ANDROID MALWARE DETECTION USING MACHINE LEARNING

TEMFACK DERICK

TABLE OF CONTENTS

IN SUPREMÆ DIGNITATIS 1343

INTRODUCTION

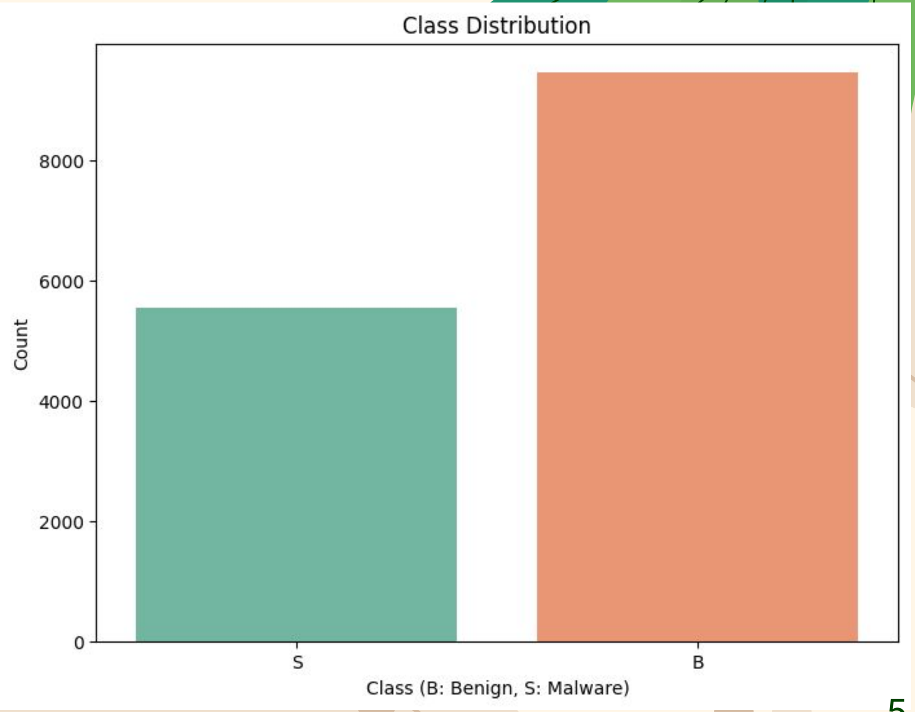
This project aims to build an effective classification model to classify a mobile application as **Benign** or **Malware**. To do so, we'll evaluate multiple classification models using different metrics and select the best model with better performance for our dataset.

01

Exploratory Data Analysis

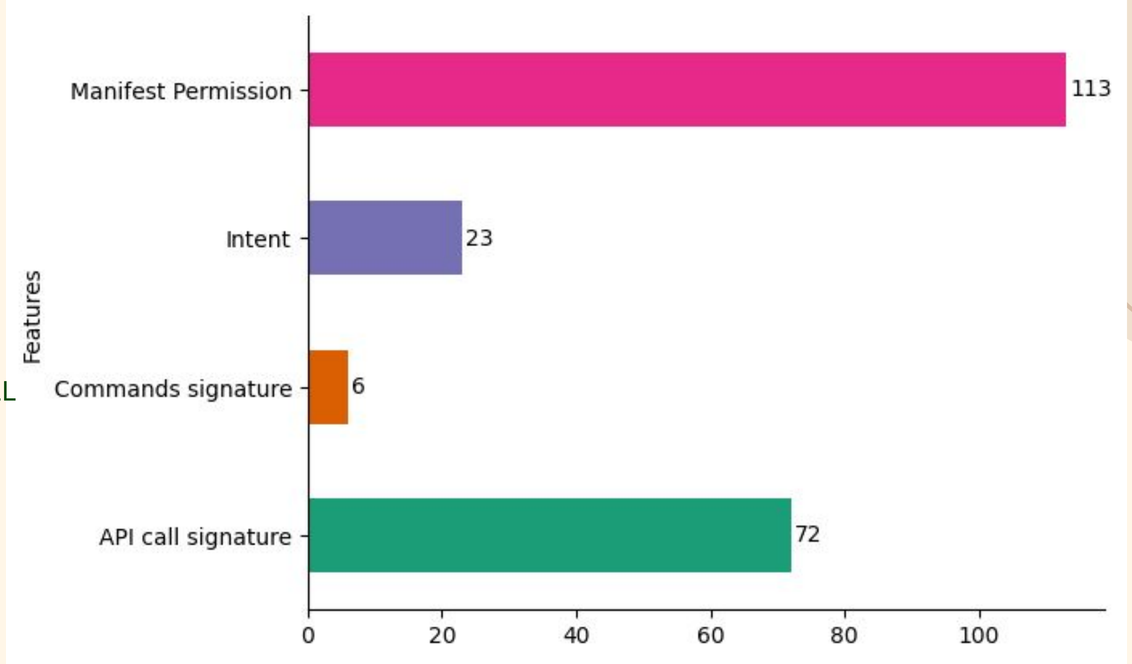
DATASET

- Data From FigShare
- 215 Features
- 15036 records (5,560 malwares and 9,476 Benign)
- Target variable is **Malware** (S) or **Benign** (B)
- Features are binary encoded

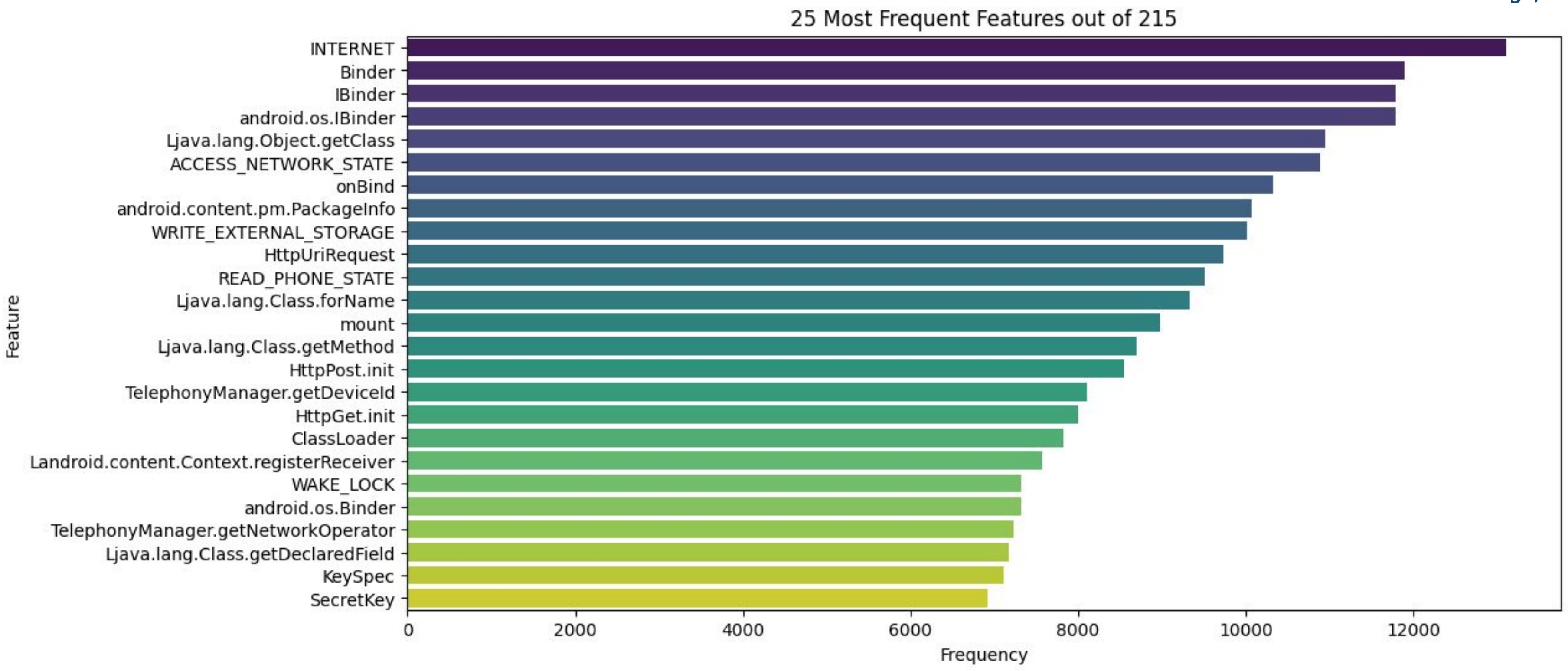


Features

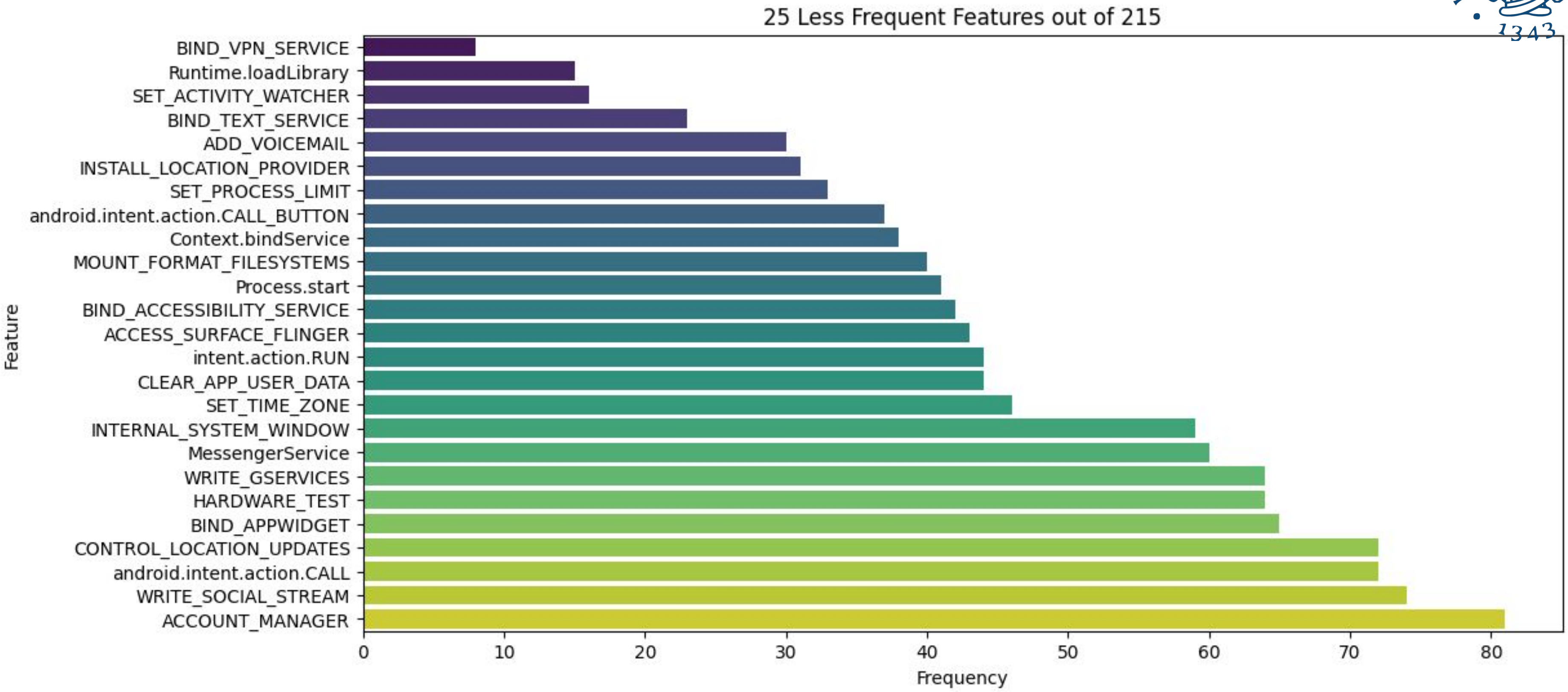
- **API Call Signature (72)**
  - signature of a method in the Java API
  - E.g. android.os.Binder;
    Ljava.lang.Class.getResource
- **Manifest Permission (113)**
  - e.g. ACCESS_FINE_LOCATION
- **Intent (23)**
  - request an action from another app component
  - e.g. android.intent.action.BATTERY_LOW, android.intent.action.NEW OUTGOING_CALL
- **Commands signature (6)**
  e.g. chown, chmod



Top 25 most frequent features



Top 25 less frequent features



02 Data Cleaning, Transformation and Splitting

Data Cleaning and Transformation

Before

TelephonyManager.getSimCountryIso	count
0	6994
0	5514
1	1330
1	1193
?	5

After

TelephonyManager.getSimCountryIso	count
0	12508
1	2523

Data Transformation

```
dataset['class'] = dataset['class'].map({'B': 0, 'S': 1})
dataset.head()
```

Data Splitting

```python
from sklearn.model_selection import train_test_split

X = dataset.drop('class', axis=1)  # Features (all columns except 'class')
y = dataset['class']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (12024, 215)
X_test shape: (3007, 215)
y_train shape: (12024,)
y_test shape: (3007,)
```

03 Models comparison

The Models We Tested

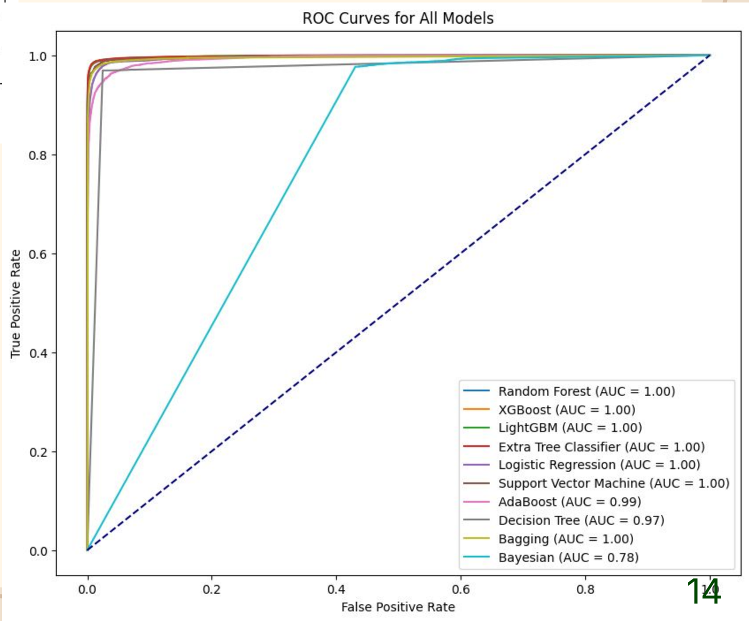
- Random Forest
- XGBoost
- LightGBM
- Extra Tree Classifier
- Logistic Regression
- SVM
- AdaBoost
- Decision Tree
- Bagging
- Bayesian

We evaluated both without sampling and with Under-sampling

Before Under-sampling

Rank	Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
1	XGBoost	0.988662	0.988112	0.981055	0.984571	0.998568
2	LightGBM	0.988495	0.987516	0.981206	0.984351	0.998513
3	Extra Tree Classifier	0.988440	0.991681	0.976846	0.984208	0.998383
4	Random Forest	0.987830	0.991592	0.975267	0.983362	0.998149
5	Support Vector Machine	0.982008	0.985944	0.964968	0.975343	0.997424
6	Logistic Regression	0.976852	0.975803	0.961058	0.968375	0.996192
7	Bagging	0.981759	0.980030	0.970305	0.975144	0.995173
8	AdaBoost	0.964959	0.960943	0.943317	0.952049	0.993118
9	Decision Tree	0.973109	0.958166	0.969403	0.963752	0.972323
10	Bayesian	0.708943	0.560313	0.978800	0.712663	0.776001

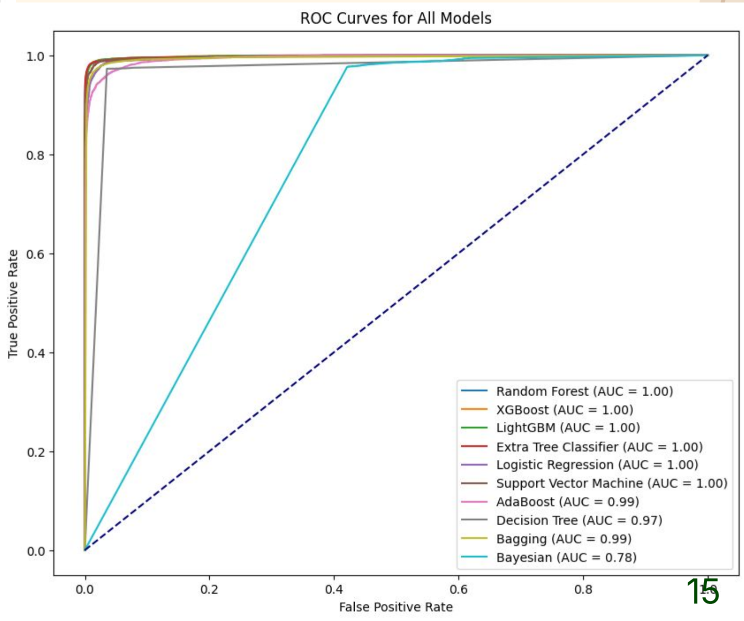
Table 4.1: Comparison of Model Performance Metrics without Sampling



After Under-sampling

Rank	Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
1	XGBoost	0.986468	0.987862	0.985040	0.986449	0.998527
2	LightGBM	0.986280	0.989186	0.983311	0.986239	0.998391
3	Extra Tree Classifier	0.986506	0.992841	0.980078	0.986418	0.998062
4	Random Forest	0.986167	0.992461	0.979777	0.986079	0.997941
5	Support Vector Machine	0.980567	0.988238	0.972711	0.980413	0.997605
6	Logistic Regression	0.974816	0.979502	0.969929	0.974692	0.996493
7	Bagging	0.976808	0.980020	0.973463	0.976730	0.994427
8	AdaBoost	0.962186	0.965546	0.958578	0.962049	0.993927
9	Decision Tree	0.968727	0.964882	0.972861	0.968855	0.969079
10	Bayesian	0.767779	0.688087	0.979627	0.808375	0.780607

Table 4.2: Comparison of Model Performance Metrics after Under-Sampling



04 Model Selection and Hyper-parameter Tuning

Before Under-sampling

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
XGBoost Training	0.998087	0.999095	0.995715	0.997402	0.999825
XGBoost Test	0.987695	0.990054	0.976806	0.983386	0.998446

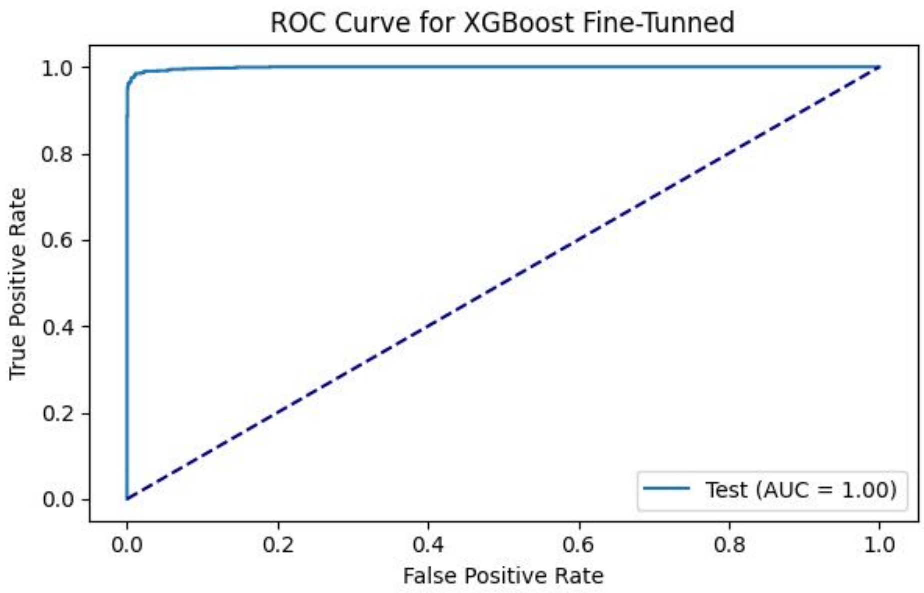
Table 5.1: Performance metrics for XGBoost model on Training and Test sets Before Sampling.

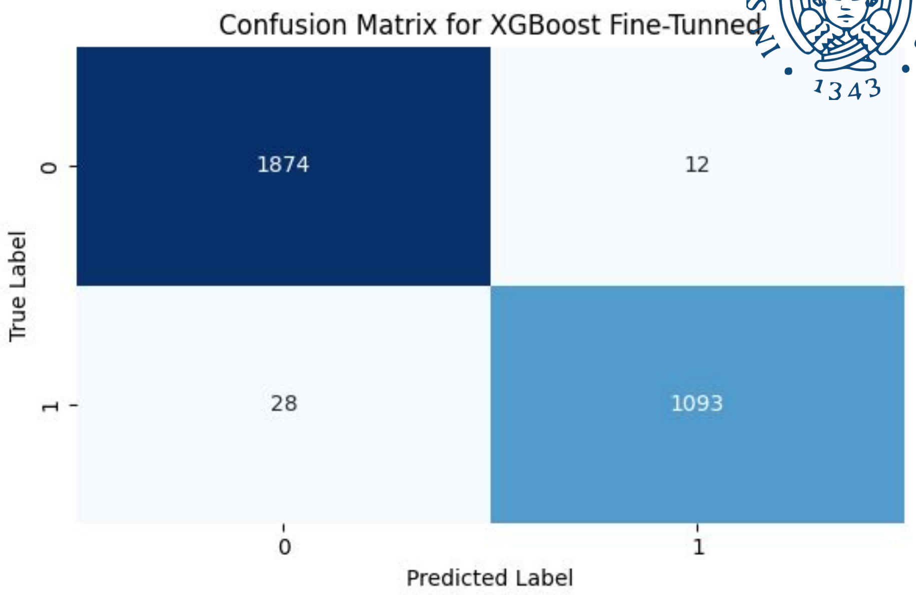
After Under-sampling

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
XGBoost Training	0.99501	0.990579	0.995940	0.993252	0.999570
XGBoost Test	0.98437	0.976909	0.981267	0.979083	0.998569

Table 5.2: Performance metrics for XGBoost model with undersampling on Training and Test sets.

Hyper-Parameter Tuning

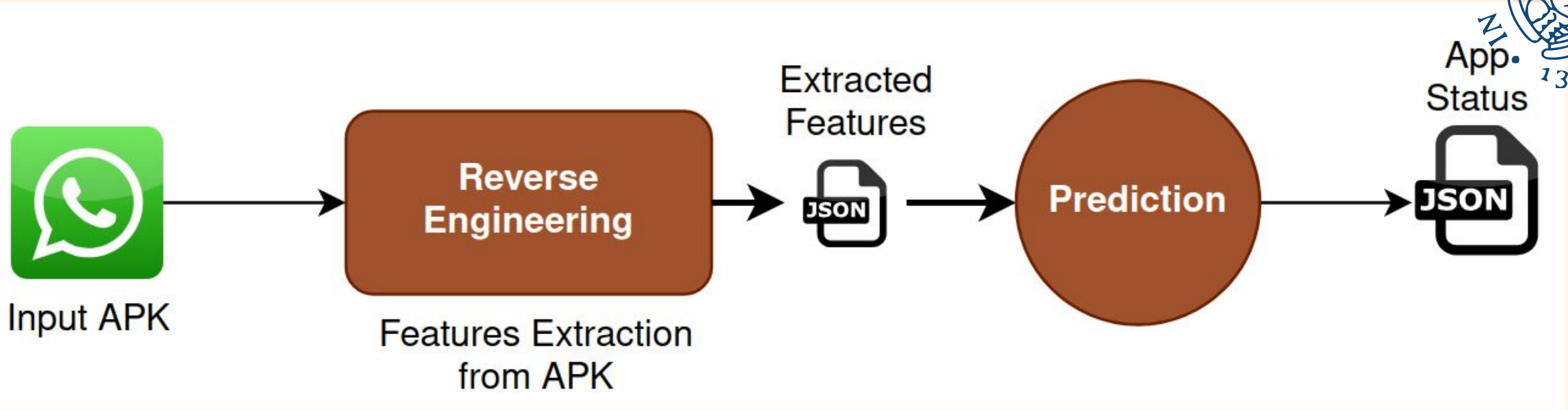




Model	Accuracy	Precision	Recall	F1 Score	ROC AUC
XGBoost Fine-Tuned	0.986698	0.98914	0.975022	0.982031	0.998764

05 Deploiement

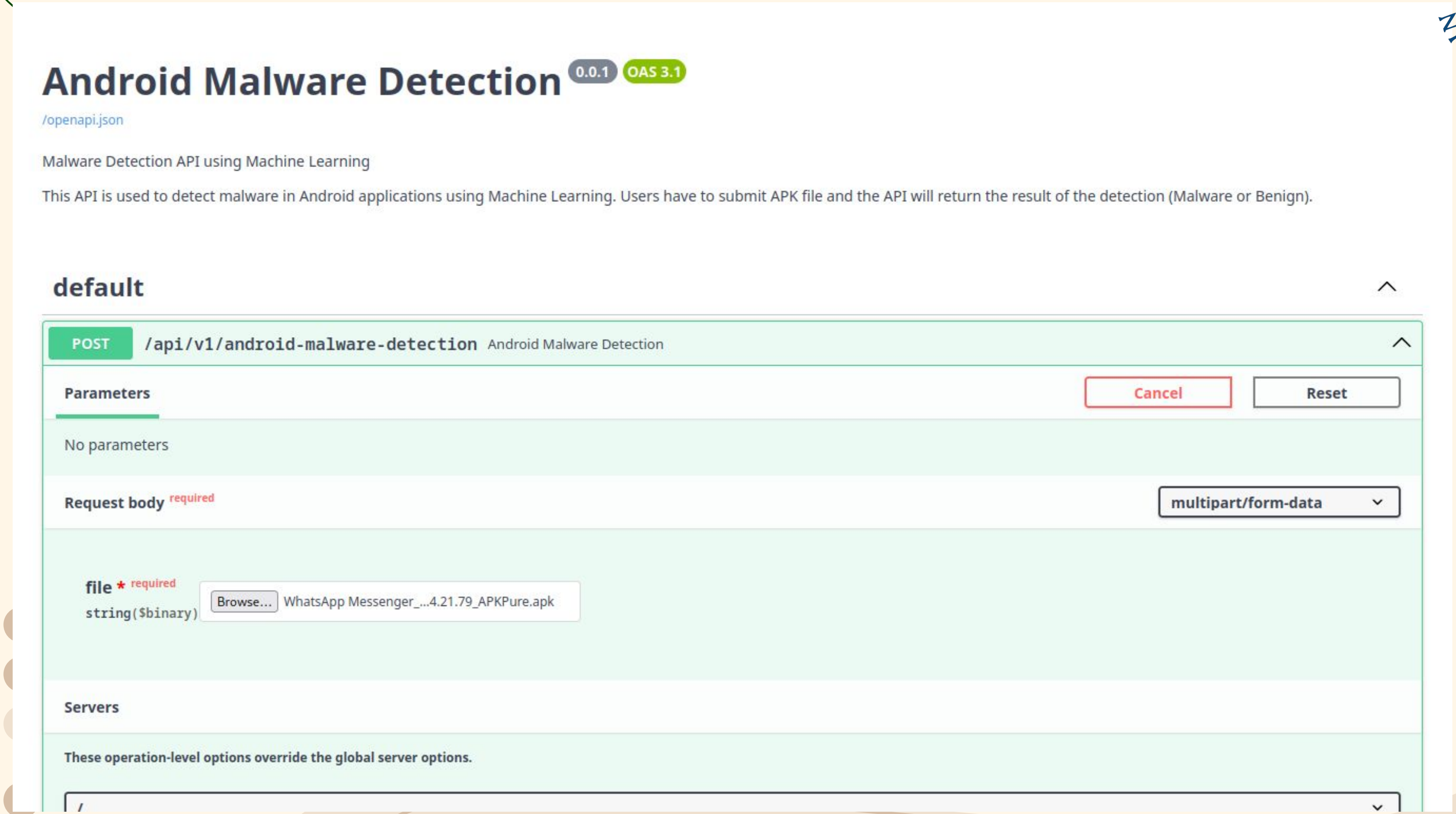
Prediction Workflow



To have access to the application, you have to follow the following steps:

1. Have Docker installed on your computer.
2. Run the following command: `docker run -p 8080:8000 tderick/android-malware-detection`
3. Go to http://localhost:8080/docs to test the application.

WhatsApp Analysis



WhatsApp Analysis

Execute | Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/v1/android-malware-detection' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'file=@WhatsApp Messenger_2.24.21.79_APKPure.apk;type=application/vnd.android.package-archive'
```

Request URL

```
http://localhost:8080/api/v1/android-malware-detection
```

Server response

Code	Details
200	Response body


```
{
  "app_name": "WhatsApp",
  "package_name": "com.whatsapp",
  "version_name": "2.24.21.79",
  "version_code": "242179005",
  "app_features": "bluetooth, location, network, gps, camera, nfc, wifi, telephony",
  "status": "Benign"
}
```


Download

Response headers

```
content-length: 218
content-type: application/json
date: Sun,03 Nov 2024 17:10:07 GMT
server: uvicorn
```

Responses

THANKS!

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, infographics & images by **Freepik**

Please keep this slide for attribution