ML Mid-Sem Project Report

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Sahil Gupta	2022430
Aditya Gupta	2022031
Avi Sharma	2022119
Rishabh Jay	2022

Motivation

- The **malware industry** is highly sophisticated, constantly creating new methods to evade security mechanisms.
- Malware infections can lead to **data theft, financial damage**, and disruptions for individuals and organizations.
- The project aims to develop a **predictive model** to determine the likelihood of a computer being compromised by malware.
- It uses a **dataset from Microsoft** with detailed information on past malware incidents.
- The goal is to **proactively identify and mitigate** malware risks.
- This project contributes to advancing **open-source methodologies** for malware prediction and enhancing security.

Literature Review

1. Malware Analysis and Detection Using Machine Learning Algorithms

   This paper addresses polymorphic malware detection, evaluating Decision Trees (DT), Convolutional Neural Networks (CNN), and Support Vector Machines (SVM). DT achieved the highest accuracy (99%), followed by CNN (98.76%) and SVM (96.41%), with low false positive rates: DT (2.01%), CNN (3.97%), and SVM (4.63%).

2. Evaluation of Machine Learning Algorithms for Malware Detection

   The study focuses on dynamic malware detection using classifiers in a simulated environment. RandomForest(RF)and Gaussian Naive Bayes (NB) achieved 100% accuracy, precision, recall, and F1-score, demonstrating effective behavior-based detection.

3. Microsoft Malware Prediction Using LightGBM Model

   This research applies the LightGBM model for Windows malware detection, emphasizing feature engineering and evaluating performance using AUC-ROC.LightGBM achieved the highest AUC-ROC score of 0.684, outperforming Catboost and XGBoost.

Dataset Details

- The dataset is sourced from **Kaggle** and aims to predict the probability of Windows machines being infected by malware based on machine-specific properties.
- **Telemetry data** is collected via Windows Defender, integrating heartbeat and threat reports.
- Each machine is uniquely identified by a **Machine Identifier**, and the target variable, **HasDetections**, indicates malware detection (1 = detected, 0 = not detected).
- **Training data** (train.csv) contains 8.9 million machines with labels, while **test data** (test.csv) includes 7.85 million machines for which predictions are required.
- The dataset consists of **83 features** representing various machine properties and configurations.

EDA And Preprocessing

- Data types were optimized for memory efficiency by setting categorical variables as categories and converting numerical values to smaller types like int8 and float32. Using a custom reduce memory usage function, memory usage was reduced by 17.1%, improving data handling and analysis speed.
- The analysis identified several columns with significant data quality issues, including PuaMode and Census ProcessorClass, which have over 99% missing values, and the DefaultBrowsersIdentifier, where 95% of entries belong to a single category.
- Additionally, 26 other columns displayed similar imbalances, with over 90% of their values concentrated in one category, suggesting they should be removed from the dataset to enhance its quality and relevance.
- Further investigation is warranted for Census IsFlightingInternal,which exhibited unusual patterns. We observe that the target classes are well-balanced, with 4,462,591 instances belonging to class 0 and 4,458,892 instances belonging to class 1 in the training dataset.
- Thus, no additional balancing techniques were necessary. We conducted Exploratory Data Analysis (EDA) on the feature

EDA on IsTouchEnabled

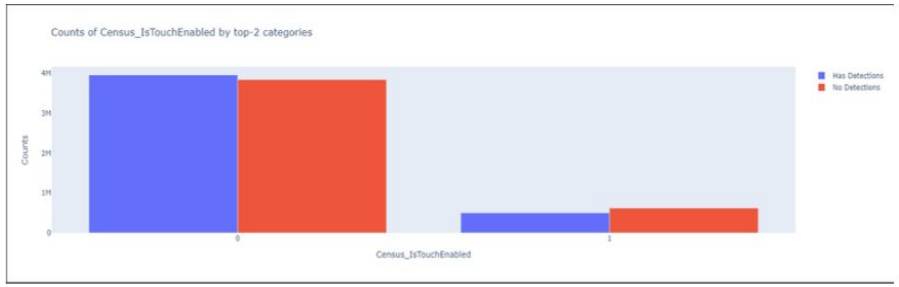

Figure 1. Touch_Based

We first examine variables with a high number of categories, followed by an analysis of those with a limited number of categories . An example of this is the feature EngineVersion, for which the plots are presented below:

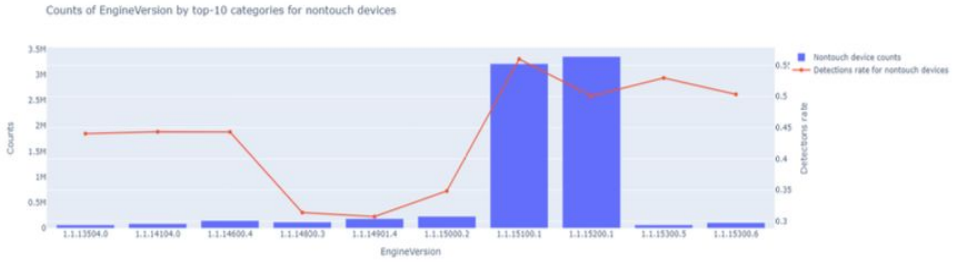

Figure 2. EngineVersion_NoTouch

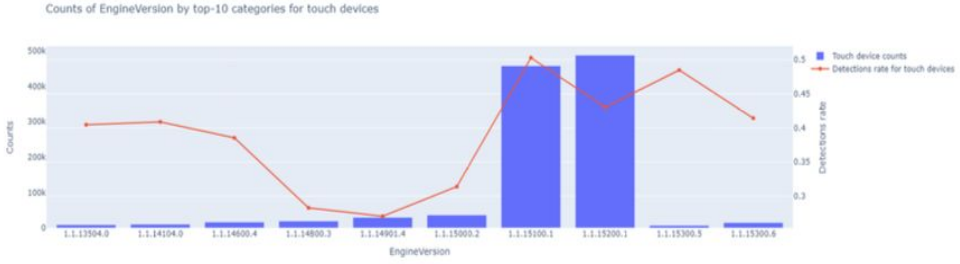

Figure 3. EngineVersion_Touch

Observation ?

We observe that two categories account for 84% of all values, indicating a significant disparity in detection rates. The remaining categories exhibit varying detection rates, primarily due to the low number of samples available in those categories. Notably, the patterns for touch and non-touch devices are quite similar . Similarly, we identified comparable observations for other variables, such as AppVersion, AvSigVersion, AVProductStatesIdentifier, AVProd- uctsInstalled, and others.

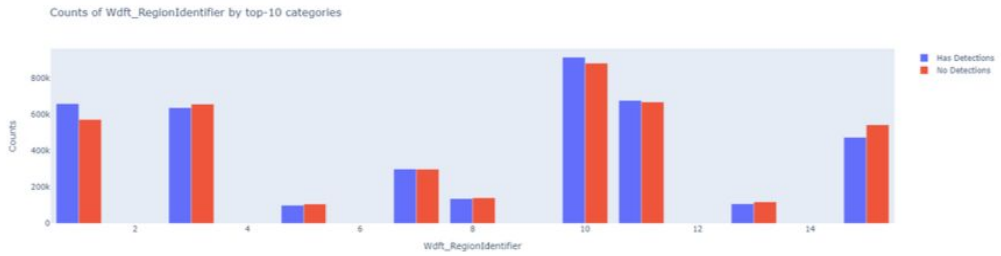

Figure 4. Categorical_Feature_Misclassified

The EDA also enabled us to identify instances where a variable was assigned an incorrect data type, such as float16/float32, despite its distribution following a categorical pattern . This way , we changed the datatype of such variables to category .

One such example is of the variable WdftRegionIdentifier

Additional plots and details are available at the provided Github Link.

Feature Engineering

Handling `OsBuildLab` Column:

- Converted to categorical data type for memory efficiency and model performance.
- Introduced a new category (0.0.0.0.0-0) to replace missing values, ensuring data consistency.

Feature Extraction:

- **New Categorical Features**: Extracted from software version details (EngineVersion, AppVersion, AvSigVersion, OsBuildLab, Census OSVersion).

Disk and Display Calculations:

1. **Primary Drive Ratio and Size**: Ratio of system volume capacity to total disk capacity; size of non-primary drive.
2. **Aspect Ratio and Display Info**: Calculated aspect ratio, dimensions, DPI (dots per inch), and screen area.
3. **Monitor Dimensions**: Combined representation of monitor dimensions as a categorical feature.

Performance-Based Features:

1. **RAM per Processor**: Amount of RAM available per processor core.
2. **Other Display Features**: Combines diagonal screen size with the number of processor cores for interaction effects.

Missing Values: Default values added for columns like Census IsFlightingInternal, Census ThresholdOptIn, Census IsWIMBootEnabled, and Wdft IsGamer.

Additional Information: More details available on the provided GitHub link.

Models and Methodology

This problem is a classification task, and we applied various methodologies to address it effectively. A range of models was explored, each chosen based on its appropriateness for the dataset and the nature of the problem. The models employed include:

1. Light Gradient Boosting Machine(LightGBM)
2. Extreme Gradient Boosting(XGBoost)
3. Random forest (RF) model
4. Decision Tree (DT) Model

Light Gradient Boosting Machine(LightGBM)

Light Gradient Boosting Machine (LightGBM) is a gradient boosting framework that uses tree-based learning algorithms, designed for high performance and efficient with k-fold cross validation(k=5) , enabling multiple models to be evaluated and selected based on their AUC performance . It handles large datasets efficiently by making chunked predictions and thus managing memory constraints during training and inference . The approach also allows for detailed insights into model performance through AUC scores and feature importance, making it suitable for complex classification problems with large datasets .

The showcase of the feature importance is as follows:

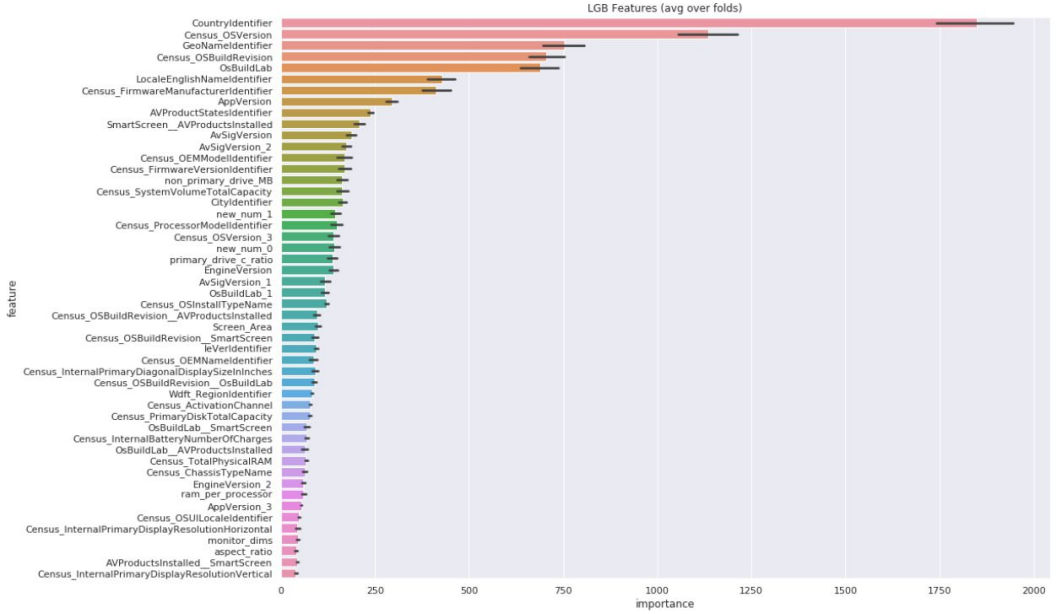

Figure 5. Feature_Selection

Extreme Gradient Boosting(XGBoost)

Extreme Gradient Boosting (XGBoost) is a highly efficient machine learning algorithm that enhances speed and performance through its parallelized tree-building and regularization techniques. This makes it a popular choice for predictive modeling in structured data, effectively minimizing overfitting. The implementation of XGBoost focuses on leveraging k-fold cross validation(stratified cross validation) to assess the model's performance robustly , while efficiently doing memory handling and AUC scoring provide insights into the model's predictive capabilities . The key hyperparameters are num boost round (set to 400) which determines the maximum number of boosting iterations early stopping rounds (set to 200), allowing early termination if validation performance does not improve.

Random forest (RF) model :

This is an ensemble-based model that uses many multiple weak learners (decision trees) to classify its final predictions. It has three main hyperparameters, the number of trees, the maximum depth of each tree, and the number of features to be sampled. We employed a stratified k-fold which is a variation of the standard K-fold cross-validation technique. This variation ensures that each fold has the same class composition as the original dataset.The model was run with the following parameters:

Number of Trees = 100

Depth = 10

and it was run for 5 folds and the final prediction was the average prediction from the 5 models trained.

Decision Tree (DT) Model :

1. **Dataset Preparation**:
   - Divide columns into continuous and categorical (version) columns.
   - Exclude columns: `HasDetections`, `MachineIdentifier`.
   - Remove columns:
     - Most frequent value > 90% of data.
     - Columns with > 500 unique values.
2. **Categorical Columns Processing**:
   - Calculate detection ratio for each unique value (mean of target `HasDetections`).
   - Sort unique values by detection ratio and map them to integers.
   - Store conversion dictionary for categorical-to-numeric conversion.
3. **Version Columns**:
   - Break down version columns into constituent parts for finer analysis.
   - Handle missing values by filling with column mean.
   - Remove single unique value columns.

Decision Tree (DT) Model :

Model Training:

- Perform hyperparameter tuning on Decision Tree classifier.
- Use **k-fold cross-validation** to find the optimal `min_sample_leaf` value.
- Maximize **AUC** (Area Under ROC Curve) for best model balance (achieved 70.23% AUC).

Model Application:

- Fill missing values in test dataset using pre-computed statistics.
- Apply trained model to predict probabilities on test data.

Results:

- Accuracy on test dataset: **63.833%**.

Link to github

Results and analysis

In this section, we present the results of our classification models applied to the dataset, focusing on their performance metrics and comparative analysis. The primary metric used for the evaluation is the accuracy in the testing set and the training in the AUC score, which provides insight into the predictive capabilities of the models. We also discuss the implications of the importance of the features and any patterns observed during the model evaluation.

Model	Accuracy	AUC Score
LightGBM	0.675	0.717
XGBoost	0.657	0.708
Random Forest	0.658	0.659
Decision Tree	0.63833	0.7023

Table 1. Model Accuracy and AUC Summary

Observations

- **LightGBM**:
  - Highest accuracy: **0.675**
  - Highest AUC: **0.675**
- **XGBoost**:
  - Accuracy: **0.657**
- **Random Forest**:
  - Accuracy: **0.658**
  - Lowest AUC: **0.659**
- **Decision Tree**:
  - Lowest accuracy: **0.63833**
  - Higher AUC than Random Forest: **0.7023**

Conclusion: LightGBM outperforms all models in both accuracy and AUC, while Decision Tree has a lower accuracy but a better AUC than Random Forest.

Conclusion

Learning from the project

Boosting Models (LightGBM & XGBoost):

- Outperform traditional models.
- Incrementally improve on errors, generalize better.
- Capture complex feature relationships.

Traditional Models (Random Forest & Decision Tree):

- Prone to overfitting.
- Random Forest has lower AUC but higher accuracy, indicating:
  - Good at class prediction.
  - Struggles with ranking predictions, especially in imbalanced datasets.

Timeline

Weeks 1-2 (August 28- September 10): Project initialization, literature review, dataset acquisition, data preprocessing, feature engineering, and exploratory data analysis (EDA).

Weeks 3-4 (September 11- September 24): Implementation of baseline models (Random Forest, SVM, Decision Trees, LightGBM), hyperparameter tuning, and model optimization.

Weeks 5-6 (September 25- October 8): Model evaluation and performance comparison focusing on AUC ROC and accuracy. Final integration, results visualization, and project documentation

Weeks 7-8 (October 9- October 22): Presentation preparation and final review. Progress report due on October 14. Mid-semester presentation on October 15. Continue adjustments based on feedback.

Work Left

We plan to do exhaustive hyperparameter tuning for optimizing the performance of the models, and also plan to explore other techniques in feature engineering.

Contribution

- **Aditya Gupta**: Conducted EDA, initial data preprocessing, and implemented and tested the LightGBM model.
- **Avi Sharma**: Performed data preprocessing, and implemented the XGBoost model.
- **Rishabh Jay**: Conducted EDA and implemented the Random Forest model.
- **Sahil Gupta**: Executed feature engineering, and implemented the Decision Tree model.