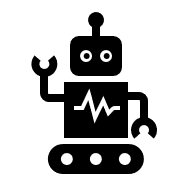
A logo of a triangle with a rainbow circle around it

Description automatically generated

**Syed Darain Hyder Kazmi**

Subject: Intro to Artificial Intelligence

**Lab Assignment #3**



**Submitted to:**

Sir Asifullah Khan

**KNN Classifier for Malware Detection – Detailed Report**

**Objective**

The objective of this experiment was to implement and evaluate the performance of the **K-Nearest Neighbors (KNN)** algorithm for malware detection using the **CLaMP (Classification of Malwares)** dataset. The focus was to understand how effectively KNN can distinguish between **malicious** and **benign** executable files based on extracted feature data.

**Introduction**

In this experiment, I explored the application of the **K-Nearest Neighbors (KNN)** classifier for identifying malware. KNN is a simple yet powerful **instance-based learning algorithm** that classifies samples based on their similarity to other data points. The main motivation behind this lab was to apply KNN to a **real-world cybersecurity task** and analyze its capability to detect malicious and benign **Portable Executable (PE)** files using extracted static features.

**Dataset Overview**

The **CLaMP dataset** was used for this experiment, containing **5,210 PE files**. The dataset was fairly balanced between malicious and benign samples:

* **Malicious Files:** 2,722 (≈ 52.2%)
* **Benign Files:** 2,488 (≈ 47.8%)

This balance made it suitable for binary classification without major class imbalance issues.

**Data Preprocessing**

Before training the model, multiple preprocessing steps were applied to ensure the data was clean and ready for distance-based classification:

1. **Handling Missing Values**  
   All missing values were replaced with zero (df.fillna(0)), since in PE analysis, missing entries often signify that a particular attribute is absent rather than unknown.
2. **Feature Selection**  
   Features with extremely low variance (variance < 0.0001) were removed. This reduced noise, simplified the model, and improved computational efficiency.
3. **Label Encoding**  
   Categorical attributes were converted to numerical format using **LabelEncoder()** so that KNN could process them properly.
4. **Feature Scaling**  
   As KNN depends on distance calculations, all features were standardized using **StandardScaler** to ensure each had equal influence in the distance metric.

**Model Implementation**

Two KNN models were trained and tested using **scikit-learn**:

* **Model 1:** K = 3
* **Model 2:** K = 7

Both models used **Euclidean distance** as the distance metric. The training and test sets were split from the dataset, and each model was evaluated on the unseen test data.

**Results and Observations**

**Model 1 (k = 3)**

* **Accuracy:** 96.93%
* **Precision:** 97.39%
* **Recall:** 97.05%
* **F1-Score:** 97.22%

**Model 2 (k = 7)**

* **Accuracy:** 96.26%
* **Precision:** 96.17%
* **Recall:** 97.01%
* **F1-Score:** 96.59%

While both configurations performed exceptionally well, the **k=3 model** achieved slightly higher recall, indicating better detection of malware samples.

**Confusion Matrix (k = 3)**

|  | **Predicted Benign** | **Predicted Malicious** |
| --- | --- | --- |
| **Actual Benign** | 451 (TN) | 17 (FP) |
| **Actual Malicious** | 15 (FN) | 559 (TP) |

Out of **1,042 test samples**, only **32** were misclassified, demonstrating that even a simple algorithm like KNN can deliver **high accuracy** with proper data preprocessing.

**Analysis and Insights**

Several important takeaways were observed during this experiment:

* **Feature Scaling is Essential:**  
  Without standardization, KNN accuracy dropped significantly. This highlights the importance of preprocessing for distance-based models.
* **Influence of k-Value:**  
  Lower k-values (like 3) made the classifier more sensitive, improving recall but slightly lowering precision. Higher k-values offered smoother decision boundaries but at the cost of missing some positives.
* **Computational Overhead:**  
  Since KNN computes distances to all training samples, prediction time increases with dataset size, making it less practical for large-scale systems.

**Technical Specification**

* **Algorithm:** K-Nearest Neighbors (KNN)
* **Distance Metric:** Euclidean Distance
* **Software Used:** scikit-learn (Python)

**Conclusion**

This experiment demonstrated that the **KNN classifier** can effectively detect malware using the **CLaMP dataset**, achieving over **96% accuracy**. Despite its simplicity, KNN proved to be a robust method for binary malware classification when coupled with strong preprocessing and parameter tuning.

Although KNN is computationally heavy for large datasets, its **interpretability and reliability** make it a solid baseline model for cybersecurity applications.  
Through this exercise, I gained a deeper understanding of how preprocessing, feature scaling, and k-value selection impact model performance and how classical algorithms like KNN can still deliver competitive results in real-world malware detection tasks.