Tweet analysis for Android malware detection in Google Play Store

by

Zhiang Fan

B.A., Kansas State University, 2016

A REPORT

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2019

Approved by:

Major Professor
Doina Caragea

Abstract

There are many approaches to detect if an app is malware or benign, for example, using static or dynamic analysis. Static analysis can be used to look for APIs that are indicative of malware. Alternatively, emulating the app's behavior using dynamic analysis can also help in detecting malware. Each type of approach has advantages and disadvantages. To complement existing approaches, in this report, I studied the use of Twitter data to identify malware. The dataset that I used consists of a large set of Android apps made available by AndroZoo. For each app, AndroZoo provides information on vt_detection, which records number of anti-virus programs in VirusTotal that label the app as malware. As an additional source of information about apps, I crawled a large set of tweets and analyzed them to identify patterns of malware and benign apps in Twitter. Tweets were crawled based on keywords related to Google Play Store app links. A Google Play Store app link contains the corresponding app's ID, which makes it easy to link tweets to apps. Certain fields of the tweets were analyzed by comparing patterns in malware versus benign apps, with the goal of identifying fields that are indicative of malware behavior. The classification label from AndroZoo was considered as ground truth.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to my advisor Dr. Doina Caragea for her continuous encouragement, patience, faith and for creating a very positive atmosphere for completing my report. In addition, I want to thank my report committee members, Dr. Mitchell Neilsen and Dr. Daniel Andresen, for monitoring my progress and providing valuable feedback.

Chapter 1

Introduction

This report is focused on Android malware detection. A large number of malware apps enter different markets such as Google Play Store, Amazon App Store, AnZhi (GoAPK), AppChina and App Store, and can cause significant damage and losses. For example, the Trojan malware is able to harvest accounts and photos stored in smart phones. Nowadays, more and more people are using payment apps like AliPay, Apple Pay, PayPal, and there are serious concerns about mobile security and issues caused by malware apps. There are many approaches that can potentially identify malware apps, in particular approaches based on static and dynamic analysis. Static analysis can be used to look for APIs and permissions that are indicative of malware, while dynamic analysis can be used to emulate the app's behavior and identify malicious behavior. Each type of approach has advantages and disadvantages. In this report, I consider an approach that can complement existing approaches. Specifically, I study the use of Twitter data in identifying malware.

Twitter is one of the most popular social media platforms in the world. Twitter records a large amount of data about many topics as posted by users. Among other things, Twitter data includes information about Android malware. We hypothesize that it is possible to use Twitter data to analyze and possibly detect malware. Even if the analysis in this report cannot definitely tell if an app is malware or not, it can potentially complement other methods for detecting malware and increase the confidence in malware classification.

Towards this goal, we start with a large set of Android apps provided by AndroZoo (Allix et al., 2016). These apps have been scanned for malicious behavior using VirusTotal (Sood, 2017), and the labeled inferred based on the scan will be used as ground truth labels. In addition to the information available in the dataset, we also crawl a large number of tweets related to Android apps based on Google Play Store app links, which facilitate matching tweets to apps. We analyze different fields in the tweets crawled, with the goal of identifying fields that are indicative of malicious behavior. In particular, we study: tweet language to identify the most frequent language for malware tweets versus benign tweets; tweet text to identify text patters used in malware versus benign; and tweet user metrics to identify characteristics of users who post tweets about malware by comparison with users who post tweets about benign apps.

Chapter 2

Data Collection and Preprocessing

There are two main data sources used in this work: tweets about Android apps and app information.

2.1 Tweet Data Collection

This section is about downloading tweets. All tweets were collected from Twitter using the Twitter Streaming API.

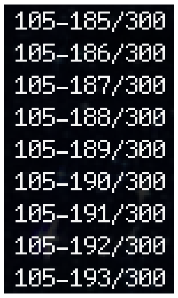
I used two crawlers. One is from Jordan DeLoach (TweetCollector.scala), and the other one was written by myself (TwitterStreamDownloader.py). For the Python crawler, each downloaded file, crawled by this crawler, contains about 300 tweets. Each tweet is saved as a single line. The structure of the tweet object can be found in Twitter developer page (https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object).

The target data consists of tweets with Google Play links. Thus, the string "play.google.com/store/apps/details?id=" would be a great keyword that allow to check if tweets contain Google Play links. However, Twitter Streaming API does not accept keywords with special characters, for example "/", so the string "play google com store apps details id" as the search keyword, instead of the exact URL in "twitterStream.filter(track=keywords)" is a better choice. This means all the words in the search string, split by space, have to occur in the tweet to satisfy the search requirement. Space in this case can be considered as "AND". (Note: If using "play,google,com,store,apps,details,id" as keyword, any words split by comma found in tweet can meet the requirement. Comma in this case is considered as "OR".)

Twitter Streaming API is provided by Twitter, and only pulls real-time tweets. Details of this API can be found in Twitter developer web page (https://developer.twitter.com).

The new script that I wrote, TwitterStreamDownloader.py, will be discussed in the rest of this section. Before running the script, a few things need to be installed: Python3, tweepy. Keys required for API are OAuthAccessToken, OAuthAccessTokenSecret, OAuthConsumerKey and OAuthConsumerSecret. These keys are authenticated by Twitter.

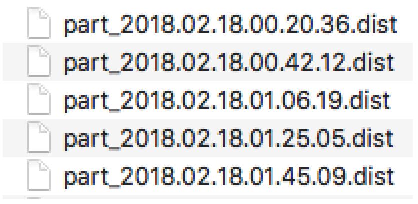
When running this crawler, one will get output like the one shown in Figure 2.1, which provides information on what file and what tweet in each file the crawler is working on.



```
105-185/300
105-186/300
105-187/300
105-188/300
105-189/300
105-190/300
105-191/300
105-192/300
105-193/300
```

Figure 2.1: *The output of the crawler is printed in the format A-B/C. Each file contains C (300) tweets, which amount to about 1.4MB. The first line says the crawler is crawling the 185th tweet (B) in the 105th file (A).*

The content of the directory where the files are written looks like in Figure 2.2



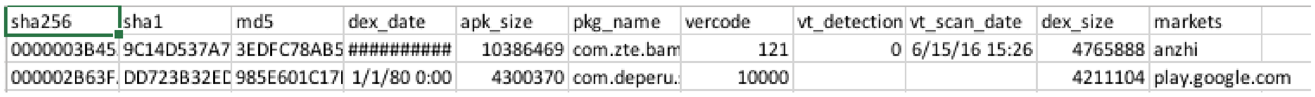
```
part_2018.02.18.00.20.36.dist
part_2018.02.18.00.42.12.dist
part_2018.02.18.01.06.19.dist
part_2018.02.18.01.25.05.dist
part_2018.02.18.01.45.09.dist
```

Figure 2.2: *In the directory with downloaded files, the number after "part" is the date and the time when the file was crawled.*

2.2 Import AndroZoo's App Data to MongoDB

The app information together with malware or benign information from VirusTotal is available from AndroZoo. The app information can be dowloaded from Androzoo's website as a file named "latest.csv". I imported the dataset to MongoDB for easier management. I also stored the app data file in some directory for future reference.

There are 11 columns in this CSV file. Details are in Fig 2.3



sha256	sha1	md5	dex_date	apk_size	pkg_name	vercode	vt_detection	vt_scan_date	dex_size	markets
0000003B45	9C14D537A7	3EDFC78AB5	##########	10386469	com.zte.bam	121	0	6/15/16 15:26	4765888	anzhi
000002B63F.	DD723B32ED	985E601C17I	1/1/80 0:00	4300370	com.deperu.:	10000			4211104	play.google.com

Figure 2.3: *Example of app in the latest.csv file. Any of sha256, sha1 and md5 can be considered as ID for the app. But uniqueness is not guaranteed for some of them. In these IDs, sha256 is the strongest one but takes more space. The fields we need to use are sha256, pkg_name and vt_detection. The vt_detection field gives the number of anti-virus programs that consider this app as malware.*

MongoDB was installed by typing:

```
$ brew install mongodb
```

Pymongo was installed by typing:

```
$ pip install pymongo
```

Subsequently, one can run MongoDB by typing:

```
$ services start mongodb
```

Finally, one can import the entire file to MongoDB by typing following command in Terminal (where "latest" is the collection name for the app data.)

```
$ mongoimport -d <database name> -c latest --type csv
--file <path of source file> --headerline
```

2.3 Preprocess Tweets and Import to MongoDB

This section is about preprocessing the tweet data and storing it into MongoDB.

The script I used is called “mongoDBConnection.py”. The script removes duplication of tweets, inserts several fields to tweet objects, and finally imports the objects into MongoDB. The reason for using NoSQL is that SQL databases do not handle json files very well.

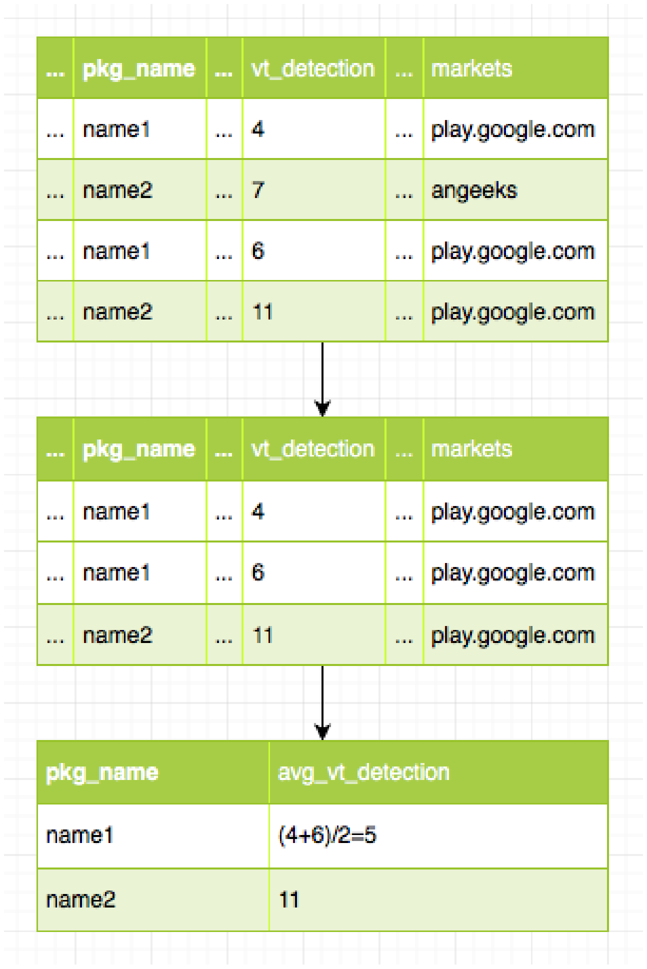
Before running the script, there are a few things to do. First, install pymongo. Then in order to avoid duplication of tweets, it is necessary to create an index for the tweet ID (“id_str”). Enter MongoDB shell by typing “mongo,” then type the following code (“tweets” in the command is collection name I use for the tweet data):

```
> db.tweets.createIndex( { id_str:1 }, { unique:true } )
```

Now, we are ready to run the script. This script firstly reads useful data from the collection “latest”. Performance would be low if the script read the entire collection, so I only select applications in Google market. Then, group the data by package name, calculate average of “vt_detection”. This processing is described in Fig 2.4.

In Google Play market, each app has unique app ID which is the package name, however, in collection latest, one package name may correspond to multiple “sha256”s, since different releases of an app are also recorded in the original Androzoo’s CSV file. So, it is useful to evaluate the final detection based on all releases for each APP. If package name is unique in collection “latest” then classify app based on the “vt_detection” as shown on the left side of Fig 2.5. However, if the package name is not unique, getting the average of “vt_detection” is the simplest way to evaluate if APP is malware or not, and the labeling in this case is described on the right side of Figure 2.5. For example, if an APP has 100 releases, 99 of them have 0 in “vt_detection” field, 1 of them has 1, then average is already greater than 0, which means it would be defined as a unknown instead of benign if using left side of the figure.

Next, the script reads files whose names end with “.dist”, from the given directory. When it finishes importing tweet data, it’s always good to check if duplicates exist in the collection by typing following code in mongo shell.



...	pkg_name	...	vt_detection	...	markets
...	name1	...	4	...	play.google.com
...	name2	...	7	...	angeeks
...	name1	...	6	...	play.google.com
...	name2	...	11	...	play.google.com

...	pkg_name	...	vt_detection	...	markets
...	name1	...	4	...	play.google.com
...	name1	...	6	...	play.google.com
...	name2	...	11	...	play.google.com

pkg_name	avg_vt_detection
name1	(4+6)/2=5
name2	11

Figure 2.4: *NoSQL query steps to select data from the latest collection and save it to a dictionary.*

```
> db.tweets.aggregate([{"$group":{"_id":"$id_str",
"count":{"$sum":1}}},{"$match":{"count":{"$gt":1}}}])
```

An example of tweet crawled in json format is shown in Figure 2.6. Several fields are added into tweet objects when running the script. These fields are “appid_list”, “appid_clas_list”, “result_clas”, “mal_counter”, “ben_counter”, “und_counter”, “unk_counter” and “une_counter”. “appid_list” is a list of Google Play app’s IDs in each tweet. Since Google Play app contains app ID in its URL, it’s easy to get the ID from the URL. Also, each tweet may have multiple links, which explains why I uses a list of IDs. Furthermore, if a tweet contains two app links, one of which is malware and the other one is benign, then this tweet can be classified as both benign and malware. Thus, “app_clas_list” contains

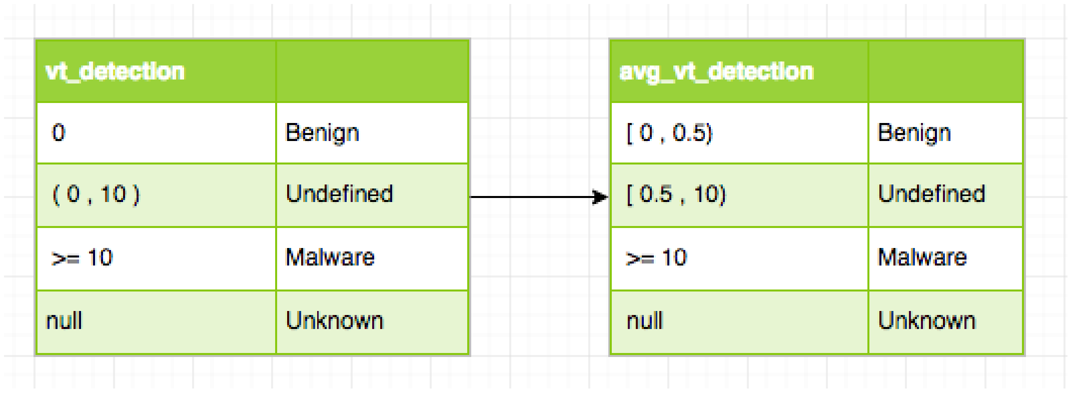


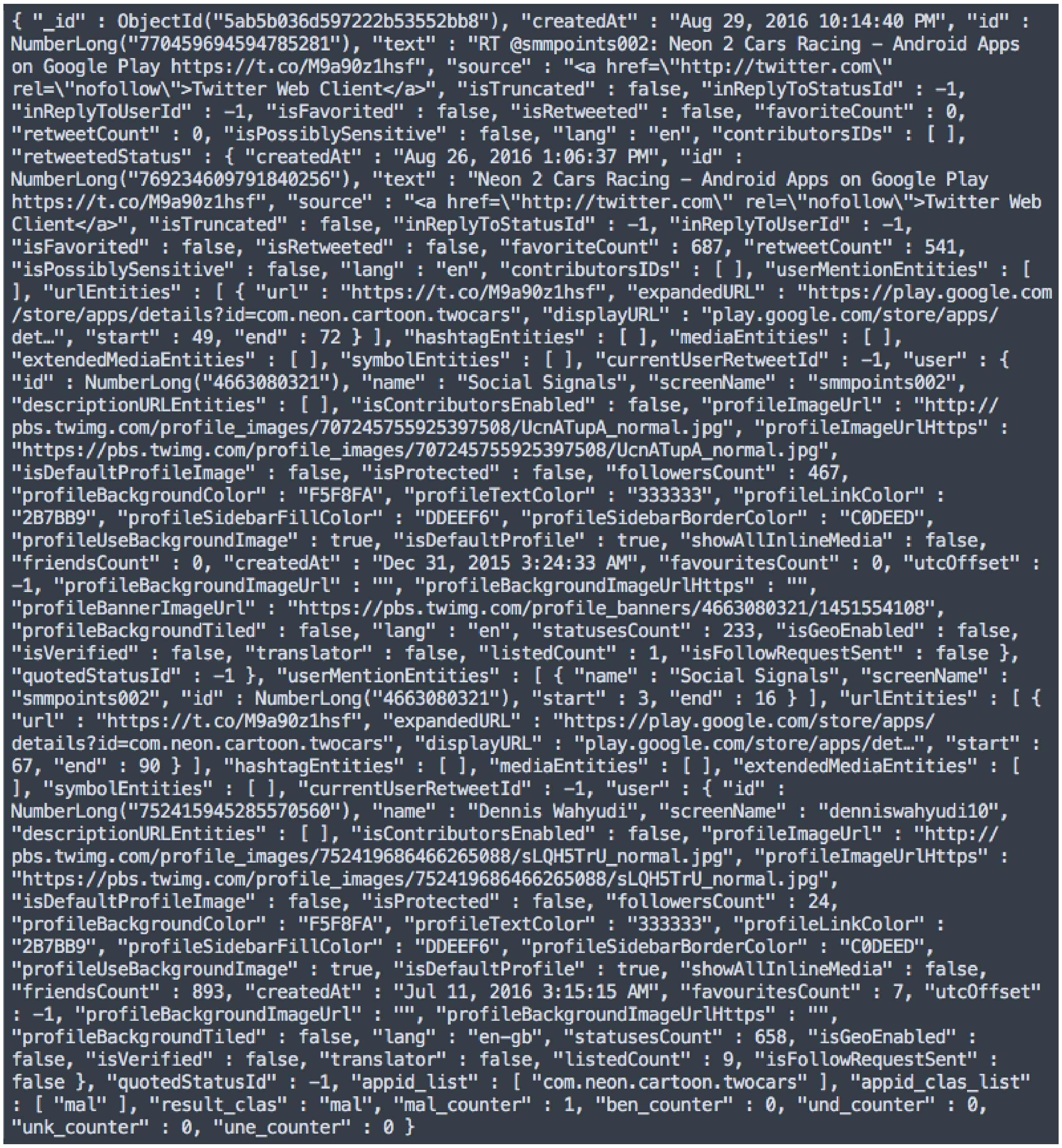
Figure 2.5: *This figure describes how to classify benign, malware, undefined and unknown based on the average of vt_detection*

the class of each app in "appid_list". The type of an app can be "mal" (malware), "ben" (benign), "und" (undefined), "unk" (unknown) and "une" (does not exist in the collection latest). "mal_counter" is a count of malware APPs in "appid_list", similar to "ben_counter", "und_counter" and "unk_counter". "result_clas" is the evaluated type of the current tweet based on appropriate defined rules. In this case, I define rules like this: if tweet contains malware's id, this tweet is classified as "mal". Then, if tweet doesn't contain malware id but contain benign id, this tweet is classified as "ben", then undefined, unknown and the IDs non-existent in the collection "latest".

Before we describe the analysis performed, here is a brief summary of the crawled data. As shown in Table 2.1, the total number of tweets with Google Play links is 1651101. Out of this, 1030641 tweets contain benign APP links, 435 contain malicious links, 113874 tweets contain undefined and 113320 tweets contain unknown links. As can be seen in the table, the total number of tweets is smaller than the sum of the four classes, as some tweets may contain multiple app links.

Table 2.1: *Data crawled*

Total Google Tweets	Benign	Malware	Undefined	Unknown
1651101	1030641	435	113874	113320



```
{ "_id" : ObjectId("5ab5b036d597222b53552bb8"), "createdAt" : "Aug 29, 2016 10:14:40 PM", "id" :
NumberLong("770459694594785281"), "text" : "RT @smmpoints002: Neon 2 Cars Racing - Android Apps
on Google Play https://t.co/M9a90z1hsf", "source" : "<a href=\"http://twitter.com\"
rel=\"nofollow\">Twitter Web Client</a>", "isTruncated" : false, "inReplyToStatusId" : -1,
"inReplyToUserId" : -1, "isFavorited" : false, "isRetweeted" : false, "favoriteCount" : 0,
"retweetCount" : 0, "isPossiblySensitive" : false, "lang" : "en", "contributorsIDs" : [ ],
"retweetedStatus" : { "createdAt" : "Aug 26, 2016 1:06:37 PM", "id" :
NumberLong("769234609791840256"), "text" : "Neon 2 Cars Racing - Android Apps on Google Play
https://t.co/M9a90z1hsf", "source" : "<a href=\"http://twitter.com\" rel=\"nofollow\">Twitter Web
Client</a>", "isTruncated" : false, "inReplyToStatusId" : -1, "inReplyToUserId" : -1,
"isFavorited" : false, "isRetweeted" : false, "favoriteCount" : 687, "retweetCount" : 541,
"isPossiblySensitive" : false, "lang" : "en", "contributorsIDs" : [ ], "userMentionEntities" : [
], "urlEntities" : [ { "url" : "https://t.co/M9a90z1hsf", "expandedURL" : "https://play.google.com
/store/apps/details?id=com.neon.cartoon.twocars", "displayURL" : "play.google.com/store/apps/
det…", "start" : 49, "end" : 72 } ], "hashtagEntities" : [ ], "mediaEntities" : [ ],
"extendedMediaEntities" : [ ], "symbolEntities" : [ ], "currentUserRetweetId" : -1, "user" : {
"id" : NumberLong("4663080321"), "name" : "Social Signals", "screenName" : "smmpoints002",
"descriptionURLEntities" : [ ], "isContributorsEnabled" : false, "profileImageUrl" : "http://
pbs.twimg.com/profile_images/707245755925397508/UcnATupA_normal.jpg", "profileImageUrlHttps" :
"https://pbs.twimg.com/profile_images/707245755925397508/UcnATupA_normal.jpg",
"isDefaultProfileImage" : false, "isProtected" : false, "followersCount" : 467,
"profileBackgroundColor" : "F5F8FA", "profileTextColor" : "333333", "profileLinkColor" :
"2B7BB9", "profileSidebarFillColor" : "DDEEF6", "profileSidebarBorderColor" : "C0DEED",
"profileUseBackgroundImage" : true, "isDefaultProfile" : true, "showAllInlineMedia" : false,
"friendsCount" : 0, "createdAt" : "Dec 31, 2015 3:24:33 AM", "favouritesCount" : 0, "utcOffset" :
-1, "profileBackgroundImageUrl" : "", "profileBackgroundImageUrlHttps" : "",
"profileBannerImageUrl" : "https://pbs.twimg.com/profile_banners/4663080321/1451554108",
"profileBackgroundTiled" : false, "lang" : "en", "statusesCount" : 233, "isGeoEnabled" : false,
"isVerified" : false, "translator" : false, "listedCount" : 1, "isFollowRequestSent" : false },
"quotedStatusId" : -1 }, "userMentionEntities" : [ { "name" : "Social Signals", "screenName" :
"smmpoints002", "id" : NumberLong("4663080321"), "start" : 3, "end" : 16 } ], "urlEntities" : [ {
"url" : "https://t.co/M9a90z1hsf", "expandedURL" : "https://play.google.com/store/apps/
details?id=com.neon.cartoon.twocars", "displayURL" : "play.google.com/store/apps/det…", "start" :
67, "end" : 90 } ], "hashtagEntities" : [ ], "mediaEntities" : [ ], "extendedMediaEntities" : [
], "symbolEntities" : [ ], "currentUserRetweetId" : -1, "user" : { "id" :
NumberLong("752415945285570560"), "name" : "Dennis Wahyudi", "screenName" : "denniswahyudi10",
"descriptionURLEntities" : [ ], "isContributorsEnabled" : false, "profileImageUrl" : "http://
pbs.twimg.com/profile_images/752419686466265088/sLQH5TrU_normal.jpg", "profileImageUrlHttps" :
"https://pbs.twimg.com/profile_images/752419686466265088/sLQH5TrU_normal.jpg",
"isDefaultProfileImage" : false, "isProtected" : false, "followersCount" : 24,
"profileBackgroundColor" : "F5F8FA", "profileTextColor" : "333333", "profileLinkColor" :
"2B7BB9", "profileSidebarFillColor" : "DDEEF6", "profileSidebarBorderColor" : "C0DEED",
"profileUseBackgroundImage" : true, "isDefaultProfile" : true, "showAllInlineMedia" : false,
"friendsCount" : 893, "createdAt" : "Jul 11, 2016 3:15:15 AM", "favouritesCount" : 7, "utcOffset"
: -1, "profileBackgroundImageUrl" : "", "profileBackgroundImageUrlHttps" : "",
"profileBackgroundTiled" : false, "lang" : "en-gb", "statusesCount" : 658, "isGeoEnabled" :
false, "isVerified" : false, "translator" : false, "listedCount" : 9, "isFollowRequestSent" :
false }, "quotedStatusId" : -1, "appid_list" : [ "com.neon.cartoon.twocars" ], "appid_clas_list"
: [ "mal" ], "result_clas" : "mal", "mal_counter" : 1, "ben_counter" : 0, "und_counter" : 0,
"unk_counter" : 0, "une_counter" : 0 }
```

Figure 2.6: *Example of one raw tweet downloaded by crawler.*

Chapter 3

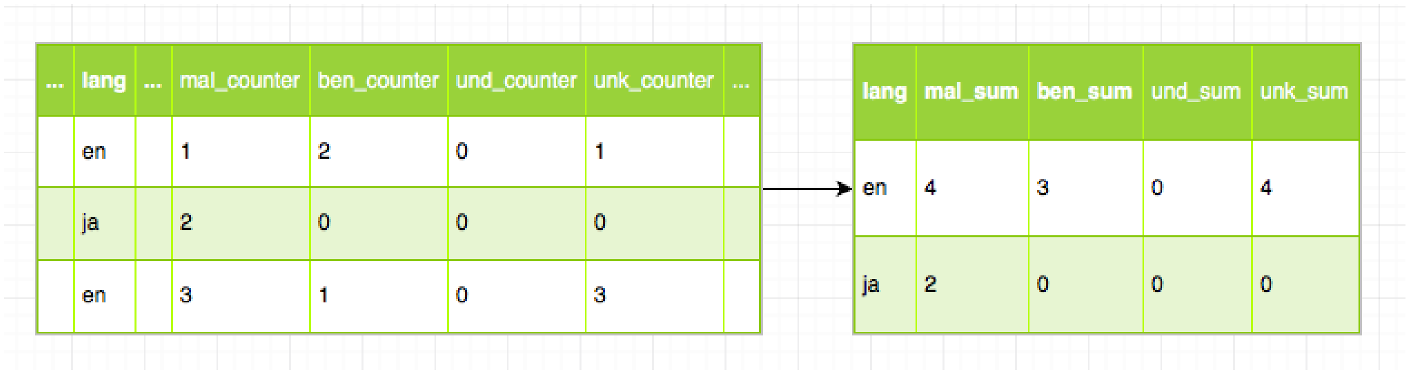
Tweet Language Analysis

The analysis in this chapter is based on data imported to MongoDB as described in Chapter 2. The focus of this chapter is on understanding how the languages of tweets relate to different classes of apps. Most crawled tweets contain Google Play links, and these links correspond to specific apps. We study if there is a correlation between malware or benign apps and the language of the tweets that refer to those apps. we assume it is not possible to identify malware just based on language, although the language information might be useful in increasing the confidence in a malware prediction.

3.1 Query Data from Database

In order to do the analysis, we need to use language("lang") frequencies of each class in "appid_list"("mal_counter", "ben_counter", "und_counter", "unk_counter"). In the tweets collection, group tweets by language and calculate summation of occurrence of corresponding class's counter. Some sample input and output tables are shown in Figure 3.1.

The script used for this analysis is "mongoDBLanguageAnalysis.py". The packages we need to use are "dash" and "pymongo". The package "dash" is used for data visualization.



...	lang	...	mal_counter	ben_counter	und_counter	unk_counter	...
	en		1	2	0	1	
	ja		2	0	0	0	
	en		3	1	0	3	

lang	mal_sum	ben_sum	und_sum	unk_sum
en	4	3	0	4
ja	2	0	0	0

Figure 3.1: *This figure shows how useful data is queried from database in this script.*

3.2 Language Analysis Results

The results in this section are based on the script described in Section 3.1.

Figure 3.2 shows that the English based tweets contain more malicious links, as compared to any other languages.

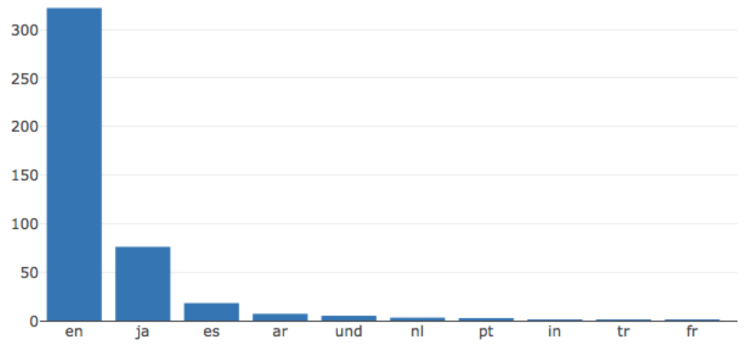


Figure 3.2: *Count of malicious tweets for each language in descending order of the counts. x-axis shows the language code, y-axis shows the number of tweets in the corresponding language.*

Figures 3.4, 3.5, and 3.6 show the counts of tweets in benign, undefined and unknown apps, respectively, sorted in decreasing order by language.

However, if we consider the total number of malicious English tweets, the result may not be surprising as the number of tweets in English is larger than the number of tweets in any other language. To gain a better understanding of language correlation to malware, we show the malware rate for each language (see Figure 3.3). The IDs not in latest.csv are not considered when calculating the malware rate.

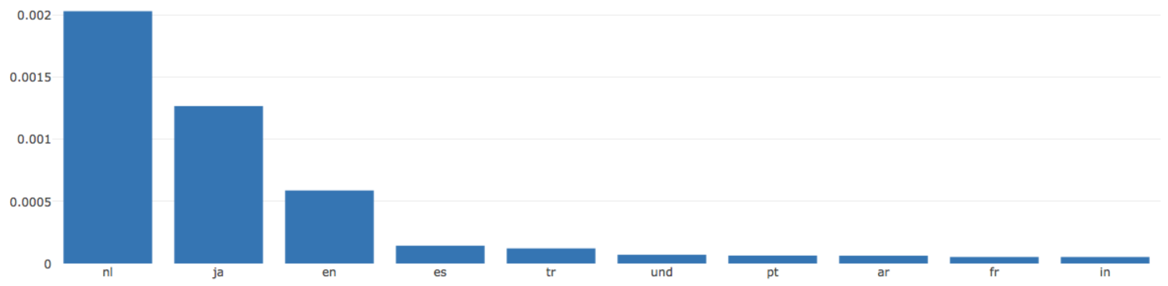


Figure 3.3: *Rate of malicious tweets relative to the total of each language in descending order.*

Figure 3.3 shows that ni (Spanish-Nicaragua) has the highest malware rate (count of malicious ni tweets / count of total ni tweets), followed by ja (Japanese), then en (English). While the rate of malicious apps per language by itself cannot be used to classify tweets, it may still help in classification. As can be seen, the malware rate of a language is very small, given that a lot more tweets are about benign apps. The malware rate only tells us which language occurs more frequently in malicious tweets. We also study if the percentage of each language from a particular class may be useful with respect to malware analysis. Figures 3.8, 3.9, 3.10 and 3.11) show the percentage of each language in each class, but we cannot gain many insights from these figures. The percentages for English (en) tweets in the four classes are higher than those of all other languages. In malware class, about 70 percent of malicious tweets' language is English.

The conclusion of this analysis, is that language analysis cannot help much in detecting malware, but it informs us about the language with more malicious tweets and higher malware rate.

Furthermore, Figure 3.7 shows the counts of tweets for each language in descending order, for tweets that refer to apps that are not in the file retrieved from AndroZoo. These apps may be missing from the AndroZoo dataset for many reasons, e.g., Google Play removed those apps based on users' reports before AndroZoo got access to them.

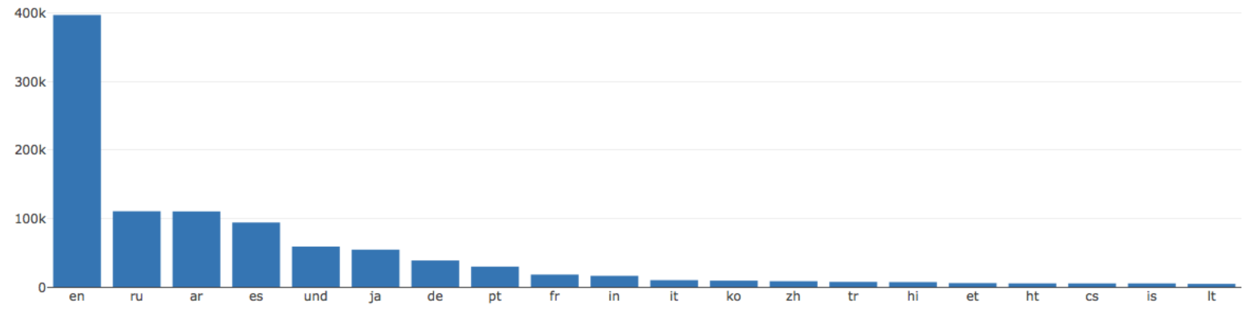


Figure 3.4: *Count of benign tweets for each language in descending order of the counts. x-axis shows the language code, y-axis shows the number of tweets in the corresponding language.*

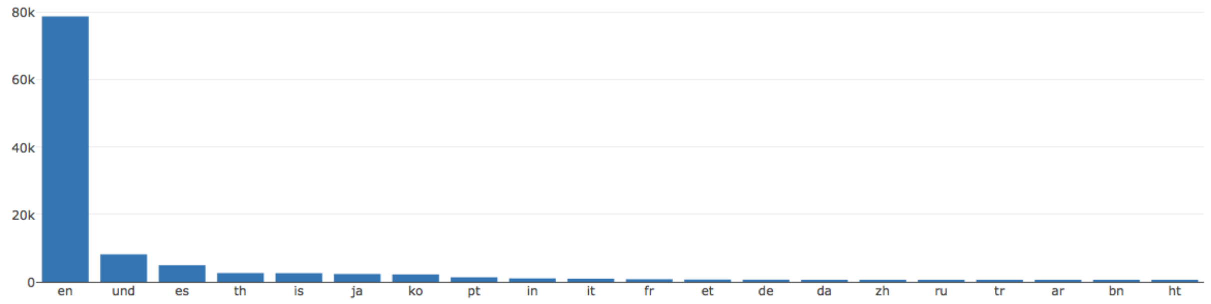


Figure 3.5: *Count of undefined tweets for each language in descending order of the counts. x-axis shows the language code, y-axis shows the number of tweets in the corresponding language.*

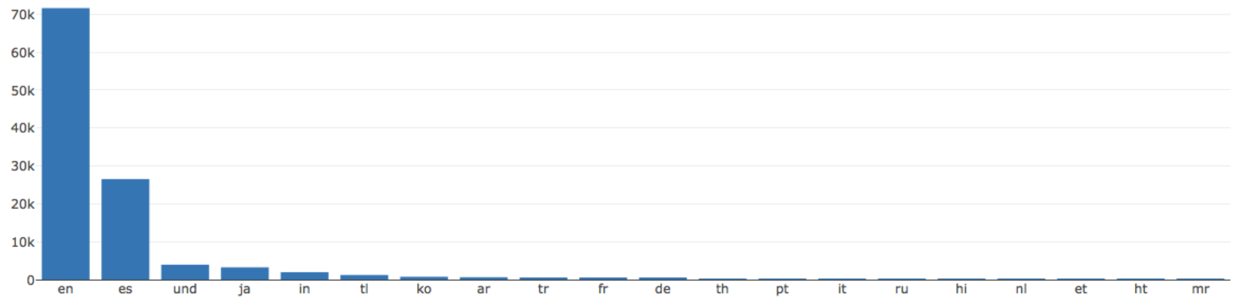


Figure 3.6: *Count of unknown tweets for each language in descending order of the counts. x-axis shows the language code, y-axis shows the number of tweets in the corresponding language.*

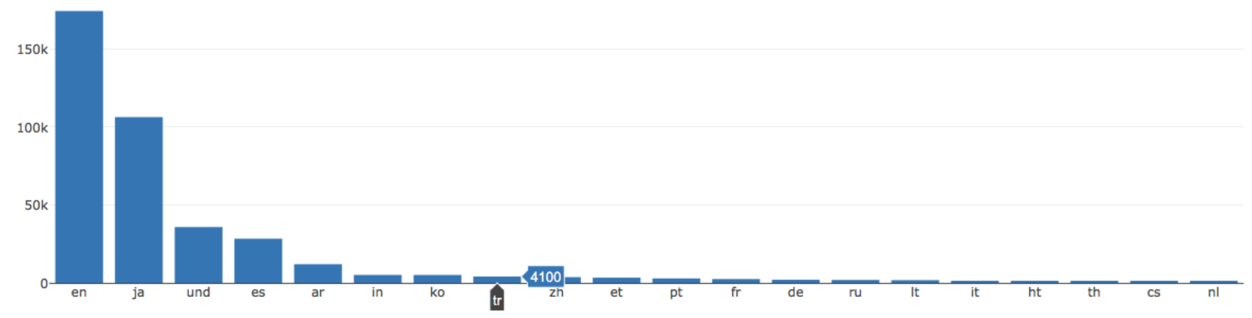


Figure 3.7: *Counts of tweets for each language in descending order. These tweets' links are not in AndroZoo's dataset. IDs may be missing from the dataset for many reasons, e.g., Google Play removed those apps based on users' reports before AndroZoo got access to them.*

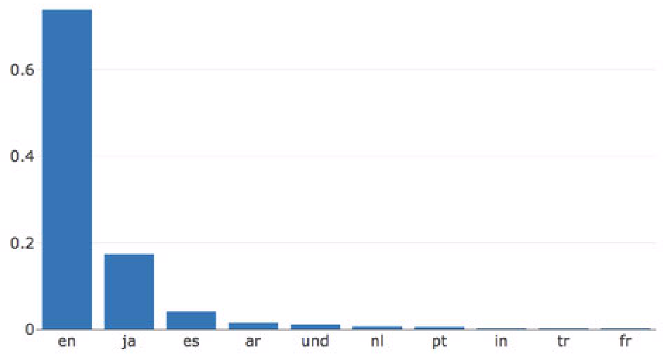


Figure 3.8: *This figure shows each language's percentage in malicious tweets. x-axis shows the language. y-axis is percentage of each language. e.g., rate of English(en) = number of English malicious tweets / number of total malicious tweets.*

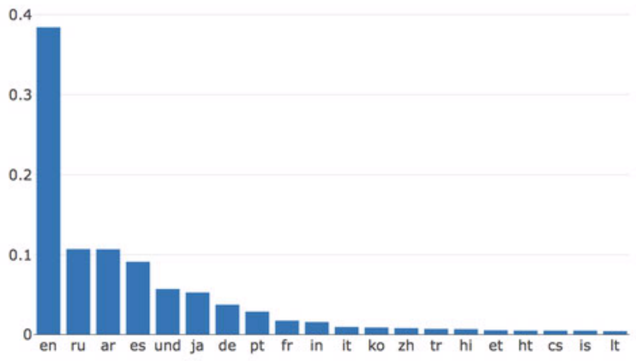


Figure 3.9: *This figure shows each language's percentage in benign tweets. x-axis shows the language. y-axis is percentage of each language.*

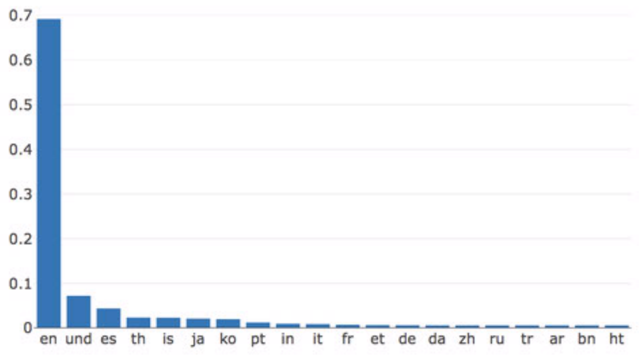


Figure 3.10: *This figure shows each language's percentage in undefined tweets. x-axis shows the language. y-axis is percentage of each language.*

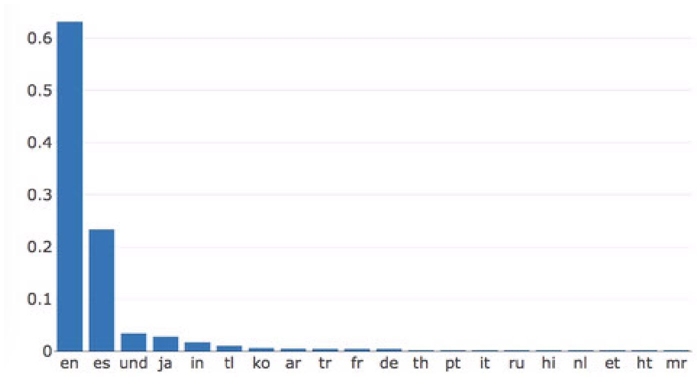


Figure 3.11: *This figure shows each language's percentage in unknown tweets. x-axis shows the language. y-axis is percentage of each language.*

Chapter 4

Word Clouds

This chapter shows tweets' word clouds for four classes which are malware, benign, undefined and unknown. A word cloud figure tells which words occur more frequently in specific classes of apps based on the size of the words.

4.1 Retrieve Tweet Text from Database and Preprocess the Text

Firstly, we need to pull the field "text" from the tweet object and save it to a file. The script I use is "tweetTextGetter.py". Third party packages we need to use are "nltk" and "pymongo". Type the following code to install "nltk". Installation of "pymongo" was introduced in Section 2.2.

```
$ pip install nltk
```

Additionally, stopwords need to be downloaded, but one can also use his or her own stopwords list.

```
$ python3
>>>import nltk
>>>nltk.download()
```

```
#click corpora select stopwords and download
```

What this script does is to pull the text from tweets whose 'mal_counter' is not empty, then clean and save all text to a file called 'text_mal_clean', and do the same thing to generate 'text_ben_clean', 'text_und_clean' and 'text_unk_clean'.

The cleaning procedure is described in what follows:

Step1: Replace all URLs by single space. Showing URLs in word clouds may also be good, but for easier coding, I remove the URLs first.

```
s = re.sub(r"http\S+", r" ", s)
```

Step2: Replace non ASCII strings with single space.

```
s = re.sub(r'[^\x00-\x7f]',r' ',s)
```

Step3: Convert strings to lowercase.

```
s = s.lower()
```

Step4: Remove punctuation.

```
translator = s.maketrans('','',string.punctuation)
s = s.translate(translator)
```

Step5: Remove extra spaces and line breaks.

```
s = s.replace('\n',' ')
s = re.sub(' +',' ',s).strip()
```

Step6: Tokenize the strings by single space and use Porter Stemmer to remove stopwords. Remove individual numbers but keep the numbers that appear in words, as it would be interesting to see some user name such as "Bob9090." 'is_number' and 'is_stopword' are functions to check if the string can be converted to a number or if it is a stopword.

```
stemmer = PorterStemmer()
s_list = [stemmer.stem(word) for word in s.split(' ')]
for ele in s_list:
    if is_number(ele) or is_stopword(ele):
        pass
    else:
        s = s + ele + ' '
```

Step7: Remove last space of the string.

```
s = s.rstrip()
```

I keep the message after "at sign" and "number sign" to see what results I get. If data after "at sign" and "number sign" has been removed, then some user name and tag related data is missing.

4.2 Generate Word Clouds

This section describes how the word cloud graph was generated. An 'R' script uses the files cleaned as described in the previous section, and outputs the word cloud graph. Simply run the script in RStudio and it would show word cloud in Figure 4.1.

Cosine similarity is also calculated between the text of all malicious tweets and the text of a test tweet, and between the text of all benign tweets and the text of a test tweet, and the results are shown in Figures 4.2 and 4.3. More specifically, in the cosine similarity analysis, I have two groups of data, one is for testing and the other one is a control group. Both of the groups have some malicious tweets. For control group, I collected text of all these malicious tweets of it, clean and save to a text file called mal_text_clean_control. For testing group, text of each tweet in it is cleaned and saved to single line of a text file called mal_text_clean_test. For each line in mal_text_clean_test, it's highly possible that there are many empty lines since some tweets only contain url which would be cleaned and that's why count of 0 is high in Fig 4.2. For testing group, I also have a file called ben_text_clean_test, which is testing queries of benign tweets. Figure 4.2 shows the cosine similarities between mal_text_clean_control and mal_text_clean_test. Figure 4.2 shows the cosine similarities between mal_text_clean_control and ben_text_clean_test. The pattern of benign histogram is clear and most of testing queries' similarities are less than 0.5 and that's what I predicted. However, malicious text doesn't have a clear pattern, one possible reason for this is that there weren't enough malicious tweets for testing.

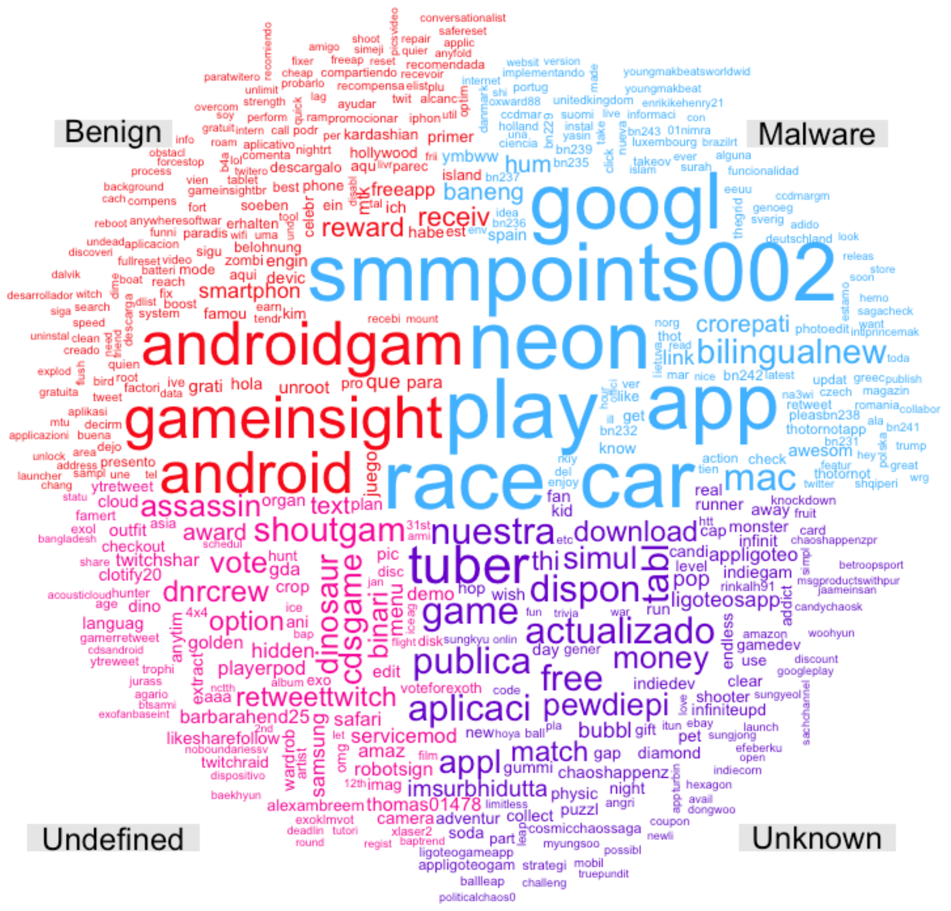


Figure 4.1: *This figure shows result of word clouds. Four categories are shown in four sectors. The larger in size, the greater frequency of words occur in documents.*

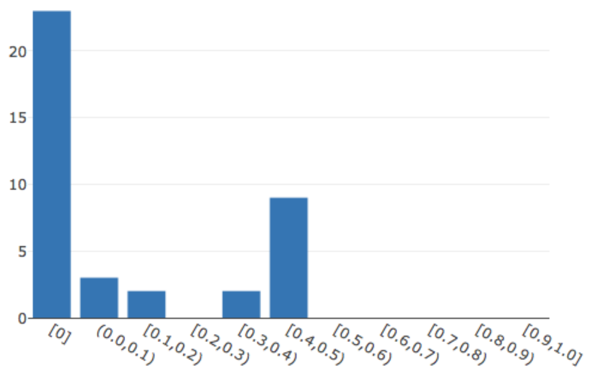


Figure 4.2: *This figure shows result of cosine similarities, y-axis is ranges of cosine similarity. x-axis is counts of tweets in that range.*

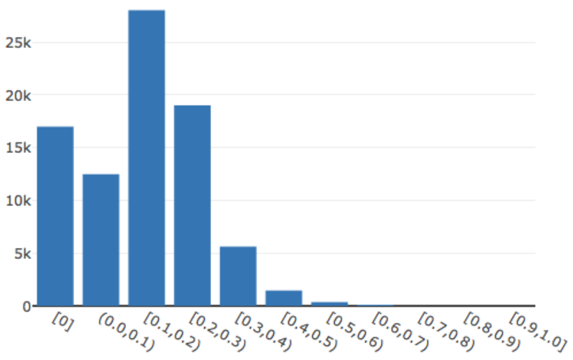


Figure 4.3: *This figure shows result of cosine similarities.*

Chapter 5

User Features Analysis

User's features are valuable fields in the tweet object, since these fields always have defined values. This ensures we have enough data to analyze. Since we already know if a tweet is malicious or benign based on the links inside the tweet, we can assume that users who posted a malicious tweets are malicious users, with respect to that tweet, while users who post benign tweets are benign users, with respect to that tweet. This categorization allows us to study if there are any differences between features of malicious users and features of benign users. The features I use are statusesCount, friendsCount, favouritesCount and followersCount under user's fields. These fields, statusesCount, friendsCount, favouritesCount and followersCount, count the number of statuses, friends, number of favourites and number of followers of a user account.

5.1 Establishing a Benign Boundary based on User Features

First of all, we plot the distribution of user's statusesCount for malicious and benign tweets in Figures 5.1 and 5.2, respectively.

The results of these two figures are not very obvious by themselves. However, what if I extend the diagram and see where count of malicious or benign tweets tend to be zero. If

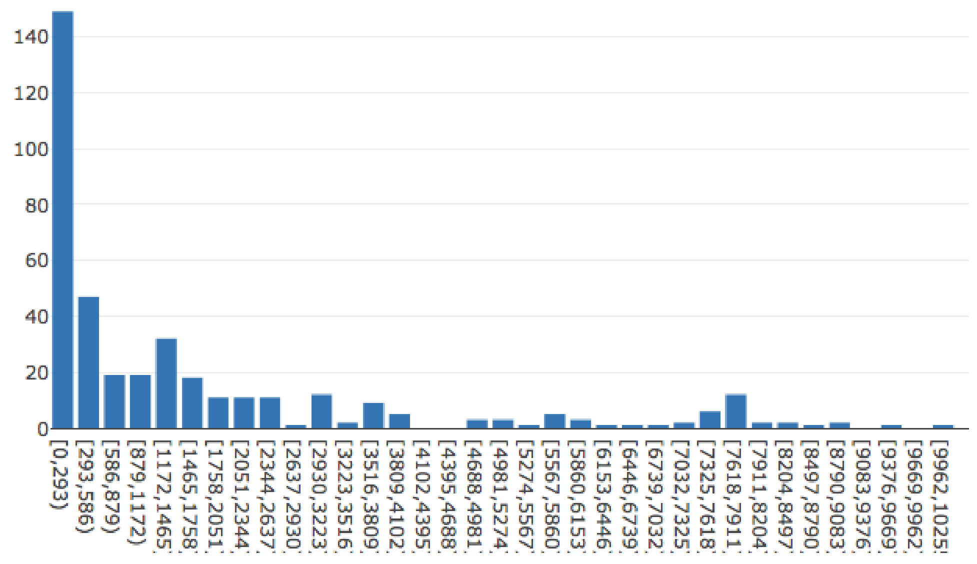


Figure 5.1: *User's statusesCount distribution for malicious tweets. x-axis is the range of statusesCount and y-axis is total tweet count whose statusesCount falls in this range and contain malicious links.*

the count of malicious tweets tends to be zero around 10000 and the count of benign tweets tends to be zero around 20000, then tweets in range 10000 to 20000 can be defined as benign tweets. Table 5.1 shows maximum value of each user's feature for benign and malware apps. As can be seen all maximum values of features from benign tweets are much greater than those from malicious tweets, and this information can be used to increase the confidence in malware predictions.

Table 5.1: *Maximum Value of Each User Feature*

User Feature	Max of Malicious	Max of Benign
statusesCount	274140	5866889
friendsCount	21386	1132141
favouritesCount	62856	851896
followingCount	122514	10838666

Figure 5.3 merges the information from Figures 5.1 and 5.2 (the y-axis shows the number of statusCounts for the corresponding statusCount value on the x-axis). Let's assume that we get maximum statusesCount of malicious tweets at point A in figure and maximum statusesCount of benign tweets at point B. Then, we can assume that the tweets whose

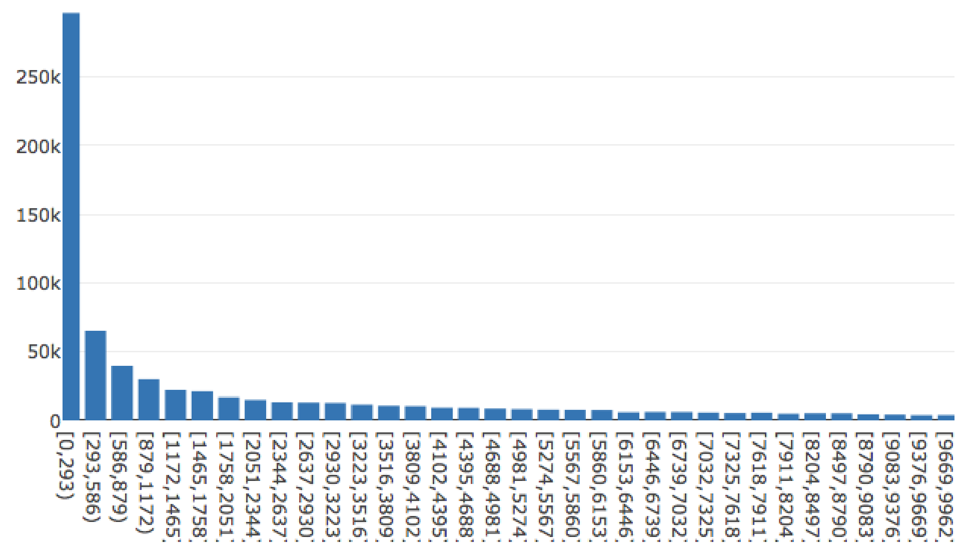


Figure 5.2: *User's statusesCount distribution for benign tweets. x-axis is the range of statusesCount and y-axis is total tweet count whose statusesCount falls in this range and contain benign links.*

statusCount is between A and B are benign tweets. Based on Figure 5.3, let's say there are n apps in the range (A,B), and the total number of apps in the dataset is N, then based on this method, (100*n/N) percent of total apps can be defined as benign. Furthermore, let's assume that I get application IDs of these APPs and save these IDs to a list, then I do the same analysis for other user features (friendsCount, favouritesCount and followersCount), and get 3 more lists, merge all these four lists, and remove duplicates. Finally, I get an application ID list without duplication. Then, I divide the length of the previous list by the total number of apps in the dataset. The result is the percentage of benign apps that I can filter out by this method. Let's see if the method work.

After running the code, I got 1298 benign app IDs and 14462 total app IDs. Thus, the rate of benign apps identified by this method is 8.975 percent, which is not very large, but it can still provide useful information that can be used to enhance the results of another malware detection approach. We will try something different in the next section.

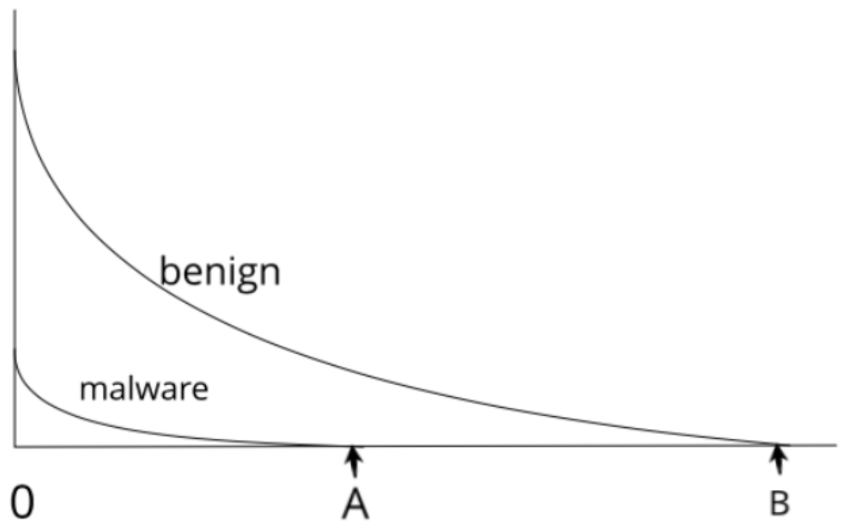


Figure 5.3: *This figure merges Figures 5.1 and 5.2. From This figure, it's obvious that if we get maximum statusesCount of malicious tweets which is point A in figure and maximum statusesCount of benign tweets which is point B, then the tweets whose statusesCount are between A ans B values can be defined as benign. Further more, the apps linked to these tweets can also be defined as benign.*

5.2 Analysis of User Features Using a Larger Data

The first method used in this section was also used in (DeLoach and Caragea, 2017), and is described in what follows. For each tweet, we get its user features and calculate the percentage of tweets whose feature values are lower. Then, calculate the average value from the above percentages, as shown in Figure 5.4. The average values for four user features are shown in Table 5.2. In this table, percentage differences between malware and benign are almost 10 percent in Statuses, Friends and Favourites fields. So when testing a tweet, the values of statusesCount, friendsCount, favouritesCount and followersCount fields in this table can be used to infer how likely it is that the tweet is a malicious tweet versus a benign tweet. Correspondingly, we can predict if the link contained inside of the tweet is malware or benign. Same way for benign, if testing data's features match the percentages listed in this table, then we can predict this tweet is benign.

Another method I used to perform analysis based on user features is as follows: for each feature, count the number of tweets in each class (malware, benign, undefined and unknown),

Table 5.2: *User Feature Analysis Results (Method 1)*

Class	Statuses	Friends	Favorites	Followers
Malicious	0.393397	0.600186	0.740938	0.538870
Benign	0.500075	0.499252	0.625510	0.505283

and save the counts in a dictionary. When testing tweets with Google Play links, search the user features' counts of these testing tweets in the dictionary and get result percentages of each classes. Following is what that dictionary looks like.

```
main_dict = {
    "statusesCount":{
        count1:{"mal":malcount1,"ben":bencount1,"und":undcount1,"unk":unkcount1},
        count2:{"mal":malcount2,"ben":bencount2,"und":undcount2,"unk":unkcount2},
        count3:{"mal":malcount3,"ben":bencount3,"und":undcount3,"unk":unkcount3},
        ...
        }
    "friendsCount":{...}
    "favouritesCount":{...}
    "followersCount":{...}
}
```

Now, I have the data for testing: "statusesCount": 233, "friendsCount": 0, "favouritesCount": 7, "followersCount": 24. Then, the program would find out what is in main["statusesCount"][233], main["friendsCount"][0], main["favouritesCount"][7] and main["followersCount"][24] - this information is shown in Table 5.3. So, the malware rate in statusesCount field is calculated like this: 4/(4+7+0+1), and benign rate is 7/(4+7+0+1), etc. Finally, the program would get the result in Table 5.4. Based on result, this testing tweet's link is more likely to be undefined in statusesCount and favouritesCount fields, malicious in friendsCount and benign in followersCount. Before using this method, it's better to prepare equal amount of tweets in four classes. Otherwise, for example, if number of benign tweets is much greater than malicious tweets, the rate of malware prediction would be lower than expected.

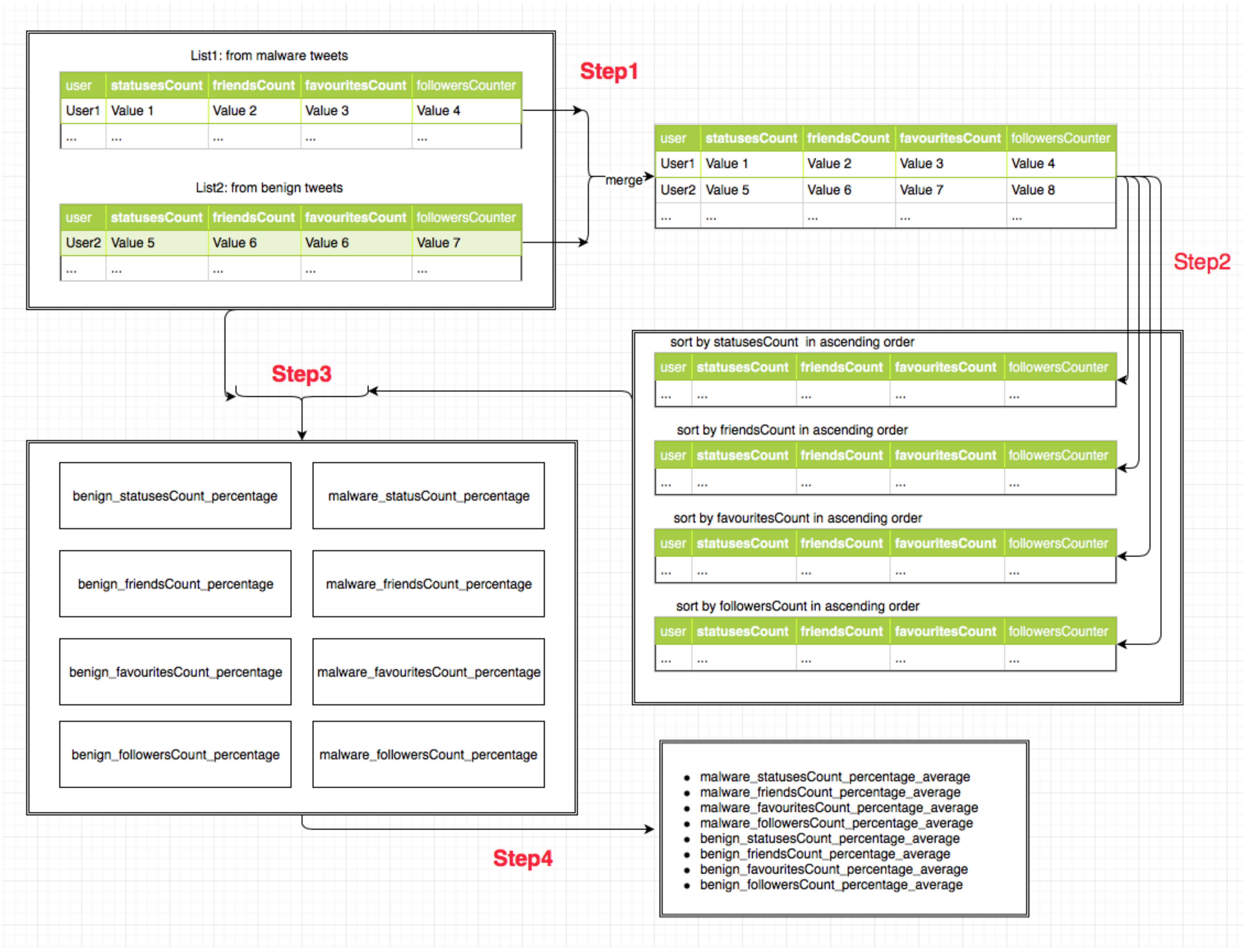
Table 5.3: *User Feature Analysis Count Results (Method 2)*

feature	malware	benign	undefined	unknown
Statuses	6	2	87	3
Friends	24	2	5	0
Favourites	55	23	86	67
Followers	21	38	26	9

Table 5.4: *User Feature Analysis Rate Results (Method 2)*

feature	malware	benign	undefined	unknown
Statuses	0.061	0.020	0.888	0.031
Friends	0.774	0.065	0.161	0
Favourites	0.238	0.010	0.372	0.290
Followers	0.223	0.404	0.277	0.096

One limitation of these two methods is that they analyze each field independently. For example, testing on one tweet may suggest a 0.70 probability that the tweet is malicious based on the statusesCount field, and 0.90 probability that the tweet is malicious based on friendsCount. It would be interesting to merge the two pieces of information into a final conclusion.



List1: from malware tweets

user	statusesCount	friendsCount	favouritesCount	followersCounter
User1	Value 1	Value 2	Value 3	Value 4
...	...	...	...	...

List2: from benign tweets

user	statusesCount	friendsCount	favouritesCount	followersCounter
User2	Value 5	Value 6	Value 6	Value 7
...	...	...	...	...

user	statusesCount	friendsCount	favouritesCount	followersCounter
User1	Value 1	Value 2	Value 3	Value 4
User2	Value 5	Value 6	Value 7	Value 8
...	...	...	...	...

sort by statusesCount in ascending order

user	statusesCount	friendsCount	favouritesCount	followersCounter
...	...	...	...	...

sort by friendsCount in ascending order

user	statusesCount	friendsCount	favouritesCount	followersCounter
...	...	...	...	...

sort by favouritesCount in ascending order

user	statusesCount	friendsCount	favouritesCount	followersCounter
...	...	...	...	...

sort by followersCount in ascending order

user	statusesCount	friendsCount	favouritesCount	followersCounter
...	...	...	...	...

benign_statusesCount_percentage	malware_statusCount_percentage
benign_friendsCount_percentage	malware_friendsCount_percentage
benign_favouritesCount_percentage	malware_favouritesCount_percentage
benign_followersCount_percentage	malware_followersCount_percentage

- malware_statusesCount_percentage_average
- malware_friendsCount_percentage_average
- malware_favouritesCount_percentage_average
- malware_followersCount_percentage_average
- benign_statusesCount_percentage_average
- benign_friendsCount_percentage_average
- benign_favouritesCount_percentage_average
- benign_followersCount_percentage_average

Figure 5.4: *This figure shows how I get results of Table 5.2. List1 and List2 are data read from mongoDB. List1 corresponds to user features from malicious tweets and List2 corresponds to user features from benign tweets. Step1 simply merges these two lists and it's used for getting sorted lists. In this experiment, four features are used which means the merged list needs to be sorted by different features. Four sorted lists are produced and these lists are sorted in ascending order. The reason I sorted these list increasingly is that these indexes can be used directly to find the number of tweets with lower feature counts (ex: statusesCount). Then, process step3. What step3 does is iterating through List1 and List2, calculating number of tweets with lower feature counts, then append to a new list. Since we are processing tweets from two 2 lists (benign and malware) for 4 groups of user features, eight lists are produced. Finally, calculate the average of the numbers of these eight lists. Those are results we get in Table 5.2.*

Chapter 6

Conclusions and Future Work

Each feature is analyzed separately. For language analysis, the malware rate of each language may not be able to suggest malware individually. One thing we can tell from this analysis is that if the tweet's language's malware rate is higher than language's benign rate, it is more likely that the tweet is a malicious tweet. For word clouds analysis, we have observed that each category (malware, benign and so on) has specific pattern of words. Furthermore, the cosine similarity can roughly tell if a tweet contains malicious links or benign links, if there is enough quantity of malicious tweets' text in the document. The more text is collected, the more accurate the similarity results get.

As mentioned before, a larger dataset gives better results. More data still needs to be crawled. Once we have enough data, some rare fields like location and coordinates can be used to understand which areas have more malicious links and potentially draw dot-density maps. Also, users who posted tweets with malicious links can be added to a black list. It would also be good to get the possibility of malware tweets by analyzing language, location, and if the user posted malicious tweets, cosine similarity of text and user features together. Additionally for some analysis in this report, using machine learning would be good choice, and that's what I plan to do as part of future work.

Bibliography

Allix, K., Bissyandé, T. F., Klein, J., and Le Traon, Y. (2016). Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, pages 468–471. ACM.

DeLoach, J. and Caragea, D. (2017). Twitter-enhanced android malware detection. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 4648–4657.

Sood, G. (2017). *virustotal: R Client for the virustotal API*. R package version 0.2.1.