# Mining permission request patterns of android applications to determine application quality

**Narita Pandhe**
Department of Computer Science
University of Georgia
Athens, GA - 30605
narita@uga.edu

## Abstract

Nowadays, because of the universality of smartphones, development and publishing of mobile apps in App Stores has boosted. But, this has also led to a drop in the quality of the published apps. Any smartphone user is aware that the quality of mobile apps greatly vary. In this paper I intend to mine for such suspicious apps and use the inferred knowledge to determine whether an unseen app is prone to be buggy. More specifically, analyze the app permissions and user reviews to identify potential correlations between error-sensitive permissions and error-related reviews. Thus, this system can act as an efficient static checker to identify buggy apps before or after they are released in the App Stores.

## 1 Introduction

Nowadays, because of the universality of smartphones, usage of smartphone apps has surged. The good part, is consequently this has boosted the development and publishing of mobile apps in App Stores. But the bad part, it has also led to a drop in the quality of the published apps. Any smartphone user is aware that the quality of mobile apps greatly vary. For instance, a large number of exceptions are raised in the wild for Android apps. As Google Play Store lacks source inspection before publishing an app, low-quality and malicious apps can easily reach the store. An article published in TechCrunch[3] states that Google planned to have an internal team of reviewers/moderators to analyze apps for policy violations prior to publication. Given the sheer number of apps published every day, scrutinizing each app is really a difficult task. So, Google's Moderators are aided with an automated system which fastens this process. Before app moderators are presented with the applications, Google uses software to pre-analyze the app for things like viruses, malware as well as other content violations. These approaches work well but do not cover another challenging threat for the app store: buggy apps, which disrupt user experience and inevitably the store quality.

An app checker is a mechanism to determine whether an app is potentially error-prone when the developer uploads it or after the developer publishes it in the market. If an app store uses efficient app checkers, the overall quality of apps would raise. This paper proposes a static app checker which can help the App Store moderators to make informed decisions based on quantitative measures. This paper also investigates the correlation between permission requests and errors reported by the end-users in their reviews.

To sum up, the primary objectives of this paper are:

- Identify common patterns in permission requests so that the applications which do not fit the predominant patterns can be flagged for additional user scrutiny. Towards this goal, I have applied, supervised machine learning techniques.

- To make informed decisions I also intend to investigate the correlation between permission requests and errors reported by end-users/user reviews which can answer questions like: To what extent apps which have error-related reviews request error-sensitive permissions?

### 1.1 Related Work

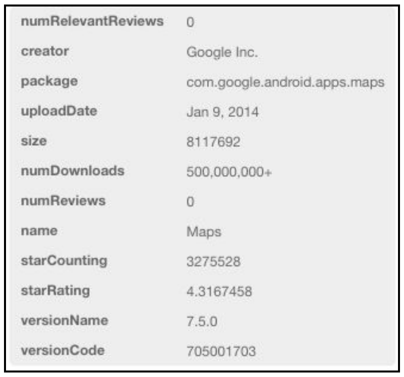
Android Platform supports extensive third-party applications. Google Play Store applications use permission systems to limit applications' access to users' private information and device resources. It also provides users with average user ratings, user reviews, descriptions, screenshots, and permissions to help them select applications. Android applications can access phone hardware and private user information via Android's API. Permissions restrict applications' ability to use the API. For example, an application can only take a photograph if it has the CAMERA permission. Developers select the permissions that their applications need to function, and these permission requests are shown to users during the installation process. The user must approve all of an application's permissions in order to install the application. Several studies have examined Android applications' use of permissions. Barrera et al. surveyed 1100 most popular applications and stated that they primarily request a small number of permissions, leaving most other permissions unused[2,4]. Felt et al. and Chia et al. surveyed Android applications and identified the most-requested permissions. Chia et al. also found several correlations between the number of permissions and other factors: a weak positive correlation with the number of installs, a weak positive correlation with the average rating, a positive correlation with the availability of a developer website, and a negative correlation with the number of applications published by the same developer[2,5,6]

Mario et al.[2] extends these past analysis. There has been research using other machine learning techniques to identify malwares. Sanz et al. applied several types of classifiers to the permissions, ratings, and static strings of 820 applications to see if they could predict application categories, using the category scenario as a stand-in for malware detection[7]. Shabtai et al. similarly built a classifier for Android games and tools, as a proxy for malware detection [8]. Zhou et al. found real malware in the wild with DroidRanger, a malware detection system that uses permissions as one input[9].

Although the techniques are similar, I am focussing understanding the correlations in permission patterns and user reviews rather than to identify malware. Maria et al.[1] devised similar system, which uses machine learning techniques like Decision Trees and Latent Dirichlet Allocation. Their system exhibits an accuracy between 61.42% to 61.96%. The approach I have followed differs in the machine learning techniques and observes a measurable improvement in accuracy.

## 2 Data

Maria et al. have open sourced their data of Playstore Apps which consists of the following metadata: name, package, creator, version code, version name, number of downloads, size, upload date, star rating, star counting, and the set of permission requests[1]. This rich data is stored in JSON format. 2 versions are available of this dataset: one from January, 2014 and the other from March, 2014. I have utilized January 2014 dataset which consists of 38, 781 apps requesting 7826 different permissions.



| | |
|---|---|
| numRelevantReviews | 0 |
| creator | Google Inc. |
| package | com.google.android.apps.maps |
| uploadDate | Jan 9, 2014 |
| size | 8117692 |
| numDownloads | 500,000,000+ |
| numReviews | 0 |
| name | Maps |
| starCounting | 3275528 |
| starRating | 4.3167458 |
| versionName | 7.5.0 |
| versionCode | 705001703 |

Figure 1: Sample App Data

This dataset is stored in the form of graph database, Neo4j[10]. This dataset therefore consists of a graph describing the apps and permissions as nodes and links between them as edges. Graph databases provide a powerful and scalable data modelling and querying technique capable of representing any kind of data in a highly accessible way [11]. The authors chose a graph database because the graph visualization helps to identify connections among data (e.g., clusters of apps sharing similar sets of permission requests)[1].

As this data set consisted only of permissions, I obtained 10,000 reviews by scraping PlayStore Apps from categories like: Health and Fitness, Media and Video, Role Playing Games and apps related to Communication. I utilized Scrapy, an application framework for crawling web sites and extracting structured data[12].

So far so good! Working good and showing correct info. The UI and especially the round widget looks really beautiful.

Thanks for listening to my request to add default clock in "tap on clock" option from settings* 5 stars here.

Thank you for considering the request, hope this beautiful app will never be bloated in future. Thank you :)

The temperature is off most of the time. It's a pity. Take a close look at your Wx provider. I do like the program, but it should deliver the Wx as accurate as possible.

Kind regards .... Well, the program still reported "Snow Blizzard" with a Outside Temp. 06 C. I live on the south side of the Jura mountain at 840 meters in Switzerland, maybe this helps a bit. Graphics 5 Stars

After downloading this app I found that you have deducted an extra amount of Rs. 50/- from my account. First an amount of Rs.10/- deducted and after few minutes again an amount of Rs.50/- was deducted. Why this cheating from the customer?

These app suks The worst app I've ever purchased why does the widget not working when i put a widget on the home screen there’s no weather coming out pls. Fix these ok

No. Hate the red clouds background when skies are grey during day time. Please add a grey clouds background and remove that hideous red cloud apocalypse garbage. It makes cloudy days worse.

Good looking and accurate The widget does look very sophisticate and it is very precise with information that displays. Love it accurate and beautiful wonderful app. worth the money

Figure 2: Sample Reviews

## 2.2 Data cleaning

I was interested in only what permissions of the app were and not how many downloads it had or what was its size. So, from the graph database, I extracted just the package name and its permissions. This data was clean enough, but the only problem was, every sample had varying number of permissions. Some apps had just 5 permissions whereas some had more than 50 permission. Due to the limitation of computational resources, I had to limit myself to 10,000 apps and their permissions.

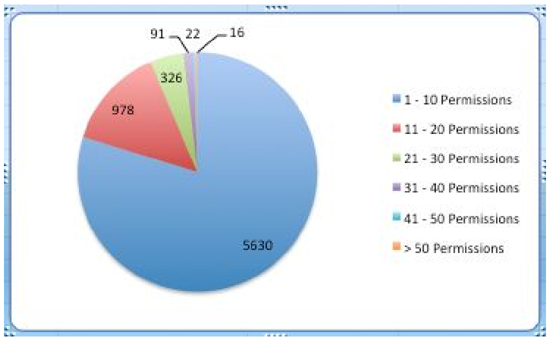


Figure 3. Distribution of apps per number of permission requests

Obviously, the extracted reviews were not clean textual data. The review text consisted of hashtags, hyperlinks, emoticons, numbers, words and acronyms used in internet slang. As a result, I couldn't use these reviews as it is. The initial pre-processing step comprised of removing the stop-words, hyperlinks and expressions (# ; ? \\ / , " @ ` !, :)).

## 3 Measuring model effectiveness

I considered the following measurement statistics.

### 3.1 Accuracy

The most obvious and straightforward measure, measured as the ratio of the correct predictions to that of the total number of test data points.

### 3.2 F1 - Score

This is another effective statistical measurement which is simply the mean of precision and recall.

$$\boxed{F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.}$$

Where precision is:

$$\boxed{\text{Precision} = \frac{tp}{tp + fp}}$$

And recall is

$$\boxed{\text{Recall} = \frac{tp}{tp + fn}}$$

Where tp stands for ‘true positive’ and fp stands for ‘false positive’ and fn stands for ‘false negative’.

## 4 Methods

The pipeline consisted of 2 phases, I first run permission pattern mining on the permissions data Next, I executed Sentiment Analysis algorithm by which I dig into and find the sentiment of reviews.

### 4.1 Permission Pattern Mining

To compare the results, I needed ground truth. Maria et al.[1] have already published a list of apps and their permissions which are error prone. I used the same as my ground truth, by labelling the same apps in the dataset of 10,000 apps as negative and rest all apps as positive.

The next stage in the pipeline was implementation of Machine Learning Classification algorithms like Naive Bayes Classifier, Support Vector Machines and Logistic Regression. I compared the results of these 3 methods to the ground truth.

#### 4.1.1 Naive Bayes

I wanted the posterior probability of the label of an app based on its permissions. Using Bayesian Statistics, I calculate the posterior probability which is a product of prior probability and likelihood. Naive Bayes makes a strong assumption of independence that given the label, features (i.e. permissions) are independent random variables. Thus, calculate the probability P(permissions|label) as the product of conditional probabilities of features given the corresponding label (i.e. P(pi |label = lj )). Thus, I implemented the following Naive Bayes theorem for binomial classification of the apps:

$$P(\text{label} = lj\ |\text{permissions}) \propto P(\text{label} = lj\ ) \times Yi\ P(pi\ |\text{label} = lj\ )$$

where P(label = lj |permissions) is the posterior probability, permissions = {p1, ..., pn} and lj ∈ L = {0, −1, +1}. Each permission is considered as a bag of words. The two main labels for classification are *positive* and *negative,* where the label positive indicates that the app is not error prone/buggy. The label with the maximum likelihood is given as the label for the app.

$$\text{labelNB} = \text{arg max } lj \in L\ P(\text{label} = lj\ |wi)$$

I used the Laplacian Smoothing technique to handle unseen words in the test permissions. I achieved an accuracy of 78% using the Naive Bayes technique. I used a training set of 7000 apps to train the model and tested the model on 3000 new/unseen apps.

### 4.1.2 SVM

Next, I used linear SVM for our binomial app classification. The parameters I used were number of iterations = 100, stepsize = 0.01 and regularization parameter = 0.1. I trained the SVM model on 7000 labeled apps and their permissions and tested it on new 3000 new apps.

In SVM, each labeled training point is a p-dimensional real vector. The intention is to find the "maximum-margin hyperplane" that divides the group of labeled training points $x_i$ for which label $y_i$ = "*positive*" from the group of points for which $y_i$ = "*negative*", which is defined so that the distance between the hyperplane and the nearest point $x_i$ from either group is maximized[13]. SVM gave an accuracy of around 74%.

### 4.3.4 Logistic Regression

Logistic Regression has proved to be a very effective method for binomial classification. Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution[14]. Here, the label of the app becomes the dependent variable. Again, I trained the model on 7000 labeled Apps and tested on 3000 new Apps. Logistic Regression achieved an accuracy of 79%.

## 4.2 Sentiment Analysis using TextBlob

For identifying the sentiment of reviews i.e. whether they speak good about an app or complain about it, I utilized the text processing tool called TextBlob. TextBlob is a Python library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more[15]. The sentiment analysis provided by TextBlob gives a *polarity* of every review in the range of -1 to 1; -1 being most negative and 1 being most positive. So implementing a threshold based policy, I classified every review as having either a *Positive* or *Negative* sentiment.

## 5 Results

Below are the results that were obtained based on the models used.

### 5.1. Permission Pattern Mining Results

Table 1: Results

| **Algorithm** | **Train Error** | **Test Error** | **F1 Score** |
|---|---|---|---|
| Naive Bayes | 0.2398 | 0.2286 | 0.8638 |
| SVM | 0.2737 | 0.2643 | 0.8456 |
| Logistic Regression | 0.2202 | 0.2156 | 0.8678 |

Train Error is the error that I got after testing the training data on the models. Test Error is the error that I got by testing the testing data on the models.

F1 score is the harmonic mean of Precision and Recall. It is an efficient method for evaluating classification model.

As per Table 1, the results given by Logistic Regression performs the best with an accuracy of 79%.

**5.2 Influence of Permissions on Error-related Reviews**
Out of all the negative apps, Table 2 presents some of the examples consisting both of the permissions and reviews.

Table 2 Sample negative apps and their reviews.

| Package Name | Permissions | Reviews |
|---|---|---|
| mohammad.adib.sidebar | RECEIVE_BOOT_COMPLETED<br>SYSTEM_ALERT_WINDOW<br>VIBRATE<br>ACCESS_WIFI_STATE<br>CHANGE_WIFI_STATE<br>GET_TASKS<br>KILL_BACKGROUND_PROCESSES<br>WRITE_SETTINGS<br>REORDER_TASKS<br>CHANGE_NETWORK_STATE<br>ACCESS_NETWORK_STATE<br>CAMERA | - Doesn't support front cam files Don't know why because everyone uses front face cameras now a days.<br><br>- Please fix. We don't have expensive equipment to see exactly what we record when we want to be in the scenes.<br><br>- Is not good to update After i update i cannot edit any video .it saying file not supported.<br><br>- Won't work I won't let me do anything it just goes back to home screen |
| com.hb.settings | WRITE_SECURE_SETTINGS<br>ACCESS_WIFI_STATE<br>CHANGE_WIFI_STATE<br>BLUETOOTH<br>BLUETOOTH_ADMIN<br>READ_SYNC_SETTINGS<br>WRITE_SYNC_SETTINGS<br>WRITE_SETTINGS<br>ACCESS_FINE_LOCATION<br>EXPAND_STATUS_BAR<br>ACCESS_NETWORK_STATE | - Force shuts down!!! Super useful app but recently its been shutting down and closing the app that I'm using!!<br><br>- Its a great app but... You have done a great job but whenever i cleared my RAM |

| | | |
|---|---|---|
| | CHANGE_NETWORK_STATE<br>CAMERA<br>CALL_PHONE<br>NFC<br>VIBRATE<br>RECEIVE_BOOT_COMPLETED<br>BILLING<br>READ_LOGS | memory (which i always do) the app gets forced stop and i have to start it again.. plz provide an option in accessibility so that the app can be excluded during clearing the ram.. and plz hurry up |
| com.jiubang.goscreenlock.theme.magiceve.getjar | READ_PHONE_STATE<br>BILLING<br>GET_ACCOUNTS<br>GET_TASKS<br>READ_LOGS<br>VIBRATE<br>EXPAND_STATUS_BAR<br>INTERNET<br>ACCESS_NETWORK_STATE<br>WRITE_EXTERNAL_STORAGE<br>CHANGE_COMPONENT_ENABLED_STATE<br>READ_EXTERNAL_STORAGE | - I hate these ads Please remove ads while charging and etc. After I unlock my screen google play store will randomly opens then its like forcing me to install the apps. It is very annoying.Please fix this and I know your team can do it and this app is very useful to us. I hope you cooperate and fix this problem. Thank you:))<br><br>- Fullscreen Ads Didn't expect to have an Ad polluted phone when I installed this. It spams fullscreen ads when you unlock or lock your device.<br><br>- It ain't working when I try to change my theme nothing shows up |

Table 2 clearly indicates that the apps which are flagged as negative, infact do have negative reviews by the end-users and vice-versa. Though these correlations can be observed, this certainly does not indicate that, the of existence of an erroneous permission pattern is necessarily the cause of the error.

**Conclusion**

In this paper, I have proposed a system that can classify potentially buggy apps by using the correlation between permission patterns and error-proneness. For this, I worked with 10,000 Android Apps from Google Play Store. I also performed sentiment analysis on 10,000 online user reviews to identify error-related reviews that point out error-suspicious apps. I used supervised machine learning techniques to automatically identify error-suspicious apps available in the Google Play Store.

From the results, it can be seen that Naive Bayes and SVM underperform than Logistic Regression owing to the fact that this problem can be solved effectively using some method pertaining to extrapolation. Thus, methods like Logistic Regression are clearly the most favorable choice for this problem. With the knowledge inferred from this analysis, we can classify new/unseen apps which either have permissions or reviews or both as buggy or non-buggy. On a dataset of 10,000 apps, this set demonstrates to have an accuracy of 78%.

In the future, accuracy of this system can be observed on a larger data set. Currently, the dataset has a fixed set of 7826 permissions. Techniques like Feature Selection, Principal Component Analysis can be used to reduce the dimensions of permissions and take into consideration only those permissions which truly determine whether an app is buggy or not.

**Acknowledgments**

I would like to thank Dr. Frederick Maier. He has always been very supportive. The lessons taught in the class have been a great help. His continuous help and guidance lead us architecting my final project with a better perspective of knowledge bases and knowledge extraction

## References

[1] Maria Gomez, Romain Rouvoy, Martin Monperrus, Lionel Seinturier. A Recommender System of Buggy App Checkers for App Store Moderators. [Research Report] RR-8626, Inria Lille; INRIA. 2014.

[2] Mario Frank, Ben Dong, Adrienne Porter Felt, Dawn Song. Mining Permission Request Patterns from Android and Facebook Applications (extended author version) University of California, Berkeley

[3] Sarah Perez. TechCrunch Article: App Submissions On Google Play Now Reviewed By Staff, Will Include Age-Based Ratings

[4] David Barrera, H. G ¨une ¸s Kayacik, Paul C. van Oorschot, and Anil Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In Proceedings of the 17th ACM conference on Computer and Communications Security, CCS, 2010

[5] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this App Safe? A Large Scale Study on Application Permissions and Risk Signals. In Proceedings of the World Wide Web Conference, WWW, 2012.

[6] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of application permissions. In Proceedings of the USENIX Conference on Web Application Development, WebApps, 2011.

[7] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, and P.G. Bringas. On the automatic categorisation of android applications. In Proceedings of the 9th IEEE Consumer Communications and Networking Conference (CCNC), 2012.

[8] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated static code analysis for classifying android applications using machine learning. In Proceedings of the 2010 International Conference on Computational Intelligence and Security, CIS, 2010.

[9] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, You, Get off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In Proceedings of the Network and Distributed System Security Symposium, NDSS, 2012.

[10] Neo4j, http://neo4j.com/

[11] I. Robinson, J. Webber, and E. Eifrem. Graph Databases. O'Reilly, June 2013.

[12] Scrapy, http://scrapy.org/

[13] Support Vector Machines, https://en.wikipedia.org/wiki/Support_vector_machine

[14] Logistic Regression, https://en.wikipedia.org/wiki/Logistic_regression

[15] Textblob, http://textblob.readthedocs.io/en/dev/index.html