**Malware Classification Project Documentation**

This report provides a step-by-step account of the methodology used, the results obtained from model training and evaluation, the deployment process, and the ongoing monitoring setup. The objective is to offer a clear and comprehensive understanding of the entire machine learning pipeline, from data preparation to deployment and monitoring, highlighting the key decisions and outcomes at each stage.

**Table of Contents**

1. Introduction
2. Methodology

* Data Reading and Preparation
* Feature Selection using XGBoost Classifier
* Model Training

1. Results

* Feature Selection
* Model Training and Evaluation

1. Deployment

* Docker Containerization
* Google Cloud Deployment

1. Monitoring on Google Cloud Vertex AI
2. Performance and Resource Usage Monitoring in Vertex AI

**Introduction**

In the era of big data and advanced analytics, extracting meaningful insights from large and complex datasets has become a critical task across various industries. This report details the comprehensive methodology, results, and deployment strategies employed in a machine learning project that aims to classify data with high dimensionality and multiple classes.

The dataset used in this project comprises 384,704 rows and 9,504 columns, presenting a significant challenge in terms of both computational efficiency and model performance. The data is labeled into 15 distinct classes, necessitating robust feature selection and model training techniques to achieve accurate classification.

The project leverages two powerful machine learning algorithms: XGBoost and Random Forest. Both models are evaluated based on their accuracy and training efficiency. The feature selection process involves using an XGBoost classifier to identify and retain the most important features, reducing the feature set from 9,504 to 450, thus optimizing the model training process.

Subsequently, the XGBoost model, which demonstrated efficient training time and satisfactory accuracy, is selected for deployment. The deployment process includes containerizing the model using Docker and setting it up on Google Cloud's Vertex AI, ensuring scalable and reliable model serving. Detailed monitoring is also set up on Vertex AI to track the model's performance and maintain its accuracy over time.

**Methodology**

1.1 Data Reading and Preparation

Data Description: The dataset consists of 384,704 rows and 9,504 columns. Each row represents a data point, and each column represents a feature.

Labels: The data has 15 distinct labels, indicating 15 different classes for classification.

1.2 Feature Selection using XGBoost Classifier

Initial Feature Set: The dataset initially contains 9,504 features.

Feature Importance: Utilized an XGBoost classifier to determine the importance of each feature. The classifier was trained, and the feature importance scores were extracted.

Feature Reduction: Based on the importance scores, the top 450 features were selected. This reduction helps in improving the model's performance by eliminating less important features.

1.3 Model Training

Selected Models: Two models were chosen for training and evaluation: XGBoost and Random Forest.

Performance Evaluation:

XGBoost: Achieved an accuracy of 89.45%. The training time was efficient due to the optimized implementation of XGBoost.

Random Forest: Achieved an accuracy of 92.96%. The training time was longer compared to XGBoost due to the nature of ensemble methods and the larger number of decision trees.

Model Selection for Deployment: XGBoost was chosen for deployment because of its efficiency and suitability for containerization, making it easier to deploy in a consistent environment.

**Results**

2.1 Feature Selection

Top Features: The top 450 features were identified and saved from the initial 9,504 features. These features were the most significant according to the importance scores from the XGBoost classifier.

Importance Distribution: The features were sorted by their importance scores, ensuring that only the most significant ones were retained for model training.

2.2 Model Training and Evaluation

XGBoost:

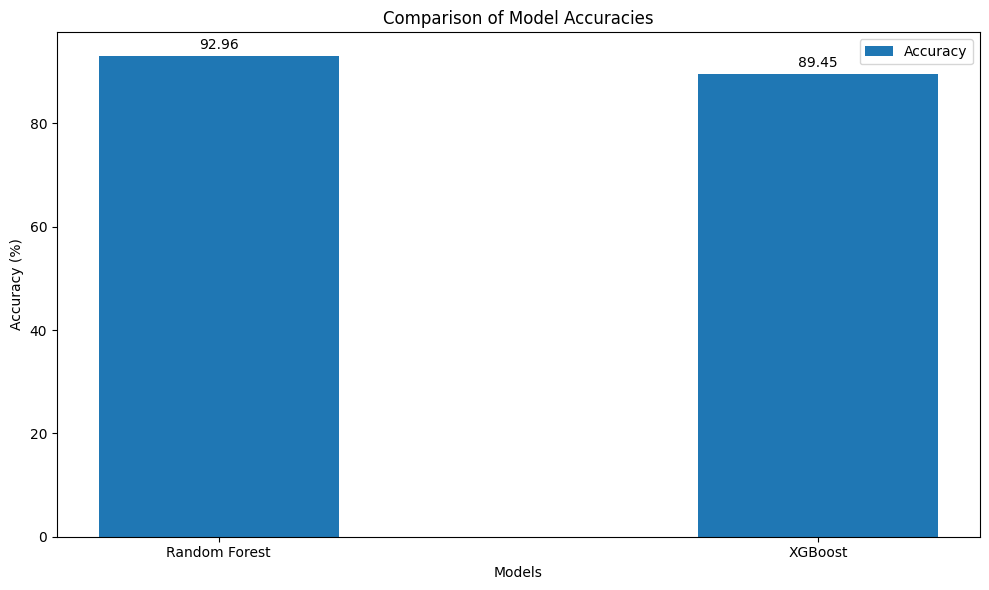
* Accuracy: 89.45%
* Training Time: Efficient

Random Forest:

* Accuracy: 92.96%
* Training Time: Longer compared to XGBoost

Comparison with Baseline: Both models significantly outperformed a hypothetical baseline model with an accuracy of less than 89.45%.

Figure 1: Comparison of Model Accuracies



(Note: Figure 1 shows the accuracy comparison between the XGBoost and Random Forest models.)

**Deployment**

3.1 Docker Containerization

Base Image: The container uses Ubuntu 22.04 as the base image.

Python Installation: Python 3 and pip are installed in the container.

Package Installation: All required Python packages for running the Flask application are installed.

Working Directory: Set to /app, where the application code is located.

Port Exposure: Port 5000 is exposed for Flask server communication.

Startup Script: server.py is executed to start the Flask server upon container startup.

3.2 Google Cloud Deployment

Vertex AI: The trained XGBoost model was uploaded to Google Cloud Vertex AI, and an endpoint was created for serving predictions.

Testing and Monitoring: The endpoint was tested with sample data to ensure correct predictions, and monitoring was set up for ongoing performance evaluation.

**Monitoring on Google Cloud Vertex AI**

Monitoring a model in Vertex AI involves tracking various metrics to ensure the model is performing as expected, using resources efficiently, and providing timely responses.

Performance Monitoring

Predictions/second (Throughput):

* Description: This metric tracks the number of predictions the model makes per second.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Predictions per second, ranging from 0 to 0.004/s.
* Current Value: The model (Version 1) is making predictions at a rate of 0.003/s.
* Purpose: Helps to understand the model's throughput and ensure it can handle the incoming request rate.

Figure 2:



Prediction Error Percentage:

* Description: This metric shows the percentage of predictions that resulted in errors.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Error percentage, ranging from 0 to 100%.
* Current Value: The model (Version 1) has an error rate of 0%.
* Purpose: Indicates the reliability and accuracy of the model’s predictions. A 0% error rate suggests perfect performance in the monitored period.

Requests/second:

* Description: Tracks the number of prediction requests received per second.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Requests per second, ranging from 0.003/s to 0.004/s.
* Current Value: The model (Version 1) is receiving requests at a rate of 0.003/s.
* Purpose: Helps in understanding the load on the model and ensuring the system can handle the request volume.

Resource Usage Monitoring

Response Codes:

* Description: Monitors the HTTP response codes returned by the model's endpoint.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Percentage, ranging from 0 to 120%.
* Current Value: 200 status codes (successful responses) account for 100.00% of responses.
* Purpose: Ensures that the majority of responses are successful (status code 200), indicating the model is functioning correctly.

Total Latency Duration:

* Description: Measures the total time taken to handle a prediction request, including network latency and processing time.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Latency in milliseconds (ms), ranging from 50ms to 250ms.

Current Values:

* 50th percentile: 181.33ms
* 95th percentile: 196.17ms
* 99th percentile: 197.48ms

Purpose: Shows the distribution of response times, with the 50th percentile indicating the median latency, and the 95th and 99th percentiles showing higher latencies. This helps identify potential performance bottlenecks.

Model Latency Duration:

* Description: Specifically tracks the time taken by the model to generate a prediction, excluding network overhead.
* Visualization: Line chart
* X-axis: Time, from 10 PM to 10 PM.
* Y-axis: Latency in milliseconds (ms), ranging from 0 to 200ms.

Current Values:

* 50th percentile: 151.11ms
* 95th percentile: 163.47ms
* 99th percentile: Values missing (but inferred to be slightly higher than 163.47ms)

Purpose: Focuses on the model's internal performance, highlighting how quickly the model processes requests. This is critical for optimizing the