Malware Classification Project Description

Project Overview

Objective: Develop a robust machine learning pipeline for malware classification using feature selection techniques to enhance accuracy and efficiency, compare it with a baseline model, containerize the solution, and deploy it on the cloud with clustering and monitoring.

Steps and Technical Specifications

1. Dataset Preparation

Download the dataset from the provided link in your feature selection lab. Initial Data Processing: Clean and preprocess the data, ensuring it's suitable for feature selection and model training. (If you don't use the whole dataset, you will sacrifice half of the project grade).

2. Feature Selection

Algorithm: Implement a feature selection algorithm (e.g., Recursive Feature Elimination, Feature Importance, or Principal Component Analysis) to reduce dimensionality and improve model performance. Documentation: Describe the chosen method, the rationale for its choice, and its impact on the dataset.

3. Model Development

Machine/Deep Learning Models: Train models (such as Random Forest, XGBoost, or a Neural Network) to classify the malware samples. Comparison: Ensure the model accuracy is at least comparable to the 92% benchmark set by the vanilla approach.

4. Dockerization

Containerization: Package the entire machine learning pipeline into a Docker container, ensuring all dependencies are included for seamless execution. Dockerfile: Create a Dockerfile that specifies the environment, dependencies, and commands to run the application.

5. API Development

Post Request API: Develop an API that can receive POST requests with malware data and return classification results. Endpoints Documentation: Clearly document the API endpoints, including expected inputs and outputs.

6. Cloud Deployment and Clustering

Cloud Service: Choose a cloud service provider (e.g., AWS, Google Cloud, Azure) for deployment. Cluster Management: Utilize Kubernetes or a similar service for managing container deployments, ensuring scalability and efficient resource management. Load Balancer: Implement a load balancer to distribute user requests evenly across pods.

7. Monitoring

Tools: Integrate monitoring tools to track the performance and health of your application in real time. Metrics: Monitor key metrics like response times, system throughput, error rates, and resource usage.

Documentation and Reporting

Provide a detailed report that covers the methodology, results, and comparisons with the baseline model. Include a README file with step-by-step instructions on setting up and running the pipeline, using the API, and deploying the Docker container. Document the API usage thoroughly with examples of requests and responses.

Submission Format

Submit a PDF report, the source code in a compressed file (including Dockerfiles and configuration files for deployment and monitoring), and a README for operational guidance.