

Malware Detection Using Machine Learning: A Comparative Analysis of Classifiers

1202-02 – DATA ANALYSIS TOOLS ANALYTICS



Prof. Sk. Md. Mizanur Rahman

Data-1202-02 - Group 5

December 9, 2024

Akshaya Bhalikha DS – 100936892

Bhavanish S Nair – 100936855

Kapeesh Vinay Bhatt – 100940689

Gift Clara Princely Israel Jebakumar – 100924338

Harshith Nagaraja - 100922585

## Content Page

### Information about the Dataset …………………………………………………………………………

### Dataset Splitting …………………………………………………………………………

### Building the Classifiers …………………………………………………………………………

### Training the Classifiers …………………………………………………………………………

### Testing the Classifiers …………………………………………………………………………

### Results and Comparison …………………………………………………………………………

Explanation of the Code …………………………………………………………………………

# 1. Information About the Dataset

The dataset contains **100,000 rows** and several numerical features. The goal is to classify samples into **malware** or **benign**. The target variable is **classification**, which has two categories:

* **Malware**: 50,000 instances
* **Benign**: 50,000 instances

To ensure accurate results, unnecessary columns with constant values were removed.

# 2. Dataset Splitting

The dataset was divided into:

* **Training set (80%)**: Used to train the models.
* **Testing set (20%)**: Used to evaluate the models on unseen data.

**Code Explanation:**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X[numerical\_columns], y, test\_size=0.2, random\_state=42, stratify=y)

1. **train\_test\_split**: Divides the dataset into training and testing sets.
2. **stratify=y**: Ensures the class distribution is the same in both sets.
3. **test\_size=0.2**: Keeps 20% of the data for testing.

# 3. Building the Classifiers

We used three machine learning methods to classify the data:

## a. Logistic Regression

* A simple linear model that tries to find a boundary between classes.
* Works well for datasets where the relationship between features and the target is linear.

**Code Explanation:**

from sklearn.linear\_model import LogisticRegression

log\_reg = LogisticRegression(max\_iter=1000, random\_state=42)

1. **max\_iter=1000**: Ensures the model gets enough iterations to converge.
2. **random\_state=42**: Ensures results are reproducible.

## b. Random Forest

* An ensemble model that combines multiple decision trees.
* Each tree makes a prediction, and the forest takes the majority vote.
* Effective for complex datasets with non-linear patterns.

**Code Explanation:**

from sklearn.ensemble import RandomForestClassifier

random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

1. **n\_estimators=100**: Builds 100 decision trees.
2. **random\_state=42**: Makes results reproducible.

## c. K-Nearest Neighbors (KNN)

* A non-parametric method that classifies data points based on their closest neighbors.
* Works well when similar samples are near each other in feature space.

**Code Explanation:**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=5)

1. **n\_neighbors=5**: Considers the 5 nearest neighbors when making predictions.

# 4. Training the Classifiers

Each model was trained using the **training set**.

**Code Explanation:**

log\_reg.fit(X\_train, y\_train\_encoded)

random\_forest.fit(X\_train, y\_train\_encoded)

knn.fit(X\_train, y\_train\_encoded)

1. **.fit**: Fits the model to the training data.
2. **y\_train\_encoded**: Encoded target variable to make it numeric.

# 5. Testing the Classifiers

The models were evaluated on the **testing set**, and performance metrics like accuracy, precision, recall, and F1-score were calculated.

**Code Explanation:**

log\_reg\_pred = log\_reg.predict(X\_test)

random\_forest\_pred = random\_forest.predict(X\_test)

knn\_pred = knn.predict(X\_test)

1. **.predict**: Generates predictions for the testing set.

To evaluate performance:

from sklearn.metrics import classification\_report

print(classification\_report(y\_test\_encoded, log\_reg\_pred))

1. **classification\_report**: Summarizes model performance, including precision, recall, and F1-score.

# 6. Results and Comparison

Here’s a summary of the results for each model:

**Logistic Regression**

* Accuracy: **74%**
* Strength: Detected malware well (84% recall).
* Weakness: Struggled to identify benign samples (64% recall).
* This model is suitable for simpler datasets but struggled with the complexity here.

**Random Forest**

* Accuracy: **100%**
* Strength: Perfectly classified both malware and benign samples.
* Random Forest’s ensemble approach makes it very robust for complex datasets.

**K-Nearest Neighbors (KNN)**

* Accuracy: **99.73%**
* Confusion Matrix:
  + Correct classifications: **9964 benign**, **9981 malware**
  + Misclassifications: **36 benign as malware**, **19 malware as benign**
* KNN performed very well but had a few errors, likely due to overlaps in feature space.

# 7. Explanation of the code:

**1. Data Preparation**

**Code for Uploading and Loading Data**

from google.colab import files

uploaded = files.upload()

data = pd.read\_csv('dataset.csv')

* **files.upload()**: Allows you to upload files from your computer into the Google Colab environment.
* **pd.read\_csv('dataset.csv')**: Reads the uploaded dataset into a pandas DataFrame for further processing.

**Separating Features and Target**

target\_column = 'classification'

X = data.drop(columns=[target\_column])

y = data[target\_column]

* **drop(columns=[target\_column])**: Removes the target column (classification) from the dataset, leaving only the features (X).
* **y = data[target\_column]**: Extracts the target column into the variable y for prediction.

**Removing Constant Columns**

for i in X.columns:

if X[i].nunique() < 2:

X.drop(columns=[i], inplace=True)

numerical\_columns.remove(i)

* **nunique()**: Counts the number of unique values in each column. If it’s less than 2, the column has constant values and is dropped.
* **inplace=True**: Makes the change directly in the DataFrame without needing to reassign it.

**2. Exploratory Data Analysis (EDA)**

**Descriptive Statistics**

numerical\_columns = X.select\_dtypes(include=['int64', 'float64']).columns

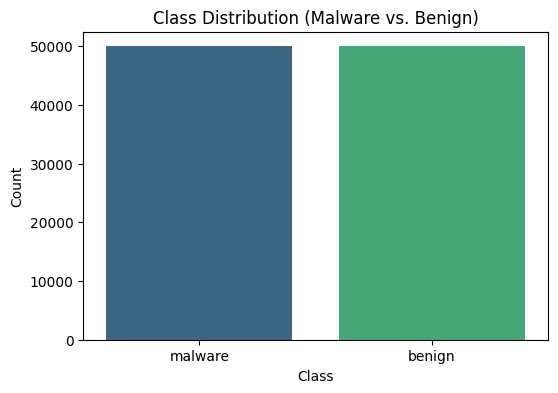
X[numerical\_columns].describe()

* **select\_dtypes**: Selects only numerical columns (int64 and float64).
* **describe()**: Provides summary statistics like mean, median, standard deviation, min, and max for numerical features. This helps understand the data’s distribution and identify potential outliers.

**Class Distribution**

sns.countplot(x=y, palette='viridis')

* **countplot**: Visualizes the number of samples for each class (malware and benign).
* **palette='viridis'**: Adds color to the plot for better visibility.

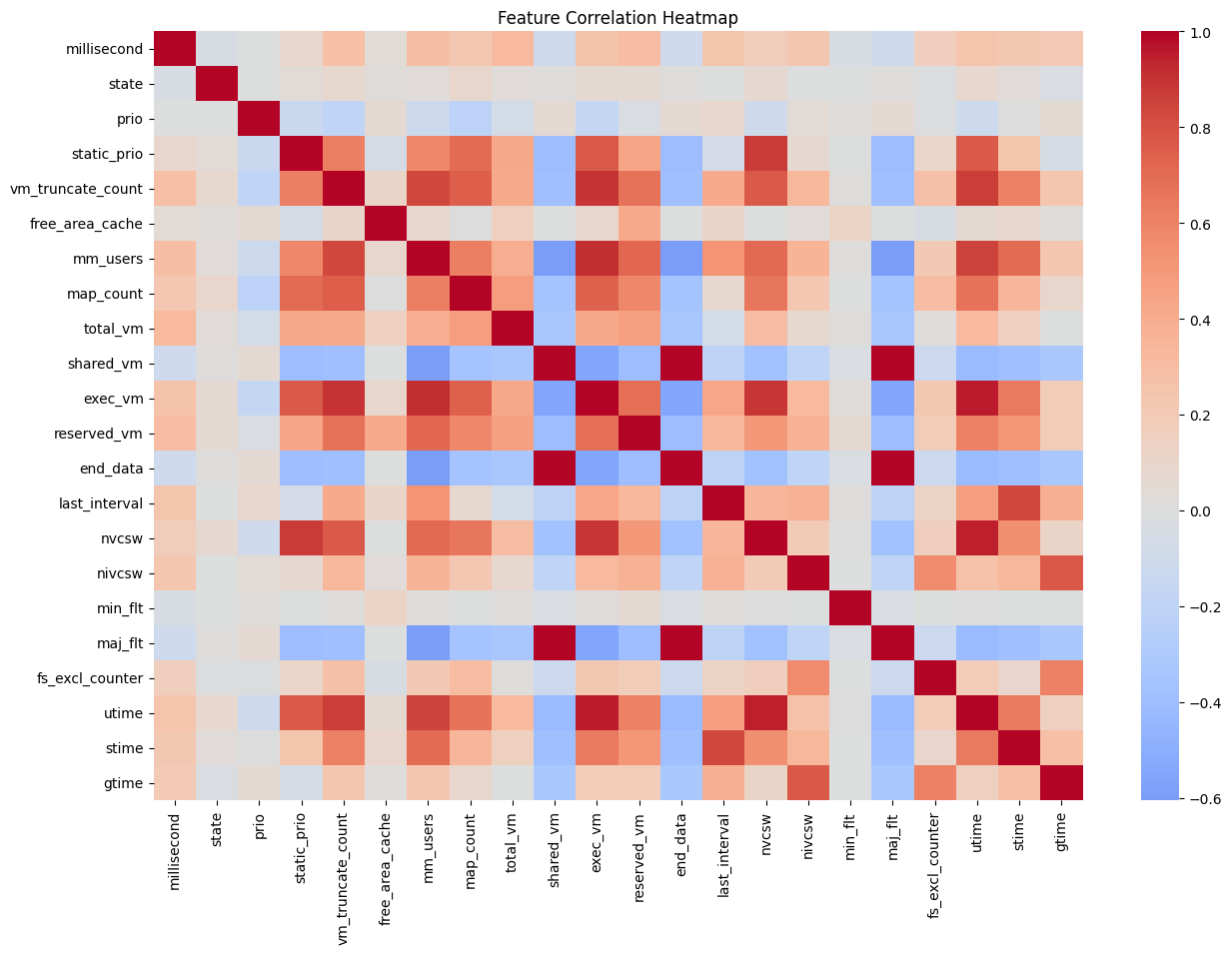


**Correlation Heatmap**

correlation\_matrix = X[numerical\_columns].corr()

sns.heatmap(correlation\_matrix, cmap='coolwarm', center=0, annot=False, cbar=True)

* **corr()**: Computes pairwise correlations between numerical features.
* **heatmap()**: Visualizes these correlations as a matrix, with colors representing the strength and direction of relationships (e.g., red for strong positive correlation, blue for strong negative correlation).



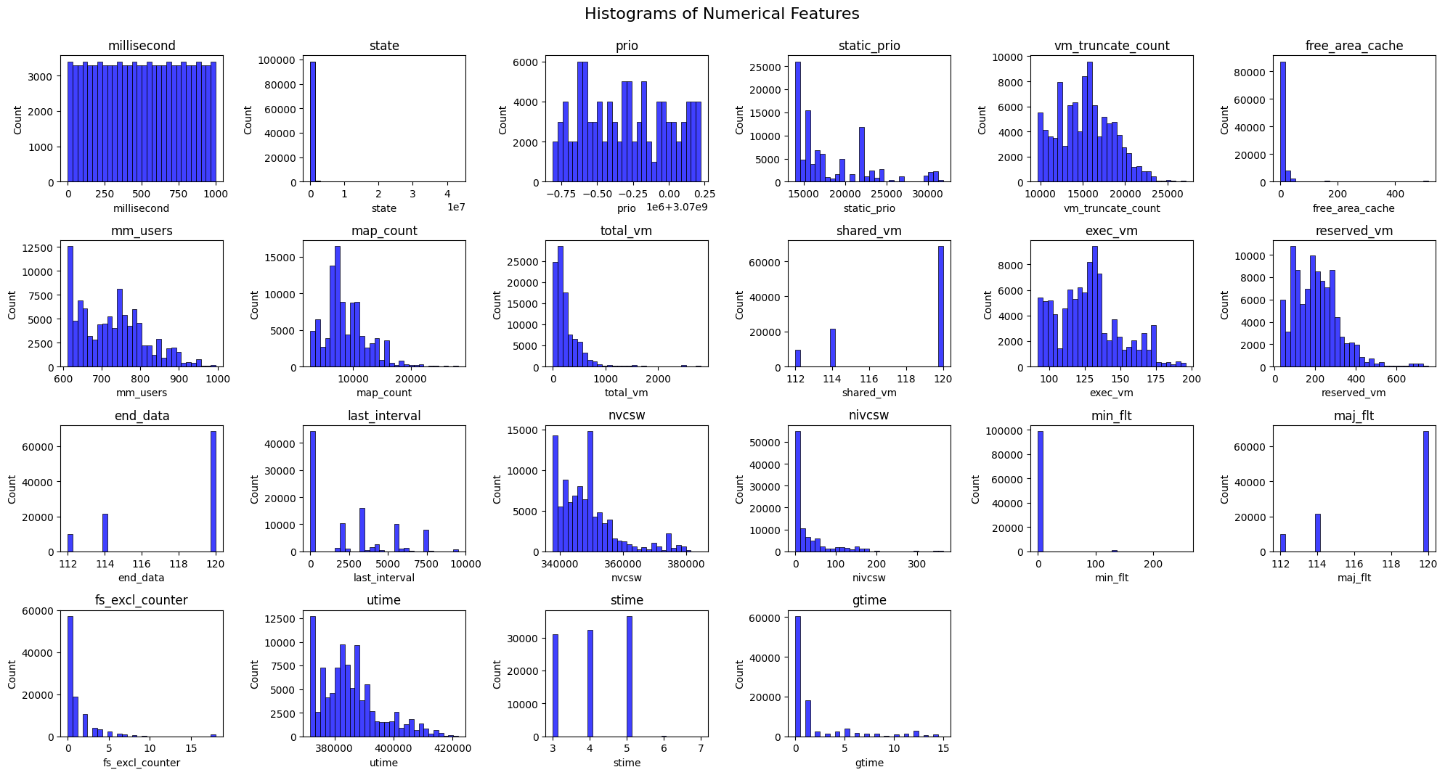
**Histograms**

for i, column in enumerate(numerical\_columns, 1):

plt.subplot(6, 6, i)

sns.histplot(X[column], bins=30, kde=False, color='blue')

* **enumerate(numerical\_columns, 1)**: Loops through numerical columns and their index, starting from 1.
* **subplot(6, 6, i)**: Creates a grid of plots with 6 rows and 6 columns to display histograms for each feature.



**3. Model Building**

**Logistic Regression**

log\_reg = LogisticRegression(max\_iter=1000, random\_state=42)

* **Logistic Regression** assumes a linear relationship between features and the target. It uses the sigmoid function to predict probabilities for each class.
* **max\_iter=1000**: Specifies the maximum number of iterations allowed to optimize the model.

**Random Forest**

random\_forest = RandomForestClassifier(n\_estimators=100, random\_state=42)

* **Random Forest** creates multiple decision trees and combines their results to make predictions.
* **n\_estimators=100**: Indicates the number of trees in the forest.

**K-Nearest Neighbors (KNN)**

knn = KNeighborsClassifier(n\_neighbors=5)

* **KNN** calculates the distance between a test point and its nearest training points. The class is assigned based on the majority vote of its neighbors.
* **n\_neighbors=5**: Considers the 5 nearest neighbors for classification.

**4. Model Training**

**Fitting the Models**

log\_reg.fit(X\_train, y\_train\_encoded)

random\_forest.fit(X\_train, y\_train\_encoded)

knn.fit(X\_train, y\_train\_encoded)

* **.fit(X\_train, y\_train\_encoded)**: Trains each model using the training data (X\_train) and corresponding labels (y\_train\_encoded).

**Label Encoding**

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

y\_train\_encoded = label\_encoder.fit\_transform(y\_train)

y\_test\_encoded = label\_encoder.transform(y\_test)

* **Label Encoding** converts class labels into numeric format (e.g., benign=0, malware=1). This is necessary for models that work with numerical inputs.

**5. Model Testing**

**Generating Predictions**

log\_reg\_pred = log\_reg.predict(X\_test)

random\_forest\_pred = random\_forest.predict(X\_test)

knn\_pred = knn.predict(X\_test)

* **.predict(X\_test)**: Uses the trained model to make predictions on the testing set.

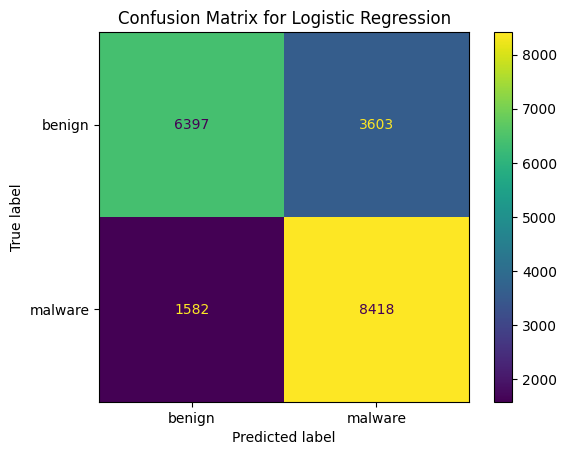
**Evaluation**

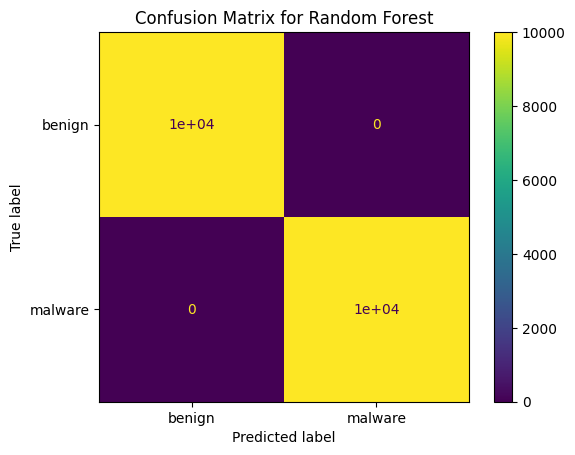
from sklearn.metrics import confusion\_matrix, classification\_report

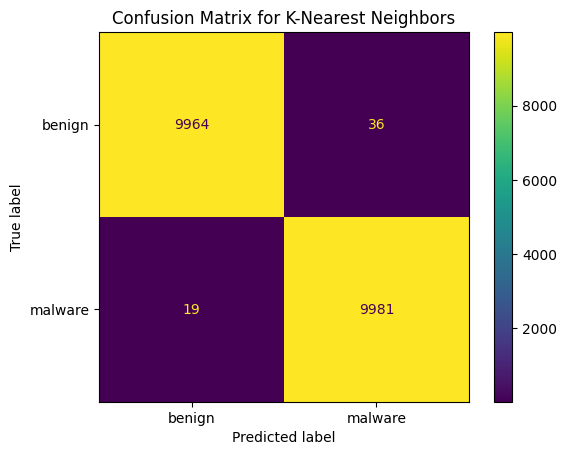
cm = confusion\_matrix(y\_test\_encoded, log\_reg\_pred)

print(classification\_report(y\_test\_encoded, log\_reg\_pred))

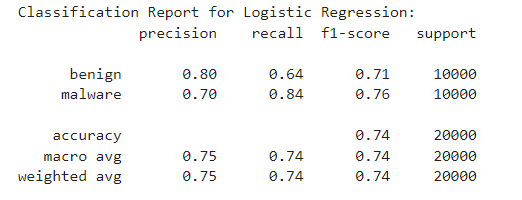
* **confusion\_matrix**: Outputs a matrix showing true positives, true negatives, false positives, and false negatives.

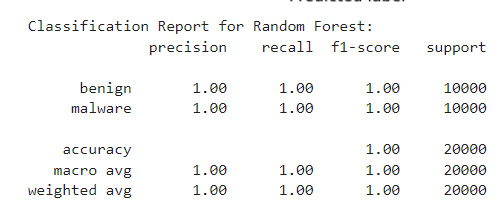


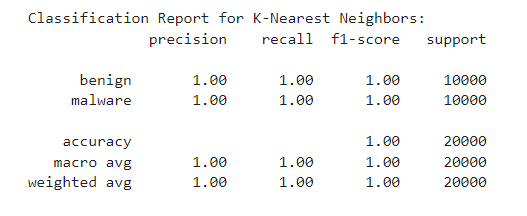




* **classification\_report**: Provides precision, recall, F1-score, and overall accuracy.







**Confusion Matrix Visualization**

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=label\_encoder.classes\_)

disp.plot(cmap='viridis')

* **ConfusionMatrixDisplay**: Displays the confusion matrix in a visually understandable format.

**6. Key Concepts and Tips**

* **Precision**: How many of the predicted positive cases are actually positive.
* **Recall**: How many actual positive cases were correctly predicted.
* **F1-Score**: Balances precision and recall into a single metric.
* **Feature Selection**: Removing constant or low-variance features helps improve model performance.
* **Scaling**: Some models like Logistic Regression may perform better if numerical features are scaled to similar ranges.

This level of explanation ensures the code is well understood and makes the implementation clear for learning and future use. Let me know if you'd like more elaboration!

# Conclusions

From this project:

1. **Random Forest** is the best choice, offering high accuracy and robustness.
2. **KNN** is a strong alternative but had minor misclassifications.
3. **Logistic Regression** works for simpler datasets but isn’t ideal for this scenario.
4. **Most preferred:** Random Forest is recommended for real-world use, especially if the dataset has complex relationships or overlaps.