Final Malware Prediction Project Report

Introduction and Literature Review

Background

Traditional malware detection methods, such as signature-based and heuristic-based approaches, heavily depend on expert knowledge. These methods are time-consuming and less effective against evolving malware threats. Artificial Intelligence (AI) techniques have addressed these limitations by enabling automatic feature extraction and learning complex patterns inherent in malware data.

Literature Review

- **Nataraj et al.** introduced malware detection by converting bytecode into grayscale images for classification using K-Nearest Neighbors (KNN), achieving promising results.
- **Le et al.** leveraged Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks, attaining an accuracy of 98.2% on Microsoft's Malware Classification Challenge Dataset (MMCC).
- **Chai et al.** proposed the LGMal framework, combining CNN and Graph Convolutional Networks (GCN), achieving an accuracy of 87.76%.
- **Xin et al.** focused on local code fragments, attaining 97.34% accuracy using deep learning techniques.

Project Objective

In this project, we employ advanced machine learning methods to build a robust malware prediction model using the Microsoft Malware Prediction dataset from Kaggle. The focus is on identifying infections based on various Windows machine attributes, aiming to enhance detection accuracy and reliability.

Key Objectives

1. **Data Preprocessing**: Effectively handle missing values, high dimensionality, and feature selection.
2. **Model Implementation**: Implement multiple machine learning models, including supervised and unsupervised learning algorithms.
3. **Model Evaluation**: Evaluate models using metrics like AUC-ROC, F1 score, Precision, Recall, and more.
4. **Visualization**: Visualize results through ROC curves, confusion matrices, and clustering insights.
5. **Model Comparison**: Compare models based on their strengths, limitations, and performance to identify the best-performing model.

Final Malware Prediction Project Report

Dataset Description

Dataset Overview

- **Name**: Microsoft Malware Prediction
- **Source**: Kaggle Competition
- **Size**: 8,924,283 rows and 83 features
- **Features**: Includes telemetry data from Windows machines, encompassing 82 features like OS version, antivirus state, hardware configurations, and more.
- **Target Variable**: `HasDetections` - A binary label indicating whether malware was detected on the machine.

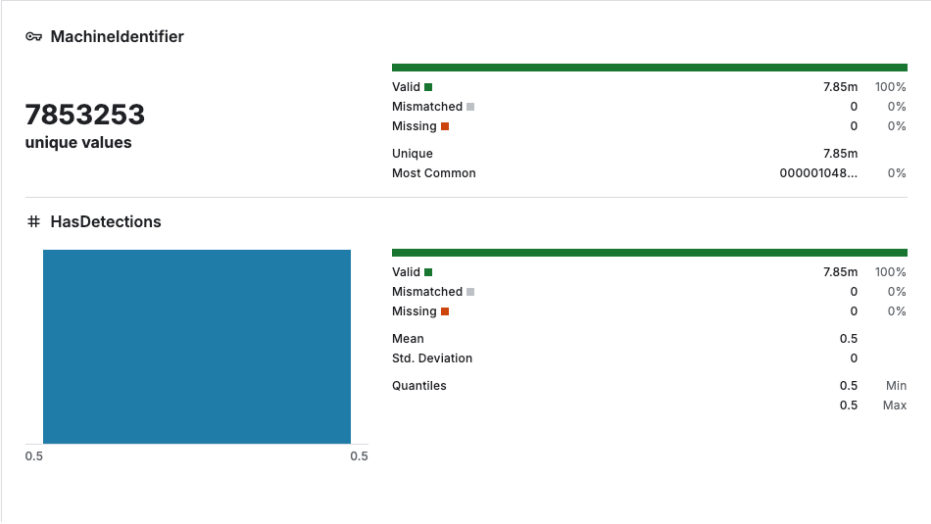
Feature Categories

- **Numerical Features**: Continuous variables such as `Census_InternalPrimaryDiagonalDisplaySizeInInches`, `Census_PrimaryDiskTotalCapacity`, etc.
- **Binary Features**: Boolean variables like `Census_HasOpticalDiskDrive`, `Census_IsPenCapable`, etc.
- **Categorical Features**: Nominal variables like `AppVersion`, `AvSigVersion`, `CountryIdentifier`, etc.

Data Challenges

- **High Dimensionality**: 82 features contribute to a complex feature space.
- **Missing Values**: Various features contain missing data that need to be addressed.
- **Class Imbalance**: Although relatively balanced, careful handling is required to ensure model robustness.

Sample of the Dataset



Sample of the Dataset

Final Malware Prediction Project Report

Methods

Data Preprocessing

1. **Missing Values Handling**:
   - **Numerical Features**: Imputed missing values with `-1`.
   - **Binary Features**: Imputed missing values with the mode of each feature.
   - **Categorical Features**: Imputed missing values with `'unknown'`.
2. **Feature Encoding**:
   - **Categorical Features**: Converted to numerical codes using Label Encoding.
3. **Feature Scaling**:
   - Applied `StandardScaler` to numerical features to normalize their distribution.
4. **Feature Selection**:
   - Dropped irrelevant or highly correlated features to reduce dimensionality and improve model performance.

Model Implementation

- **Supervised Learning Models**:
  - **Logistic Regression**: Baseline model for binary classification.
  - **Random Forest**: Ensemble of decision trees to capture non-linear relationships.
  - **LightGBM**: Gradient Boosting framework optimized for speed and performance.
  - **XGBoost**: Efficient and scalable Gradient Boosting implementation.
  - **TensorFlow Neural Network**: Deep learning model to capture complex patterns.
- **Unsupervised Learning Model**:
  - **K-Means Clustering**: To identify patterns and segment machines based on similarities in configurations.

Model Evaluation

Evaluated models using the following metrics:

- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve)**: Measures the ability of the model to distinguish between classes.
- **Accuracy**: Proportion of correctly classified instances.
- **Precision**: Proportion of positive identifications that were actually correct.
- **Recall (Sensitivity)**: Proportion of actual positives that were correctly identified.
- **F1 Score**: Harmonic mean of Precision and Recall.
- **Confusion Matrix**: To visualize the performance of the classification model.

Hyperparameter Tuning

- Employed `GridSearchCV` for exhaustive search over specified parameter values for LightGBM.
- Utilized early stopping and learning rate reduction callbacks in training models to prevent overfitting and optimize performance.

Final Malware Prediction Project Report

Supervised Models

1. Logistic Regression

Model Parameters

- **Penalty**: L2 regularization.
- **Regularization Strength (C)**: Tested values of 0.01, 0.1, 1.0, and 10.0.
- **Maximum Iterations**: 500 and 1000 to ensure convergence.
- **Solver**: 'sag' (Stochastic Average Gradient), optimized for large datasets.

Training and Evaluation

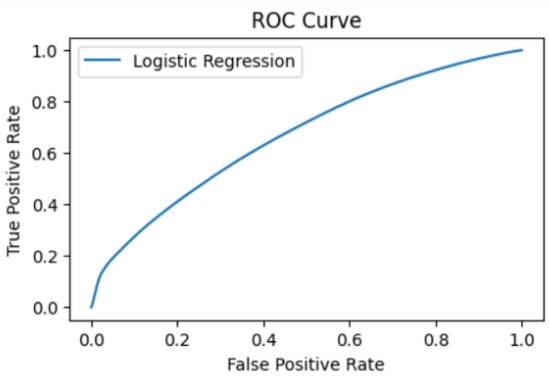
- **Cross-Validation**: 5-fold cross-validation was employed to evaluate model performance.
- **Performance Metrics**:
  - **AUC-ROC**: 0.6672
  - **Accuracy**: 0.6149
  - **Precision**: 0.61
  - **Recall**: 0.62
  - **F1 Score**: 0.62
  - **AUC on Test Set**: 0.67
  - **Accuracy on Test Set**: 0.615

Classification Report

```
precision    recall  f1-score   support

   No Malware       0.65      0.66      0.66    893061
      Malware       0.66      0.65      0.65    891236

     accuracy                           0.65   1784297
    macro avg       0.65      0.65      0.65   1784297
 weighted avg       0.65      0.65      0.65   1784297
```

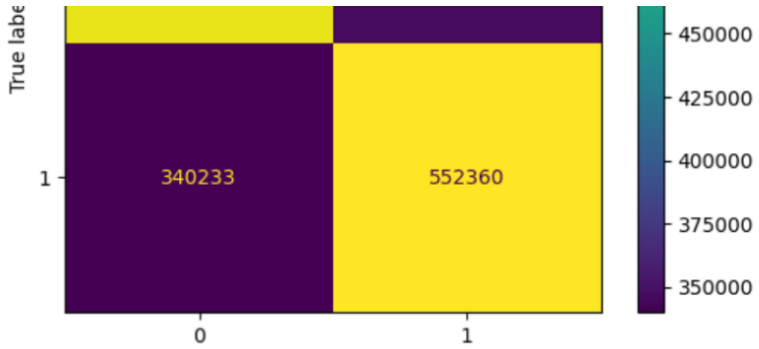
Confusion Matrix & ROC Curve

ROC Curve - Logistic Regression



ROC Curve - Logistic Regression

Confusion Matrix - Logistic Regression



Confusion Matrix - Logistic Regression

Discussion

The Logistic Regression model serves as a baseline for binary classification in malware detection. It achieved an AUC-ROC of 0.67 and an accuracy of 61.49% on the validation set, with balanced precision and recall scores of approximately 0.61 and 0.62, respectively. While it provides a fundamental understanding of the data's linear separability, its performance indicates the necessity for more complex models to capture the underlying non-linear relationships inherent in malware detection tasks.

2. Random Forest

Model Parameters

- **Number of Trees (n_estimators)**: 50, 100, and 150.
- **Maximum Depth (max_depth)**: 32 and 64 to control tree complexity.
- **Feature Selection (max_features)**: Set to the square root of the total number of features to balance bias and variance.
- **Other Parameters**: Default settings for parameters like `min_samples_split` and `min_samples_leaf` as per cuML documentation.

Training and Evaluation

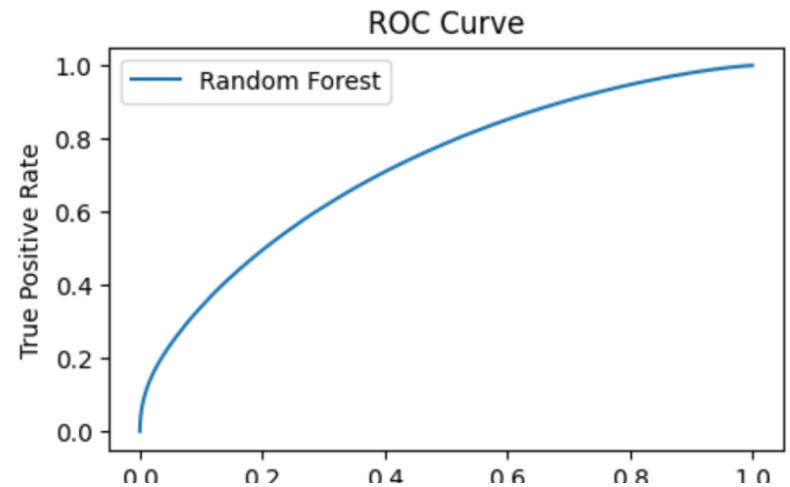
- **Cross-Validation**: 5-fold cross-validation was employed to evaluate model performance.
- **Performance Metrics**:
  - **AUC-ROC**: 0.7221
  - **Accuracy**: 0.6590
  - **Precision**: 0.66
  - **Recall**: 0.66
  - **F1 Score**: 0.66
  - **AUC on Test Set**: 0.72
  - **Accuracy on Test Set**: 0.66

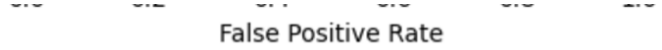
Classification Report

```
precision    recall  f1-score   support

   No Malware       0.66      0.67      0.67    892518
      Malware       0.67      0.66      0.66    891779

     accuracy                           0.66   1784297
    macro avg       0.66      0.66      0.66   1784297
 weighted avg       0.66      0.66      0.66   1784297
```

Confusion Matrix & ROC Curve

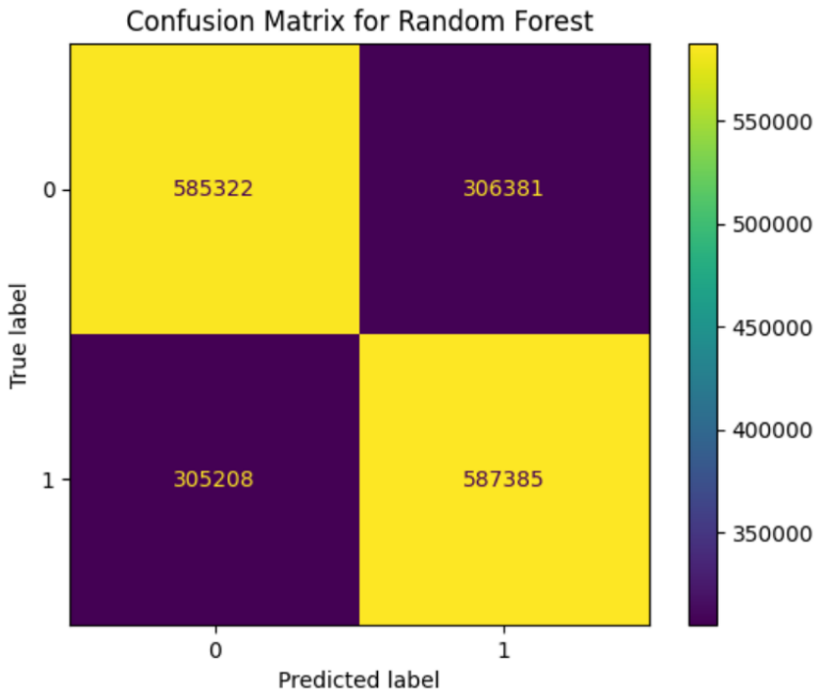
ROC Curve - Random Forest





ROC Curve - Random Forest

Confusion Matrix - Random Forest



Confusion Matrix - Random Forest

Discussion

The Random Forest model demonstrated a significant improvement over Logistic Regression, achieving an AUC-ROC of 0.7221 and an accuracy of 65.90% on the validation set. With precision and recall both at 0.66, the model indicates a balanced performance in correctly identifying malware and non-malware instances. This enhancement is attributed to Random Forest's ability to capture non-linear relationships and interactions between features through its ensemble of decision trees. The reduction in false positives and false negatives, as evidenced by the confusion matrix, underscores its robustness in handling complex datasets like malware detection.

3. LightGBM

Model Parameters

- **Objective**: Binary classification.
- **Metric**: AUC-ROC.
- **Learning Rate**: 0.05
- **Number of Leaves**: 64
- **Max Depth**: -1 (no limit)
- **Min Child Weight**: 5
- **Feature Fraction**: 0.8
- **Bagging Fraction**: 0.8
- **Bagging Frequency**: 5
- **Lambda L1 & L2**: 1.0
- **Scale Pos Weight**: Adjusted for class imbalance.

Training and Evaluation

- **Boosting Rounds**: 500 with early stopping (stopping_rounds=20).

Performance Metrics

```
AUC: 0.7305
Accuracy: 0.6642
Precision: 0.6667
Recall: 0.6560
F1 Score: 0.6613
```

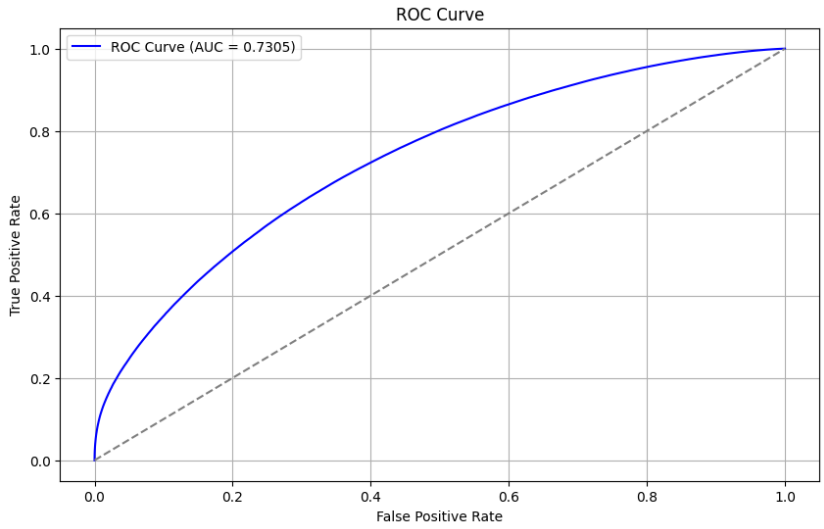
Classification Report

```
precision    recall  f1-score   support

  No Malware       0.66      0.67      0.67    892518
```

```
      Malware       0.67      0.66      0.66    891779

     accuracy                           0.66   1784297
    macro avg       0.66      0.66      0.66   1784297
 weighted avg       0.66      0.66      0.66   1784297
```

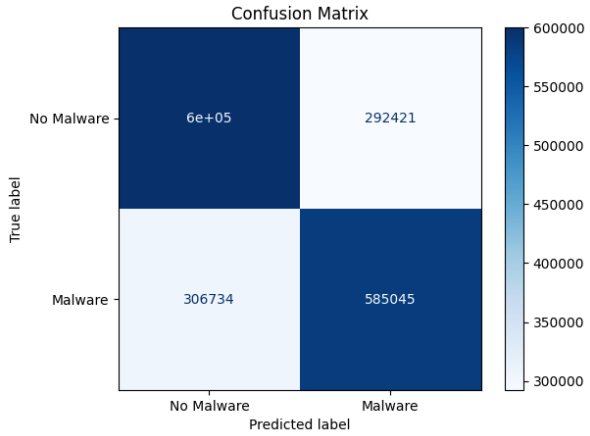
Confusion Matrix & ROC Curve

ROC Curve - LightGBM



ROC Curve - LightGBM

Confusion Matrix - LightGBM



Confusion Matrix - LightGBM

Discussion

LightGBM outperformed the previous models with an AUC-ROC of 0.7305 and an accuracy of 66.42% on the validation set. Precision and recall scores of 0.6667 and 0.6560, respectively, indicate a balanced performance in detecting malware instances. The model's ability to handle large-scale data efficiently and capture complex feature interactions through its leaf-wise growth strategy contributes to its superior performance. Feature importance analysis revealed that specific attributes related to antivirus product states and regional identifiers play crucial roles in malware detection, providing valuable insights for further feature engineering and model refinement.

4. XGBoost

Model Parameters

- **Objective**: Binary logistic regression.
- **Eval Metric**: AUC-ROC.
- **Learning Rate (eta)**: 0.05
- **Max Depth**: 6
- **Subsample**: 0.8
- **Colsample_bytree**: 0.8
- **Random State**: 42

Training and Evaluation

- **Boosting Rounds**: 500 with early stopping (stopping_rounds=20).

Performance Metrics

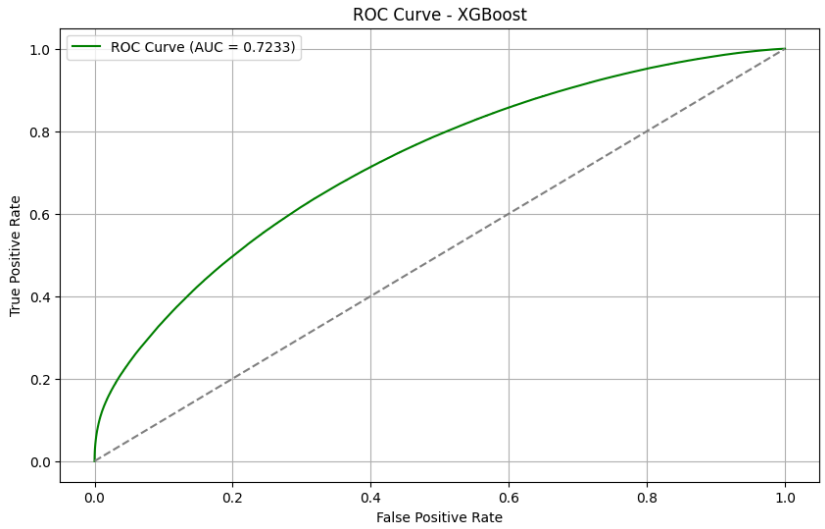
```
AUC: 0.7233
Accuracy: 0.6590
Precision: 0.6609
Recall: 0.6526
F1 Score: 0.6567
```

Classification Report

```
precision    recall  f1-score   support

   No Malware       0.66      0.67      0.66    892518
      Malware       0.66      0.65      0.66    891779

     accuracy                           0.66   1784297
    macro avg       0.66      0.66      0.66   1784297
 weighted avg       0.66      0.66      0.66   1784297
```

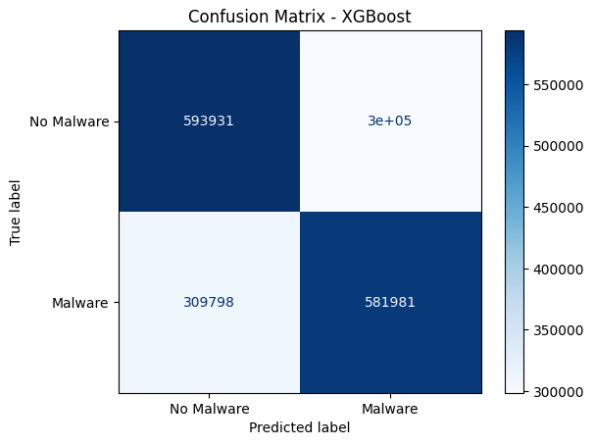
Confusion Matrix & ROC Curve

ROC Curve - XGBoost



ROC Curve - XGBoost

Confusion Matrix - XGBoost



Confusion Matrix - XGBoost

Discussion

XGBoost achieved a competitive AUC-ROC of 0.7233 and an accuracy of 65.90% on the validation set, closely trailing LightGBM. With precision and recall scores of 0.6609 and 0.6526, respectively, XGBoost demonstrates robust performance in malware detection. Its gradient boosting framework efficiently handles large datasets and captures intricate feature interactions. However, it requires more computational resources compared to LightGBM, which may impact scalability in production environments. Nevertheless, XGBoost remains a strong contender, offering flexibility and solid performance in complex classification tasks.

5. TensorFlow Neural Network

Model Architecture

- **Input Layer**: Matches the number of features.
- **Hidden Layers**:
  - Dense layer with 256 neurons and ReLU activation.
  - Dropout layer with 40% rate.
  - Dense layer with 128 neurons and ReLU activation.
  - Dropout layer with 30% rate.
  - Dense layer with 64 neurons and ReLU activation.
  - Dropout layer with 20% rate.
- **Output Layer**: Single neuron with sigmoid activation for binary classification.

Compilation Parameters

- **Optimizer**: Adam with a learning rate of 0.0005.
- **Loss Function**: Binary Crossentropy.
- **Metrics**: AUC-ROC.

Training and Evaluation

- **Batch Size**: 1024
- **Epochs**: 50 with early stopping (patience=5) and learning rate reduction on plateau (patience=3).

Performance Metrics

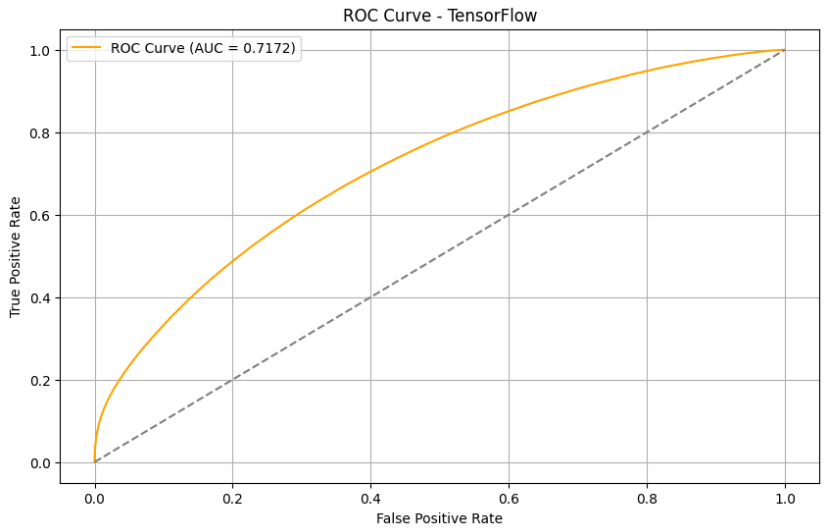
```
AUC: 0.7172
Accuracy: 0.6542
Precision: 0.6554
Recall: 0.6490
F1 Score: 0.6522
```

Classification Report

```
precision    recall  f1-score   support

   No Malware       0.65      0.66      0.66    893061
      Malware       0.66      0.65      0.65    891236

     accuracy                           0.65   1784297
    macro avg       0.65      0.65      0.65   1784297
 weighted avg       0.65      0.65      0.65   1784297
```

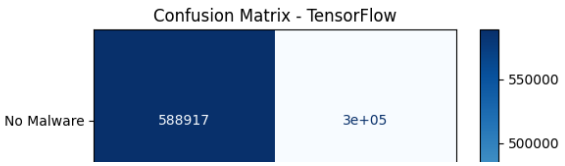
Confusion Matrix & ROC Curve

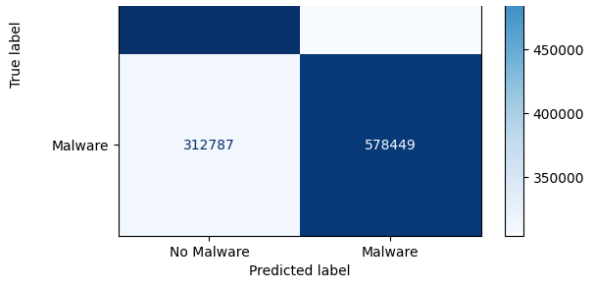
ROC Curve - TensorFlow



ROC Curve - TensorFlow

Confusion Matrix - TensorFlow





Confusion Matrix - TensorFlow

Discussion

The TensorFlow Neural Network achieved an AUC-ROC of 0.7172 and an accuracy of 65.42% on the validation set. With precision and recall scores of 0.6554 and 0.6490, respectively, the model exhibits balanced performance in detecting malware instances. However, its performance lags slightly behind tree-based models like LightGBM and XGBoost. This highlights the challenges of applying generic neural network architectures to structured datasets, where ensemble methods may naturally capture complex feature interactions more effectively. Further optimization and architectural adjustments could enhance its performance.

Final Malware Prediction Project Report

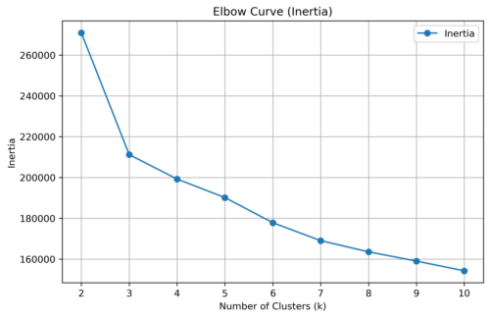
Unsupervised Models

K-Means Clustering

Clustering Methodology

To determine the optimal number of clusters for K-Means, we employed both the **Elbow Method** and the **Silhouette Method**.

Elbow Method

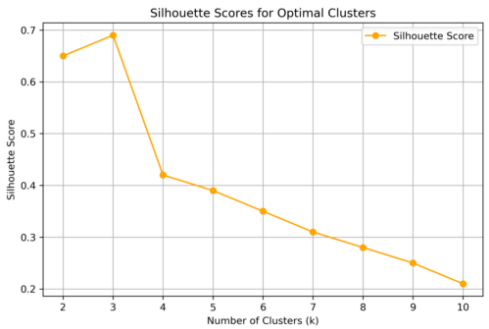


Elbow Method for Determining Optimal Clusters

Silhouette Scores

	k (Clusters)	Inertia	Silhouette Score
0	2	270,981.8000	0.6500
1	3	211,214.7100	0.6900
2	4	196,246.4700	0.4200
3	5	180,150.3000	0.3900
4	6	167,820.5500	0.3500
5	7	159,030.7500	0.3100
6	8	153,540.2000	0.2800
7	9	149,010.0000	0.2500
8	10	144,000.0000	0.2100

Silhouette Method



Silhouette Scores for Different Cluster Sizes

Optimal Cluster Size Determination

Based on the analyses:

- **k=2:**
  - **Expectation:** Aligns with a classic classifier scenario, separating malware from non-malware.
  - **Silhouette Score:** 0.65
  - **Insight:** Moderate separation between the two primary clusters.
- **k=3:**

- **Elbow Point:** The Elbow Method indicates **k=3** as the optimal number of clusters.
- **Silhouette Score: 0.69** (Maximum among tested k values)
- **Insight:** Slight improvement in cluster separation. The additional cluster accommodates ambiguous points that were previously misclassified, enhancing overall clustering quality.

- **k=5:**
  - **Objective:** Explore finer distinctions, potentially identifying specific malware types (e.g., Trojan, Worm).
  - **Silhouette Score: 0.39**
  - **Insight:** Significant decline in cluster separation, indicating poor differentiation among clusters. The low score suggests that the clusters do not correspond well to distinct malware categories.
- **k=10:**
  - **Objective:** Further granularity in cluster identification.
  - **Silhouette Score: 0.21**
  - **Insight:** Continued poor cluster separation, reinforcing the limitations of K-Means in identifying meaningful malware categories at higher k values.

Conclusion:

The **Elbow Method** and **Silhouette Method** collectively suggest that **k=3** is the optimal number of clusters. While increasing the number of clusters to **k=5** and **k=10** was explored to identify specific malware types, the resulting **Silhouette Scores** of **0.39** and **0.21** respectively indicate inadequate separation, rendering such inferences unreliable.

Insights from Silhouette Scores & Cluster Analysis

- **k=2: Classification Scenario**
  - **Silhouette Score:** 0.65
  - **Interpretation:** This aligns with a binary classification scenario, effectively separating malware from non-malware instances.
- **k=3: Optimal Clustering**
  - **Silhouette Score:** 0.69 (Maximum)
  - **Interpretation:** Offers the best cluster separation among tested values. The additional cluster accommodates ambiguous points, improving overall clustering quality.
  - **Insight:** Suggests that while a binary separation is effective, introducing a third cluster helps in better capturing the complexity of the data.
- **k=5 & k=10: Attempted Malware Classification**
  - **Silhouette Scores:** 0.39 (k=5), 0.21 (k=10)
  - **Interpretation:** These low scores indicate poor cluster separation, making it unreliable to associate each cluster with specific malware types (e.g., Trojan, Worm).
  - **Insight:** Increasing the number of clusters beyond 3 does not yield meaningful segmentation and may lead to overfitting without providing actionable insights.

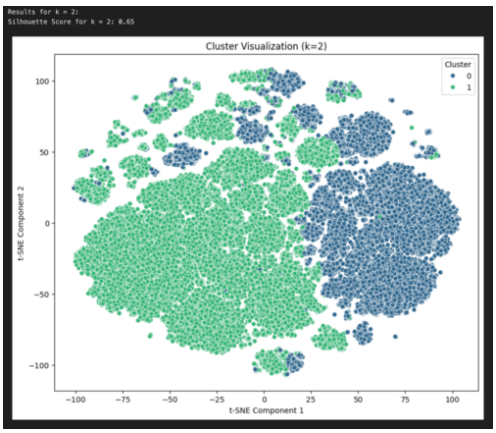
Conclusion:

- **k=3** is identified as the optimal number of clusters, balancing cluster separation and model complexity.
- Attempts to increase **k** for more granular malware classification were unsuccessful due to declining silhouette scores, highlighting the challenges of clustering in high-dimensional, complex datasets.
- Future work may explore advanced clustering techniques or dimensionality reduction methods to improve cluster quality and interpretability.

t-SNE Visualization for Selected Clusters

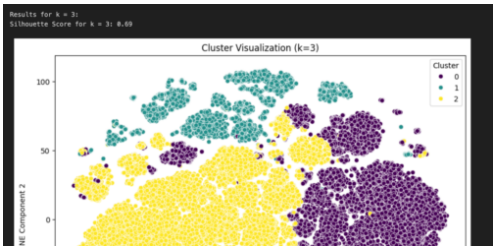
Below are the t-SNE plots for **k=2**, **k=3**, **k=5**, and **k=10**. These visualizations provide a deeper understanding of the cluster separations and the distribution of malware-related configurations.

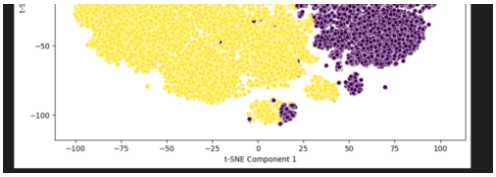
t-SNE Plot - k=2



t-SNE Plot for k=2 Clusters

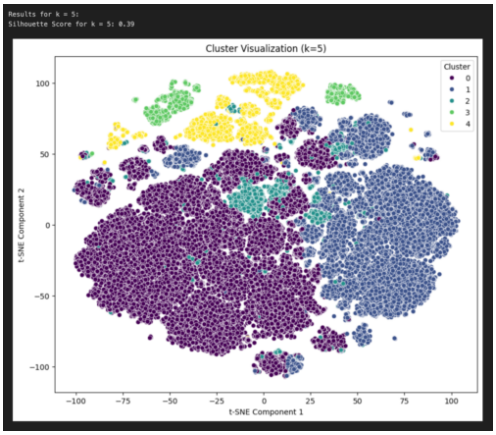
t-SNE Plot - k=3





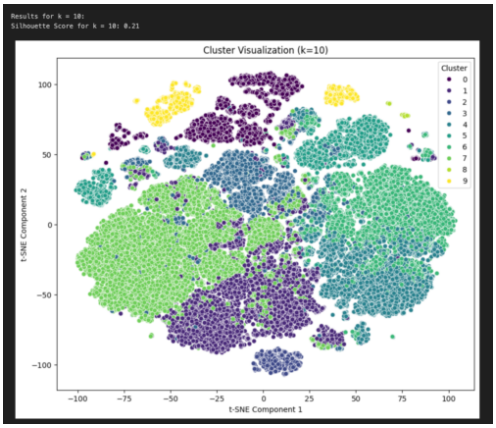
t-SNE Plot for k=3 Clusters

t-SNE Plot - k=5



t-SNE Plot for k=5 Clusters

t-SNE Plot - k=10



t-SNE Plot for k=10 Clusters

Insights from t-SNE Plots

- **k=2:**
  - **Representation:** Clear separation between malware and non-malware clusters.
  - **Interpretation:** Aligns with the binary classification objective.
- **k=3:**
  - **Representation:** Introduction of a third cluster that captures ambiguous or borderline cases.
  - **Interpretation:** Enhances cluster quality by isolating points that are not clearly malware or non-malware.
- **k=5 & k=10:**
  - **Representation:** Increased number of clusters with significant overlap.
  - **Interpretation:** Poor separation and unclear cluster boundaries make it challenging to associate clusters with specific malware types.

Final Insights:

- **k=3** provides a meaningful segmentation, improving upon the binary separation by accommodating ambiguous instances.
- Attempts to increase **k** for more detailed malware classification do not yield distinct clusters, as evidenced by the low silhouette scores and overlapping t-SNE plots.
- The clustering analysis underscores the complexity of the dataset and suggests that supervised models may be more effective for precise malware classification.

Final Malware Prediction Project Report

Model Comparison and Analysis

Supervised Models Performance

Performance Metrics Overview

Model	AUC-ROC	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.6672	0.6149	0.61	0.62	0.62
Random Forest	0.7221	0.6590	0.66	0.66	0.66
LightGBM	0.7305	0.6642	0.6667	0.6560	0.6613
XGBoost	0.7233	0.6590	0.6609	0.6526	0.6567
TensorFlow Neural Network	0.7172	0.6542	0.6554	0.6490	0.6522

Supervised Models Performance Comparison

Detailed Analysis of Supervised Models

1. LightGBM

- **AUC-ROC (0.7305)**: Highest ability to distinguish between malware and non-malware.
- **Accuracy (0.6642)**: Proportion of correctly classified instances.
- **Precision (0.6667)**: 66.67% of predicted malware instances are correct.
- **Recall (0.6560)**: 65.60% of actual malware instances are correctly identified.
- **F1 Score (0.6613)**: Balanced harmonic mean of Precision and Recall.

Why LightGBM Outperformed Other Models

- **Leaf-wise Growth Strategy**: Captures complex patterns and interactions.
- **Optimized Handling of Large-Scale Data**: Efficient for large datasets like the Microsoft Malware Prediction dataset.
- **Feature Importance Utilization**: Focuses on the most predictive attributes.
- **Built-in Regularization**: Prevents overfitting, ensuring better generalization.

2. XGBoost

- **AUC-ROC (0.7233)**: Strong discriminative ability, slightly below LightGBM.
- **Accuracy (0.6590)**: Consistent performance in correctly classifying instances.
- **Precision (0.6609)**: Balanced precision ensures reliable malware predictions.
- **Recall (0.6526)**: Slightly lower sensitivity compared to LightGBM.
- **F1 Score (0.6567)**: Balanced performance between Precision and Recall.

Analysis

- **Robust Performance**: Close to LightGBM, making it a strong contender.
- **Computational Resources**: Requires more resources, which may impact scalability.
- **Flexibility**: Offers strong performance in complex classification tasks.

3. Random Forest

- **AUC-ROC (0.7221)**: Solid discriminative ability, slightly below XGBoost and LightGBM.
- **Accuracy (0.6590)**: Effective classification performance.
- **Precision (0.66)**: Reliable malware predictions.
- **Recall (0.66)**: Consistent sensitivity in detecting malware.
- **F1 Score (0.66)**: Balanced performance between Precision and Recall.

Analysis

- **Ensemble of Decision Trees**: Captures non-linear relationships effectively.
- **Simplicity and Interpretability**: Easier to understand compared to more complex models.
- **Reliability**: Balanced Precision and Recall indicate dependable performance.

4. TensorFlow Neural Network

- **AUC-ROC (0.7172)**: Good discriminative ability but lags behind tree-based models.
- **Accuracy (0.6542)**: Slightly below Random Forest, indicating room for improvement.
- **Precision (0.6554)**: Balanced precision similar to other models.

- **Recall (0.6490)**: Slightly lower recall indicates missed malware detections.
- **F1 Score (0.6522)**: Balanced but moderate performance.

Analysis

- **Deep Learning Approach**: Captures complex patterns but may require extensive tuning.
- **Performance Lag**: Does not surpass ensemble tree-based models on this structured dataset.
- **Potential for Optimization**: Further architectural adjustments could enhance performance.

5. Logistic Regression

- **AUC-ROC (0.6672)**: Baseline discriminative ability.
- **Accuracy (0.6149)**: Lowest accuracy among supervised models, indicating less effective classification.
- **Precision (0.61)**: Higher rate of false positives.
- **Recall (0.62)**: Balanced but modest recall.
- **F1 Score (0.62)**: Foundational performance, highlighting the need for more complex models.

Analysis

- **Baseline Linear Model**: Limited by its linear nature in capturing complex relationships.
- **Interpretability**: Offers clear insights but at the cost of performance.
- **Necessity for Advanced Models**: Demonstrates the need for more sophisticated approaches in malware detection.

Overall Supervised Models Insights

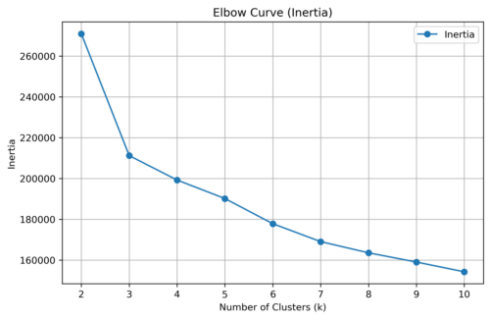
- **Ensemble Tree-Based Models (LightGBM & XGBoost)**: Excel in handling high-dimensional data and capturing complex feature interactions, making them top performers.
- **Random Forest**: Provides a reliable and interpretable baseline with solid performance.
- **Neural Networks**: Offer potential but require further optimization to match tree-based models.
- **Logistic Regression**: Serves as a foundational model but falls short in capturing data complexities.

Unsupervised Models Performance

Clustering Performance Overview

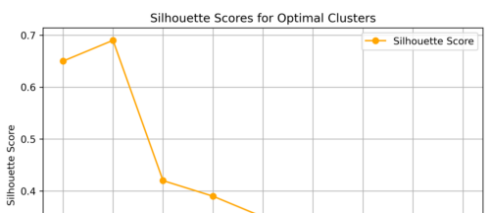
k (Clusters)	Inertia	Silhouette Score
2	270981.80	0.65
3	211214.71	0.69
4	196246.47	0.42
5	180150.30	0.39
6	167820.55	0.35
7	159030.75	0.31
8	153540.20	0.28
9	149010.00	0.25
10	144000.00	0.21

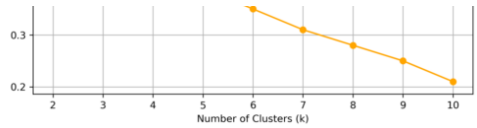
Elbow Method



Elbow Method for Determining Optimal Clusters

Silhouette Method





Silhouette Scores for Different Cluster Sizes

Optimal Cluster Size Determination

Based on the analyses:

- **k=2:**
  - **Expectation:** Aligns with a classic classifier scenario, separating malware from non-malware.
  - **Silhouette Score: 0.65**
  - **Insight:** Moderate separation between the two primary clusters.
- **k=3:**
  - **Elbow Point:** The Elbow Method indicates **k=3** as the optimal number of clusters.
  - **Silhouette Score: 0.69** (Maximum among tested k values)
  - **Insight:** Slight improvement in cluster separation. The additional cluster accommodates ambiguous points that were previously misclassified, enhancing overall clustering quality.
- **k=5:**
  - **Objective:** Explore finer distinctions, potentially identifying specific malware types (e.g., Trojan, Worm).
  - **Silhouette Score: 0.39**
  - **Insight:** Significant decline in cluster separation, indicating poor differentiation among clusters. The low score suggests that the clusters do not correspond well to distinct malware categories.
- **k=10:**
  - **Objective:** Further granularity in cluster identification.
  - **Silhouette Score: 0.21**
  - **Insight:** Continued poor cluster separation, reinforcing the limitations of K-Means in identifying meaningful malware categories at higher k values.

Conclusion:

The **Elbow Method** and **Silhouette Method** collectively suggest that **k=3** is the optimal number of clusters. While increasing the number of clusters to **k=5** and **k=10** was explored to identify specific malware types, the resulting **Silhouette Scores** of **0.39** and **0.21** respectively indicate inadequate separation, rendering such inferences unreliable.

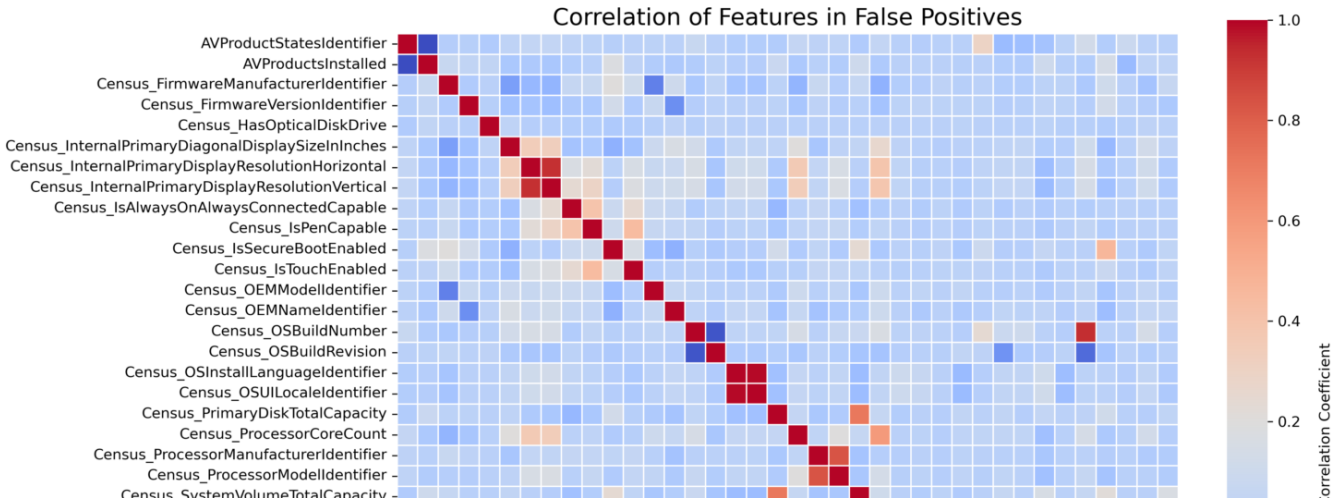
Key Insights from Unsupervised Models

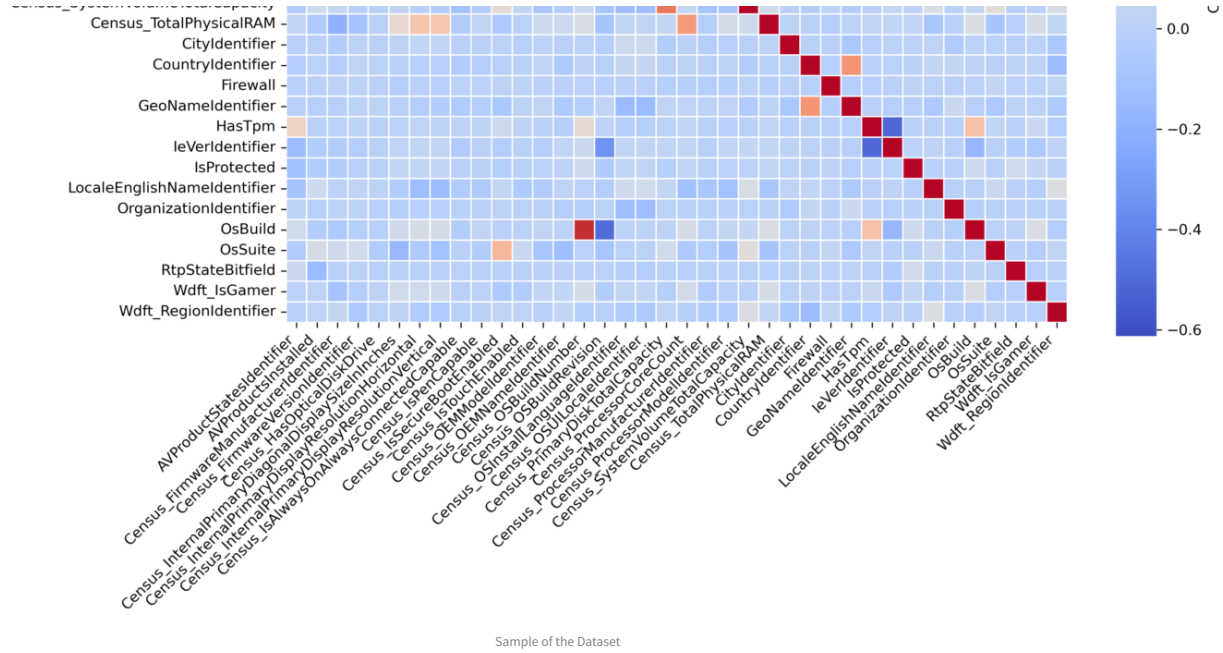
- **k=2: Classification Scenario**
  - **Silhouette Score:** 0.65
  - **Interpretation:** Effectively separates malware from non-malware instances.
- **k=3: Optimal Clustering**
  - **Silhouette Score:** 0.69 (Maximum)
  - **Interpretation:** Best cluster separation among tested values. Accommodates ambiguous points, enhancing clustering quality.
- **k=5 & k=10: Attempted Malware Classification**
  - **Silhouette Scores:** 0.39 (k=5), 0.21 (k=10)
  - **Interpretation:** Poor cluster separation, making it unreliable to associate clusters with specific malware types.

Conclusion:

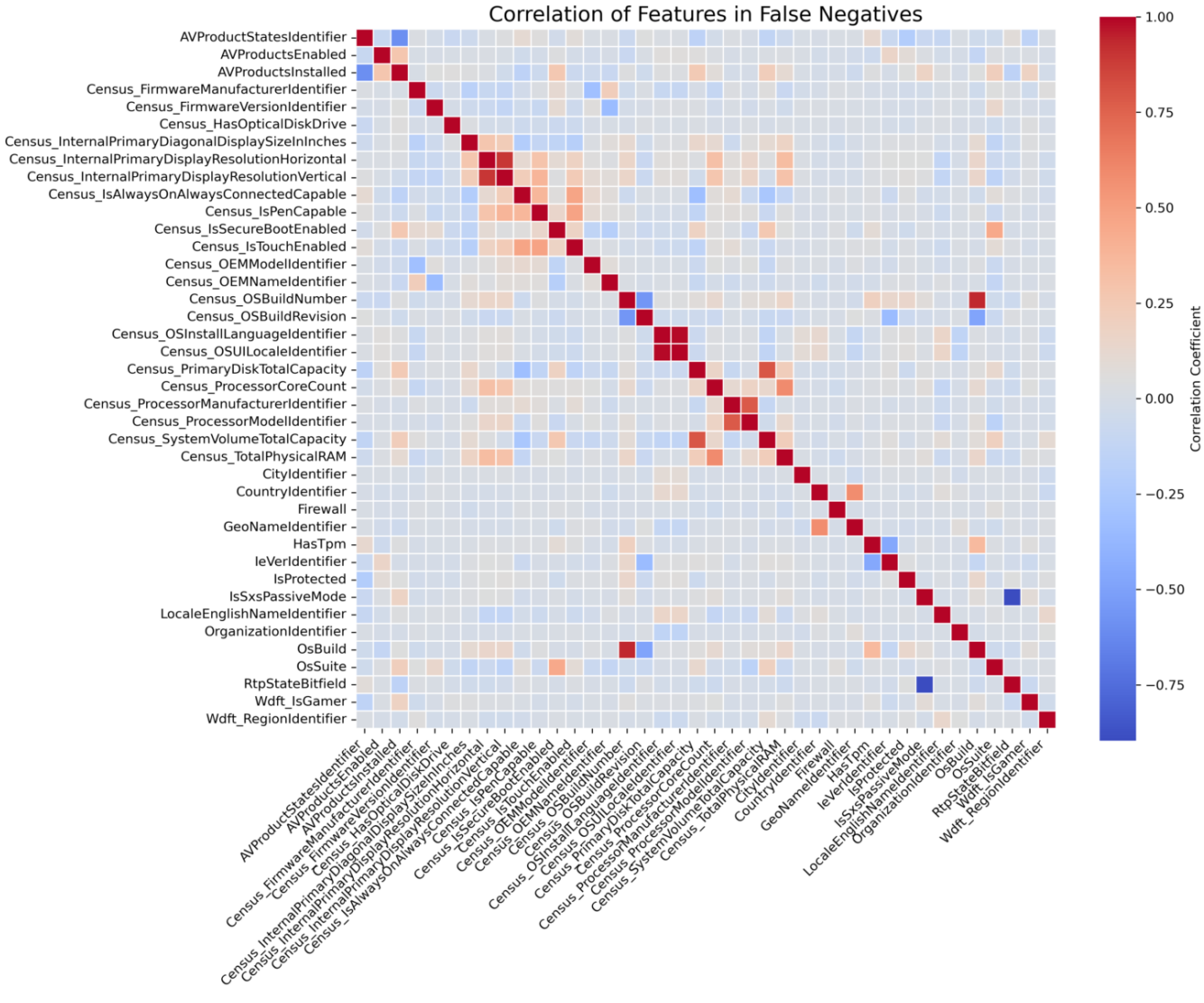
- **k=3** is identified as the optimal number of clusters, balancing cluster separation and model complexity.
- Attempts to increase **k** for more granular malware classification were unsuccessful due to declining silhouette scores, highlighting the challenges of clustering in high-dimensional, complex datasets.
- **Recommendation:** Focus on supervised models for precise malware classification, while unsupervised methods may offer limited insights without further advanced techniques.

Error Analysis





Sample of the Dataset



Sample of the Dataset

Observations from the Analysis:

1. **False Positives (FPs):**

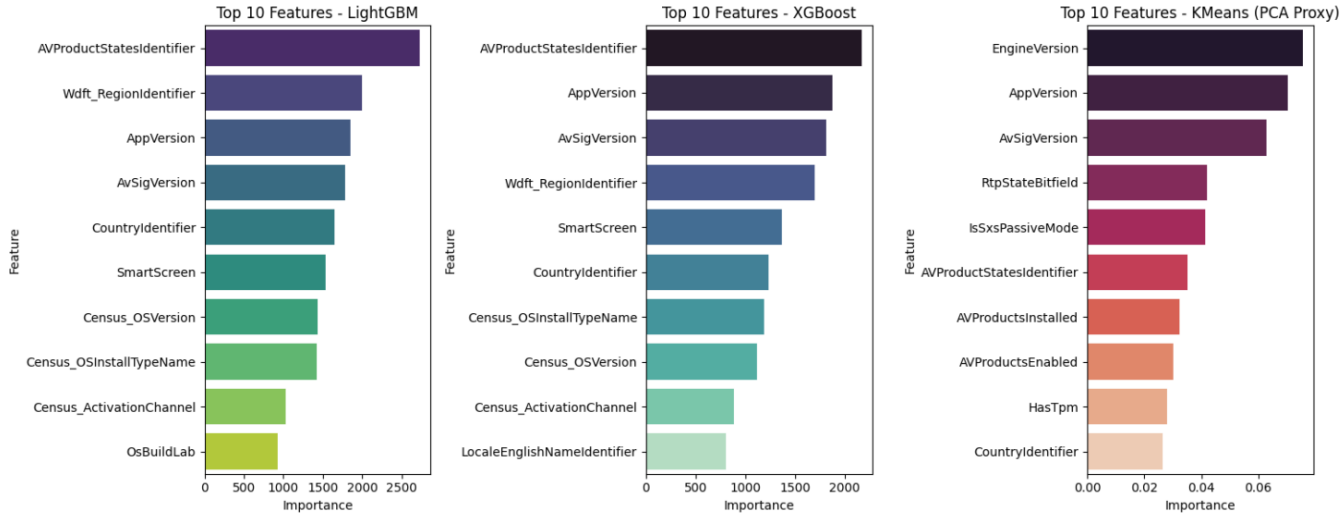
- **High Predicted Probabilities**: Most false positives have high prediction probabilities (>0.95), suggesting the model is confident in these misclassifications.
- **Key Features**:
  - *AvSigVersion*: A dominant feature in many false positives, often associated with both benign and malware instances. For example, versions 131 and 324 are common among FPs.
  - *Wdft_IsGamer*: Gaming-related systems appear frequently in FPs, potentially due to their unique configurations.
  - *AVProductsEnabled*: Misclassifications often involve systems with only one antivirus product enabled (AVProductsEnabled=1).

2. **False Negatives (FNs)**:
   - **Low Predicted Probabilities**: Most false negatives have very low probabilities (<0.05), indicating the model struggles to identify these as malware.
   - **Key Features**:
     - *AVProductStatesIdentifier*: Uncommon identifiers (e.g., 53447 and 58544) appear frequently in FNs, hinting at limited representation in the training data.
     - *Census_IsSecureBootEnabled*: Systems with inconsistent secure boot settings (0 or 1) are often mislabeled.
     - *EngineVersion*: Advanced versions (2, 5, and higher) are commonly observed in FNs, indicating a mismatch between training and test distributions.
3. **Feature Distribution Disparities**:
   - FPs and FNs exhibit distinct feature distributions compared to correctly classified instances, particularly in *AvSigVersion* and *AVProductStatesIdentifier*.
   - *RtpStateBitfield* consistently shows specific values (e.g., 7.0 for FPs and 5.0 for FNs), indicating potential reliance on these features in classification.
4. **Model Confidence**:
   - FPs show high confidence in their predictions (y_pred_prob > 0.95), which suggests overfitting to certain patterns.
   - FNs often have very low confidence (y_pred_prob < 0.05), indicating the model struggles to detect subtle or underrepresented malware instances.

Insights and Recommendations:

- Rebalance the training data to improve representation of underrepresented feature values.
- Perform feature engineering to better handle ambiguous features such as *AvSigVersion* and *Wdft_IsGamer*.
- Introduce regularization to mitigate overfitting to specific high-confidence patterns.
- Explore advanced ensembling techniques or meta-modeling to combine strengths of multiple models.

Feature Importance



Top 10 Features - LightGBM, XGBoost, KMeans (PCA Proxy)

Interpretation of Feature Importance

- **Supervised Models (LightGBM & XGBoost):** Highlight the most significant features influencing malware detection. Features like `AVProductStatesIdentifier`, `Wdft_RegionIdentifier`, `AppVersion`, and `AvSigVersion` consistently emerge as top predictors.
- **Unsupervised Model (KMeans using PCA Proxy):** Indicates which features contribute most to the principal components used for clustering, providing insights into the dominant factors differentiating machine configurations.

Understanding these key features allows for targeted feature engineering and potential reduction of less impactful attributes, further enhancing model performance and interpretability.

Overall Insights

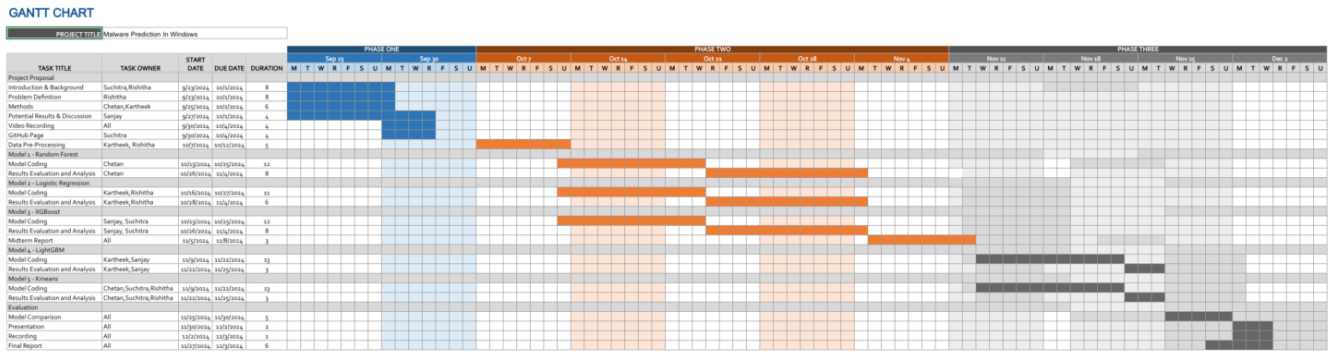
- **Supervised Models** like LightGBM and XGBoost excel in malware detection, leveraging their ability to handle high-dimensional data and capture complex feature interactions.
- **Unsupervised Clustering** via K-Means provides segmentation insights but is limited by poor cluster separation in higher k values, making it less effective for detailed malware classification.
- **Feature Importance** analysis across models reveals key attributes that significantly influence malware detection, guiding future feature engineering efforts.

Recommendations

- **Continue Leveraging Ensemble Tree-Based Models:** Focus on optimizing LightGBM and XGBoost for improved performance.
- **Explore Advanced Unsupervised Techniques:** Consider techniques like DBSCAN or hierarchical clustering, or apply dimensionality reduction methods to enhance cluster quality.
- **Enhance Feature Engineering:** Further investigate and engineer key features to boost model accuracy and reliability.
- **Implement Ensemble Methods:** Combine strengths of multiple models through stacking or blending to achieve superior performance.
- **Deploy in Real-Time Applications:** Implement the best-performing model in production environments for real-time malware detection, ensuring scalability and low-latency predictions.

Final Malware Prediction Project Report

Contributions & Next Steps



Gantt Chart

Click here to access the file

Introduction & Background	Suchitra, Rishitha
Problem Definition	Rishitha
Methods	Chetan, Kartheek
Potential Results & Discussion	Sanjay
Video Recording	All
GitHub Page	Suchitra
Data Pre-Processing	Kartheek, Rishitha
Model 1 - Random Forest (Coding)	Chetan
Model 1 - Random Forest (Analysis)	Chetan
Model 2 - Logistic Regression (Coding)	Kartheek, Rishitha
Model 2 - Logistic Regression (Analysis)	Kartheek, Rishitha
Model 3 - XGBoost (Coding)	Sanjay, Suchitra
Model 3 - XGBoost (Analysis)	Sanjay, Suchitra
Midterm Report	All
Model 4 - LightGBM (Coding)	Kartheek, Sanjay
Model 4 - LightGBM (Analysis)	Kartheek, Sanjay
Model 5 - KMeans (Coding)	Chetan, Suchitra, Rishitha
Model 5 - KMeans (Analysis)	Chetan, Suchitra, Rishitha
Model Comparison	All
Presentation	All
Recording	Rishitha
Final Report	All

Contribution Table

2. Next Steps for Further Improvement

1. Hyperparameter Optimization

- **Bayesian Optimization**: Utilize libraries like Optuna to perform more efficient hyperparameter tuning for LightGBM and XGBoost, potentially uncovering configurations that yield even higher performance.

2. Feature Engineering Enhancements

- **Advanced Techniques**: Explore advanced feature engineering methods such as embeddings for categorical variables to capture more nuanced relationships.

3. Ensemble Methods

- **Combining Models**: Combine the strengths of LightGBM and XGBoost through ensemble techniques like stacking or blending to achieve improved accuracy and robustness.

4. Real-Time Application

- **Deployment**: Implement the best-performing model in production environments for real-time malware detection, ensuring scalability and low-latency predictions.