DATA ANALYSIS TOOLS ANALYTICS

TITLE: ENHACING MALWARE DETECTION USING MACHINE LEARNING

COURSE CODE: 1202

PROFESSOR: Sk. Md. Mizanur Rahman

DATE: 10/08/2024

GROUP 2

OLUWAFEMI IGE
MIHARI JASON
MYMUNNISA BEGUM SHAIK
PREETI SHARMA
MANJU PONNAN VIJAYALAKSHMI
ANISH DSA

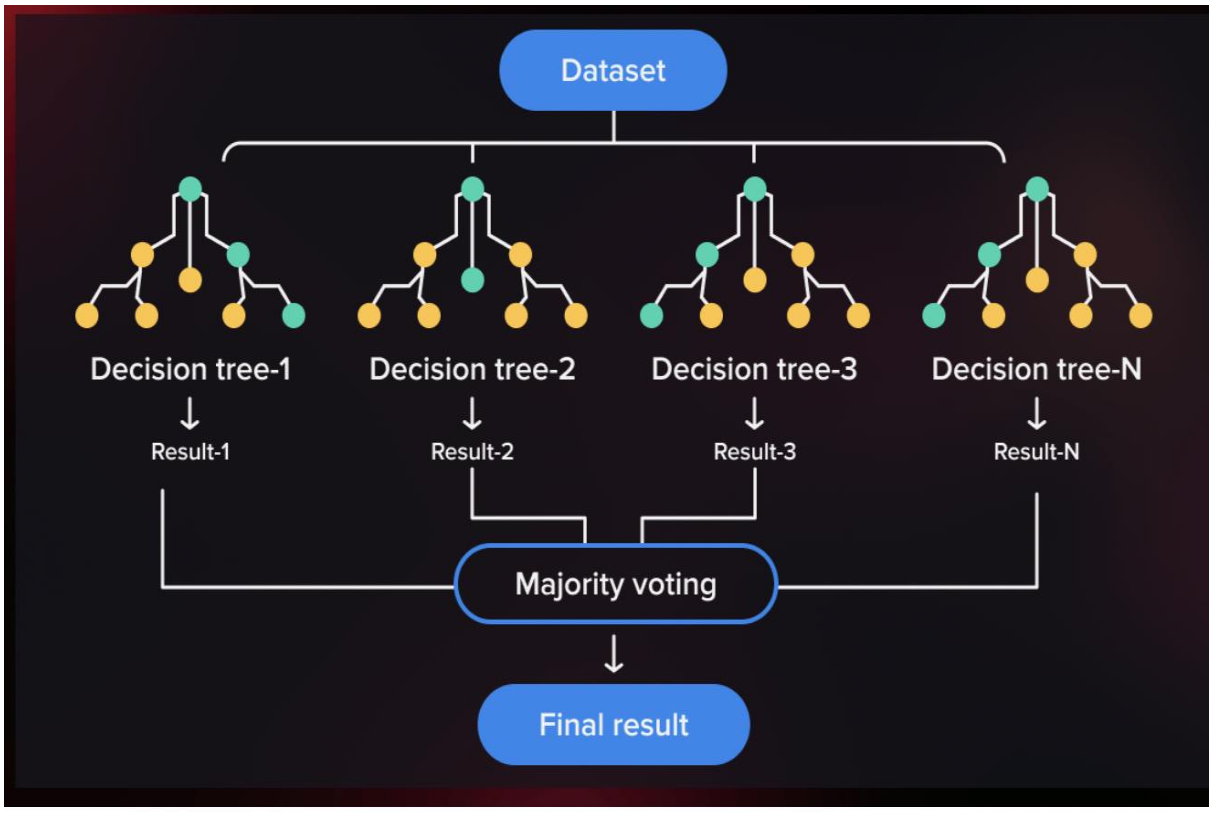
Table of Contents

INTRODUCTION

In today's digital landscape, the rise of cyber threats necessitates advanced tools to accurately identify and distinguish between malicious software (malware) and benign software. This project aims to develop a sophisticated classification system that leverages machine learning algorithms to differentiate between these two categories by analyzing a comprehensive set of system-level features. The features under consideration include utime (user time), gtime (guest time), cgtime (cgroup time), stime (system time), static_prio (static priority), prio (dynamic priority), normal_prio (normal priority), min_flt (minor faults), and maj_flt (major faults). By focusing on these indicators, the project seeks to create a reliable model capable of predicting the nature of a given file, whether it is malware or benign.

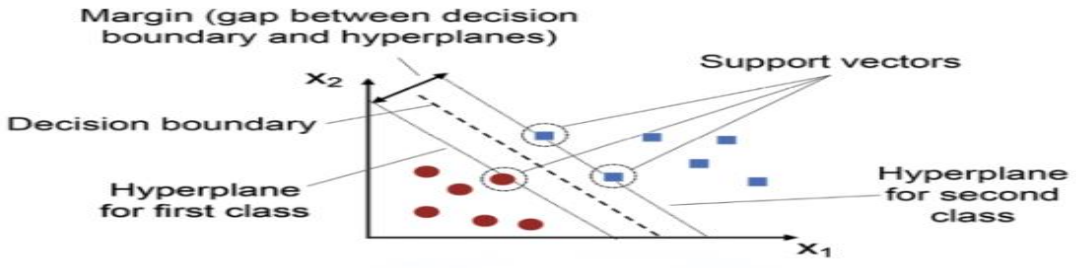
To achieve this goal, the project will utilize three well-established machine learning algorithms: Random Forest, Logistic Regression, and Support Vector Machines (SVM). Each of these algorithms offers unique strengths that make them suitable for this classification task:

1. **Random Forest**: This ensemble learning method is highly effective in classification tasks due to its ability to build multiple decision trees and output the mode of the classes (classification) of the individual trees. Random Forest is particularly valued for its high accuracy, ability to handle datasets with high dimensionality, and robustness against overfitting. It works well even when the data contains noisy or missing values, making it an ideal candidate for this project.



2. **Logistic Regression**: Logistic Regression is a statistical model that excels in binary classification problems. By modeling the probability of a binary outcome based on one or more predictor variables, Logistic Regression provides a straightforward and interpretable model. Its effectiveness in situations where the relationship between the dependent and independent variables is linear makes it a strong contender for this classification task.

3. **Support Vector Machines (SVM)**: SVM is a powerful classification algorithm known for its ability to find the optimal hyperplane that best separates the classes in the feature space. SVM is particularly effective in high-dimensional spaces and can handle cases where the number of dimensions exceeds the number of samples. Its flexibility and precision in drawing boundaries between classes make it an invaluable tool for distinguishing between malware and benign files.



The project will involve data preprocessing, feature selection, and the application of the chosen algorithms to classify files as either malware or benign. The models' performances will be evaluated using metrics such as accuracy, precision, recall, and F1-score. Additionally, techniques like cross-validation and hyperparameter tuning will be employed to optimize the models and enhance their generalizability to new, unseen data.

By harnessing machine learning and a rich dataset of system-level features, this project aspires to contribute to cybersecurity efforts by providing an effective tool for malware detection. The resulting classification system could play a crucial role in protecting systems from potential threats, thereby enhancing overall cybersecurity measures.

Exploratory Data Analysis (EDA)

Exploratory data analysis is a statistical way of evaluating data sets to summarise their essential properties, frequently utilising statistical graphics and other data visualisation techniques. EDA can be used with or without a statistical model, however it is primarily used to examine what the data can tell us without formal modelling or hypothesis testing.

2.01 Cyber Security Data set contains 10000 rows and 34 columns

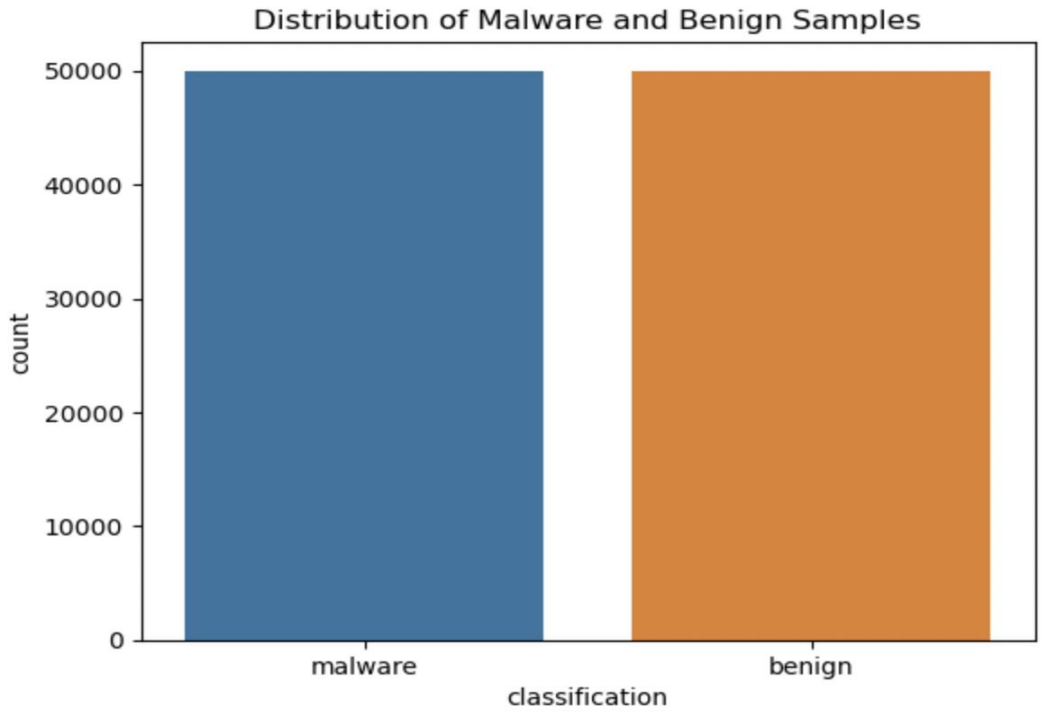
	hash	millisecond	classification	state	usage_counter	prio	static_prio	normal_prio	policy	vm_pgoff
0	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	0	malware	0	0	3069378560	14274	0	0	0
1	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	1	malware	0	0	3069378560	14274	0	0	0
2	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	2	malware	0	0	3069378560	14274	0	0	0
3	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	3	malware	0	0	3069378560	14274	0	0	0
4	42fb5e2ec009a05ff5143227297074f1e9c6c3ebb9c914...	4	malware	0	0	3069378560	14274	0	0	0

2.02 Number of Instances, features and Number of Instances from each class

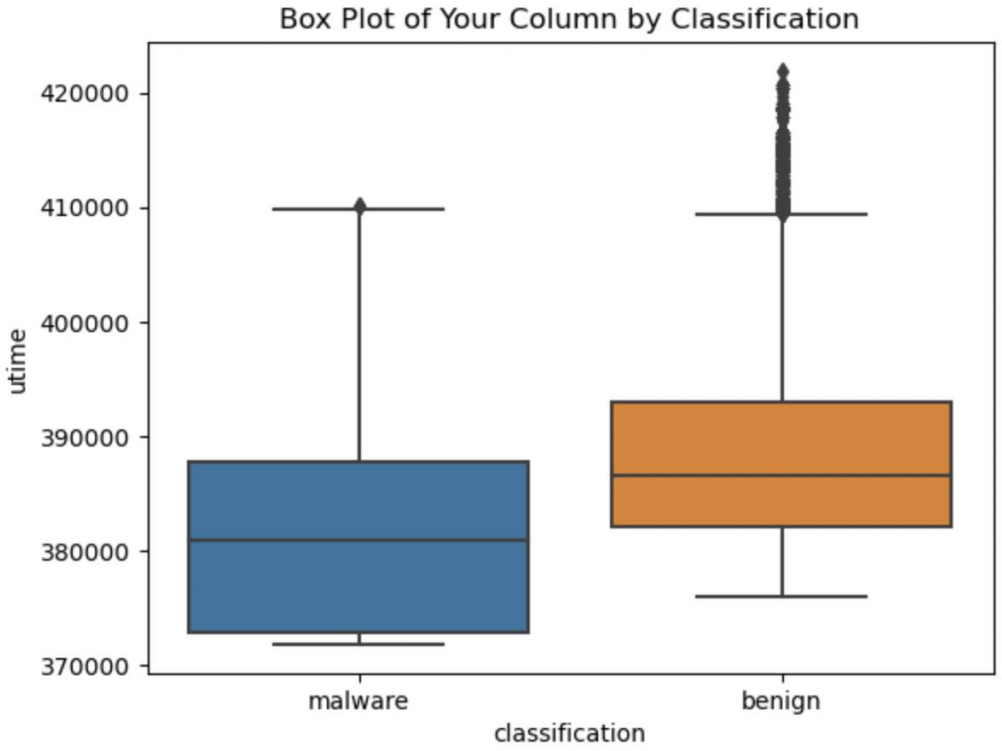
```
Number of instances: 100000
Number of features: 34

Instances for each class:
malware    50000
benign     50000
Name: classification, dtype: int64
```

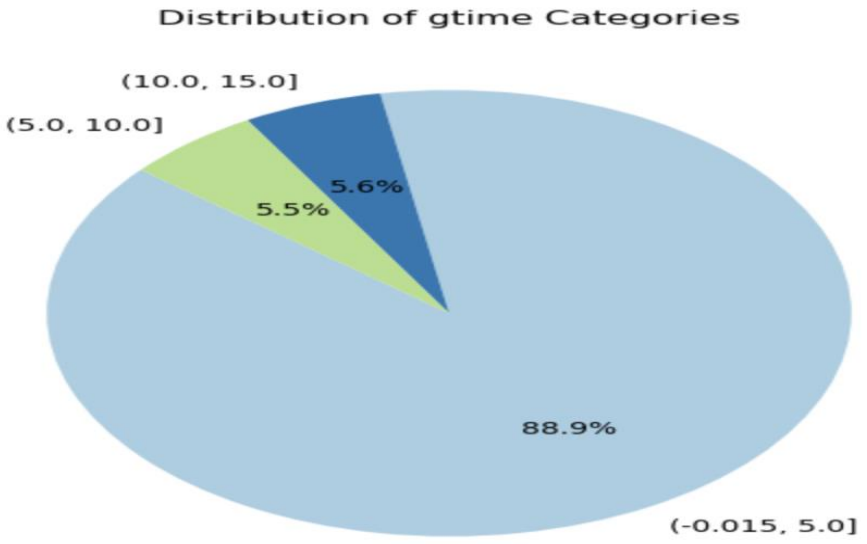
2.03 Distribution of Malware and Benign Samples



2.04 Box Plot of classifications against utime



2.05 Distribution of gtime Categories

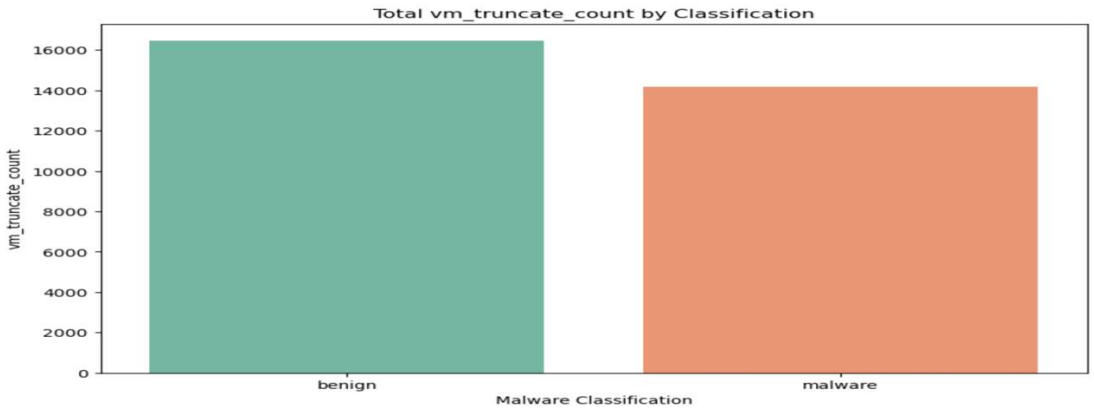


Distribution plots shows that the range between -0.15 and 5.0 have the majority of gtime of about 88.9%.

2.06 Total Major Page Faults and Total Minor page Fault by Classification

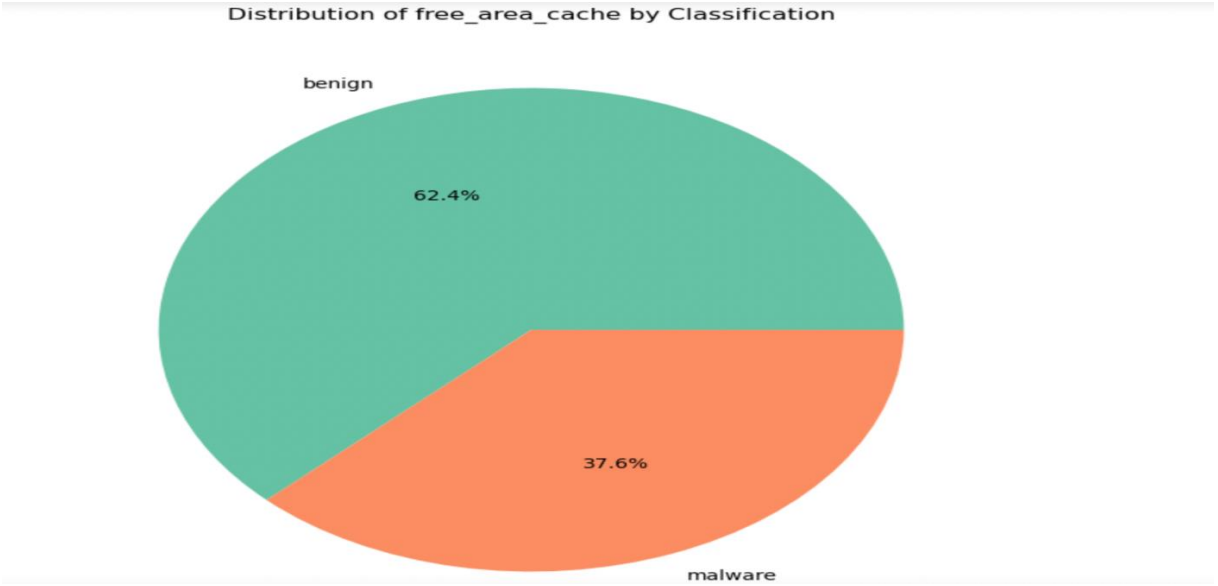
While the plot shows Maj_flt for benign and malware class, the malware has a slightly higher total minor page faults than benign.

2.07 Total Major Page Faults and Total Minor page Fault by Classification



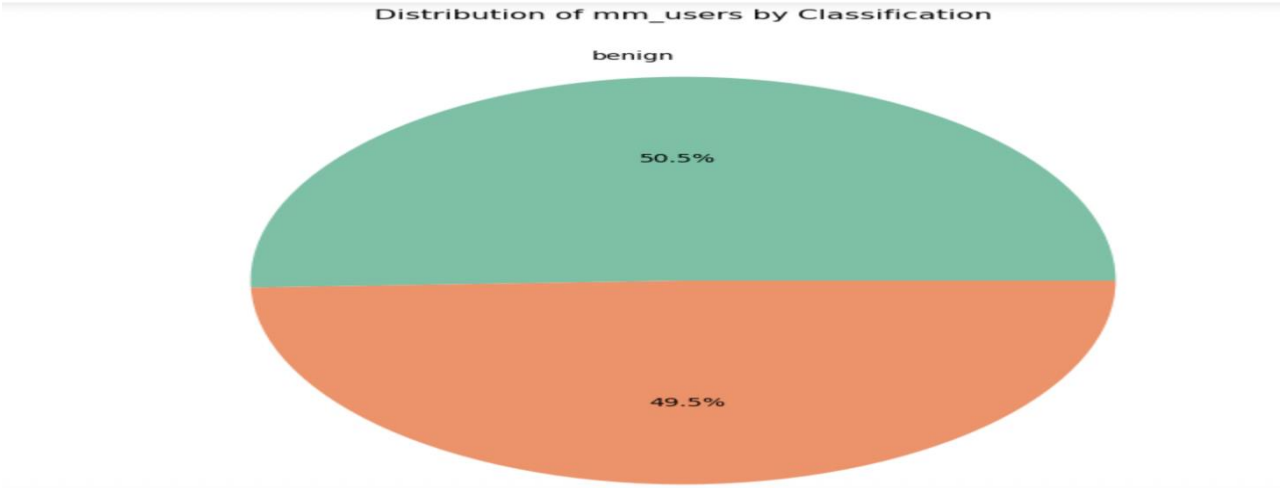
The comparison of Total vm_truncate_count showed benign having a higher count than malware.

2.08 Distribution of free_area_cache by Classification



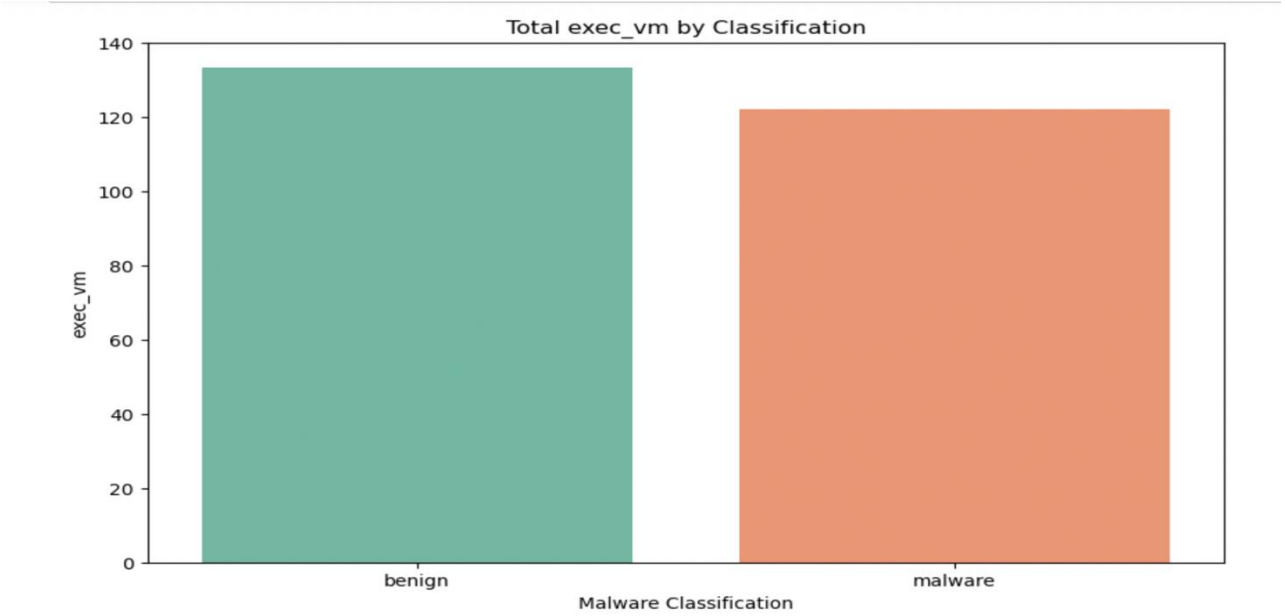
By free_area _cache, the dataset shows 62.4% for benign while 37.6% for Malware.

2.09 Distribution of mm_users by Classification



Distribution by class of mm_users shows 50.5% for benign while 49.5% show

2.10 Total execution on virtual machine by Classification



MODELLING APPROACH

3.1. Random Forest

Why Random Forest?

Random Forest is an ensemble learning method that is particularly well-suited for classification tasks. It combines the output of multiple decision trees to make a final prediction, which reduces the risk of overfitting—a common problem in machine learning. This algorithm is highly accurate, especially when dealing with large datasets, and is known for its robustness and ability to handle a high number of input features.

How It Works

Random Forest operates by creating a multitude of decision trees during training and outputs the mode of the classes (classification) of the individual trees. The method employed is known as Bagging (Bootstrap Aggregation), where multiple subsets of the data are created by sampling with replacement. Each decision tree in the forest is trained on a different subset of the data, and the final prediction is made by aggregating the predictions from all the trees, usually through majority voting.

For example, in the dataset containing features such as `utime` (user time), `gtime` (guest time), `cgtime` (cgroup time), `stime` (system time), `static_prio` (static priority), `prio` (dynamic priority), `normal_prio` (normal priority), `min_flt` (minor faults), and `maj_flt` (major faults), Random Forest would build multiple decision trees based on these features. Each tree would split the data based on different criteria (e.g., whether `utime` is greater than a certain threshold) and classify the software as malware or benign. The final classification is made based on the majority vote of all the trees.

Advantages

- **Accuracy**: Random Forest is known for its high accuracy, especially with large datasets.
- **Robustness**: The ensemble approach reduces the risk of overfitting and increases model stability.
- **Scalability**: It can handle large datasets with numerous features efficiently.

3.2. Logistic Regression

Why Logistic Regression?

Logistic Regression is a widely-used statistical method for binary classification tasks. It is simple, interpretable, and effective when the relationship between the dependent variable and the independent variables is approximately linear. Logistic Regression models the probability that a given input belongs to a particular class (e.g., malware or benign).

How It Works

Logistic Regression estimates the probability that a given input belongs to a certain class using the logistic function, also known as the sigmoid function. The sigmoid function outputs a probability

value between 0 and 1, which is then used to classify the input. For example, if the probability of the input being malware is greater than 0.5, the model classifies it as malware; otherwise, it is classified as benign.

The logistic model is defined as:

$$P(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+\beta_1 X_1+\beta_2 X_2+...+\beta_n X_n)}}$$

Where:

- $P(Y = 1|X)$ is the probability that the dependent variable Y is 1 (e.g., malware).
- $\beta_0, \beta_1, ..., \beta_n$ are the coefficients of the model.
- $X_1, X_2, ..., X_n$ are the independent variables (e.g., system-level features).

Advantages

- **Interpretability**: The model coefficients provide insights into the importance of each feature.
- **Simplicity**: It is easy to implement and computationally efficient.
- **Probabilistic Output**: Logistic Regression provides probability estimates, which can be useful for decision-making.

3.3 Support Vector Machines (SVM)

Why SVM?

Support Vector Machines are powerful for classification tasks, especially when the data is high-dimensional. SVM works by finding the optimal hyperplane that separates the data into different classes with the maximum margin. This algorithm is particularly effective when the classes are not linearly separable, as it can utilize kernel functions to map the input features into higher-dimensional spaces.

How It Works

SVM attempts to find the hyperplane that best separates the classes of interest (malware and benign). The optimal hyperplane is the one that maximizes the margin, which is the distance between the hyperplane and the nearest data points from either class, known as support vectors. In cases where the data is not linearly separable, SVM employs kernel functions (e.g., polynomial, radial basis function) to transform the data into a higher-dimensional space where a linear separator can be found.

Mathematically, the decision function for SVM is:

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right)$$

Where:

- x_i are the support vectors.
- y_i are the class labels of the support vectors.
- α_i are the Lagrange multipliers.
- $K(x_i, x)$ is the kernel function.
- b is the bias term.

Advantages

- **Effective in High-Dimensional Spaces**: SVM performs well when the number of features is large.
- **Versatility**: The use of different kernel functions allows SVM to handle various types of data.
- **Robustness**: SVM is less prone to overfitting, especially in cases where the number of features exceeds the number of observations.

DATA PREPARATION

Before applying these models, the data needs to be carefully prepared. In our dataset, we have several system-level features such as `utime`, `gtime`, `cgtime`, `stime`, `static_prio`, `prio`, `normal_prio`, `min_flt`, and `maj_flt`. The first step in data preparation is to remove any irrelevant columns, such as the hash column that might not provide meaningful information for classification.

Next, it is essential to inspect the data for correlations. Features with high correlations to the target variable (malware or benign) are particularly valuable for the models. However, it is also important to consider multicollinearity, where multiple features are highly correlated with each other, as this can affect model performance.

Another crucial step is to address class imbalance. In many real-world datasets, the number of samples belonging to each class is not equal. For example, there might be many more benign software instances than malware instances. This imbalance can lead to biased models that favor the majority class. WE ensured we haveequal balances pof both the benign and Malware instances.

Finally, the data is split into training and testing sets, we employed the 80-20 split. This ensures that the models are trained on one portion of the data and evaluated on another to assess their performance.

4.1 Model Implementation and Evaluation

For each of the models—Random Forest, Logistic Regression, and SVM—the implementation begins with selecting the best hyperparameters. This is achieved using techniques like Grid Search Cross-Validation (Grid Search CV), which systematically tests combinations of parameters to find the most effective ones.

Once the optimal parameters are determined, the models are trained on the training data. The performance of each model is then evaluated using the testing set. Common metrics for evaluation include accuracy, precision, recall and F1-score. These metrics provide insights into the models' ability to correctly classify malware and benign software.

Random Forest: For Random Forest, we focus on parameters such as the number of trees (n_estimators), the maximum depth of the trees, and the minimum number of samples required to split a node.

Logistic Regression: For Logistic Regression, we consider the regularization parameter (C) and the choice of penalty (e.g., L1 or L2).

SVM: For SVM, the key parameters include the choice of kernel (linear, polynomial, RBF), the regularization parameter (C), and the kernel coefficient (gamma).

After training, the models are tested on the unseen data to determine their generalization capabilities. The results are compared to identify which model performs best for the task of malware classification

4.2 Procedure

Firstly, we ascertained the number of instances and features, then checked to ensure that the malware and the benign instances where equal to avoid bias.

```
# Number of instances (rows)
num_instances = df.shape[0]

# Number of features (columns excluding the class column)
num_features = df.shape[1] - 1  # Subtract 1 to exclude the class column

# Number of instances for each class
class_distribution = df['classification'].value_counts()

# Output the results
print(f"Number of instances: {num_instances}")
print(f"Number of features: {num_features}")
print("\nInstances for each class:")
print(class_distribution)
```

```
Number of instances: 100000
Number of features: 34

Instances for each class:
malware    50000
benign     50000
Name: classification, dtype: int64
```

We ran a summary statistic to identify features whose values are ‘0’ or not useable in our analysis. The identified features were dropped.

```
# Assuming you've already loaded your data into a DataFrame called df
summary_statistics = df.describe()

# Display the summary statistics
print(summary_statistics)

```

```
        millisecond         state  usage_counter          prio  \
count  100000.000000  1.000000e+05       100000.0  1.000000e+05
mean      499.500000  1.577683e+05            0.0  3.069706e+09
std       288.676434  9.361726e+05            0.0  2.963061e+05
min         0.000000  0.000000e+00            0.0  3.069190e+09
25%       249.750000  0.000000e+00            0.0  3.069446e+09
50%       499.500000  0.000000e+00            0.0  3.069698e+09
75%       749.250000  4.096000e+03            0.0  3.069957e+09
max       999.000000  4.326605e+07            0.0  3.070222e+09

         static_prio  normal_prio    policy  vm_pgoff  vm_truncate_count  \
count  100000.000000     100000.0  100000.0  100000.0      100000.000000
mean    18183.900070          0.0       0.0       0.0       15312.739510
std      4609.792765          0.0       0.0       0.0        3256.475008
min     13988.000000          0.0       0.0       0.0        9695.000000
25%     14352.000000          0.0       0.0       0.0       12648.000000
50%     16159.000000          0.0       0.0       0.0       15245.000000
75%     22182.000000          0.0       0.0       0.0       17663.000000
max     31855.000000          0.0       0.0       0.0       27157.000000

       task_size  ...          nivcsw        min_flt        maj_flt  \
count   100000.0  ...  100000.000000  100000.000000  100000.000000
mean         0.0  ...      32.991160       2.053130     117.920240
std          0.0  ...      52.730176      13.881382       3.116892
min          0.0  ...       0.000000       0.000000     112.000000
25%          0.0  ...       1.000000       0.000000     114.000000
```

we carried out an EDA on the dataset as seen in the prior section to ascertain things such as relationships and proportions between features.

We defined our features, identified numerical and categorical features and created preprocessors for them. Pipelines for the classifier was also created before splitting the data into training and test sets. We then initialized the classifiers , trained the classifier, made predictions and then finally evaluated the classifier. The process was carried out for the random forest and repeated for the logistic regression and SVM.

```
import seaborn as sns


# Define features and target
X = df2.drop('classification', axis=1)  # Assuming 'classification' is the target column
y = df2['classification']

# Identify numerical and categorical columns
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns
categorical_cols = X.select_dtypes(include=['object', 'category']).columns

# Create preprocessor for numerical and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_cols),
        ('cat', OneHotEncoder(), categorical_cols)
    ])

# Create pipelines for each classifier
def create_pipeline(classifier):
    return Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', classifier)
    ])

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize classifiers
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)


# Create pipelines for each classifier
rf_pipeline = create_pipeline(rf_classifier)
```

```


# Create pipelines for each classifier
rf_pipeline = create_pipeline(rf_classifier)


# Train classifiers
rf_pipeline.fit(X_train, y_train)


# Make predictions
rf_predictions = rf_pipeline.predict(X_test)


# Evaluate classifiers
def evaluate_model(predictions, true_labels, model_name):
    print(f"\n{model_name} Classification Report:")
    print(classification_report(true_labels, predictions))
    print(f"{model_name} Confusion Matrix:")
    cm = confusion_matrix(true_labels, predictions)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malware'], yticklabels=['Benign',
    plt.title(f'Confusion Matrix for {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

evaluate_model(rf_predictions, y_test, 'Random Forest')


```

CONCLUSION

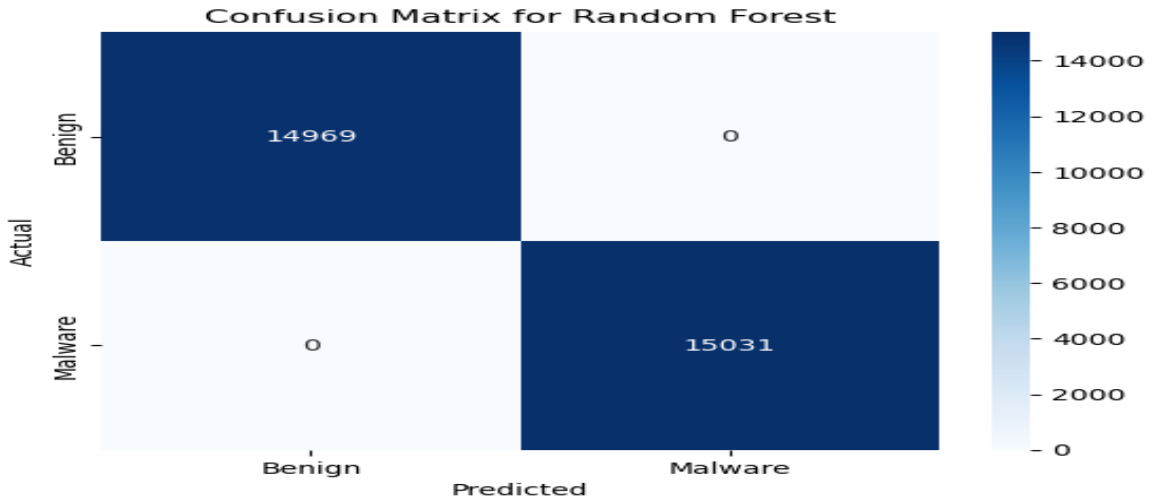
5.1 RANDOM FOREST

```
Random Forest Classification Report:
              precision    recall  f1-score   support

      benign       1.00      1.00      1.00     14969
     malware       1.00      1.00      1.00     15031

    accuracy                           1.00     30000
   macro avg       1.00      1.00      1.00     30000
weighted avg       1.00      1.00      1.00     30000

Random Forest Confusion Matrix:
```



The classification report indicates that the Random Forest model performs flawlessly in distinguishing between "benign" and "malware" instances. Both precision and recall are perfect at 1.00, showing no errors in predictions. The F1-score is also 1.00, indicating a perfect balance between precision and recall. With 100% accuracy across 30,000 samples, the model accurately classifies every instance, showcasing exceptional performance in this binary classification task.

The confusion matrix for the Random Forest model displays perfect classification results:

- **Benign (True Positives)**: All 14,969 "Benign" instances were correctly identified as "Benign."
- **Malware (True Negatives)**: All 15,031 "Malware" instances were accurately classified as "Malware."
- **False Positives**: There were no incorrect classifications of "Malware" as "Benign."
- **False Negatives**: There were no incorrect classifications of "Benign" as "Malware."

This matrix underscores the model's flawless performance.

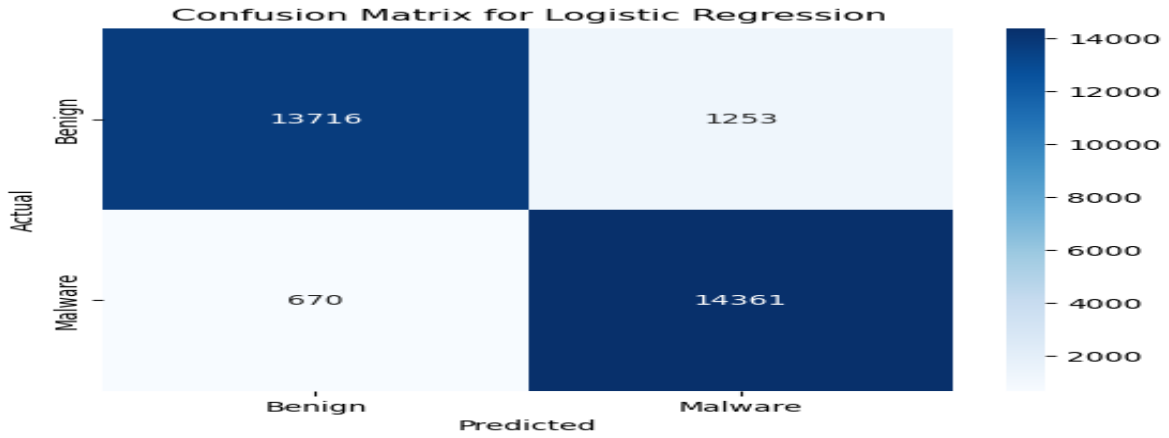
5.2 LOGISTIC REGRESSION

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

      benign       0.95      0.92      0.93     14969
     malware       0.92      0.96      0.94     15031

    accuracy                           0.94     30000
   macro avg       0.94      0.94      0.94     30000
weighted avg       0.94      0.94      0.94     30000

Logistic Regression Confusion Matrix:
```



The Logistic Regression model demonstrates solid performance in classifying "benign" and "malware" instances. It achieves a precision of 0.95 for "benign" and 0.92 for "malware," with recall values of 0.92 and 0.96, respectively. The F1-scores are 0.93 for "benign" and 0.94 for "malware." With an overall accuracy of 0.94, the model correctly classifies 94% of the 30,000 samples, showing balanced effectiveness across both categories.

The Logistic Regression model's performance is summarized in the above confusion matrix:

Out of the total instances, 13,716 "Benign" cases were correctly identified as "Benign", and 14,361 "Malware" cases were accurately classified as "Malware".

However, the model misclassified 1,253 "Benign" instances as "Malware" (false positives), and 670 "Malware" instances as "Benign" (false negatives).

While the model shows overall good performance, there's room for improvement in reducing these misclassifications, particularly in distinguishing between true "Benign" and "Malware" cases.

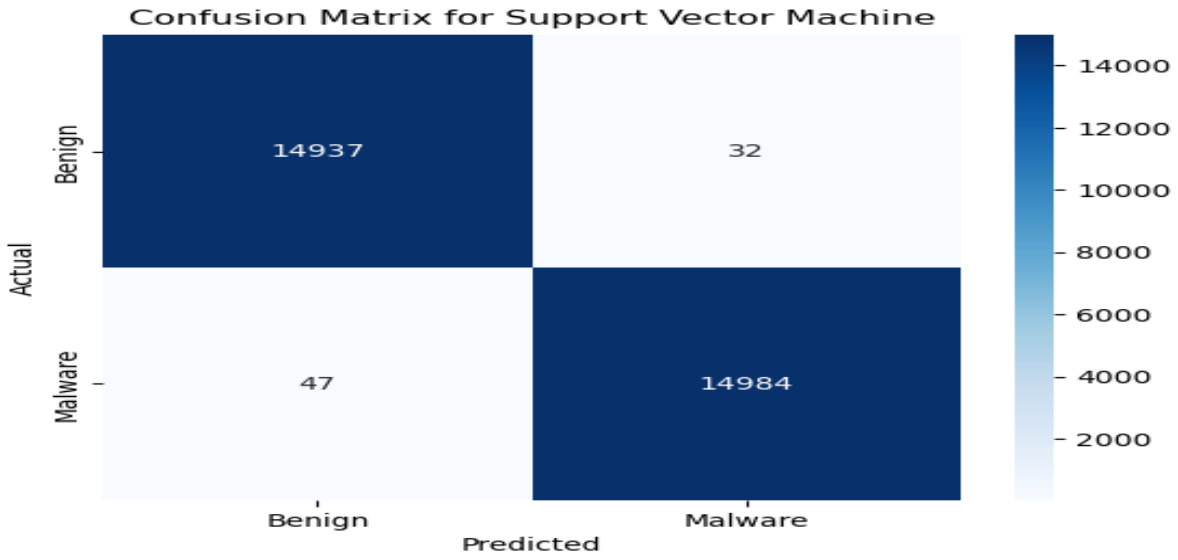
5.3 SVM

```
SVM Classification Report:
              precision    recall  f1-score   support

      benign       1.00      1.00      1.00     14969
     malware       1.00      1.00      1.00     15031

    accuracy                           1.00     30000
   macro avg       1.00      1.00      1.00     30000
weighted avg       1.00      1.00      1.00     30000

SVM Confusion Matrix:
```



The SVM model delivers flawless performance in distinguishing between "benign" and "malware" instances, achieving perfect scores of 1.00 for precision, recall, and F1-score in both categories. The model's accuracy is 1.00, meaning it correctly classified all 30,000 samples. Both the macro and weighted averages are also 1.00, indicating exceptional and consistent performance across the entire dataset.

The Support Vector Machine (SVM) confusion matrix shows strong performance in classifying instances. True positives (Benign): 14,937, true negatives (Malware): 14,984. The model had minimal false positives (32) and false negatives (47), indicating effective differentiation between "Benign" and "Malware" cases. Overall, the SVM demonstrates high accuracy with a low rate of misclassification, suggesting it efficiently identifies both classes in the dataset.

REFERENCES

Logunova, I. (2022). Random Forest Classifier: Basic Principles and Applications. *Serokell*.

Swaminathan, S. (2018). Logistic Regression — Detailed Overview. *Towards Data Science*.

Kumar, A. (2023). Support Vector Machine (SVM) Python. *Data Science Machine Learning Deep Learning Statistics Gneerative AI Courses Admissions Interview Questions Educational Presentations Privacy policy Contact us Analytics Yogi*.