Android malware classification

Emilie TRAN and François BARNOUIN

Our GitHub repository:
https://github.com/Franz1234567/MLCS_final

1 ABSTRACT

The exponential rise in Android device usage has made the platform a prime target for malware attacks, posing significant risks to data security, user privacy, and financial stability. Traditional malware detection methods, such as signature-based and heuristic approaches, are increasingly ineffective against sophisticated or obfuscated threats. This study investigates the application of machine learning techniques for Android malware detection, leveraging features like permissions, API calls, and metadata to classify applications as benign or malicious. A publicly available dataset was preprocessed and used to train and evaluate multiple machine learning classifiers, including Support Vector Machines (SVM), Gradient Boosting, Random Forests, and more. Each model was fine-tuned through hyperparameter optimization and evaluated using F1-score and accuracy. Results indicate that SVM outperformed other models with the highest F1-score of 0.9665 and an accuracy of 0.9630. The findings demonstrate the potential of machine learning to enhance the scalability and robustness of Android malware detection systems, providing a dynamic defense against evolving cyber threats.

2 INTRODUCTION

2.1 Context

The exponential growth in Android device usage has positioned the platform as a prime target for cybercriminals. This has led to a substantial increase in the prevalence of malware specifically tailored to exploit Android's open-source ecosystem (see figure 1). Such malicious software threatens to compromise data security, lead to financial losses, and infringe on user privacy. Traditional detection methods, including signature-based and heuristic approaches, are becoming increasingly ineffective against the evolving landscape of sophisticated and obfuscated malware. Given the limitations of conventional techniques, it is essential to explore innovative and adaptive solutions. Machine learning offers a dynamic approach that leverages patterns and anomalies in app behaviors to detect malicious intent. By utilizing features such as permissions, API calls, and metadata, machine learning models can effectively classify applications as benign or malicious. The integration of such solutions is critical for maintaining the security and integrity of mobile ecosystems.

2.2 Goals

This study aims to evaluate the effectiveness of machine learning classifiers in detecting Android malware. A selection of algorithms was applied to a publicly available dataset to assess their ability to accurately and efficiently classify applications as benign or malicious. By comparing these models, the research seeks to identify the most effective approach(es) for building scalable and robust solutions to enhance mobile cybersecurity.

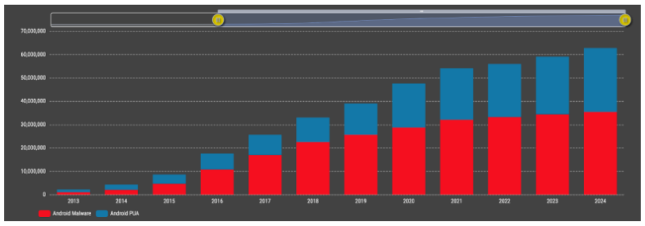


Figure 1: Android Malware Number Evolution[1]

3 DATASET

3.1 Overview

The Android Malware Detection Dataset is a collection of features extracted from Android applications designed to aid in research and development of malware detection techniques for the Android platform. The dataset contains a variety of attributes that characterize both the behaviors and functionalities of Android apps. By exploring these features, it allows to build and refine machine learning models aimed at identifying malicious Android apps.

3.2 Features

The dataset includes features related to permissions (like access to location, camera, and contacts), system behaviors (like Bluetooth or system settings), and security functions like encryption and authentication. It also covers communication features (such as interactions with SMS, phone calls, or network management) and data access features (such as reading and writing to external storage or accessing user data like contacts). It also includes app lifecycle (like boot completion event).

4 CONTENT

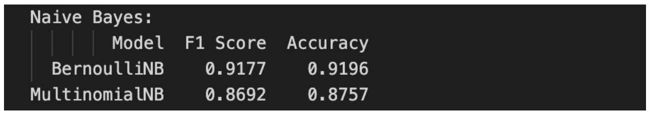
4.1 Data preprocessing

First, we had to do some preprocessing. Since the data was sorted between malware/benign samples, rows were randomized to shuffle the distribution of malware and benign samples. Columns with unique values were removed as they provided no useful information. Labels were standardized by replacing "Benign" with 0 and "Malware" with 1. The dataset was then split into training, validation, and test sets, using a 60/20/20 ratio. For model training and evaluation, the training and validation sets were concatenated. We used both of them using a 10-fold cross validation process to ensure consistent results.

4.2 Model Tuning

Model tuning was conducted for several algorithms by testing a range of hyperparameter combinations. For KNN, the number of neighbors and distance metrics were varied. Naive Bayes was evaluated using both Bernoulli and Multinomial variants. Logistic Regression was tuned by modifying the regularization parameter C and the maximum number of iterations. SVM models were adjusted by varying C, gamma, and kernel type. Decision Trees and Random Forests were optimized for maximum depth, number of estimators, and splitting criteria. Gradient Boosting was tuned by varying maximum depth, number of estimators, and learning rate. For each model, the top 10 combinations of parameters were identified based on F1-score and accuracy using 10-fold cross-validation. The best-performing configuration was selected, finding compromise if best options are not the same for the two metrics.

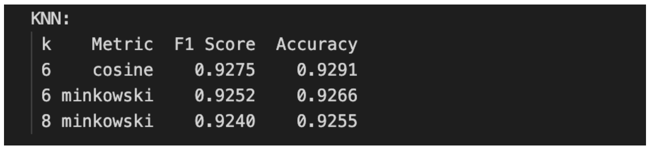
4.3 Top three combinations for each model



```
Naive Bayes:
            Model  F1 Score  Accuracy
  BernoulliNB    0.9177    0.9196
MultinomialNB    0.8692    0.8757
```

Figure 2: Top2 Naive Bayes models

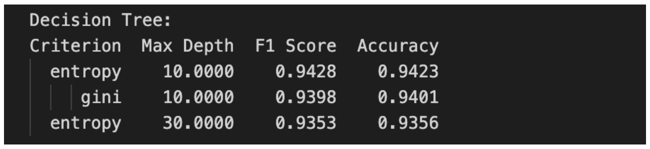
Bernoulli outperforms Multinomial, likely due to the dataset's binary attributes aligning well with Bernoulli assumptions. This model performs the least, likely due to the fact that it is unlikely that the features are independent.



```
KNN:
 k    Metric  F1 Score  Accuracy
 6    cosine    0.9275    0.9291
 6 minkowski    0.9252    0.9266
 8 minkowski    0.9240    0.9255
```

Figure 3: Top3 KNN hyperparameters

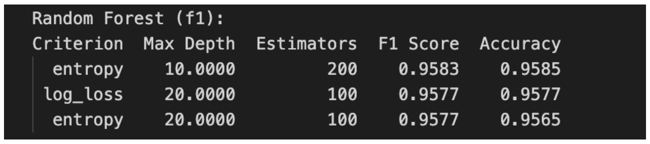
The best performance is achieved with 6 neighbors and cosine distance.



```
Decision Tree:
Criterion  Max Depth  F1 Score  Accuracy
  entropy    10.0000    0.9428    0.9423
     gini    10.0000    0.9398    0.9401
  entropy    30.0000    0.9353    0.9356
```

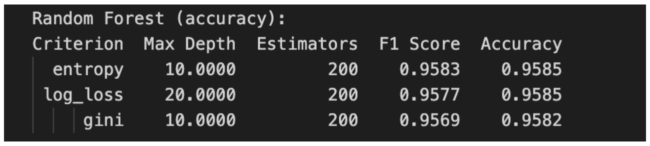
Figure 4: Top3 Decision Tree hyperparameters

Model with depth constraints at 10 and entropy as a criterion deliver the most consistent results.



```
Random Forest (f1):
Criterion  Max Depth  Estimators  F1 Score  Accuracy
  entropy    10.0000         200    0.9583    0.9585
 log_loss    20.0000         100    0.9577    0.9577
  entropy    20.0000         100    0.9577    0.9565
```

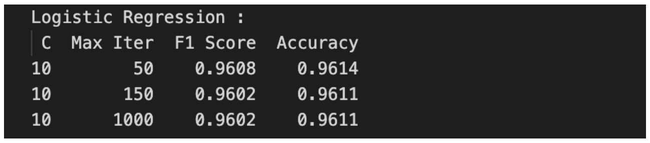
Figure 5: Top3 Random Forest hyperparameters sorted by f1 score



```
Random Forest (accuracy):
Criterion  Max Depth  Estimators  F1 Score  Accuracy
  entropy    10.0000         200    0.9583    0.9585
 log_loss    20.0000         200    0.9577    0.9585
     gini    10.0000         200    0.9569    0.9582
```

Figure 6: Top3 Random Forest hyperparameters sorted by accuracy

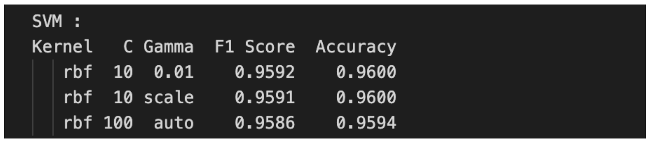
According to the two previous figures, entropy as the criterion, max-depth equals to 10 and number of estimators equals to 200 is the model which performs well for f1 score and accuracy. We can see for the next combinations some differences in two sorts. So the first combination seems to be the best in this case.



```
Logistic Regression :
 C  Max Iter  F1 Score  Accuracy
10        50    0.9608    0.9614
10       150    0.9602    0.9611
10      1000    0.9602    0.9611
```

Figure 7: Top3 Logistic Regression hyperparameters

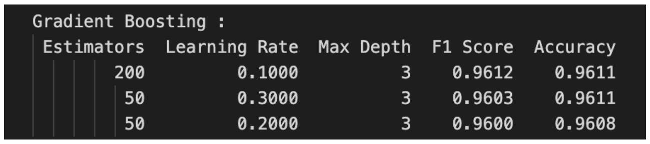
The best performance is achieved with a regularization parameter C=10 and a maximum of 50 iterations,



```
SVM :
Kernel   C Gamma  F1 Score  Accuracy
   rbf  10  0.01    0.9592    0.9600
   rbf  10 scale    0.9591    0.9600
   rbf 100  auto    0.9586    0.9594
```

Figure 8: Top3 SVM hyperparameters

A radial basis kernel with C=10 and gamma=0.01 provides optimal performance.



```
Gradient Boosting :
 Estimators  Learning Rate  Max Depth  F1 Score  Accuracy
        200         0.1000          3    0.9612    0.9611
         50         0.3000          3    0.9603    0.9611
         50         0.2000          3    0.9600    0.9608
```

Figure 9: Top3 Gradient Boosting hyperparameters

A shallow depth (max-depth=3) with learning rate 0.1 ensures a balance between accuracy and overfitting.

4.4 Final choice

Model	Hyperparameters
KNN	n-neighbors=6, metric=cosine
NB	Bernouilli
LR	C=10, max-iter=50
DT	max-depth=10, criterion=entropy
RF	max-depth=10, criterion=entropy, n-estimator=200
GB	max-depth=3, learning-rate=0.1, n-estimator=200
SVM	C=10, gamma=0.01, kernel=rbf

5 MODEL COMPARISON

5.1 Confusion Matrices

As we can see in the pictures below (figures 10-16), while simpler models like Naive Bayes and KNN show higher misclassification rates, ensemble methods such as Random Forest and Gradient Boosting achieve a better balance between false positives and false negatives. The SVM stands out with the lowest misclassification rates, confirming its effectiveness in handling the dataset's complexities.

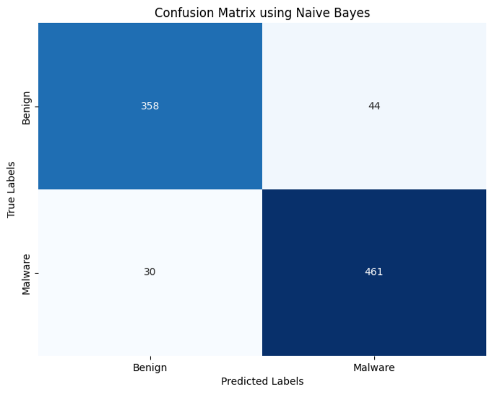


Figure 10: Naive Bayes Confusion Matrix

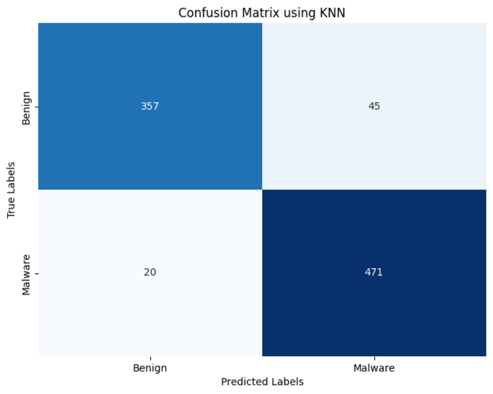


Figure 11: KNN Confusion Matrix

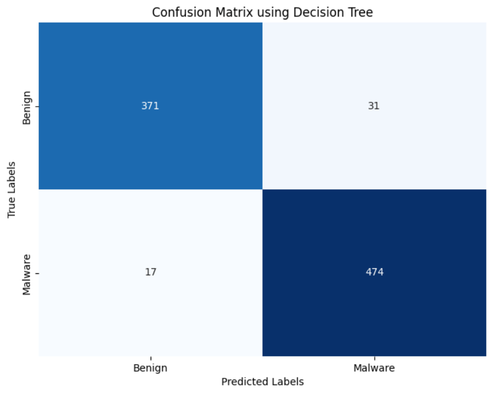


Figure 12: Decision Tree Confusion Matrix

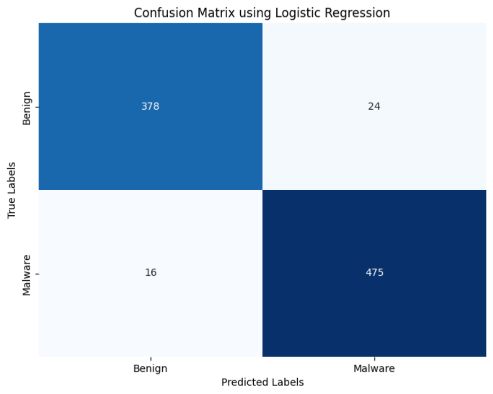


Figure 13: Logistic Regression Confusion Matrix

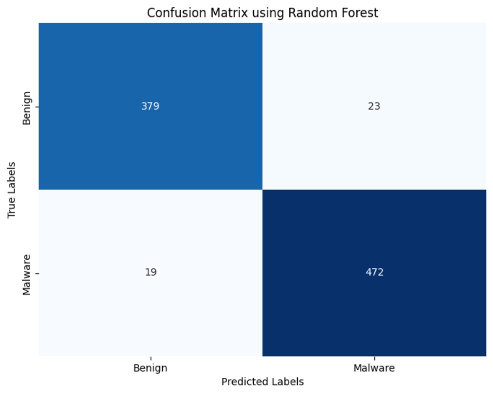


Figure 14: Random Forest Confusion Matrix

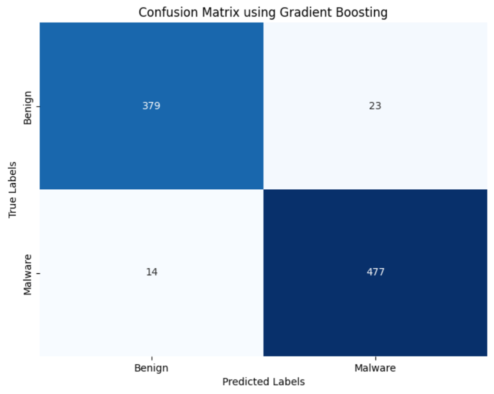


Figure 15: Gradient Boosting Confusion Matrix

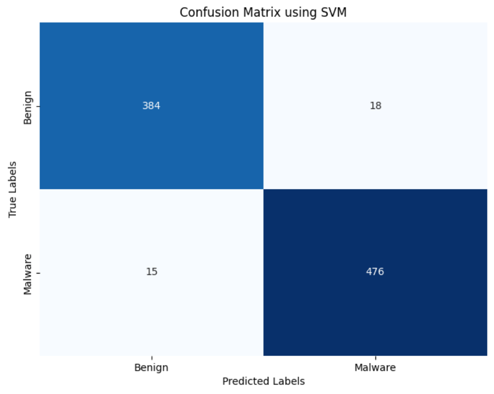


Figure 16: SVM Confusion Matrix

5.2 Evaluation on train

We evaluated the models on the training dataset to check for overfitting, and the graph below compares the performance of each model on both the training and test datasets, highlighting any discrepancies in their generalization capabilities.

Most models show minimal overfitting, with close performance between training and test sets. For example, Random Forest, Logistic Regression and Gradient Boosting exhibit small differences in F1 Score and Accuracy, indicating good generalization. However, Decision Tree model shows slight signs of overfitting, with higher training metrics compared to the test set. We can notice that SVM is the one that shows the best generalization, with very close results for test/train sets.

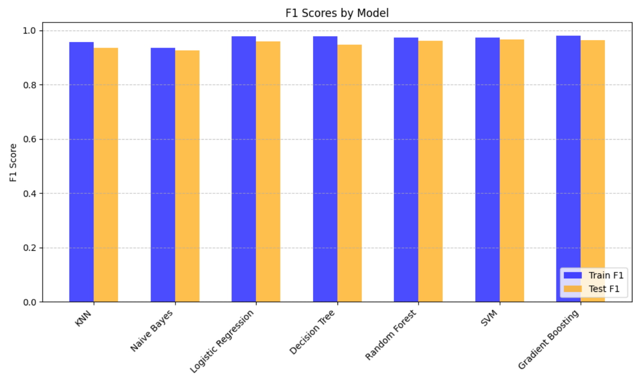


Figure 17: F1-score on train/test set

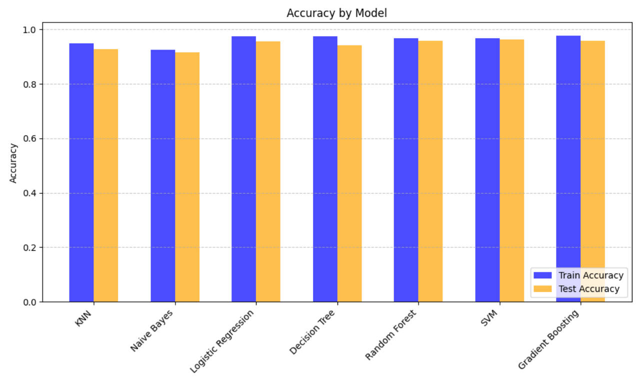
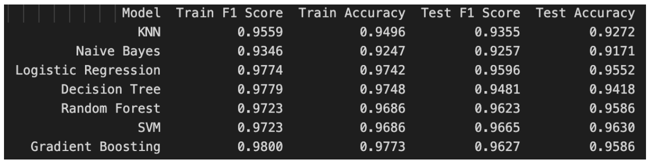


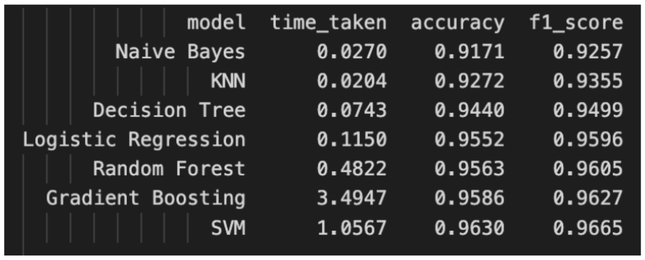
Figure 18: Accuracy on train/test set



```
              Model  Train F1 Score  Train Accuracy  Test F1 Score  Test Accuracy
                KNN          0.9559          0.9496         0.9355         0.9272
        Naive Bayes          0.9346          0.9247         0.9257         0.9171
Logistic Regression          0.9774          0.9742         0.9596         0.9552
      Decision Tree          0.9779          0.9748         0.9481         0.9418
      Random Forest          0.9723          0.9686         0.9623         0.9586
                SVM          0.9723          0.9686         0.9665         0.9630
  Gradient Boosting          0.9800          0.9773         0.9627         0.9586
```

Figure 19: Train/test f1-score and accuracy on each model

5.3 Conclusion



```
              model  time_taken  accuracy  f1_score
        Naive Bayes      0.0270    0.9171    0.9257
                KNN      0.0204    0.9272    0.9355
      Decision Tree      0.0743    0.9440    0.9499
Logistic Regression      0.1150    0.9552    0.9596
      Random Forest      0.4822    0.9563    0.9605
  Gradient Boosting      3.4947    0.9586    0.9627
                SVM      1.0567    0.9630    0.9665
```

Figure 20: Model results

The results demonstrate that Support Vector Machine (SVM) achieved the best performance for classifying Android malware based on the flags used, with the highest F1-score (0.9665) and the top accuracy score (0.9630). Moreover, it was the model that had the least signs of overfitting (even though those signs were small on all models). Gradient Boosting and Random Forest also delivered strong results, with Gradient Boosting achieving the second highest accuracy (0.9586) and a near-optimal F1-score (0.9627), while Random Forest followed closely (accuracy: 0.9563, F1-score: 0.9605). Logistic Regression performed well, with an accuracy of 0.9552 and an F1-score of 0.9596, making it a competitive model for this task. Simpler models like Decision Tree and KNN demonstrated solid but slightly lower performance, with F1-scores of 0.9499 and 0.9355, respectively. Naive Bayes, while being the least accurate (accuracy: 0.9171, F1-score: 0.9257), offered the simplest and fastest computation. Overall, all models were computationally efficient given the size and nature of the dataset, with runtimes ranging from milliseconds to just a few seconds.

6 COLLABORATION IN THE TEAM

François:

- Preprocessing
- Tuning svm, random forest, decision tree, naive bayes

Emilie:

- Tuning knn, logistic regression, gradient boosting
- Run all tuned models on train and test set

Both:

- Plots
- Report
- GitHub management

REFERENCES

[1] AV-ATLAS Portal. (n.d.). Malware Statistics. Available from: https://portal.av-atlas.org/malware/statistics

[2] Revaldo, D. (2024). Android Malware Detection Dataset. Available from: https://www.kaggle.com/datasets/dannyrevaldo/android-malware-detection-dataset/code