Android File Classifier Based on Feature Signatures

GROUP MEMBERS: HARIS HUMAYON GHUMMAN (21L-7526),
RANA ABDULLAH JAVED (21L-7553), ABU HURAIRA ZAHEER (21L-7586)

SECTION: BCS-7D

COURSE: Information Security

SUBMITTED TO: Sir Ammar Haider

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

Android File Classifier Based on Feature Signatures

Introduction

Android Analyzer is a machine learning-based application designed to classify Android APK files as **Suspicious (malicious)** or **Benign (safe)** based on their **feature signatures**. By analyzing API call behaviors and permissions, the tool efficiently identifies potentially harmful applications. This project uses machine learning techniques to distinguish between malicious and safe apps, achieving high accuracy and interpretability.

Datasets

The project utilizes two datasets sourced from the **Android Malware Dataset for Machine Learning** on Kaggle. The main dataset consists of **15,036 APK samples**, categorized as **9476 benign** and **5560 suspicious** applications. Each application is described by **215 features**, capturing API calls, permissions, and behaviors.

The second dataset maps the 215 features to their corresponding API call signatures, providing interpretability and insights into which features are indicative of suspicious or benign behavior.

Project Structure

1. Preprocessing

The dataset initially contained **15,036 samples**, but after removing missing values using `dropna()`, the final size was reduced to **15,031 records**. The cleaned dataset exhibited a significant class imbalance, with **63% benign** and **37% suspicious** samples (Figure 1). Such an imbalance can bias the model toward the majority class, making balancing techniques essential.

2. Feature Analysis

Feature analysis revealed that some features had strong correlations with suspicious behavior. For example, the `SEND_SMS` feature was present in nearly **100% of suspicious applications**, while it was rare in benign applications (Figure 2). This makes `SEND_SMS` a critical indicator of malicious intent. Other features, such as `READ_PHONE_STATE` and `TelephonyManager.getDeviceId`, were also more prevalent in suspicious apps, highlighting their significance in classification.

This analysis underscores the importance of understanding feature-level behavior, which directly impacts the effectiveness of the classification model.

3. Data Splitting and Balancing

The dataset was divided into training and testing sets, with the training set having distribution of **63.1% benign** and **36.9% suspicious** applications (Figure 3).

To ensure fair training, **Random Over-Sampling** was applied to the training set, resulting in a balanced dataset with equal numbers of benign and suspicious samples (Figure 4). This balancing process ensured that the model learned equally from both classes, preventing bias toward the majority class.

4. Model Training and Selection

Three machine learning models were trained and evaluated:

1. **Random Forest Classifier**: Achieved an accuracy of **98.6%**, with precision and recall scores of **99%** and **98%**, respectively. Its ability to handle complex feature interactions made it the most effective model (Figure 5)

2. **Logistic Regression**: Achieved an accuracy of **97.6%**. While simpler and interpretable, it was less effective in capturing intricate relationships between features (Figure 6).

3. **Decision Tree**: Achieved an accuracy of **97.1%**. Although interpretable, it was more prone to overfitting compared to Random Forest (Figure 7)

The Random Forest Classifier emerged as the best-performing model, with its robust performance across all metrics (Figure 8). The following table summarizes the results:

Model	Accuracy	Precision (Macro Avg)	Recall (Macro Avg)	F1-Score (Macro Avg)
Random Forest Classifier	98.6%	99%	98%	99%
Logistic Regression	97.6%	97%	98%	97%
Decision Tree	97.1%	97%	97%	97%

5. Evaluation

The **Random Forest Classifier**, trained on a comprehensive dataset of Android applications, was tested on the `whatsapp.apk` file to validate its predictive accuracy. The analysis began with feature extraction using **Androguard**, a reverse engineering tool for Android applications. Key features, including permissions, API calls, and intents, were extracted from the APK file and matched against a predefined set of 215 attributes used during model training. A binary feature vector was generated to represent the presence or absence of these attributes.

Permissions were extracted from the APK's manifest, identifying typical attributes like `SEND_SMS`, `READ_PHONE_STATE`, and `INTERNET`. API calls such as `Landroid.content.Context.registerReceiver` and `HttpPost.init` were also analyzed, alongside intents like `android.intent.action.SEND` and `android.intent.action.VIEW`. These features were then processed to form a comprehensive vector representation of the APK, capturing its behavioral patterns.

The feature vector was fed into the **Random Forest Classifier**, which predicted the APK as **benign**. This analysis of `whatsapp.apk` reinforces the model's reliability in distinguishing between suspicious and benign applications, showcasing its robustness and practical utility for Android malware detection. The classifier's accurate prediction highlights its effectiveness in processing unseen APK files and maintaining high detection precision.

Conclusion

The **Android Analyzer** project successfully demonstrates how machine learning can classify Android APK files based on their feature signatures. By addressing class imbalance, performing detailed feature analysis, and evaluating multiple classifiers, the project achieved high accuracy and reliable predictions with the Random Forest Classifier. This tool provides valuable insights into app behavior, enabling researchers and developers to identify potentially harmful apps with confidence.

Future work could focus on extending the dataset, improving real-time detection capabilities, and integrating this tool into broader Android security solutions.

Figures

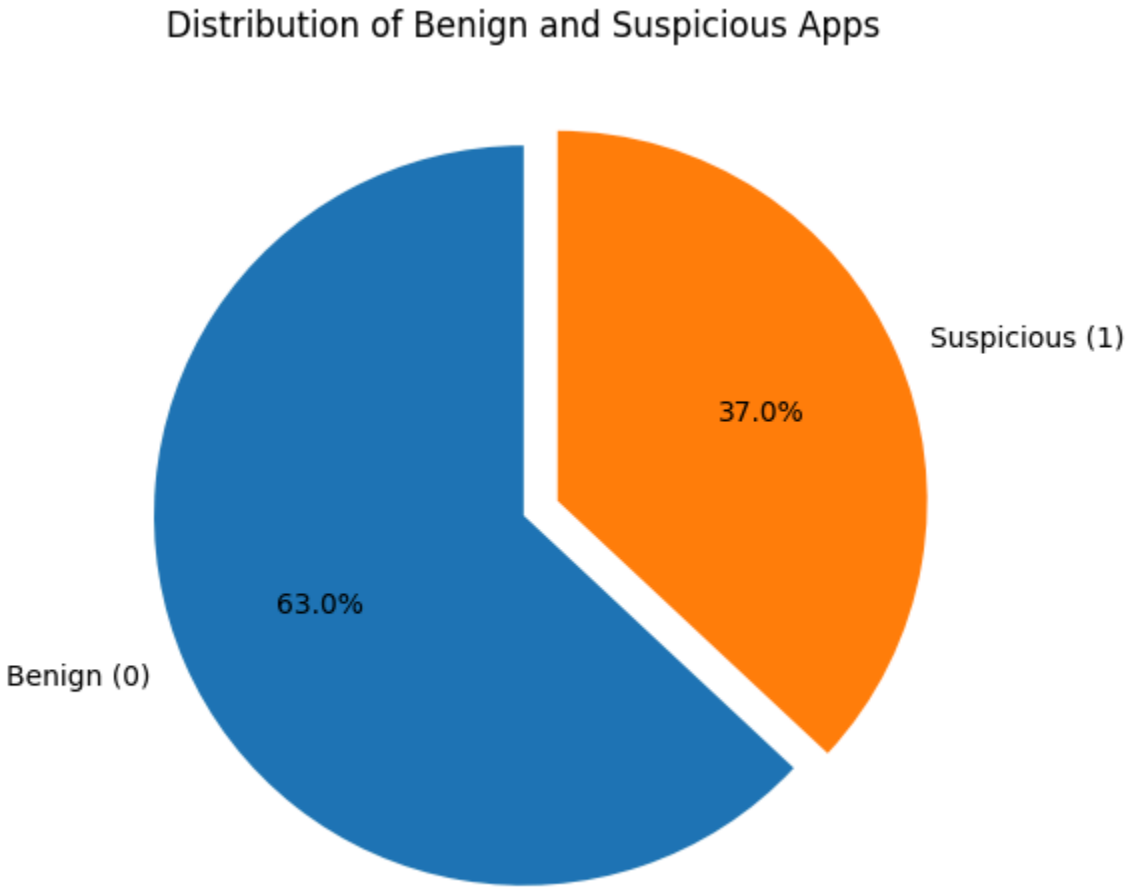

Figure 1 - Distribution of Benign and Suspicious Apps

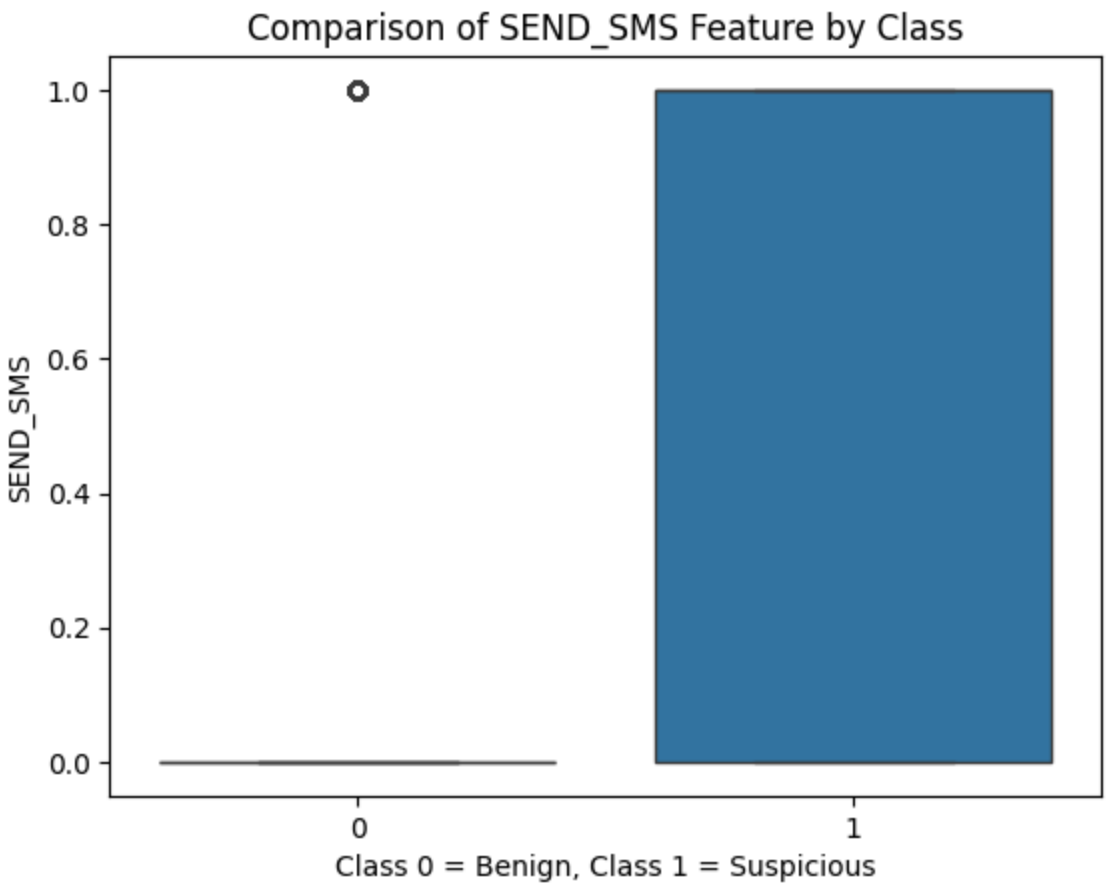

Figure 2 - Comparison of SEND_SMS Feature by Class

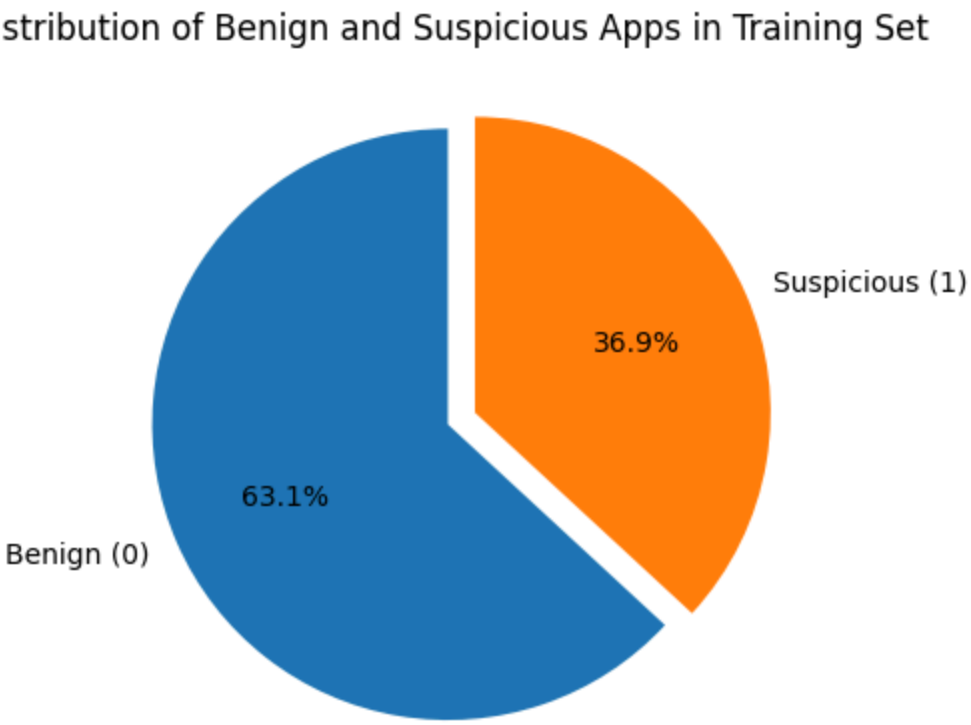

Figure 3 - Distribution of Benign and Suspicious Apps in Training Set

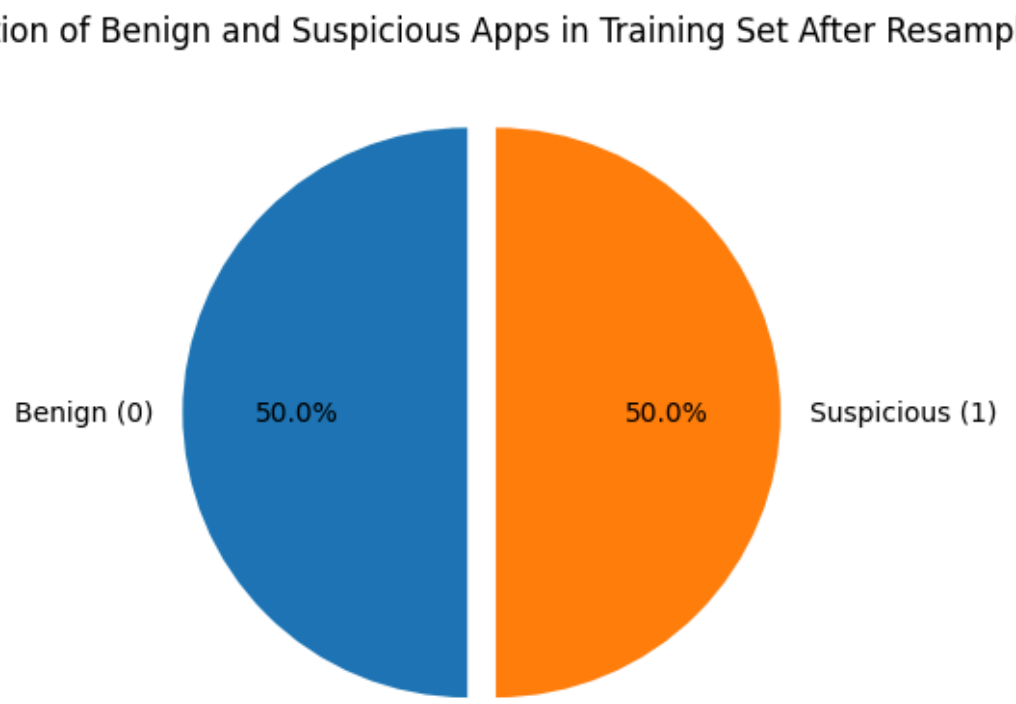

Figure 4 - Distribution of Benign and Suspicious Apps in Training Set After Resampling

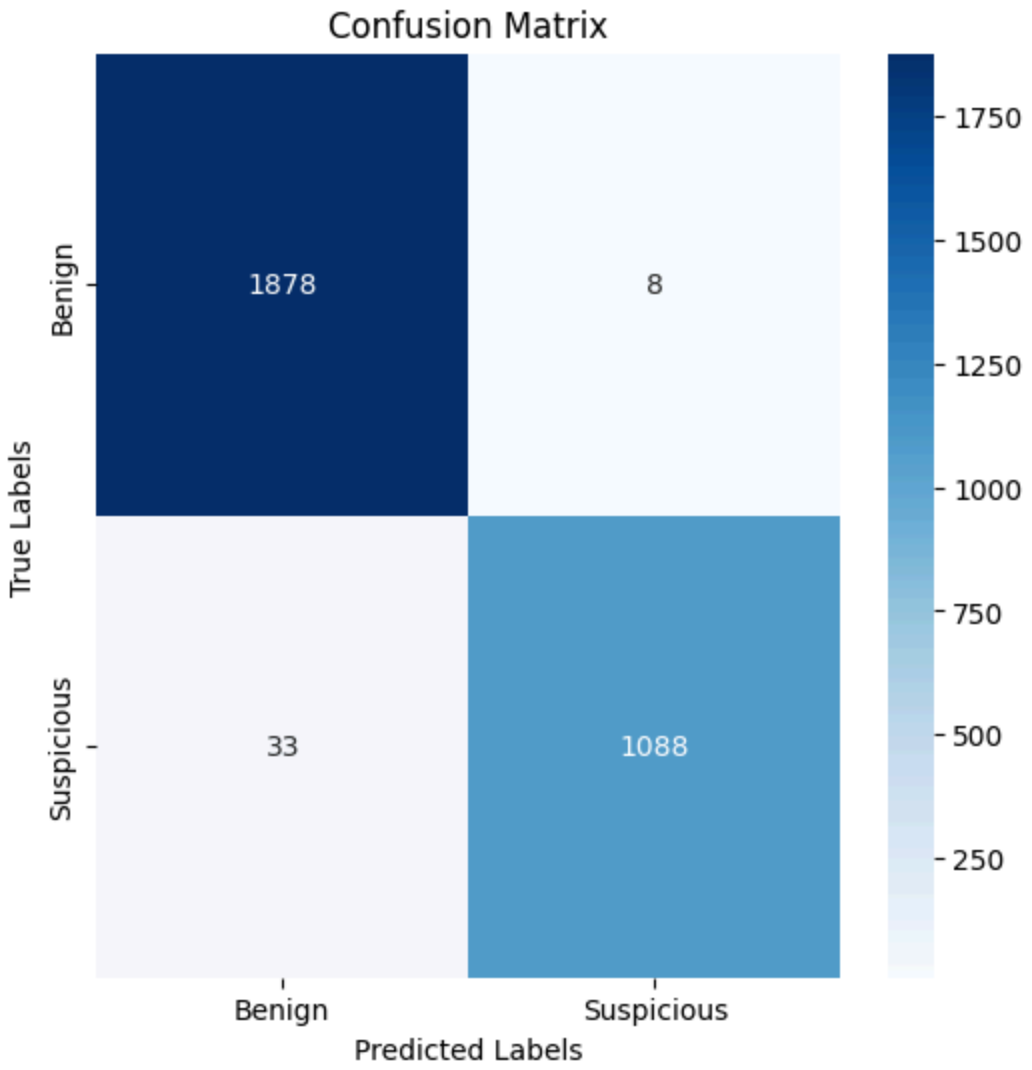

Figure 5 - Confusion Matrix of Random Forest Classifier

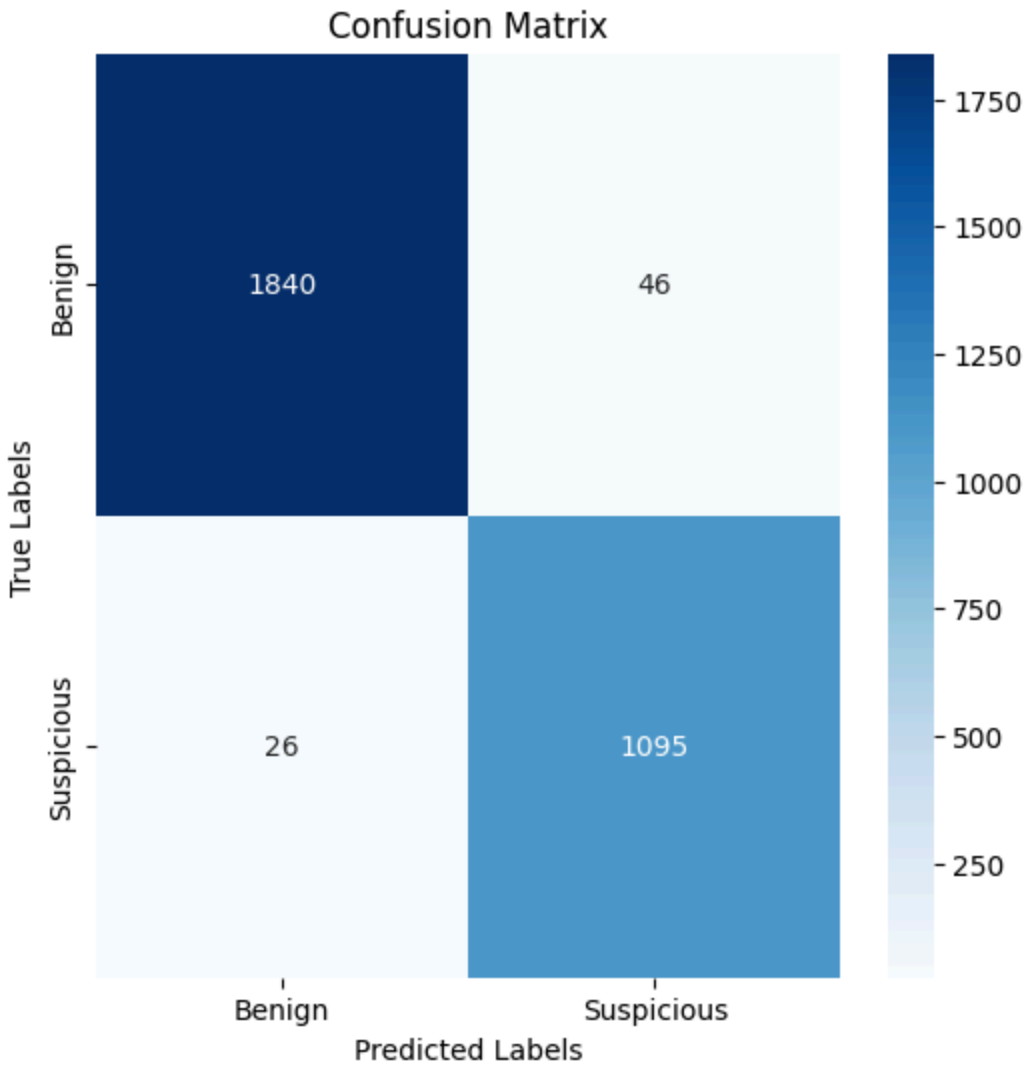

Figure 6 - Confusion Matrix of Logistic Regression Classifier

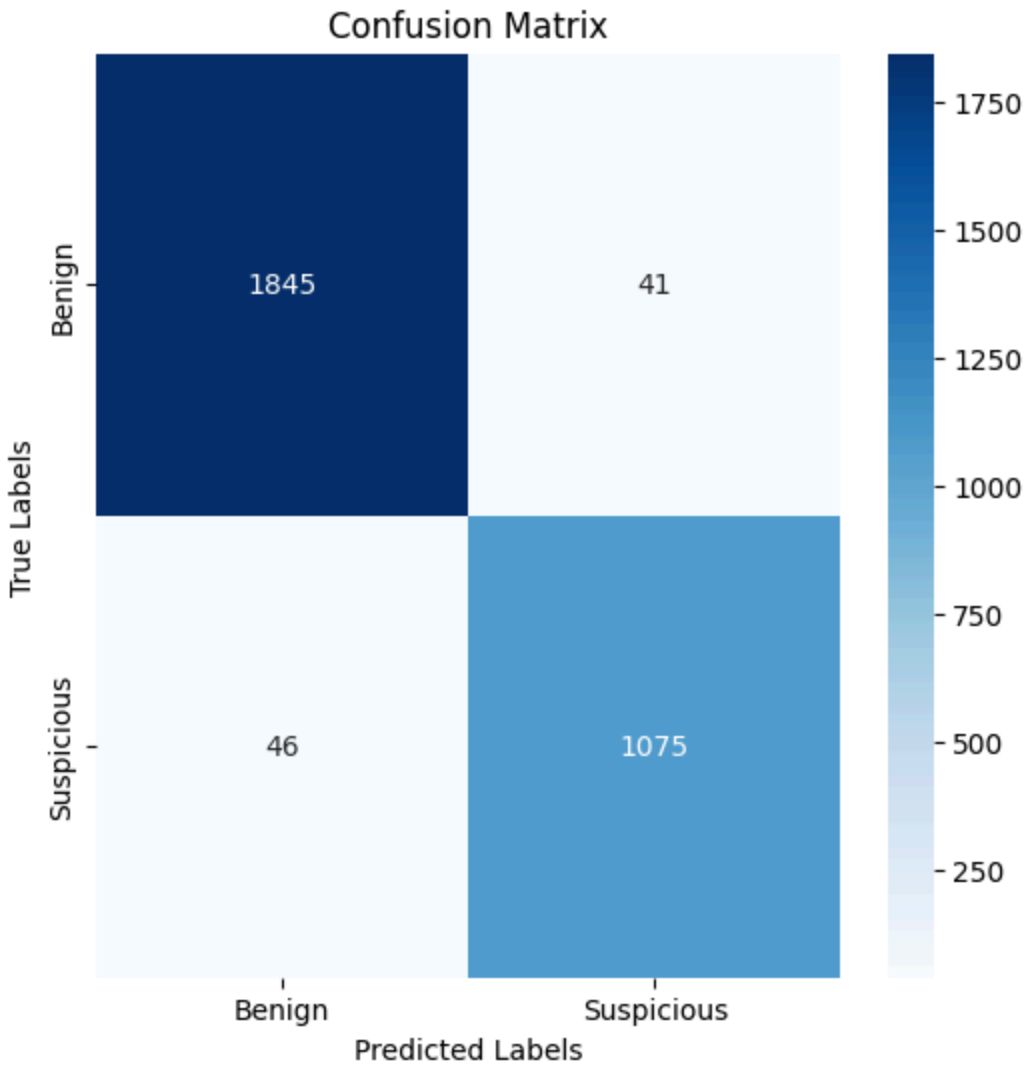

Figure 7 - Confusion Matrix of Decision Tree Classifier

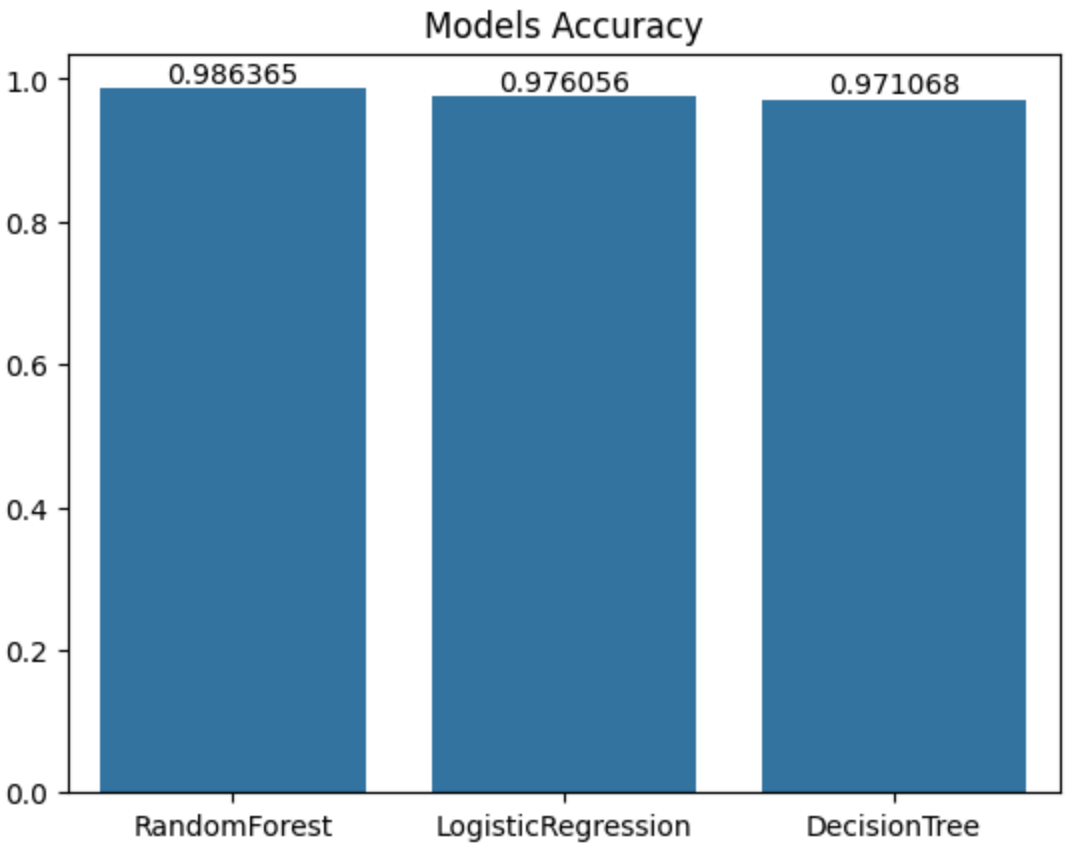

Figure 8 - Comparison of Random Forest, Logistic Regression and Decision Tree Classifier accuracies.