Static Malware Classifier API

This project explores feature engineering, model training, deployment and evaluation of a machine learning model for malware detection. A random forest classifier, trained on Portable Executable (PE) file features, is deployed as an API endpoint on AWS SageMaker. A Python client is then developed to interact with the deployed model. The client extracts relevant features from executable files and retrieves classification results from the SageMaker endpoint. Finally, the model's performance is benchmarked on a randomly selected dataset of 100 malware and 100 benign samples from EMBER 2018.

Introduction

The ever-evolving landscape of malware adversarial attacks necessitates the development of robust and efficient detection methods. Machine learning offers a promising approach for identifying malicious software based on extracted features of PE files.

The project is divided into three primary stages:

1. **Model Training and Deployment:** The random forest classifier, trained to identify malware based on PE file features, is deployed as an API endpoint on AWS SageMaker. SageMaker simplifies the process of building, training, and deploying machine learning models.
2. **Client Development:** A Python client is developed to utilize the deployed model. This client takes an executable file as input, extracts relevant features from the PE file, and sends those features to the SageMaker endpoint for classification. The client then receives and displays the model's prediction on whether the file is malicious or benign.
3. **Performance Benchmarking:** To evaluate the model's effectiveness, a random selection of 100 malware and 100 benign samples is drawn from the EMBER 2018 dataset. The model's performance is then benchmarked on this dataset, providing insights into its accuracy and generalizability on an out-of-distribution dataset.

By deploying the model as an API endpoint and developing a user-friendly client, this project demonstrates the practical application of machine learning for malware detection. Additionally, benchmarking the model's performance on a representative dataset provides valuable information for further development and refinement.

Methodology

Feature Extraction

This project employed a multi-faceted approach to extract relevant features from PE files:

- **Byte Sequence n-grams:** The code extracted consecutive sequences of n bytes (n-grams) from the binary content of each file, capturing patterns in the raw byte sequences.
- **Imported DLLs:** The names of dynamically linked libraries (DLLs) imported by each file were extracted and preprocessed for feature representation.
- **Section Names:** The names of sections within the PE files were extracted and processed to reveal potential structural patterns. Number of sections in a PE file is also extracted as a feature.

Feature Representation

- **N-gram Features:** The code used a K-most-common-features approach, retaining the K most frequently occurring n-gram counts across the dataset for feature representation.
- **Textual Features:** For imported DLL names and section names, the code employed a natural language processing approach.
    - **Hashing Vectorizer:** This technique transformed text data into fixed-length numerical vectors, enabling efficient representation of varying-length text features.
    - **TF-IDF:** This technique was applied to weigh the importance of words within the corpus, reducing the influence of common words and highlighting those with higher discriminative power.

Model Training:

- **Classifier:** The project used a random forest classifier for malware detection. Random forest models construct an ensemble of decision trees, effectively handling non-linear relationships and reducing overfitting.
- **Training Data:** Features extracted from a PE file dataset were used to train the model.

Model Deployment:

Model Preparation

- The trained random forest model was packaged into a TAR.GZ archive, including model file, and entrypoint script and dependencies for deployment.
- This model archive was uploaded to an S3 bucket *s3://simplified-midterm-03/* for access by SageMaker.

SageMaker Deployment

- The AWS SageMaker Python SDK facilitated model deployment:

- **Role Assumption:** The `get_execution_role` function acquired the necessary permissions for model deployment.
- **Model Configuration:** A `SKLearnModel` object was created, specifying:
  - Model data location in S3
  - Execution role
  - Entry point script for model execution
  - Source directory containing additional model code
  - Python version
  - Scikit-learn framework version
  - Required dependencies
- **Endpoint Creation:** The deploy method initiated model deployment, establishing a REST API endpoint:
  - Instance type: "ml.t2.medium"
  - Initial instance count: 1
  - **Serialization and Deserialization:** JSONSerializer and JSONDeserializer were specified for request and response formatting.

Client Interaction

- A Python script acted as a client for model interaction:
  - **Test Data:** Sample test data (a sparse matrix) was used for prediction.
  - **JSON Payload:** The data was structured into a JSON payload for endpoint transmission:
    - Data: Converted to a list of values
    - Row and column indices for the sparse matrix.
    - Matrix shape
  - **Endpoint Configuration:** The predictor object's `accept` and `content_type` properties were set to "application/json" for JSON communication.
  - **Prediction:** The `predict` method sent the JSON payload to the endpoint and retrieved the model's prediction.

Client Development

- **Python Client:** A Python client application was developed to:
  - Accept a new PE file as input
  - Extract the relevant features using the same techniques as in model training
  - Send the extracted features to the deployed model endpoint on SageMaker
  - Receive and display the model's prediction on file maliciousness.

Performance Benchmarking

- **Dataset:** 100 malware and 100 benign samples were randomly selected from the EMBER 2018 dataset for evaluating model performance.

- **Metrics:** The model's performance was assessed using accuracy, precision, recall, F1-score, and potentially other relevant metrics.
- **Benchmarking:** The model's performance was compared against in-distribution benchmarks for malware detection.

Note: In order to benchmark the performance of the Random Forest classifier it needed to be retrained on unigrams of byte sequences of PE files. It is because the PE files for EMBER dataset are not available but byte frequency counts are available which can be repurposed for this model.

Performance Evaluation and Benchmarking

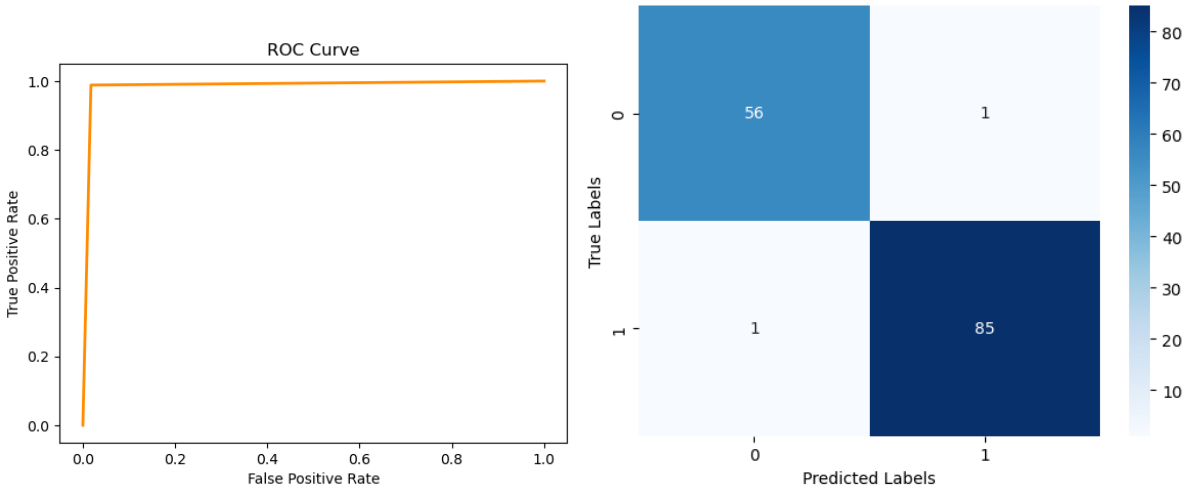
This section evaluates the model's performance on both in-distribution and out-of-distribution (OOD) data. We compare the model's accuracy, precision, recall, F1 score, and AUC-ROC score for each data type. Additionally, we analyze confusion matrices to gain further insight into classification patterns.

In-distribution Data:

The model achieved strong performance on in-distribution data, demonstrating its ability to learn the underlying patterns effectively. Here's a breakdown of the metrics:

- **Accuracy:** 0.9860 (very high, indicating a high proportion of correct predictions)
- **Precision:** 0.9884 (very high, signifying the model rarely makes false positive errors)
- **Recall:** 0.9884 (very high, implying the model effectively identifies most relevant cases)
- **F1 Score:** 0.9884 (very high, reflecting the balance between precision and recall)
- **AUC-ROC Score:** 0.9854 (very high, suggesting excellent ability to distinguish between positive and negative cases)

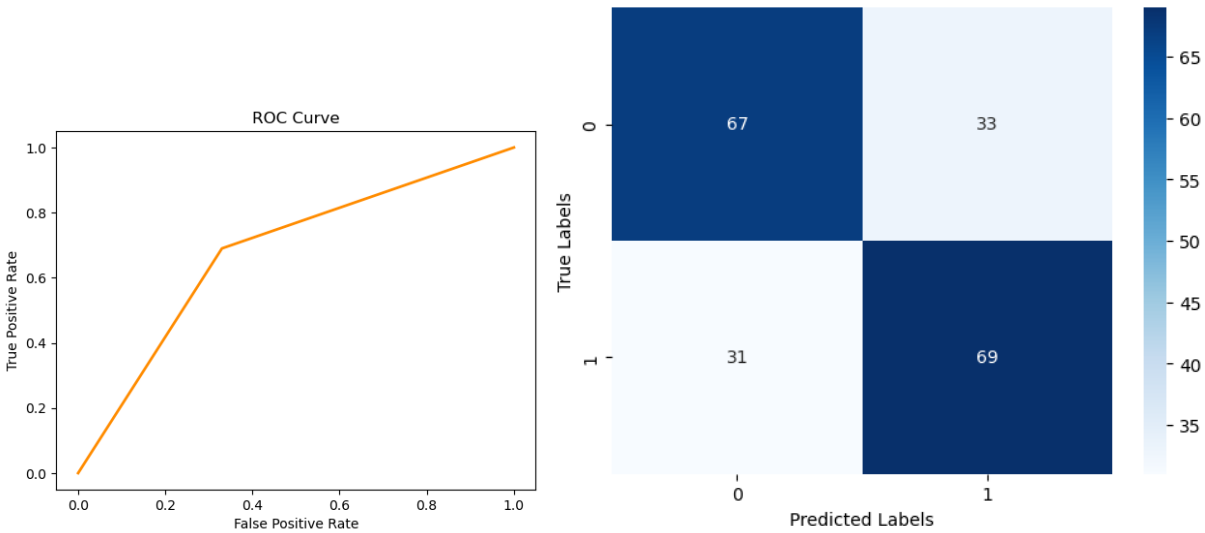
The high values across all metrics and the ROC curve (assuming it visually confirms a curve closer to the top-left corner) indicate the model is well-suited for tasks involving in-distribution data.



Out-of-distribution Data:

The model's performance on OOD data differed significantly from its performance on in-distribution data. This highlights the importance of considering OOD scenarios during model evaluation. Here's a breakdown of the OOD metrics:

- **Accuracy:** 0.6800 (moderate, suggesting the model makes more incorrect predictions on OOD data)
- **Precision:** 0.6765 (moderate, indicating the model struggles with some false positive errors)
- **Recall:** 0.6900 (moderate, implying the model misses some relevant cases in OOD data)
- **F1 Score:** 0.6832 (moderate, reflecting the imbalance between precision and recall)
- **AUC-ROC Score:** 0.6800 (moderate, suggesting the model has difficulty separating positive and negative cases in OOD data)



The confusion matrix for OOD data visually confirms the challenges. It likely shows a higher number of off-diagonal cells compared to the in-distribution matrix, signifying a large number of misclassifications on OOD data points.

Conclusion:

The evaluation demonstrates the model's effectiveness for in-distribution data. However, its performance degrades significantly on OOD data. This highlights the need for further exploration of OOD detection and handling techniques to improve the model's robustness in real-world scenarios where encountering unseen data is a possibility.