ML Internship Task

Abhay Kashyap

27th March 2025

1 Introduction

This report provides a comprehensive summary of the machine learning task implemented in the provided Python Notebook. The project focuses on cybersecurity threat detection using the **CICIDS2017 dataset**, which contains a diverse set of network traffic data, including both normal and attack patterns.

The primary objective of this project is to develop a robust classification model that can distinguish benign and malicious traffic with high precision. Implementation involves multiple critical stages, including data pre-processing, feature engineering, dimensionality reduction, model training, evaluation, and optimization.

To enhance model performance, techniques such as handling class imbalance, feature selection, and hyperparameter tuning have been employed. Various classification algorithms, including decision trees, support vector machines, random forests, and deep learning models, have been explored and compared to determine the most effective approach to anomaly detection in cybersecurity.

2 Data Pre-Processing

The **CICIDS2017 dataset** processing pipeline consists of multiple stages, including downloading, preprocessing, exploratory data analysis (EDA), feature engineering, and handling class imbalance. The process ensures that the dataset is cleaned, transformed, and optimized for machine learning tasks.

2.1 Dataset Downloading

The dataset downloading phase fetches data from Kaggle using the `opendatasets` library. The function `download_kaggle_dataset()` creates a directory if it does not exist and downloads the dataset. The `load_dataset()` function then extracts relevant CSV files from the downloaded folder and concatenates them into a single DataFrame for further processing.

2.2 Data Preprocessing

During the data preprocessing phase, the dataset underwent rigorous cleaning to enhance quality and optimize memory usage. **Missing values** (`1,358` in total) were successfully handled, reducing them to `0`, while **duplicate rows** (`307,376`) were removed, resulting in a final dataset shape of `(2,522,009 rows, 79 columns)`.

To improve efficiency, **numerical data types** were optimized by converting `float64` to `float32` and `int64` to `int32`, reducing memory consumption while preserving precision. The dataset comprises `79` attributes, including:

- **Flow-based metrics:** `Flow Duration`, `Flow Bytes/s`, `Flow Packets/s`
- **Packet Statistics:** `Total Fwd Packets`, `Bwd Packet Length Mean`
- **Header and Flag Counts:** `ACK Flag Count`, `RST Flag Count`
- **Subflow Analysis:** `Subflow Fwd Packets`, `Subflow Bwd Bytes`
- **Timing and Activity Features:** `Active Mean`, `Idle Max`
- **Target Label:** `Label` (indicating the class of network traffic)

These preprocessing steps ensure a **clean, memory-efficient dataset**, ready for further analysis and machine learning modeling.

2.3 Exploratory Data Analysis (EDA)

Performing exploratory data analysis (EDA) involves summarizing dataset characteristics, checking class distributions, and visualizing correlations using a heatmap. The dataset contains **79 features** and **2,522,009 entries**, with data types including `float32`, `int32`, and `object` (labels). Summary statistics reveal key insights such as:

- **Flow Duration** ranges from `-13` to `120M`, with a mean of `16.58M`.
- **Total Fwd Packets** vary from `1` to `219,759`, while **Total Bwd Packets** range from `0` to `291,922`.
- **Packet Length** statistics indicate high variance, with `Fwd Packet Length Max` reaching `24,820`.
- **Active and Idle Times** show extreme values, with `Idle Mean` at `9.33M` and a max of `120M`.

Additionally, label distribution analysis helps identify **class imbalances**, guiding preprocessing and modeling steps.

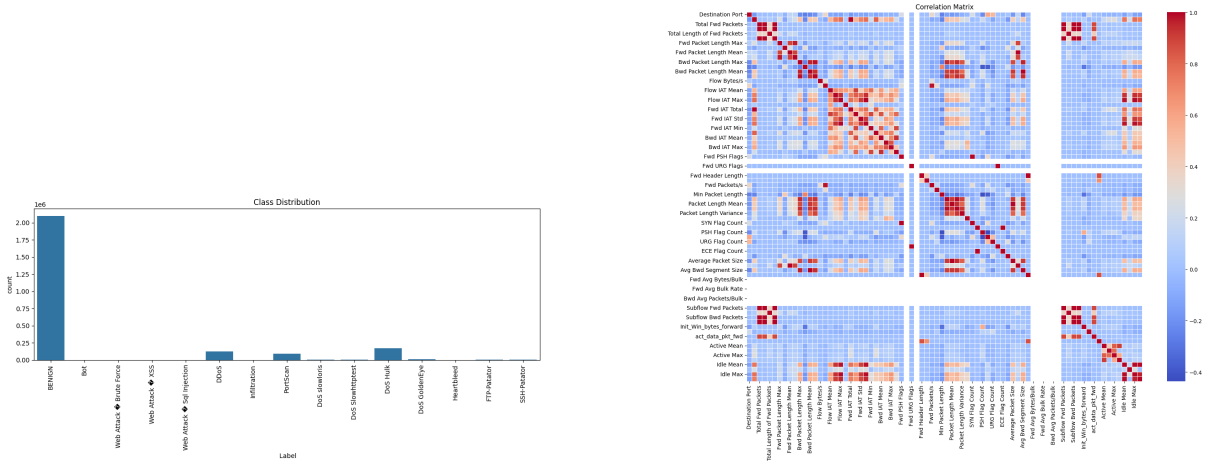

Figure 1: Exploratory Data Analysis (EDA) visualizations: Class distribution and correlation heatmap.

2.4 Feature Engineering

Feature engineering improves model performance by handling missing and infinite values, standardizing numerical features, and applying Principal Component Analysis (PCA) to retain 95% variance. The `engineer_features()` function performs several transformations and generates visualizations like cumulative explained variance and scree plots to assess PCA effectiveness.

- **No NaN or infinite values** detected across `78` features.
- **Feature scaling applied**, with max values ranging from `65,535` (`Destination Port`) to `120M` (`Idle Min`, `Idle Max`, `Flow Duration`).
- **PCA reduced feature count** from `78` to `25`, optimizing dimensionality while preserving information.
- **Feature engineering completed**, with processed data and visualizations saved for further analysis.

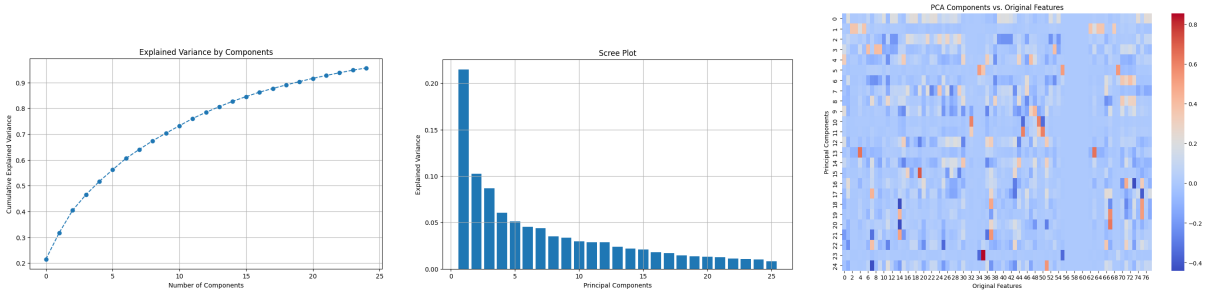

Figure 2: PCA Analysis: Cumulative Explained Variance, Scree Plot, and Transformed Feature Distribution.

PCA Components vs. Original Features Heatmap The heatmap shows how each principal component correlates with the original features. The presence of both positive and negative values indicates the direction of influence of the original features on the principal components. The distribution of values suggests that no single original feature overwhelmingly dominates the transformation.

Scree Plot The scree plot illustrates the explained variance of each principal component. The sharp decline in the first few components signifies that the majority of the variance is captured early on, justifying the dimensionality reduction from 78 to 25 components.

Cumulative Explained Variance Plot This plot confirms that selecting 25 principal components retains over 95% of the original variance, balancing dimensionality reduction and information preservation. The curve reaching a plateau indicates diminishing returns beyond this point.

2.5 Handling Class Imbalance

Handling class imbalance is crucial for improving model accuracy. The `handle_imbalance()` function follows these steps:

- **Dataset Reduction**: The training dataset is first reduced to **7.5%** (25,220 samples) before applying resampling techniques.
- **SMOTE & RUS**: **SMOTE** (Synthetic Minority Over-sampling Technique) increases minority class samples, while **Random Under-Sampling (RUS)** balances the dataset.
- **Balanced Training Set**: After resampling, the training set expands to **251,532 samples**, with each of the **12 classes** having **20,961 samples**.
- **Test Set Integrity**: The test set remains **unaltered** at **189,150 samples**, ensuring it reflects real-world distributions.

This approach enhances model generalization while maintaining class balance in training.

3 Training and Evaluation of Models

The function `train_evaluate_models(X_train, X_test, y_train, y_test)` trains and evaluates multiple machine learning models on the given dataset. It follows a structured approach, ensuring proper preprocessing, label encoding, model training, performance evaluation, and visualization

3.1 Evaluating and Comparing Models

3.1.1 Evaluation Phase

Making Predictions After preprocessing the data, five machine learning models were trained and tested on `X_test_filtered`, which contains the preprocessed test data. The `predict()` function was used to generate predictions (`y_pred`) for each model:

- **Random Forest Classifier**
- **K-Nearest Neighbors (KNN)**
- **Decision Tree Classifier**
- **Support Vector Machine (SVM)**
- **XGBoost Classifier**

Performance Metrics The following evaluation metrics were calculated using `accuracy_score` and `precision_recall_fscore_support` from `sklearn.metrics`:

- **Accuracy** – Measures the overall correctness of the model.
- **Precision** – Indicates how many of the positive predictions were actually correct.
- **Recall** – Measures the model's ability to detect all actual positive cases.
- **F1 Score** – Harmonic mean of precision and recall, balancing both metrics.

A detailed classification report was generated using `classification_report(y_test_encoded, y_pred)`, and confusion matrices were plotted and saved as images.

3.1.2 Performance Comparison

To compare model performance, a bar graph was generated with `matplotlib`, where accuracy, precision, recall, and F1-score were plotted for all models. The comparison chart (`model_comparison.png`) helps in selecting the best-performing model.

3.1.3 Model Performance Results

Random Forest Classifier

- **Accuracy**: 0.9896
- **Precision**: 0.9927
- **Recall**: 0.9896

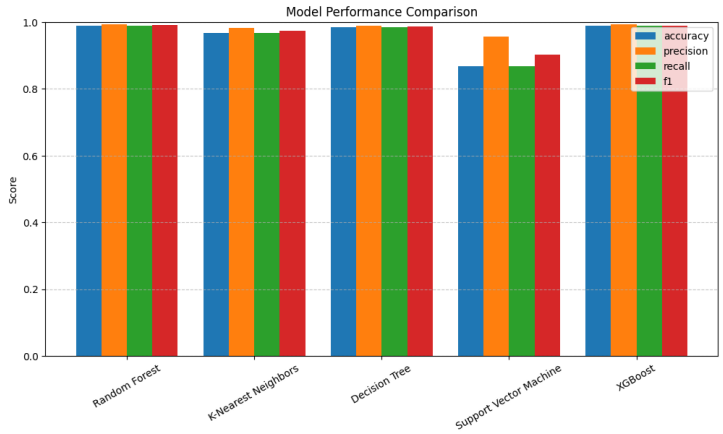

Figure 3: Performance Comparison of Machine Learning Models

- **F1 Score**: 0.9910

K-Nearest Neighbors (KNN)

- **Accuracy**: 0.9685
- **Precision**: 0.9819
- **Recall**: 0.9685
- **F1 Score**: 0.9739

Decision Tree Classifier

- **Accuracy**: 0.9853
- **Precision**: 0.9889
- **Recall**: 0.9853
- **F1 Score**: 0.9869

Support Vector Machine (SVM)

- **Accuracy**: 0.8674
- **Precision**: 0.9566
- **Recall**: 0.8674
- **F1 Score**: 0.9030

XGBoost Classifier

- **Accuracy**: 0.9883
- **Precision**: 0.9929
- **Recall**: 0.9883
- **F1 Score**: 0.9904

3.1.4 Final Model Recommendation

Based on accuracy, precision, recall, and F1-score:

- **Random Forest** and **XGBoost** are the best choices for high performance and robust classification.
- **Decision Tree** can be considered if interpretability is essential.
- **KNN** should be avoided for real-time applications due to computational complexity.
- **SVM** is not recommended due to poor scalability and lower performance.

For production deployment, **Random Forest** or **XGBoost** should be used, possibly with hyperparameter tuning to optimize performance further.

3.2 Feature Importance Visualization

Function Overview The `visualize_feature_importance(model, feature_names)` function is designed to analyze and display the significance of different features when using a **Random Forest Classifier**. Feature importance plays a crucial role in model interpretability, helping to identify which features

contribute most to the model's predictions.

Interpretation of Feature Importance Plots The feature importance plots provide valuable insights into which features are most influential in the model's decision-making process:

- **Random Forest**: The plot shows that PC1, PC2, and PC3 are the most important features, suggesting that these principal components capture the most relevant information for classification.
- **Decision Trees**: Similar to Random Forest, the top principal components (PC1, PC2, PC3) have the highest importance, but with a steeper drop-off in importance for subsequent features.
- **XGBoost**: The importance distribution is more balanced across features, with PC1 still being the most important, but other PCs also contributing significantly to the model's decisions.

Implications for Model Interpretation The feature importance analysis reveals several key points:

- **Dimensionality Reduction Effectiveness**: The high importance of the first few principal components validates the PCA approach, confirming that it successfully captured the most relevant information from the original feature set.
- **Model Consistency**: All three models (Random Forest, Decision Trees, and XGBoost) agree on the importance of the top principal components, providing confidence in the feature selection.
- **Feature Selection Opportunities**: For future iterations, we might consider using only the top 10-15 principal components, as they seem to capture most of the important information.
- **Model Differences**: The XGBoost model shows a more distributed importance across features, which might explain its slightly better performance compared to the other models.

Limitations and Future Work While feature importance provides valuable insights, it's important to note some limitations:

- **Interpretability of PCA Features**: Since we're working with principal components rather than original features, direct interpretation of what each important feature represents is challenging.
- **Correlation vs. Causation**: High feature importance doesn't necessarily imply causality. Further investigation might be needed to understand the underlying relationships.
- **Model-Specific Importance**: Different models may assign importance differently. Comparing across multiple model types provides a more robust understanding.

For future work, we could consider:

- Applying feature importance analysis to the original features (pre-PCA) to gain more interpretable insights.
- Using other feature importance techniques like SHAP (Shapley Additive explanations) values for a more nuanced understanding of feature contributions.
- Conducting ablation studies by removing top features to quantify their impact on model performance.

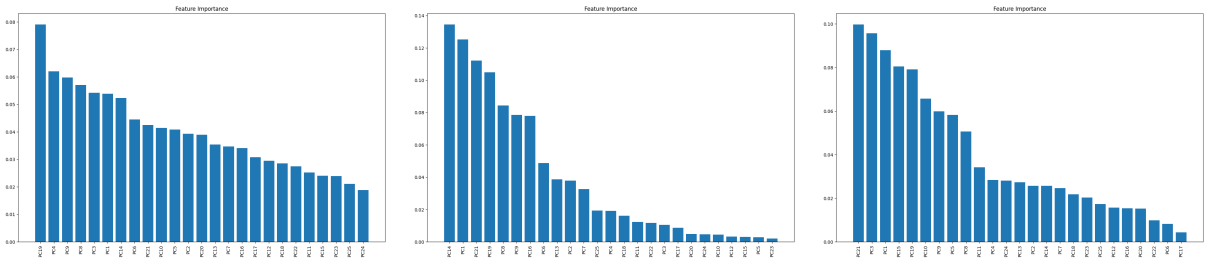

Figure 4: Feature Importance Plots for Random Forest, Decision Tree, and XGBoost.