Android Malware Detection



Intelligence Compute Research Center

01/01/2018

**Catalog**

[1. Introduction 3](#_Toc503219127)

[2. Data Preprocessing 3](#_Toc503219128)

[2.1 Concatenating smali files 3](#_Toc503219129)

[3. Feature Engineering 4](#_Toc503219130)

[3.1 Opcode n-gram 4](#_Toc503219131)

[3.2 Permission 6](#_Toc503219132)

[3.3 API 7](#_Toc503219133)

[4. Model 9](#_Toc503219134)

[4.1 Random Forest 9](#_Toc503219135)

[4.2 XGboost 9](#_Toc503219136)

[5. Experiment and result analysis Feature Selection 10](#_Toc503219137)

[5.1 Dataset 10](#_Toc503219138)

[5.2 Evaluation 10](#_Toc503219139)

[5.3 Framework 11](#_Toc503219140)

[5.4 Dimensionality Reduction 11](#_Toc503219141)

[5.5 Feature Selection 14](#_Toc503219142)

[5.6 Model Training and result analysis 16](#_Toc503219143)

[6. Submission Result 18](#_Toc503219144)

[7. Conclusion 18](#_Toc503219145)

# Introduction

In recent years, with explosive growth of Android malware and due to the severity of its damages to smart phone users, the detection of Android malware has become an increasingly important topic in cyber security.

The project is about Android malware detection. It provides the train data that contains 1000 positive examples and 1000 negative examples. We need to extract features from the data to train machine learning models to determine whether an Android software is malicious.

# Data Preprocessing

## 2.1 Concatenating smali files

Since read all smali files in all Android apks in training set needs hours, we concatenate all the smali files in an apk in one file. After the concatenation we can read all of smali files in several minutes.

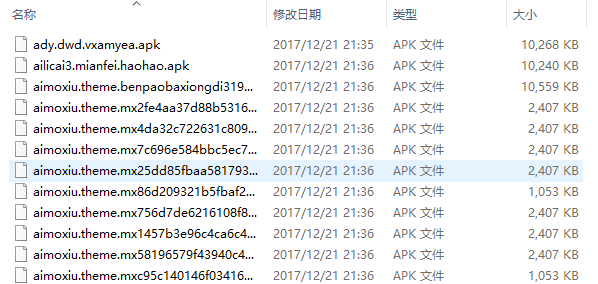


Figure 2.1 concatenated smali files

# Feature Engineering

## 3.1 Opcode n-gram

N-gram is a concept of natural language processing. Early speech recognition technology and statistical language model are closely related to it. N-gram is based on a simple assumption that the probability of a word is only related to the previous n-1 word, which can be calculated from a large number of corpus. For example, behind the word “eat”, the probability of “apple” or “pizza” is bigger than the probability of “road”, because the normal corpus doesn’t appear the combination of “eat the road”. In conclusion, n-gram contains some linguistic features.

The idea of applying n-gram to malware detection was first proposed by Tony .et al in 2004 in the paper “n-gram-based Detection of New Malicious Code”. And their method was based on ByteCode. In 2008, the paper of Moskovitch et al., “Unknown Malcode Detection Using OPCODE Representation”, made use of OPCODE instead of ByteCode and that is more reasonable.

In our project, we extract 1-gram, 2-gram, 3-gram, 4-gram opcode separately first, and then we try some combination of above four features using random forest classifier, the features are on the table 3-1. And figure 3-1 shows part of the opcode 3-gram feature.

Since the dimensions of 3-gram feature and 4-gram feature are too large, after extracting both of them we reduce those dimensions according to the importance of each feature, which is in section 4.

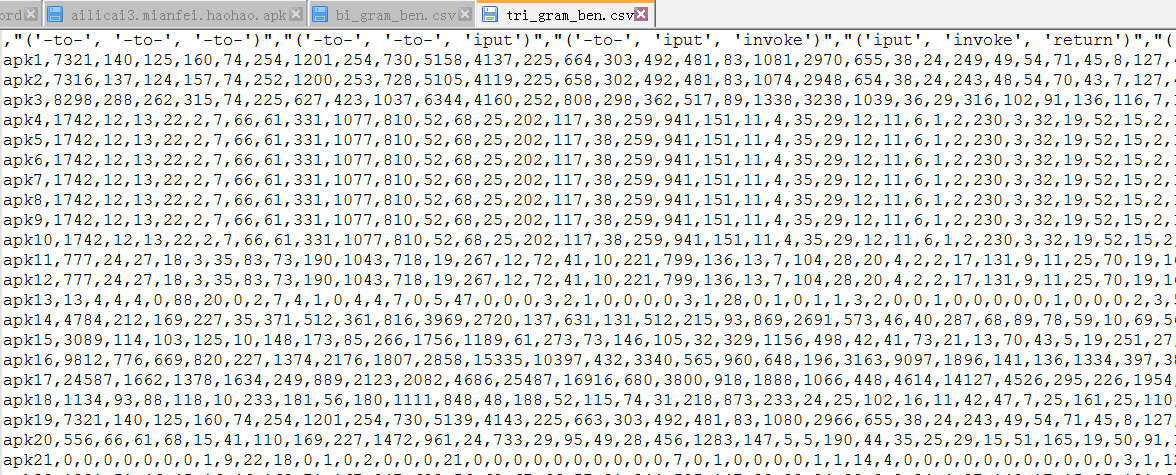


Figure ‑ opcode 3-gram feature

Table 3- N-gram Features

|  |
| --- |
| **N-gram Features** |
| 1-gram |
| 2-gram |
| 3-gram |
| reduced\_3-gram |
| 4-gram |
| reduced\_4-gram |
| 1-gram+2-gram |
| 2-gram+3-gram |
| 2-gram+reduced\_3-gram |
| 1-gram+2-gram+3-gram |
| 1-gram+2-gram+3-gram+4-gram |

## 3.2 Permission

The Android permission system controls the access of applications on all kinds of resources. Some resources of Android system contain the privacy of users or mobile phone, such as mobile phone number, location and short massage. Therefore, the permissions on the access of these resources are important.

Permission Feature can describe the malicious tendency of Android applications, which can distinguish the malware. The permission sequences for Android application to apply at installation and running are recorded in the configuration Android Manifest.xml in an Android application, which is in the file of APK, as the figure 3-2 and the figure 3-3.

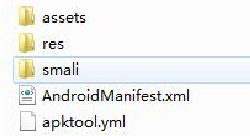


Figure ‑ The structure of APK file



Figure ‑ A sample of configuration Android Manifest.xml

After extracting the Permission Feature, we reduce those dimensions according to the importance of each feature. The details are in section 5.

## 3.3 API

The process of stealing privacy information such as phone numbers in a user's mobile phone requires a lot of system API functions in Android system. Apparently, the acquirement of user privacy sensitive data about malicious applications is ultimately realized by calling the system API. Different from Android Permission Features, the access to sensitive privacy resources can control a lot of system APIs to access these resources. However, the malicious behaviors described by the system API is more straightforward, and it will show which system to access sensitive privacy resources directly. Therefore, in the Malicious opcode detection, we can find some malicious actions from API.

For the extracting the System API features, we first process the Android applications by decompiling. Folder smalis store the Dalvik byte, and we use it to extract the API features. Each .smali file corresponds to a class of Android application source code. It is worth noting that the command operation code for the system API calling in the Dalvik assembly language is the invoke command. So we retrieve the invoke instruction in the .smali file. Then we can realize the extractions of API calling to the Android application. The figure 3-4 shows the anti-assembly code snippet of an Android application. In sixth line, the invoke command makes a call to the system API.

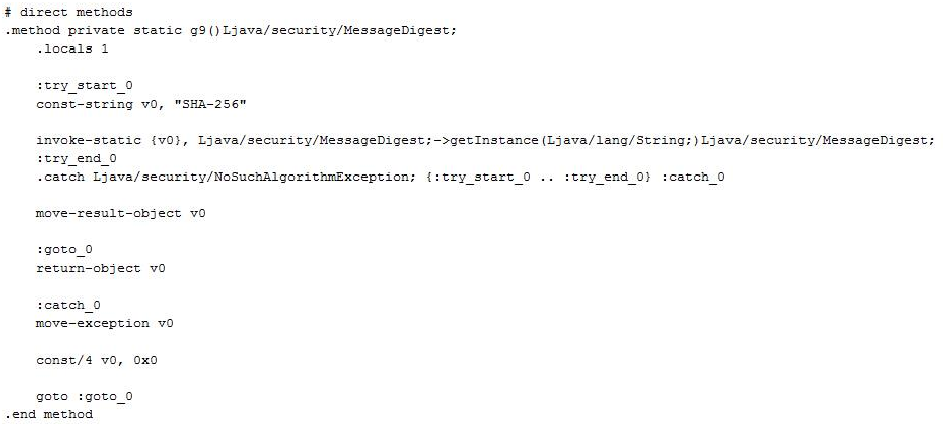


Figure ‑ An anti-assembly code snippet of an Android application

# Model

## Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classification or mean prediction of the individual trees.

## XGboost

XGBoost is an algorithm that has been dominating applied machine learning and Kaggle competitions for structured or tabular data. It is an implementation of gradient boosting machines created by Tianqi Chen, now with contributions from many developers. It belongs to a broader collection of tools under the umbrella of the Distributed Machine Learning Community or DMLC who are also the creators of the popular mxnet deep learning library.

# Experiment and result analysis Feature Selection

## 5.1 Dataset

In this project, due to the usage of machine learning, we need to divide the dataset into training data and test data. There are 2000 instances. We use the 80% of dataset as training data, and 20% of dataset as test data.

Table 5 - 1 The partitionof dataset

|  |  |
| --- | --- |
| **Type of data** | **Numbers of data** |
| Training data | 1600 |
| Test data | 400 |

## 5.2 Evaluation

We use F1 score to evaluate the results. the F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score:



p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples. Relevant is equivalent to actually having the disease. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

## 5.3 Framework

**Feature engineering**

**Feature selection**

**Final submission**

Xgboost

**Modeling**

Opcode count:

1,2,3,4-grams

Android

Permission features

System API features

RandomForest

Figure 5‑1 The framework

## 5.4 Dimensionality Reduction

Because the dimension of features are very high and some of features are over 100,000 dimension which can result in overfitting. Almost all the feature we extracted have a lot of redundant dimensions. So we use random forests to reduce dimensionality of features.

We add a prefix “reduced” to represent that feature has been dimensionally reduced.

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classification or mean prediction of the individual trees.

Before training the model, we extract different features from data and save them in documents. Then, we read features from these documents and separately label positive examples and negative examples with “0” and “1”. All these examples and labels are stored in variable “data”.

In Random forests model, we use five-fold cross validation to get the more reliable performance of model. First, we randomly separate data into train and test data. Then, we train the random forest model and calculate two evaluation parameters which are “f1-score” and “accuracy”. After five folds validations, we get the average “f1-score” and “accuracy” as the final performance of model.

In random forests model, we only tune “n\_estimators” parameters with 200 and 400. And other parameters are default.

We use random forests to select features, and the results of random forests model with different features will be shown in chapter 5.

We select important features by using feature\_importances calculated in the model as follows. And we sort the features by importance.

*feature\_sorted=sorted(zip(map(lambda x: round(x, 4), srf.feature\_importances\_), names), reverse=True)*

The top important features of 3-gram are shown as follows.

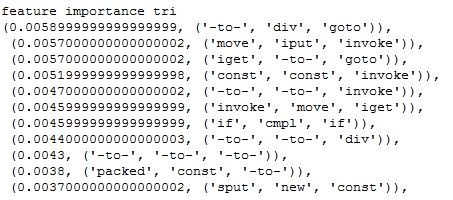


Figure ‑2 The top important features of 3-gram

The top important features of API are shown as follows.

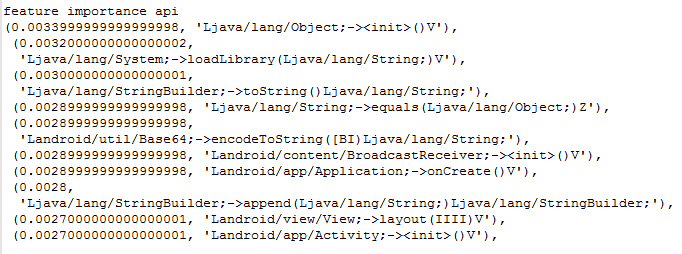


Figure ‑3 The top important features of API

The top important features of permission are shown as follows.

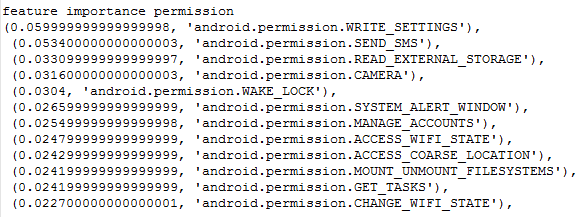


Figure ‑4 The top important features of permission

The top important features of reduced\_3-gram, reduced\_API and reduced\_permission are shown as follows.

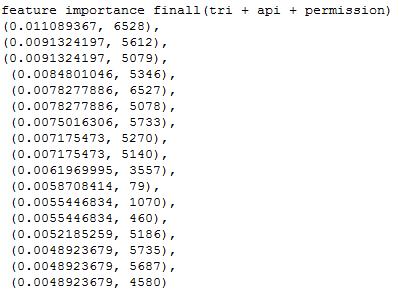


Figure ‑5 The top important features of reduced\_3-gram, reduced\_API and reduced\_permission

We reduce the dimension of features to retain the most important features by setting a threshold and only keep dimensions that above that threshold.

## 5.5 Feature Selection

Different feature may capture different aspect of attributes of malwares. More feature lead to longer training time. Since our time is limited, we just try some com

The performance of different features in random forests model are shown as follows.

Table 5 - The performance of different features

|  |  |  |
| --- | --- | --- |
| Feature | F1-score | Accuracy |
| 1-gram | 0.8844 | 0.8787 |
| 2-gram | 0.9002 | 0.8960 |
| 3-gram | 0.9040 | 0.9001 |
| reduced\_3-gram | 0.9025 | 0.8985 |
| reduced\_4-gram | 0.8995 | 0.8959 |
| 1-gram+2-gram | 0.8953 | 0.8909 |
| 2-gram+3-gram | 0.9009 | 0.8970 |
| 2-gram + reduced 3-gram | 0.8994 | 0.8954 |
| permission | 0.8818 | 0.8817 |
| 2-gram + permission | 0.8986 | 0.8944 |
| reduced\_3-gram + reduced\_permission | 0.9046 | 0.9015 |
| API | 0.9142 | 0.9101 |
| reduced\_API | 0.9177 | 0.9147 |
| reduced\_3-gram + reduced\_API | 0.9112 | 0.9076 |
| reduced\_3-gram + reduced\_API + reduced\_permission | 0.9268 | 0.9248 |

From the table, we can find that 3-gram performances the best in top n-gram features. And we can find that the accuracy of reduced 3-gram is almost the same as 3-gram. So, we decide to use reduced 3-gram as one feature in the final model which can shorten training time of the model and avoid overfitting. On the basis of reduced 3-gram feature, we combine reduced permission feature and get an accuracy of 0.9015, which is higher than permission feature and a little higher than reduced 3-gram. So we decide to use reduced permission as another feature. Finally, we train the model with API feature and get a accuracy of 0.9101. Similarly, we reduce the dimension of API and find that reduced API get a higher accuracy than API. So, we use reduced API as the third feature.

In the end, we select reduced\_3-gram, reduced\_API and reduced\_permission as the final features by training random forests model. The accuracy of model with the three features is 0.9248.

## 5.6 Model Training and result analysis

After the feature selection by training random forests, we train the XGBoost model with these features.

The expression of data in XGBoost is the same as random forests model. All examples and labels are also stored with variable “data”.

We use reduced\_3-gram, reduced\_API and reduced\_permission features to train and tune parameters of our model.

We tune parameters by using grid search. First, we import the libraries we will need to do grid search for XGBoost by following code.

*import xgboost as xgb*

*from sklearn.grid\_search import GridSearchCV*

We start tuning on the n\_estimators first. We set the objective to ‘binary:logistic’ since this is a binary classification problem. We can see which parameters perform the best by check the grid scores. The grid scores of n\_estimators parameter are shown as follows. We can find that 200 is the best parameter value of {100, 200, 300, 400, 500}. After several similar times of tuning , we get the best parameter value 225 { 225, 250, 275} and use it as the final parameter value.

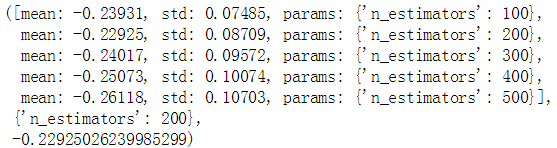


Figure ‑ The grid scores of different n\_estimators

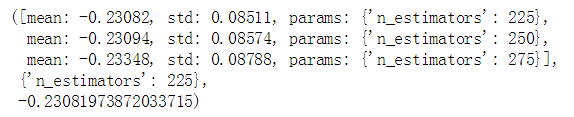


Figure ‑ The grid scores of different n\_estimators

Then, we tune on the maximum depth of the trees, along with the min\_child\_weight, which is very similar to min\_samples\_split in sklearn’s version of gradient boosted trees. The grid scores of max\_depth parameter are shown as follows. We can find that 3 is the best parameter value of {3, 5, 6, 7} and use it as the final parameter value.

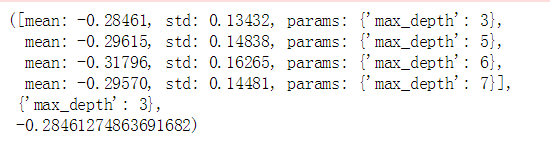


Figure ‑ The grid scores of different max\_depth

By tuning parameters of XGBoost model, we get the accuracy of about 0.93 which is the best performance in previous models. So we use the XGBoost with reduced\_3-gram, reduced\_API and reduced\_permission features as the final test model.

# Submission Result

We label the test data by the trained XGBoost model (on all of training data) and on the test set our performance is shown on figure 6-1.

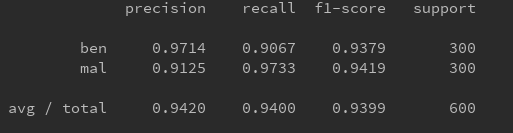


Figure ‑ The result of test data

# Conclusion

We learned a lot from this project. It is important to search for information and have a study on this field before we start coding. Proper preprocessing on data saves us lots of time.

In addition, organize the code properly is very important. Owing to time constraints, our code is a little messy. So, next we want to do is to learn more about python and refactor our code. Besides, we just tune some of parameters of our model in this project and do not have enough time to fine tune the parameters. When we have spare time, we will do more experiments on the data, fine tune the parameters to gain more insight and getting deep into the world of machine learning.