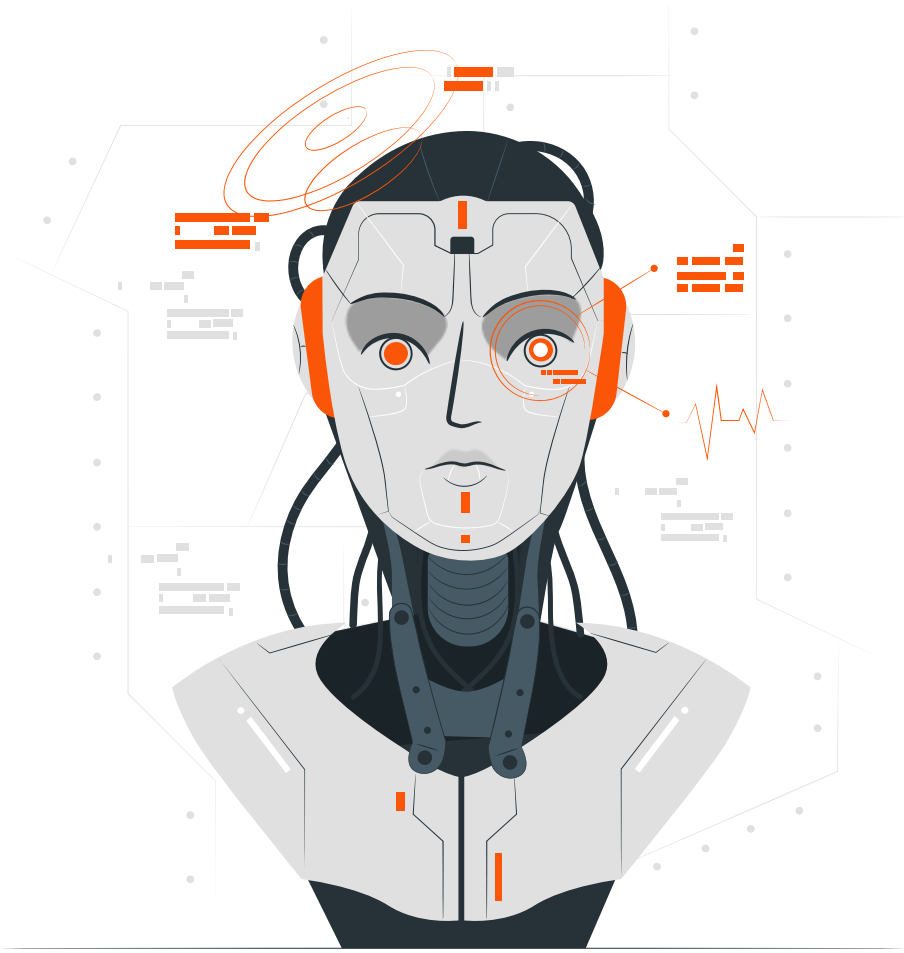
Android Application Malware Detection using

Random **F**orest **C**lassification

- Parsa Moslem (5755015)
- Amir Mohammad Azimi (5795736)



Overview

- Dataset
- Visualization
- Problem
- Solution
- Preprocessing
- Model
- Hyperparameters
- Grid Search
- Results

Dataset

Android Application Malware

Dataset

The '**Android Malware Detection Dataset**' is a comprehensive collection of data designed to facilitate research in the detection and analysis of malware targeting the Android platform. This dataset encompasses a **wide range of features extracted from Android applications**, providing valuable insights into their behaviors and functionalities.

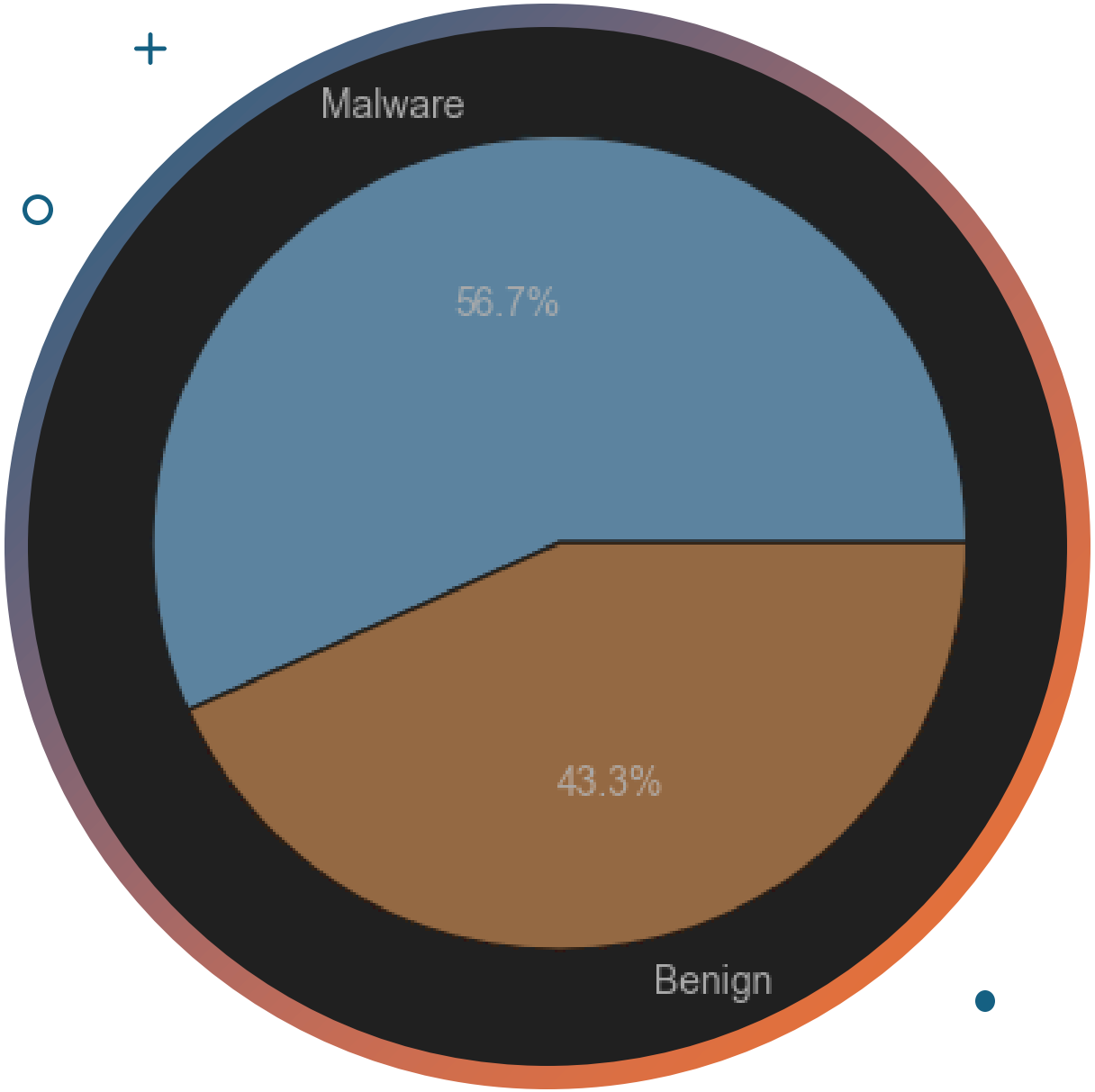
Type	Columns	Rows
All	328	4464
Binary	326	4464
String	1	4464

Dataset downloaded from Kaggle

Features are **permissions** that are extracted from different applications, and a **label** that tells if the application is **malware** or **not**.

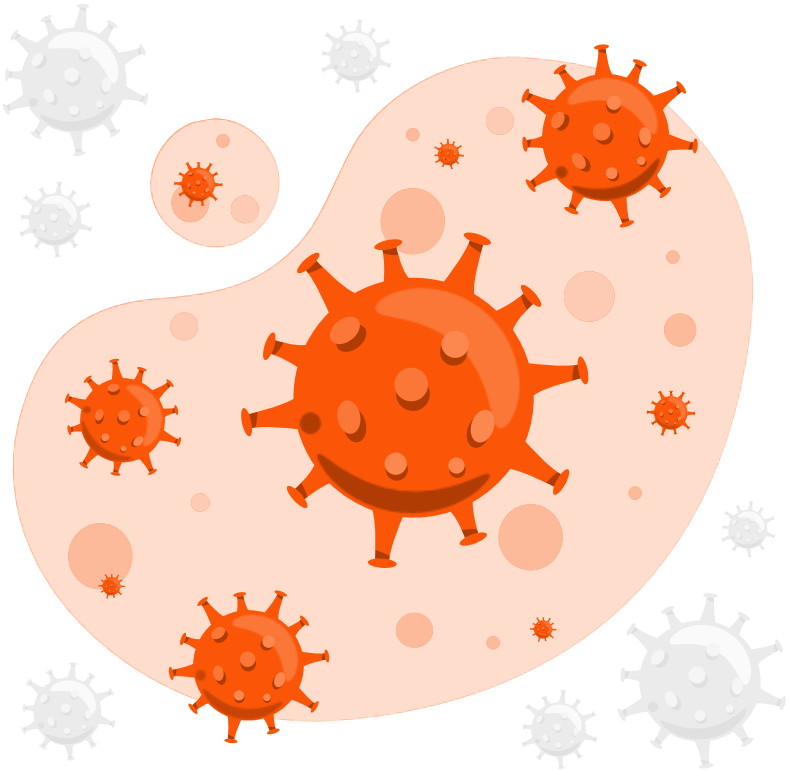
To understand our data, we did the

Visualization



Dataset Distribution

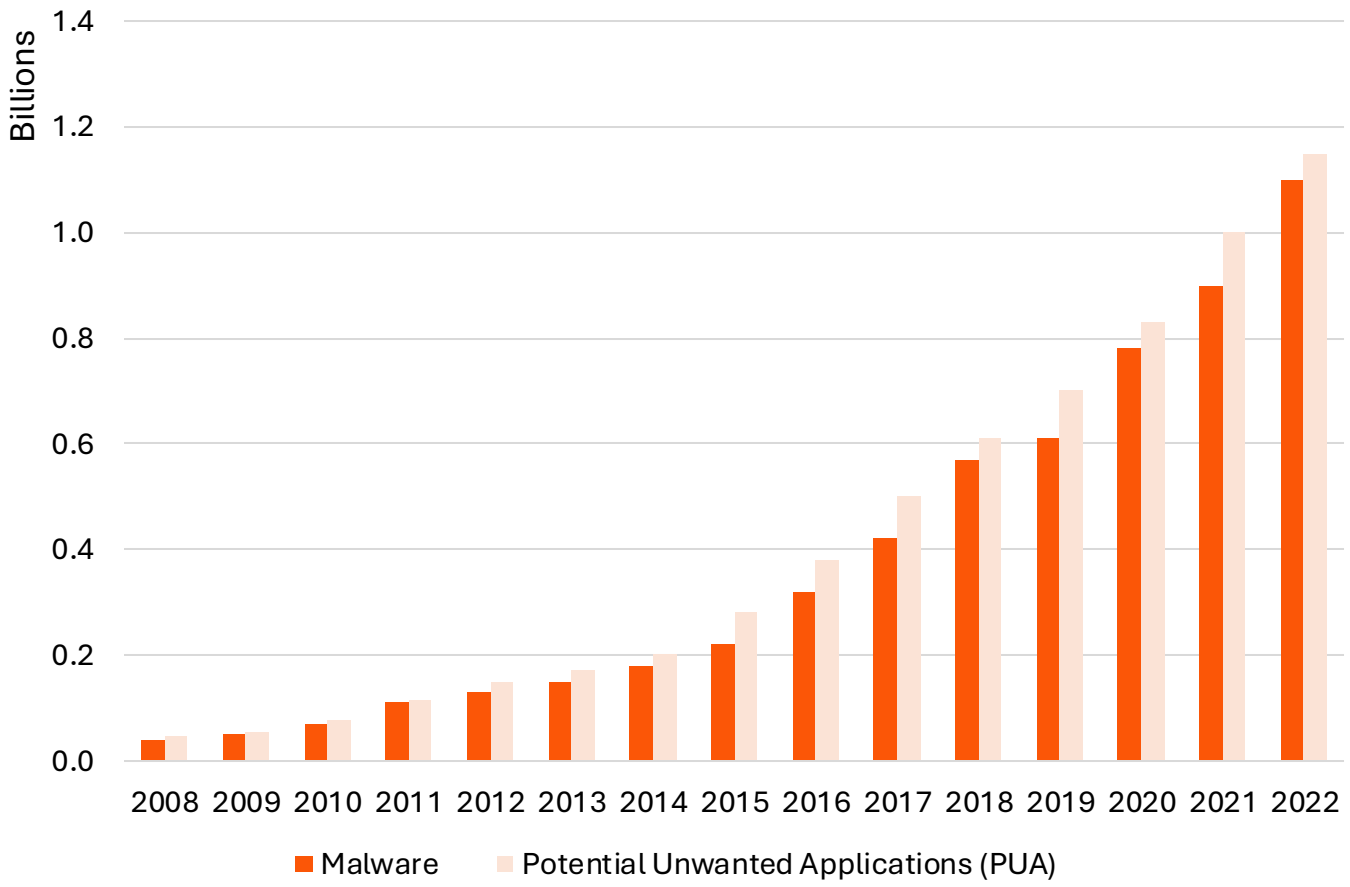
The dataset is **almost balanced** as 56.7% of applications are **malware**, and the other part are **benign**.



Rise of Android Malwares is the

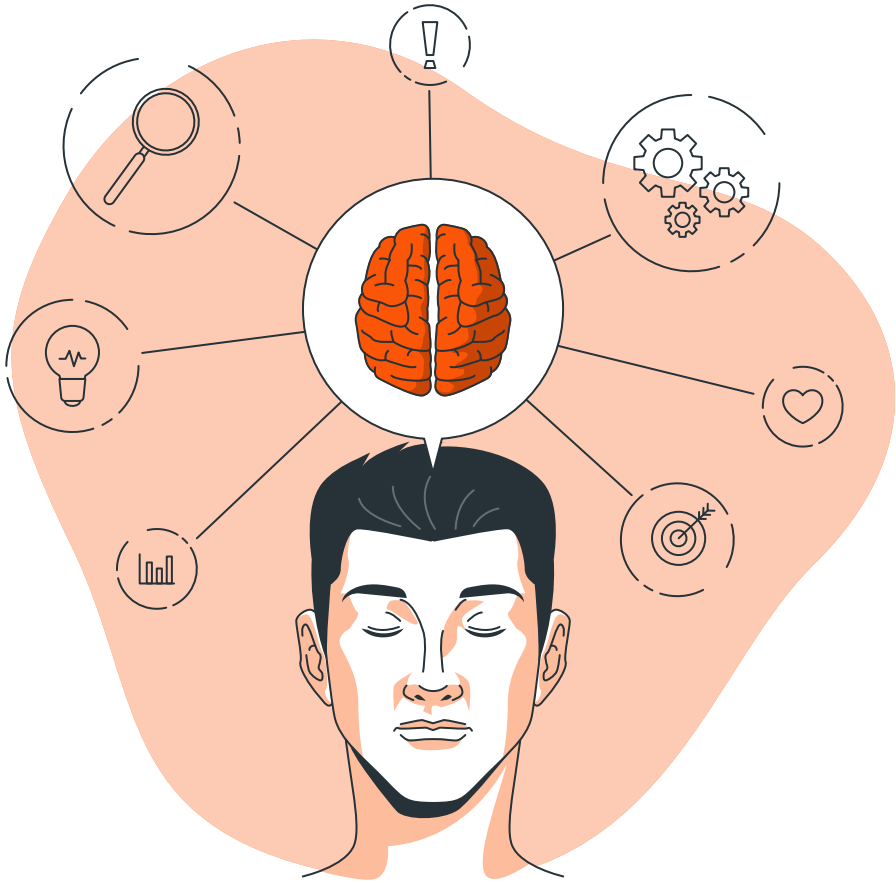
Problem

The rise of Android Malware



- Android is the most popular mobile operating system globally.
- This popularity makes it a target for malware developers.
- Malware can steal sensitive data, disrupt phone functionality or install unwanted software.

Source

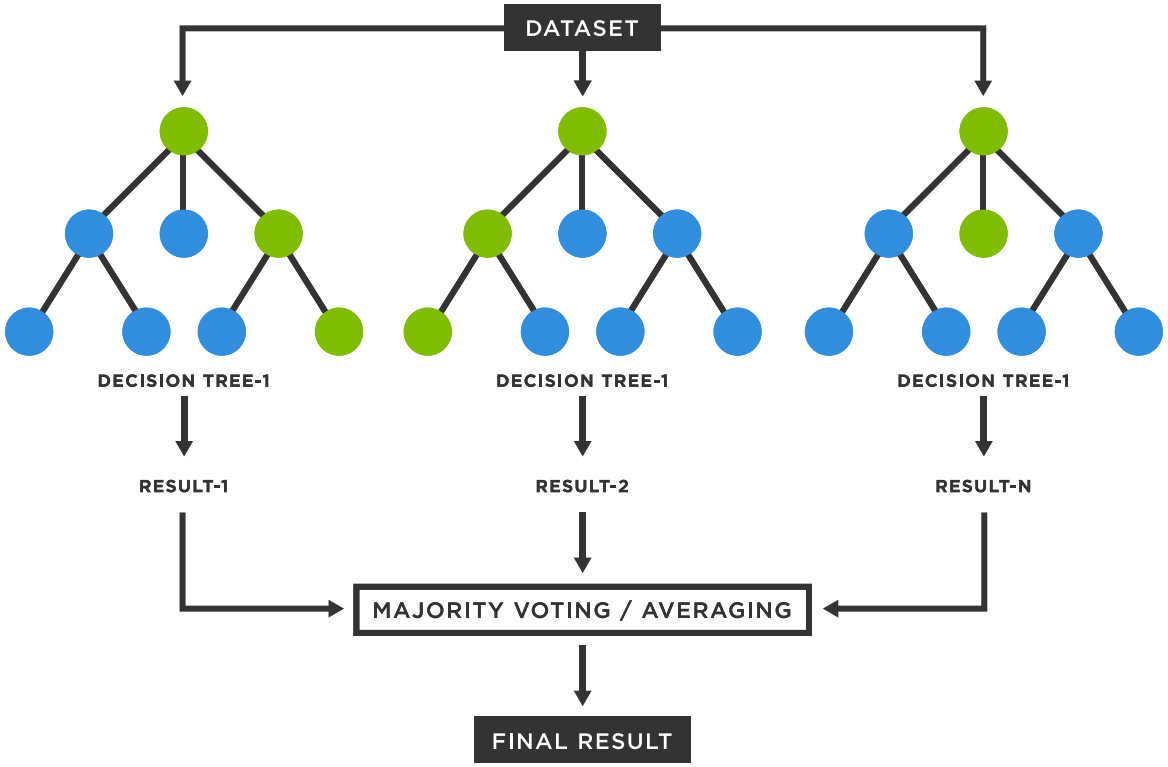


But what are the

Solution?

Leveraging **Machine Learning**

- Machine learning models can learn from data and improve over time.
- They can identify complex patterns that are not easily detectable by traditional methods.
- Random Forest Classification is a powerful machine learning technique well-suited for this task.



To make the dataset ready, it is needed to do

Preprocessing

Preprocessing

```
ds.isnull().sum()
```

Executed at 2024.03.26 16:14:24 in 80ms

319-328 | Length: 328, dtype: int64

	123 <unnamed>
com.android.launcher.permission.UNIN…	0
com.sec.android.iap.permission.BILLI…	0
com.htc.launcher.permission.UPDATE_S…	0
com.sec.android.provider.badge.permi…	0
android.permission.ACCESS_NETWORK_ST…	0
com.google.android.finsky.permission…	0
com.huawei.android.launcher.permissi…	0
android.permission.READ_SMS	0
android.permission.PROCESS_INCOMING_…	0
Label	0

There is no **Null** value in the used dataset.

```python
# Find columns with all zeros
zero_columns = ds.columns[(ds == 0).all()]

# Print the list of columns with all zeros
print("Columns with all zeros:")
print(zero_columns.tolist())
print("Number of columns with all zeros:", len(zero_columns))

#drop the columns that are zero always
ds = ds.drop(zero_columns, axis=1)
```

Executed at 2024.03.26 16:16:55 in 46ms

```
Columns with all zeros:
['ACCESS_PROVIDER', 'ACCESS_SERVICE', 'ACCESS_SHARED_DATA', 'activityCalled', 'ACTIVITY_RECOGNITION', 'ANT', 'ANT_ADMIN',
'AUTORUN_MANAGER_LICENSE_MANAGER', 'AUTORUN_MANAGER_LICENSE_SERVICE(.autorun)', 'BIND_GET_INSTALL_REFERRER_SERVICE', 'CHANGE_DISPLAY_MODE',
'CHECK_LICENSE', 'DATABASE_INTERFACE_SERVICE', 'DOWNLOAD_SERVICE', 'EXTENSION_PERMISSION', 'FULLSCREEN.FULL', 'GOOGLE_AUTH', 'GOOGLE_PHOTOS',
'JPUSH_MESSAGE', 'MAPS_RECEIVE', 'MESSAGE', 'PERMISSION', 'PERMISSION_RUN_TASKS', 'PLUGIN', 'READ', 'READ_ATTACHMENT', 'READ_AVESTTINGS',
'READ_CONTENT_PROVIDER', 'READ_DATA', 'READ_DATABASES', 'READ_GMAIL', 'READ_GSERVICES', 'READ_MESSAGES', 'RECEIVE', 'RECEIVE_SIGNED_DATA_RESULT',
'RESPOND', 'REQUEST', 'SEND', 'WRITE', 'WRITE_AVSETTING', 'WRITE_DATA', 'WRITE_DATABASES', 'android.permission.PROCESS_INCOMING_CALLS']
Number of columns with all zeros: 43
```

There were some permissions that are not used by any application. We have decided to **drop** them as they do not have any effect on the overall model accuracy.

Preprocessing

As we need to have non-string values in our dataset, we have replaced the ‘label’ column values with 1 and 0.

```
android_app_ds["Label"] = android_app_ds["Label"].replace(["Malware", "Benign"], [1, 0])
```

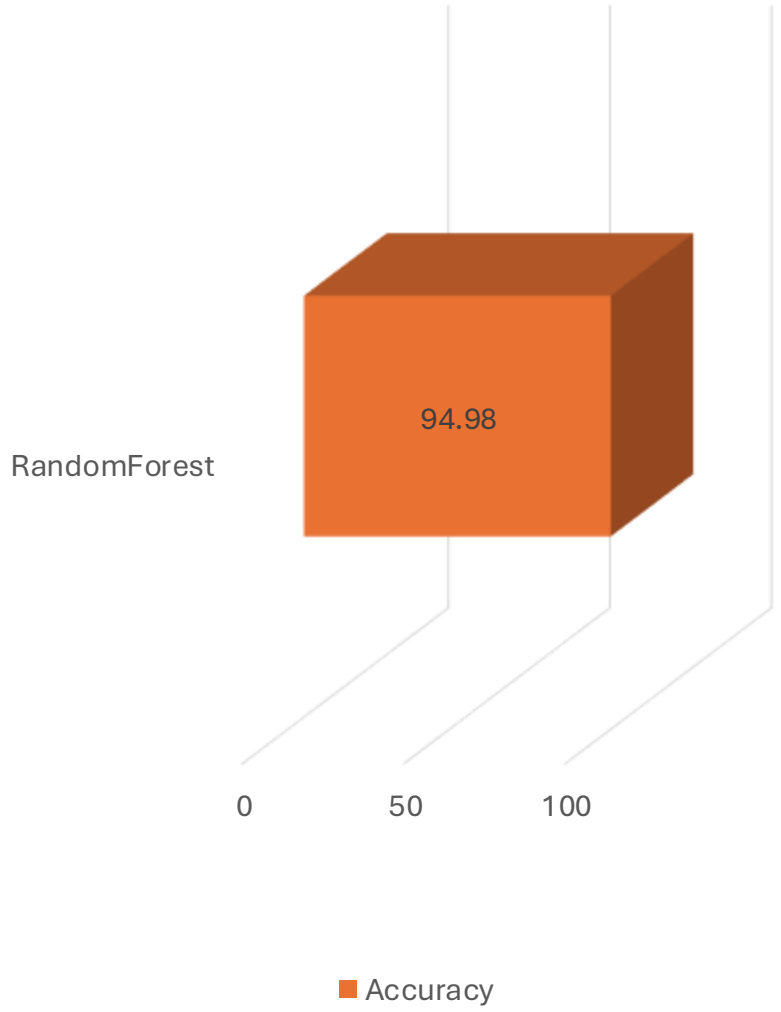
After converting this feature, we have no more things to do for our dataset to make the model.

To automate tasks and gain insights from data, we need to make a machine learning

Model

Machine Learning **Model** Creation

- Implemented a **R**andom **F**orest **C**lassifier (**RFC**), an ensemble learning technique.
- Random Forest combines **multiple decision trees** for improved accuracy and robustness.
- **Hyperparameter** tuning was performed to optimize the model's performance.



It was necessary to optimize

Hyperparameters,

But what are they?

Hyperparameters necessity

Random Forest models have settings called **hyperparameters** that control how the model learns from data. These parameters are not directly learned from the data itself, but rather need to be set before training. By adjusting these hyperparameters, we can fine-tune the model's performance to better suit our specific dataset and avoid overfitting.

- **Improved Accuracy:** Finding the right hyperparameter settings can lead to a more **accurate** model that generalizes well to unseen data.
- **Reduced Overfitting:** Hyperparameter tuning helps prevent the model from memorizing the training data instead of learning underlying patterns.
- **Better Model Behavior:** Tuning allows you to understand how different hyperparameters affect the model's behavior and achieve a desired result.

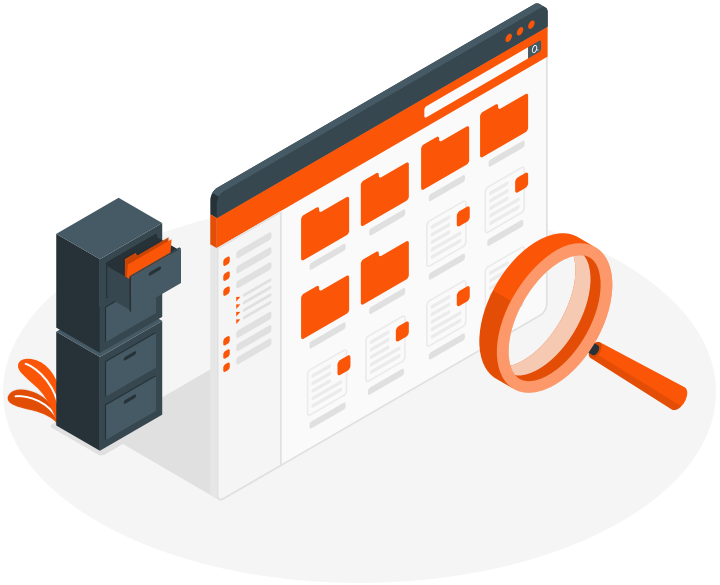
Which

Hyperparameters

do we used?

Used Hyperparameters

- **max_features:** This hyperparameter determines the number of features randomly considered at each split point in a decision tree within the forest.
- **max_depth:** This hyperparameter controls the maximum depth a tree in the random forest can grow. In simpler terms, it limits the number of splits a tree can make.



To optimize the hyperparameters, we have used

Grid Search

Grid Search

```
grid = {
    "n_estimators": [1000],
    "max_features": [1, 'sqrt', 'log2'],
    "max_depth": [None, 5, 10, 20, 30]
}
MS = GridSearchCV(estimator= RandomForestClassifier(),
                  cv=10,
                  param_grid=grid,
                  scoring= 'accuracy')
MS.fit(Xl, Yl)
```

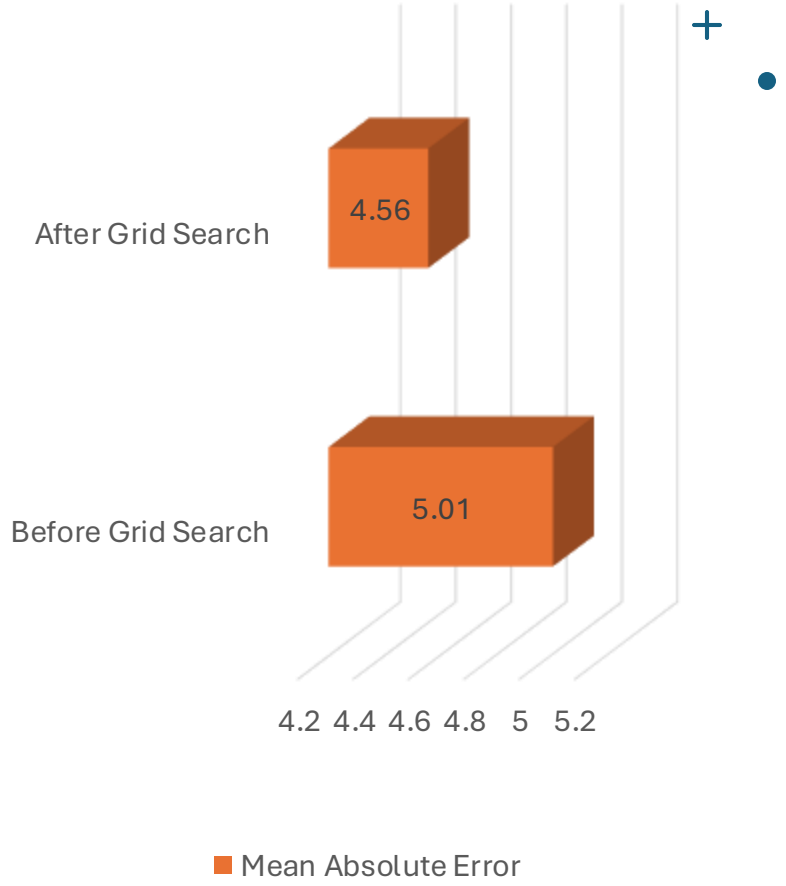
What are the

Results

that we have obtained?

Random Forest **Mean Absolute Error**

- **MAE** calculates the average absolute difference between the predicted probabilities (usually between 0 and 1) and the actual labels (0 for benign, 1 for malware)
- Hyperparameter tuning reduced the model's average error by 0.5% (MAE).



Confusion Matrix

- We have used confusion matrix to visualize the performance of our classification model.
- It helps us to understand how many predictions the model got right and wrong for each class in the data.