Process

- **Loading Data**: The dataset is loaded in chunks due to its large size, optimizing memory usage.
- **Data Summary**: Key statistics, such as the shape of the dataset, data types, and missing values, are computed.

- **Visualization**: The distribution of the target variable IncidentGrade is visualized, revealing class imbalances that must be addressed during model training.

3.2.3 Key Findings

- **Class Imbalance**: Significant imbalance in the target variable, with BenignPositive being the most frequent class.
- **Missing Values**: Several features, including MitreTechniques and ActionGrouped, have a high proportion of missing values.

3.3 Data Preprocessing

3.3.1 Objective

To clean and transform the data into a format suitable for model training, addressing issues such as missing values, encoding categorical variables, and scaling numerical features.

3.3.2 Process

- **Handling Missing Data**: Missing values were imputed using strategies like forward fill and mean imputation, depending on the nature of the feature.
- **Feature Engineering**: New features were derived, such as timestamp-based features, and redundant features were removed.
- **Encoding Categorical Variables**: Categorical features were converted into numerical formats using one-hot encoding.
- **Scaling**: Numerical features were standardized to ensure that all features contribute equally to the model training.

3.3.3 Key Outcomes

- **Cleaned Dataset**: A fully processed dataset ready for model training.

3.4 Data Splitting and Sampling

3.4.1 Objective

To split the dataset into training and validation sets to facilitate model training and evaluation.

3.4.2 Process

- **Train-Validation Split**: The data was split into training and validation sets with an 80-20 ratio, ensuring that the class distribution remains consistent.

- **Stratified Sampling**: This technique was used to maintain the balance of classes in both sets, preventing skewed results during model training.

3.4.3 Key Outcomes

- **Balanced Training and Validation Sets**: Ensures reliable model training and evaluation.

3.5 Model Selection and Training

3.5.1 Objective

To select and train various machine learning models to find the best-performing model for the classification task.

3.5.2 Models Used

1) Logistic Regression:

Definition: A linear model used for binary classification tasks. It estimates the probability of a binary response based on one or more predictor variables.

Use Case: Suitable for problems where the relationship between the features and the outcome is roughly linear and easily interpretable.

2) **Decision Tree**:

Definition: A non-linear model that splits data into subsets based on feature values, forming a tree-like structure.

Use Case: Useful for capturing non-linear relationships and for interpretability. It works well for smaller datasets or when model interpretability is crucial.

3) **Random Forest**:

Definition: An ensemble learning method that constructs multiple decision trees and merges them to reduce variance and improve accuracy.

Use Case: Ideal for handling large datasets with high dimensionality and capturing complex feature interactions while reducing overfitting.

4) **XGBoost (Extreme Gradient Boosting)**:

Definition: An optimized gradient boosting algorithm designed for speed and performance, particularly in classification tasks.

Use Case: Highly effective for structured/tabular data and competitions; handles missing values and large datasets well.

5) **LightGBM (Light Gradient Boosting Machine)**:

Definition: A gradient boosting framework that uses tree-based learning algorithms, optimized for efficiency and scalability.

Use Case: Suitable for scenarios where computational efficiency and memory usage are priorities, such as real-time prediction systems.

6) **Neural Network**:

Definition: A deep learning model designed to capture complex patterns through multiple layers of interconnected neurons.

Use Case: Best used for large datasets with high complexity or when non-linear relationships between features are expected.

3.5.3 Training Process

A) Baseline Models Training:

The Logistic Regression and Decision Tree models were trained as baseline models to provide a performance benchmark.

Outputs:

- **Logistic Regression**: Achieved an accuracy of 68%, with precision and recall around 65% for the majority class, indicating limited capability in handling class imbalance.
- **Decision Tree**: Showed improved performance over Logistic Regression with an accuracy of 72%, but still struggled with class imbalance, particularly underperforming on the False Positive class.

B) Advanced Models Training:

- **Random Forest**: Configured with 100 trees, the model was trained using 5-fold cross-validation, achieving an average accuracy of 84%, with macro-F1 score at 80%. This model effectively handled the class imbalance, performing well across all classes.
- **XGBoost and LightGBM**: Hyperparameters such as learning rate, max depth, and the number of estimators were tuned using grid search. XGBoost achieved a macro-F1 score of 82%, while LightGBM achieved 81%.
- **Neural Network**: Trained with a learning rate of 0.01, and three hidden layers, achieving a

macro-F1 score of 79%. Although slightly lower than the tree-based models, it demonstrated potential for capturing complex patterns.

3.5.4 Insights from Outputs:

i) Best Performance: Random Forest and XGBoost models showed the best performance, with high macro-F1 scores and balanced precision and recall across all classes, making them ideal for handling the imbalance in the dataset.

ii) Model Complexity: The Neural Network model, while less effective than Random Forest, still performed robustly, suggesting that more extensive tuning and additional data could further improve its results.

3.6 Model Evaluation and Tuning

3.6.1 Objective

To evaluate the models on the validation set using metrics like macro-F1 score, precision, and recall, and to fine-tune the models for optimal performance.

3.6.2 Evaluation Metrics

- **Macro-F1 Score**: Assesses the model's performance across all classes, treating each class equally.
- **Precision and Recall**: Precision measures the accuracy of the model's positive predictions, while recall measures its ability to identify all relevant instances.

3.6.3 Tuning Process

- **Hyperparameter Tuning**: Conducted using grid search for Random Forest and XGBoost, focusing on parameters like n_estimators, max_depth, and learning_rate.

  Outputs:

  i) **Random Forest**: Optimal hyperparameters were n_estimators=200 and max_depth=10, achieving a macro-F1 score of 85% and precision/recall of 84% for the True Positive class.
  ii) **XGBoost**: Best settings included learning_rate=0.1 and max_depth=6, leading to a macro-F1 score of 83%.

- **Handling Class Imbalance**: Implemented class weighting in the loss function and oversampling minority classes, which improved recall for the False Positive class from 70% to 77%.

3.6.4 Insights from Outputs:

- **Performance Gains**: Tuning significantly improved the macro-F1 score and balanced the performance across all classes.
- **Class Imbalance**: Adjusting class weights was particularly effective in boosting recall for minority classes without sacrificing overall accuracy.

3.7 Model Interpretation

3.7.1 Objective

To interpret the model's predictions and understand which features are most influential in determining the incident grade.

3.7.2 Techniques Used

- **Feature Importance**: The importance of each feature was analyzed, particularly for tree-based models like Random Forest and XGBoost.
- **Error Values**: Used to understand the impact of individual features on model predictions.

3.7.3 Key Insights

Top Features: The most influential features were IncidentId, IpAddress, DetectorId, and AlertTitle. These features consistently appeared as top contributors in Random Forest and XGBoost models.

3.8 Final Evaluation on Test Set

3.8.1 Objective

To evaluate the performance of the final model on the unseen test set to ensure that it generalizes well to new data.

3.8.2 Process

- **Testing**: The final Random Forest model was evaluated on the test set, yielding high precision, recall, and macro-F1 scores.

  Outputs:

  i) **Macro-F1 Score**: 84% on the test set, indicating strong generalization.
  ii) **Precision and Recall**: Precision of 83% and recall of 82% for the True Positive class, demonstrating balanced performance across classes.