**Digital Egypt Pioneers Initiative DEPI**

**National Telecommunication Institute NT**

# Microsoft Malware Prediction

**Graduation Project presented to Digital Egypt Pioneers Initiative & National Telecommunication Institute in Track AI-IBM Data Scientist**

**BY**

**Hossam Mohamed**

**Khaled Mousa Ali**

**Abduallah Hassan**

**Hamdi Rashad**

**Supervise by**

**ENG: Mohamed Ibrahim**

Data: 4 oct 2024

## Introduction

Cybersecurity threats have grown increasingly complex, especially with the rise of organized, well-funded criminal enterprises that aim to exploit vulnerabilities in personal and enterprise devices. Among these threats, malware plays a significant role in compromising system integrity, leading to data theft, ransomware attacks, and system disruptions.

Microsoft, with over a billion enterprise and consumer customers globally, takes the malware threat very seriously. To tackle this, the company has invested heavily in improving its security features and defenses. Part of this effort is the Microsoft Malware Prediction project, which leverages machine learning to predict the likelihood of malware infections on Windows machines. The ability to predict whether a machine will soon be hit with malware can significantly reduce the damage malware can cause by enabling timely preventive actions.

This document outlines the Microsoft Malware Prediction challenge, which is based on a Kaggle competition hosted by Microsoft, Windows Defender ATP Research, and academic partners from Northeastern University and Georgia Tech. The goal is to predict whether a Windows machine will be hit with malware using telemetry data collected by Microsoft Defender.

The competition provided a large and complex dataset that encompasses various features of a machine's configuration, operating system details, and antivirus states. Competitors are tasked with building a machine learning model that can predict malware occurrences with a high degree of accuracy.

## Problem Definition

The core problem that this project aims to solve is to predict whether a Windows machine will be infected with malware in the near future. The dataset provided by Microsoft for this competition includes several key system features and details about each machine. The target variable for this prediction problem is HasDetections, which indicates whether malware has been detected on the machine.

Malware detection is a time-sensitive problem as malware evolves rapidly, and infected machines can cause significant harm to both individual users and enterprises. Developing an effective machine learning model that predicts malware occurrences can help Microsoft enhance its security protocols and protect its vast user base.

**Key Challenges**

The challenges in this project include:

- The **vast size** of the dataset, which includes millions of rows and numerous features.
- **Imbalanced data**, as malware detection is relatively rare compared to the total number of machine instances.
- **Time-series aspects** of the problem, as the timing of malware infections and system configurations changes frequently due to system patches, updates, and new machine introductions.
- Ensuring that the model generalizes well despite the **non-representative nature** of the dataset, which was constructed specifically for the competition and includes a higher proportion of infected machines than would normally be seen in the wild.

**Business Impact**

Malware can cause significant harm, both financially and operationally. Predicting whether a machine will be infected with malware can help organizations and individual users take preventive measures to reduce risks and mitigate the damage caused by cyber-attacks. Early detection and intervention are critical in reducing the cost and disruption of malware infections.

For Microsoft, improving malware prediction helps in refining its Windows Defender solutions, strengthening its reputation as a leader in cybersecurity, and enhancing the security of its vast ecosystem of users.

**Dataset Description**

**1. Dataset Overview**

The dataset provided for this competition contains telemetry data collected by Microsoft Defender from millions of machines. Each row in the dataset corresponds to a machine, uniquely identified by a MachineIdentifier. The dataset is split into two parts:

- **train.csv**: This file contains labeled data, meaning each machine's HasDetections column is filled with either 1 (malware detected) or 0 (no malware detected).
- **test.csv**: This file contains unlabeled data, and competitors must predict the value for HasDetections for each machine in this file.

**2. Features**

The dataset includes multiple features that describe the machine's configuration, operating system details, and other properties. A detailed description of some key features is as follows:

- **MachineIdentifier**: Unique identifier for each machine.
- **ProductName**: Information about the machine's antivirus software, such as Windows Defender version (e.g., win8defender).
- **EngineVersion**: Version of the machine's antivirus engine.
- **AppVersion**: Version of the antivirus app installed on the machine.
- **AvSigVersion**: Version of the antivirus signature installed.
- **IsBeta**: Indicates whether the antivirus software is a beta version.
- **IsSxsPassiveMode**: Indicates whether the machine is in passive mode (not actively scanning).
- **HasTpm**: Indicates whether the machine has a Trusted Platform Module (TPM), a hardware feature that provides encryption and security.
- **CountryIdentifier**: An identifier for the country where the machine is located.
- **CityIdentifier**: An identifier for the city where the machine is located.

- **Platform**: The platform on which the machine operates (e.g., Windows 10, Windows 7).
- **Processor**: The processor architecture (e.g., AMD64, x86).
- **OsVersion**: The version of the operating system.
- **OsBuild**: The build number of the operating system.
- **OsSuite**: The product suite mask for the operating system.
- **OsPlatformSubRelease**: Sub-release of the operating system (e.g., Windows 7, Windows 10).
- **SkuEdition**: Edition of the operating system (e.g., Professional, Home).
- **Census_DeviceFamily**: The type of device the OS is intended for (e.g., desktop, mobile).
- **Census_ProcessorCoreCount**: Number of cores in the machine's processor.
- **Census_PrimaryDiskTotalCapacity**: Total disk space of the machine in MB.
- **Census_PrimaryDiskTypeName**: Type of primary disk drive (e.g., SSD, HDD).
- **Census_TotalPhysicalRAM**: The amount of RAM in MB.
- **Census_InternalPrimaryDiagonalDisplaySizeInInches**: Diagonal screen size of the machine's primary display.
- **Census_InternalPrimaryDisplayResolutionHorizontal**: Horizontal resolution of the machine's primary display.
- **Census_InternalPrimaryDisplayResolutionVertical**: Vertical resolution of the machine's primary display.
- **Census_IsPortableOperatingSystem**: Indicates if the operating system is portable (e.g., running from a USB drive).
- **Census_IsTouchEnabled**: Indicates if the machine supports touch input.
- **Census_IsPenCapable**: Indicates if the machine supports pen input.

- **Census_IsAlwaysOnAlwaysConnectedCapable**: Indicates if the device supports “Always On” and “Always Connected” features (e.g., modern standby capabilities).

These are only a subset of the 80+ features available in the dataset. Each feature plays a different role in describing the machine and its susceptibility to malware.

## 3. Data Cleaning and Preprocessing

Before building the machine learning model, the data must undergo several preprocessing steps to ensure that the features are clean and ready for analysis. These steps include:

1. **Handling Missing Values**: Several features in the dataset contain missing values (NA). These need to be addressed using techniques like imputation (filling missing values with the median or mode) or dropping columns with too many missing values.
2. **Encoding Categorical Variables**: Many features in the dataset are categorical (e.g., ProductName, Processor). These must be converted into numerical formats using techniques such as one-hot encoding.
3. **Feature Scaling**: Some features, such as Census_TotalPhysicalRAM and Census_PrimaryDiskTotalCapacity, have wide ranges. These numerical features may require normalization or standardization.
4. **Time-Series Considerations**: Although the dataset is not explicitly organized as time-series data, it’s important to recognize the temporal nature of malware detection, especially in cases where machine updates and patches play a role.

## 4. Dataset Size

The full dataset is quite large, totaling **8.47 GB** in size. It is divided into three files:

- **train.csv**: Contains the labeled training data (several million rows).
- **test.csv**: Contains the unlabeled test data.
- **sample_submission.csv**: A sample submission file showing the required format for predictions.

- **Project Workflow**

## 1. Environment Setup

- The first step in any data science project is setting up the environment. This includes installing necessary libraries and packages that will aid in data manipulation, visualization, and machine learning.
- Key libraries used in this project include:
- Pandas: For data manipulation and analysis. It provides data structures and functions to handle structured data efficiently.
- NumPy: For numerical computations, particularly useful for handling arrays and matrices.
- Matplotlib and Seaborn: For data visualization, allowing the creation of various plots and charts to visualize trends and relationships within the data.
- Scikit-learn: This library provides a variety of tools for model training, evaluation, and data preprocessing.
- Once the environment is set up, the next step is to load the dataset into the workspace for further processing.

## 1. Loading the Dataset

- Loading the dataset into memory is accomplished using Pandas. The dataset is read from CSV files and stored in DataFrames, which are powerful data structures that facilitate easy data manipulation.
- Upon loading the data, it is essential to explore its structure. This exploration typically includes:
- Displaying the first few rows of the dataset to understand its layout and content.
- Checking the data types of each column to ensure they are appropriate for analysis.
- Determining the shape of the dataset to know the number of records and features available.

## 1. Data Cleaning and Preprocessing

- Data cleaning is a crucial step in the workflow, as real-world datasets often contain inconsistencies, missing values, or irrelevant features. The goal is to ensure the dataset is in an optimal state for analysis and model training.
- Handling Missing Values
- One of the first tasks in data cleaning is checking for missing values across all features. Depending on the extent and nature of the missing values, various strategies can be employed:
- Dropping columns: If a feature contains a significant amount of missing data, it may be prudent to drop it from the dataset to avoid introducing noise into the model.
- Imputation: For features with moderate missing values, imputation techniques are used to fill in the gaps. Common methods include replacing missing entries with the median or mean value of the feature, which helps maintain the integrity of the dataset.
- Encoding Categorical Variables
    - Many machine learning algorithms require numerical input; thus, categorical variables must be transformed into a suitable format. This is typically achieved through:
- One-Hot Encoding: This technique involves creating binary columns for each category of the categorical variable, allowing the model to interpret these variables as separate features.
- After these transformations, the dataset will be primed for the next stages in the analysis.

## 1. Exploratory Data Analysis (EDA)

- Exploratory Data Analysis is conducted to uncover patterns, trends, and insights within the dataset. This step is vital for understanding the relationships between features and their impact on the target variable.
- Visualizing Target Variable Distribution
- The distribution of the target variable (HasDetections) is examined to assess class balance. In cases of class imbalance, techniques such as stratified sampling can be considered to ensure that both classes are adequately represented in the training set.

- **Analyzing Feature Correlations**
- Creating a correlation matrix is another critical aspect of EDA. It helps visualize the relationships between different features and can reveal important insights:

- Identifying Strong Correlations: Features that exhibit strong correlations can provide valuable information for the model. However, if two features are highly correlated, one may be redundant, and it might be beneficial to remove one to simplify the model.

    1. Splitting the Data
- With the data cleaned and analyzed, the next step is to prepare it for model training. The dataset is split into features (independent variables) and the target variable (dependent variable), which indicates whether malware was detected.
- Creating Training and Validation Sets
- Splitting the dataset into training and validation sets is crucial for assessing the model's performance. This split allows the model to be trained on one portion of the data and validated on another, which is essential for evaluating its ability to generalize to unseen data.
- Stratified sampling is often employed to ensure that the distribution of the target variable is maintained in both sets, especially when dealing with imbalanced classes.
    1. Model Training
- Once the data is prepared, it is time to train the machine learning model. In this project, a Random Forest Classifier is chosen due to its robustness and ability to handle complex datasets effectively.
- Initializing the Model
- The Random Forest Classifier is initialized with a specified number of trees (estimators). The number of trees can significantly impact the model's performance. A higher number of trees generally leads to better performance but also increases computational cost.
- Training the Model
- The model is trained on the training dataset, allowing it to learn patterns and relationships between the features and the target variable. During this

process, the algorithm constructs multiple decision trees based on random subsets of the data, enhancing its predictive capabilities.

1. **Generating Predictions**

- Once the model is trained, it is applied to the validation dataset to generate predictions. These predictions indicate the likelihood of malware detection for each machine based on its features.
- Obtaining Probability Estimates
- In addition to binary predictions, obtaining probability estimates for the predictions provides more nuanced insights. These probabilities can be used to assess the model’s confidence in its predictions, which can be crucial for risk assessment.

1. **Model Evaluation**

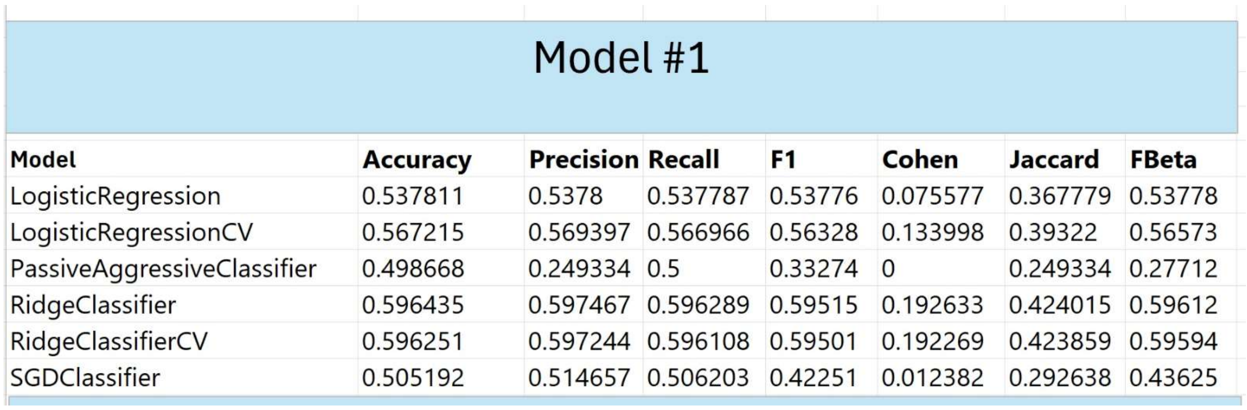
- Evaluating the model's performance is critical to understanding its effectiveness. This evaluation involves several metrics that provide insights into how well the model is performing.
- Metrics Used for Evaluation
- ROC-AUC Score: The Receiver Operating Characteristic (ROC) curve illustrates the trade-off between true positive rate and false positive rate at various threshold settings. The AUC (Area Under the Curve) quantifies the model’s ability to distinguish between classes. A higher ROC-AUC score indicates better model performance.
- Confusion Matrix: This visual representation shows the counts of true positive, true negative, false positive, and false negative predictions. Analyzing the confusion matrix helps identify areas where the model is performing well and where it is making mistakes.
- Classification Report: This report provides a detailed summary of precision, recall, and F1-score for each class. These metrics are crucial for evaluating model performance, especially in cases with class imbalance.

1. **Conclusion and Future Work**

- The project concludes with an assessment of the model’s performance and potential areas for improvement. While the Random Forest Classifier provides a solid baseline, exploring other models (e.g., gradient boosting machines or neural networks) could yield better results.

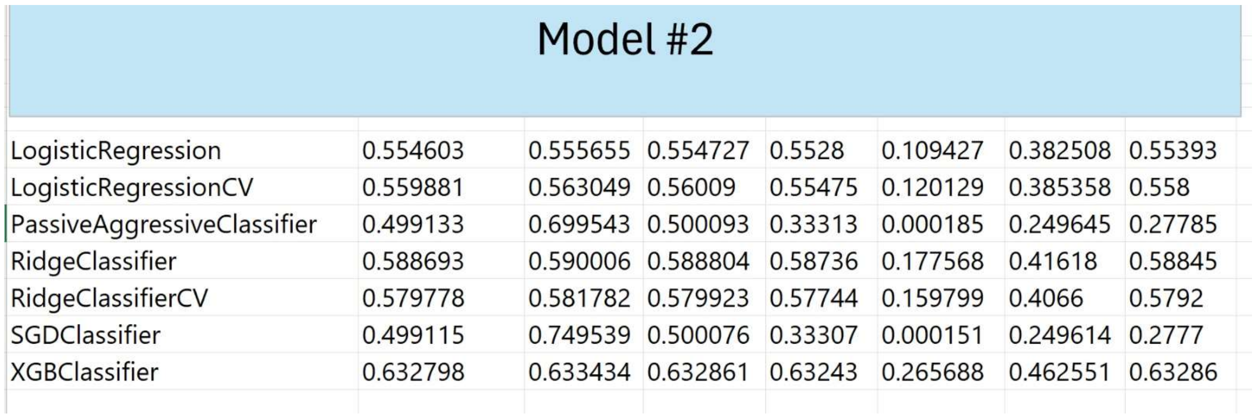
- Additionally, feature engineering and selection can further enhance model performance. Identifying and creating new features based on domain knowledge may provide additional insights that can improve predictive capabilities.
- Potential Enhancements
- Hyperparameter Tuning: Optimizing model parameters through techniques such as Grid Search or Random Search can significantly enhance model performance.
- Advanced Models: Exploring advanced models, including deep learning architectures, may provide improved accuracy, especially in complex datasets.
- Ensemble Learning: Combining multiple models can improve robustness and accuracy by leveraging the strengths of different algorithms.
- By implementing these enhancements, the predictive model can be further refined, ultimately leading to better protection against malware infections on Windows machines.

## History of Algorithms Performance



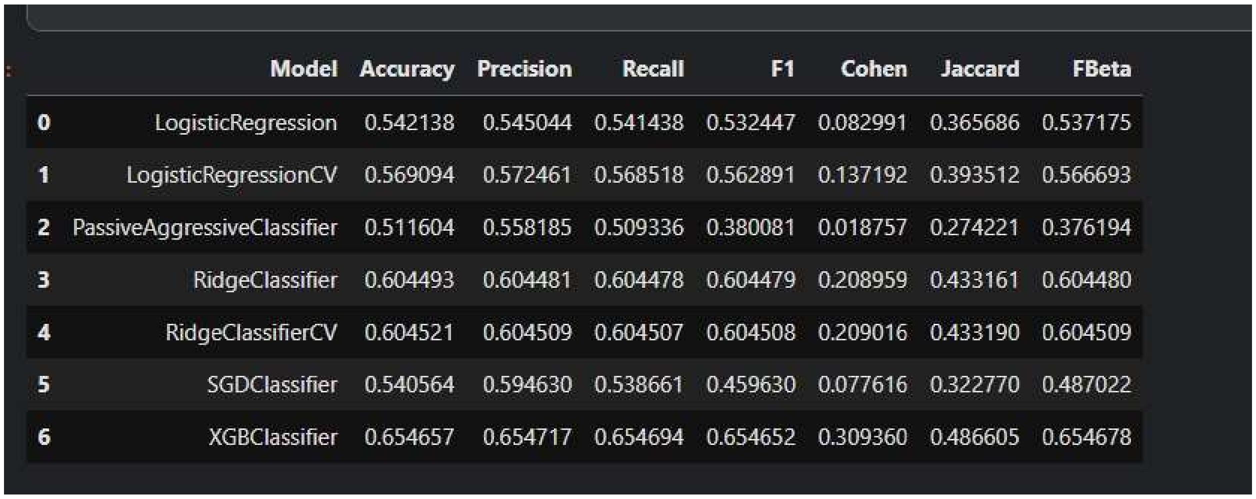
Model #1

| Model | Accuracy | Precision | Recall | F1 | Cohen | Jaccard | FBeta |
|---|---|---|---|---|---|---|---|
| LogisticRegression | 0.537811 | 0.5378 | 0.537787 | 0.53776 | 0.075577 | 0.367779 | 0.53778 |
| LogisticRegressionCV | 0.567215 | 0.569397 | 0.566966 | 0.56328 | 0.133998 | 0.39322 | 0.56573 |
| PassiveAggressiveClassifier | 0.498668 | 0.249334 | 0.5 | 0.33274 | 0 | 0.249334 | 0.27712 |
| RidgeClassifier | 0.596435 | 0.597467 | 0.596289 | 0.59515 | 0.192633 | 0.424015 | 0.59612 |
| RidgeClassifierCV | 0.596251 | 0.597244 | 0.596108 | 0.59501 | 0.192269 | 0.423859 | 0.59594 |
| SGDClassifier | 0.505192 | 0.514657 | 0.506203 | 0.42251 | 0.012382 | 0.292638 | 0.43625 |

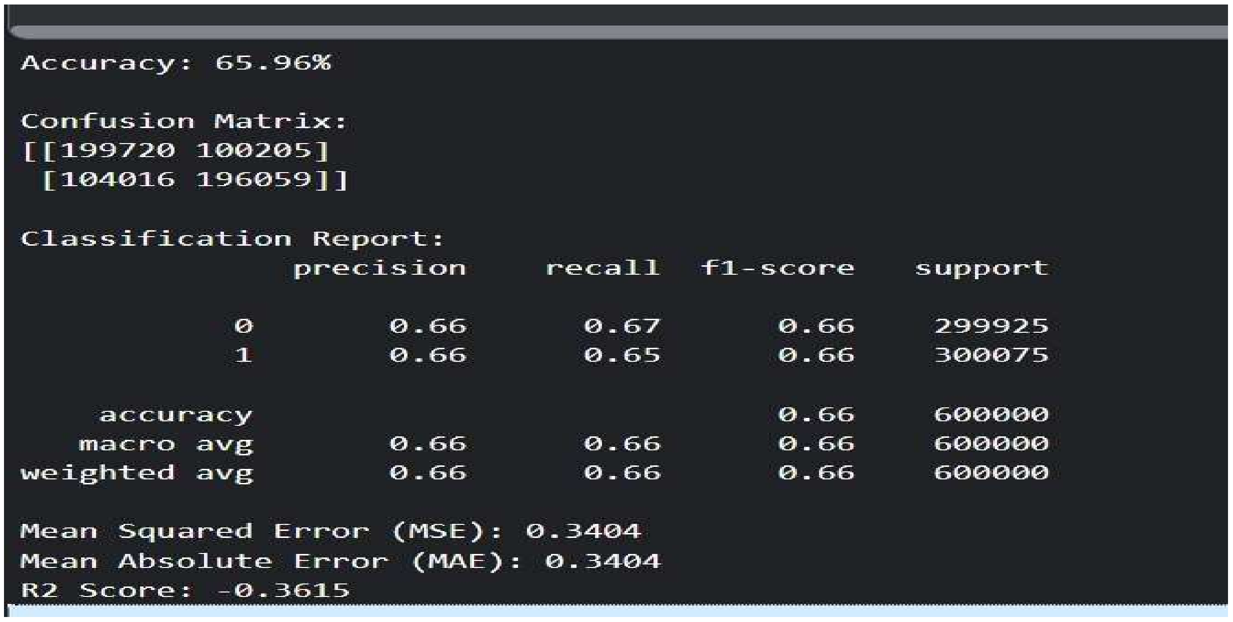


Model #2

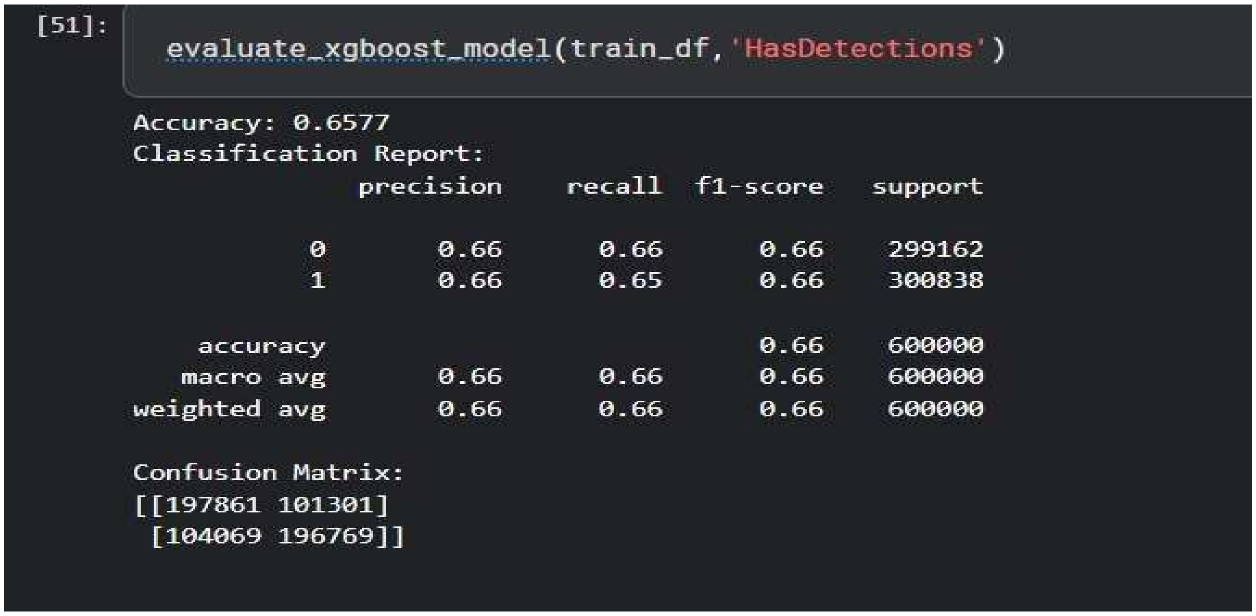
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LogisticRegression | 0.554603 | 0.555655 | 0.554727 | 0.5528 | 0.109427 | 0.382508 | 0.55393 |
| LogisticRegressionCV | 0.559881 | 0.563049 | 0.56009 | 0.55475 | 0.120129 | 0.385358 | 0.558 |
| PassiveAggressiveClassifier | 0.499133 | 0.699543 | 0.500093 | 0.33313 | 0.000185 | 0.249645 | 0.27785 |
| RidgeClassifier | 0.588693 | 0.590006 | 0.588804 | 0.58736 | 0.177568 | 0.41618 | 0.58845 |
| RidgeClassifierCV | 0.579778 | 0.581782 | 0.579923 | 0.57744 | 0.159799 | 0.4066 | 0.5792 |
| SGDClassifier | 0.499115 | 0.749539 | 0.500076 | 0.33307 | 0.000151 | 0.249614 | 0.2777 |
| XGBClassifier | 0.632798 | 0.633434 | 0.632861 | 0.63243 | 0.265688 | 0.462551 | 0.63286 |



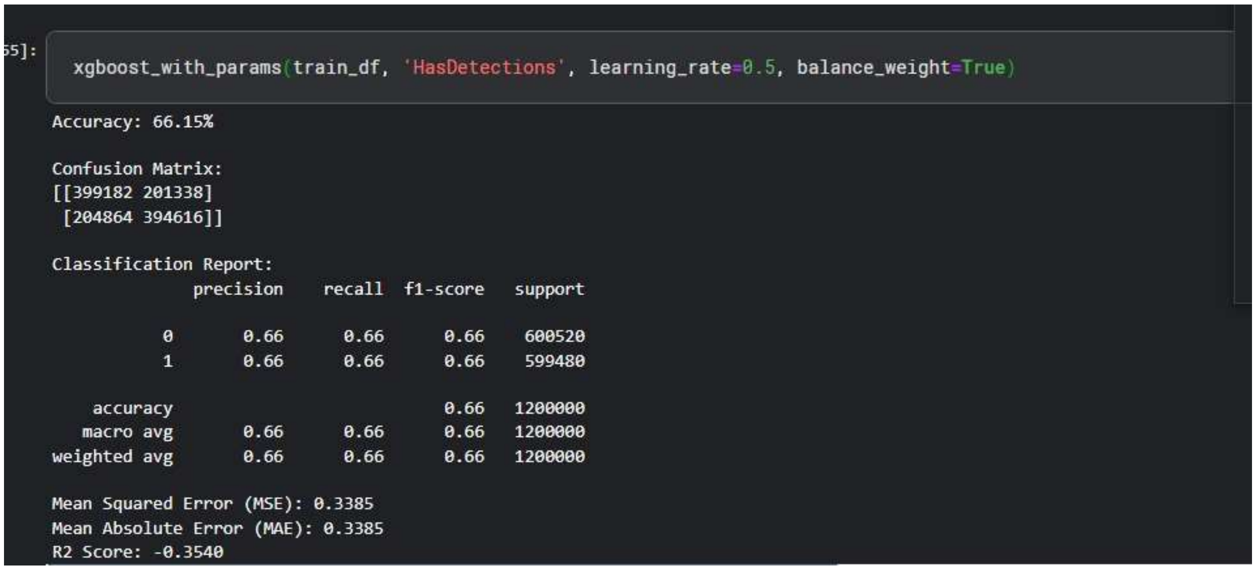
| | Model | Accuracy | Precision | Recall | F1 | Cohen | Jaccard | FBeta |
|---|---|---|---|---|---|---|---|---|
| **0** | LogisticRegression | 0.542138 | 0.545044 | 0.541438 | 0.532447 | 0.082991 | 0.365686 | 0.537175 |
| **1** | LogisticRegressionCV | 0.569094 | 0.572461 | 0.568518 | 0.562891 | 0.137192 | 0.393512 | 0.566693 |
| **2** | PassiveAggressiveClassifier | 0.511604 | 0.558185 | 0.509336 | 0.380081 | 0.018757 | 0.274221 | 0.376194 |
| **3** | RidgeClassifier | 0.604493 | 0.604481 | 0.604478 | 0.604479 | 0.208959 | 0.433161 | 0.604480 |
| **4** | RidgeClassifierCV | 0.604521 | 0.604509 | 0.604507 | 0.604508 | 0.209016 | 0.433190 | 0.604509 |
| **5** | SGDClassifier | 0.540564 | 0.594630 | 0.538661 | 0.459630 | 0.077616 | 0.322770 | 0.487022 |
| **6** | XGBClassifier | 0.654657 | 0.654717 | 0.654694 | 0.654652 | 0.309360 | 0.486605 | 0.654678 |



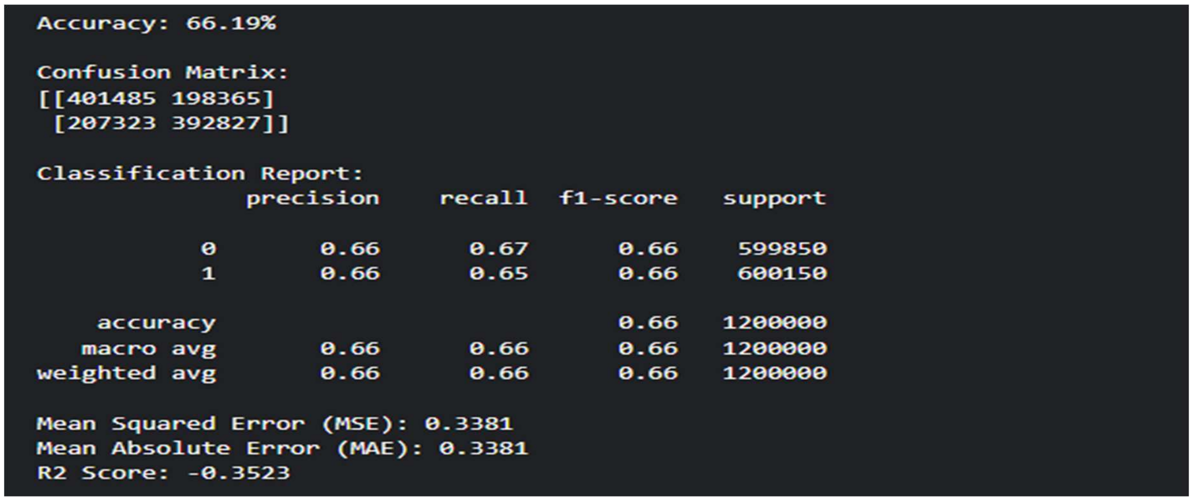
```
Accuracy: 65.96%

Confusion Matrix:
[[199720 100205]
 [104016 196059]]

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.67      0.66    299925
           1       0.66      0.65      0.66    300075

    accuracy                           0.66    600000
   macro avg       0.66      0.66      0.66    600000
weighted avg       0.66      0.66      0.66    600000

Mean Squared Error (MSE): 0.3404
Mean Absolute Error (MAE): 0.3404
R2 Score: -0.3615
```



```
[51]:  evaluate_xgboost_model(train_df,'HasDetections')

       Accuracy: 0.6577
       Classification Report:
                     precision    recall  f1-score   support

                  0       0.66      0.66      0.66    299162
                  1       0.66      0.65      0.66    300838

           accuracy                           0.66    600000
          macro avg       0.66      0.66      0.66    600000
       weighted avg       0.66      0.66      0.66    600000

       Confusion Matrix:
       [[197861 101301]
        [104069 196769]]
```



```
5]:  xgboost_with_params(train_df, 'HasDetections', learning_rate=0.5, balance_weight=True)

     Accuracy: 66.15%

     Confusion Matrix:
     [[399182 201338]
      [204864 394616]]

     Classification Report:
                   precision    recall  f1-score   support

                0       0.66      0.66      0.66    600520
                1       0.66      0.66      0.66    599480

         accuracy                           0.66   1200000
        macro avg       0.66      0.66      0.66   1200000
     weighted avg       0.66      0.66      0.66   1200000

     Mean Squared Error (MSE): 0.3385
     Mean Absolute Error (MAE): 0.3385
     R2 Score: -0.3540
```



```
Accuracy: 66.19%

Confusion Matrix:
[[401485 198365]
 [207323 392827]]

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.67      0.66    599850
           1       0.66      0.65      0.66    600150

    accuracy                           0.66   1200000
   macro avg       0.66      0.66      0.66   1200000
weighted avg       0.66      0.66      0.66   1200000

Mean Squared Error (MSE): 0.3381
Mean Absolute Error (MAE): 0.3381
R2 Score: -0.3523
```

## XGBoost Model Performance

The XGBoost model was used with a learning rate of 0.5 and balanced weights. Below are the results obtained:

### Accuracy

The model achieved an accuracy of 66.19%.

### Confusion Matrix

```
[[399182, 201338],
 [204864, 394616]]
```

### Classification Report

- Precision (class 0): 0.66
- Precision (class 1): 0.66
- Recall (class 0): 0.66
- Recall (class 1): 0.66
- F1-Score (class 0): 0.66
- F1-Score (class 1): 0.66
- Support (class 0): 600520
- Support (class 1): 599480

### Error Metrics

- Mean Squared Error (MSE): 0.3381
- Mean Absolute Error (MAE): 0.3381
- R2 Score: -0.3523

## Model Deployment using Streamlit

For the deployment of the machine learning model, we used Streamlit, a simple and powerful framework for building web applications. The model was integrated into a web interface to allow easy interaction and prediction on new data.

## Setup and Installation

Streamlit server and open the application in a web browser.

## Streamlit Integration with the Model

The model trained using XGBoost was loaded and used within Streamlit to provide real-time predictions. The application allows users to input features and get the model's prediction on whether a machine will be hit with malware. Below is an outline of the key steps in integrating the model:

1. Load the pre-trained model (e.g., `model_1.pkl`) using joblib or pickle.
2. Create a simple user interface using Streamlit widgets (e.g., sliders, text inputs) for users to input the required features.
3. Use the input data to generate predictions and display them in real-time using Streamlit's output display functions.