An Investigation into Android Run-time Permissions from the End Users' Perspective

Gian Luca Scoccia
Gran Sasso Science Institute
L'Aquila, Italy
gianluca.scoccia@gssi.it

Stefano Ruberto
Gran Sasso Science Institute
L'Aquila, Italy
stefano.ruberto@gssi.it

Ivano Malavolta
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
i.malavolta@vu.nl

Marco Autili
University of L'Aquila
L'Aquila, Italy
marco.autili@univaq.it

Paola Inverardi
University of L'Aquila
L'Aquila, Italy
paola.inverardi@univaq.it

ABSTRACT

To protect the privacy of end users from intended or unintended malicious behaviour, the Android operating system provides a permissions-based security model that restricts access to privacy-relevant parts of the platform. Starting with Android 6, the permission system has been revamped, moving to a run-time model. Users are now prompted for confirmation when an app attempts to access a restricted part of the platform.

We conducted a large-scale empirical study to investigate how end users perceive the new run-time permission system of Android, collecting and inspecting over 4.3 million user reviews about 5,572 apps published in the Google Play Store. Among them, we identified, classified, and analyzed 3,574 permission-related reviews, employing machine learning and Natural Language Processing techniques. Out of the permission-related reviews, we determined recurring points made by users about the new permission system and classified them into a taxonomy. Results of our analysis suggest that, even with the new system, permission-related issues are widespread, with 8% of collected apps having user reviews with negative comments about permissions. We identify a number of points for improvement in the Android run-time permission system, and provide recommendations for future research.

KEYWORDS

Android, Permissions, Apps, Opinion Mining, Review Analysis, Privacy, Security

ACM Reference Format:
Gian Luca Scoccia, Stefano Ruberto, Ivano Malavolta, Marco Autili, and Paola Inverardi. 2018. An Investigation into Android Run-time Permissions from the End Users' Perspective. In *MOBILESoft '18: MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3197231.3197236

MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5712-8/18/05…$15.00
https://doi.org/10.1145/3197231.3197236

1 INTRODUCTION

As mobile devices provide more advanced features, more sensitive data are manipulated and stored, including not only personal information but also data collected via sensors [1]. Aiming at protecting the privacy of end users from intended or unintended malicious behavior, the Android operating system provides a permissions-based security model that restricts the access to security- and privacy-relevant parts of the platform.

Past research in the field has evidenced the existence of usability issues within the permission system [2–4]. Only a minority of users are aware of the implications of their privacy decisions and warning dialogs are not easily understood.

Towards addressing these problems, the permission system has been revamped and, starting with Android 6 (i.e., Android API level 23), access to privacy- and security-relevant parts of the platform is enforced by a new run-time permission system. Under the new permission system, users are prompted for confirmation when an app attempts to access a restricted part of the platform for the first time [5].

In this paper, we investigate **how end users perceive the new run-time permission system of Android**, with the ultimate goal of identifying possible points of improvement present in the permission system, despite the recent changes. For this purpose, we conducted a large-scale empirical study on over 4.3 million user reviews about 5,572 apps published on the Google Play Store that adopt the run-time permission system (identified within an initial dataset of over 18 million user reviews belonging to 15,124 apps). By using a combination of an established keyword-based approach [6], we identified among them potential permission-related reviews regarding the new Android permission system. We manually analyzed a statistically representative sample of the reviews, and categorized the main concerns expressed by end users about the new system into a taxonomy. Then, by making use of machine learning and Natural Language Processing (NLP) techniques, we classified a complete set of 3,574 permission-related reviews according to the previously-built taxonomy. Finally, by analyzing the achieved classification, we identified a number of points for improvement in the new permission system related to, e.g., the lack of clarity of developers when requesting permissions, excessive number of requested permissions, and permission-related bugs. We

discuss each point in details and provide insights for future research and for improving the Android permission system.

The **main contributions** of this paper are the following:

- a taxonomy of the main concerns expressed by users in reviews about the Android run-time permission system;
- a semi-automatic classification pipeline to classify Android reviews according to specific concerns (i.e., run-time permissions in our case);
- evidence about the accuracy of different combinations of machine learning techniques when classifying Android users' reviews;
- empirical results about user reviews mentioning the Android run-time permission system involving 18,326,624 reviews about 15,124 apps;
- the identification of a number of potential issues of the run-time permission system; and a discussion about how those issues can be addressed in the future.

The **target audience** of this paper is composed of Android developers, researchers, and Android platform maintainers. We support Android developers by providing a set of actionable and evidence-based insights about how to manage Android run-time permissions. We support researchers by (i) informing them on the state of the practice about Android run-time permissions, and (ii) providing a tool-based classification pipeline for investigating on arbitrary aspects of Android user reviews. Finally, we support Android platform maintainers by providing a set of insights about how to improve the Android run-time permission system according to comments expressed by end users in app reviews.

The remainder of the paper is structured as follows. Section 2 provides background concepts about the Android run-time permissions. Section 3 describes the design of our study and Section 4 describes the semi-automatic classification pipeline we developed. The main results of our study are presented in Sections 5, 6, and 7, and they are discussed in Section 8. Section 9 discusses the threats to the validity of our study, whereas Section 10 describes related work. Section 11 illustrates the future research challenges we identified, and Section 12 closes the paper.

2 ANDROID RUN-TIME PERMISSIONS

Android relies on a *permission system* to inform users about access to sensitive information and enforce restrictions on the operations an app can perform. As several usability issues were discovered in older versions of the platform [2–4], most recent Android releases (i.e., from Android API version 23 onwards) moved to a *run-time permission system*. According to this system, permissions are divided into several protection levels, where the most important ones are about the so-called *normal* and *dangerous permissions*. Normal permissions cover areas where the app needs to access data or resources outside the app's sandbox, but with very little risk for the user's privacy (e.g., activation of the bluetooth, setting an alarm). Dangerous permissions cover areas where the app accesses data or resources that involve the user's private information (e.g., reading contacts, accessing the camera).

Granting dangerous permissions could potentially affect the user's stored data or the operation of other apps [7]. Hence, to obtain access to resources protected by dangerous permissions, developers must prompt the end user to request those permissions at run-time. To meet this requirement, developers are provided with a dedicated API to perform run-time permission requests [8]. Of course, the final decision about whether a permission is granted is in the hands of the end user, who can deny access to requested resources. Before performing a sensitive operation, developers should alway ask for required permissions, as users also have the option to enabled and disable permissions one-by-one in system settings, although they are not prompted again for accesses to resources for which permissions are already granted.

3 DESIGN OF THE STUDY

This section describes how we designed our study. In order to perform an objective and replicable study we followed the guidelines on empirical software engineering in [9] and [10].

3.1 Goal and Research Questions

The **goal** of the study is to evaluate the Android run-time permission system for the purpose of characterizing the way end users perceive its issues and benefits in the **context** of 15,124 free Android apps published in the Google Play Store. We refined this goal into the following research questions:

RQ1 - How *accurate* is an automated approach in classifying user reviews via different combinations of machine learning techniques?
RQ2 - To what *extent* app reviews express concerns about the Android run-time permission system?
RQ3 - What are the *main concerns* about the Android Run-time permissions system in app reviews?

RQ1 is a meta research question. By answering it we aim at objectively assessing the accuracy of different combinations of machine learning techniques, e.g., Naive Bayes classifier and support vector machines. Indeed, since the proposed software pipeline can include different components, it is expected that different combinations will result in different levels of accuracy. Obviously, we use the results of the most accurate software configuration when answering RQ2 and RQ3.

RQ2 aims at assessing how end users consider issues and benefits related to the Android run-time permission system, and if they vary across app categories.

The rationale behind RQ3 is to identify the main concerns of end users about the Android run-time permission system, what issues are still unresolved, but also positive reactions about it (e.g., praises).

3.2 Subjects Selection

In this section, we describe how we built the dataset used as a basis for our study. Collection process is shown in Figure 1.

Apps selection – As a starting point for our app selection we considered the top 500 most popular free apps from each of the 35 categories of the Google Play Store, as ranked by the App-Annie service for app ranking analysis[1] as of October 11, 2016. The total amount of entries we extracted is 15,517. At the time, some new categories (such as *Dating* and *Parenting*) had recently been

[1] www.appannie.com/apps/google-play/top-chart/united-states

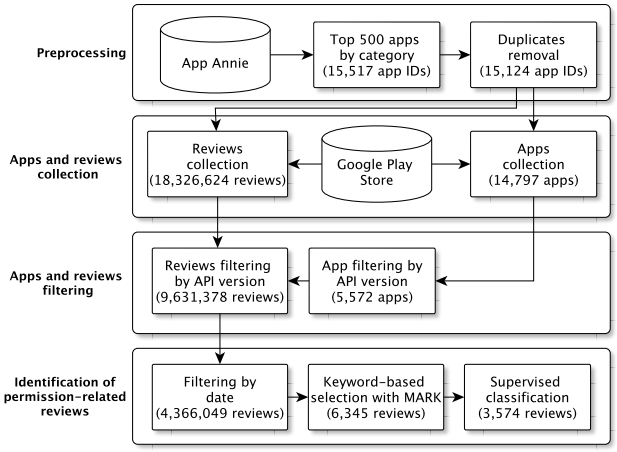

Figure 1: Summary of the data collection process

introduced in the store, and they contained less than 500 entries. Finally, we removed those duplicate apps that appeared in more than one category, achieving a set of 15,124 unique app IDs. Selection was limited to free apps, as binaries are necessary to identify apps that adopt the new permission system. In fact, free apps represent 93.99% of all Google Play Store apps and they are downloaded more often [11].

Apps and reviews collection – We downloaded from a third party service[2] the binary files (i.e., the APKs) of the apps identified in the app selection step. Some of the apps were not existing on the Google Play store, and have therefore been excluded. This can happen if the developers decide to remove the app from the store or if Google decides to remove the app for violation of some publishing policies. This led to the discarding of 327 apps, resulting in a total of 14,797 apps. Using an open-source web scraper[3], we collected all the user reviews published in the Google Play store for all the apps identified in the previous step. For each review, we collected the full *review text*, the *publication date*, and the *review rating*, i.e., a grade, on a scale from one to five, assigned by the reviewer to the app. As the Google Play store exhibits only a limited amount of reviews for each app at a given time [12], we repeated the collection process multiple times to collect a more extensive dataset. Afterwards, we merged the results of each collection iteration, discarding duplicates, leading to an acquired total of 18,326,624 individual reviews. The whole process lasted 8 weeks. The most recent review in our dataset is dated 28 February 2017.

Apps and reviews filtering – We developed a tool for automatically disassembling the APK of an Android app, analyzing its manifest file, and identifying the Android API Level targeted by the application (i.e., the value of the `androidtargetSdkVersion` attribute of the `uses-sdk` tag[4]). With such a tool, we identified the apps using an Android API version equal or greater than 23, which is the Android version in which the run-time permission system was first introduced. A total of 5,572 apps fell into this last category. Since we are only interested in reviews that discuss the new run-time permission system, we excluded all reviews belonging to apps that still use an API version earlier than 23. This filtering step resulted in a set of 9,631,378 reviews. From this set, we further filtered out all the reviews predating 5 October 2015, the release date of Android API 23, hence achieving a total of 4,366,049 reviews.

Keyword-based selection – In order to identify potential permission-related reviews we performed an additional keyword-based selection, following the intuition that users often use semantically similar or related keywords to describe an issue or a feature, as suggested in [6]. To select our keywords we relied on the MARK tool by Vu et al. [6, 13]. We choose MARK over other tools (e.g., CALAPPA [14] or AR-Miner [15]) for its ability to grasp semantic similarity among words and provide suggestions of keywords similar to the ones given as input. We provided the keyword *permission* as input and, by following the tool suggestions, we added to the set of keywords the words *privacy* and *consent*. We choose not to expand our keywords set further and limit ourselves to a concise set of neuter, pertinent keywords. Indeed, expanding it further would allow us to identify a larger set of potentially permission-related reviews. However, it would come at the expense of precision, while potentially introducing biases into our subsequent analysis stemming from positive/negative connotations associated with some words or expressions. The result of this filtering step is a set of 6,345 reviews. The high discard-rate in this step is not surprising, rather it is in accordance with existing research confirming that only a (relatively) small fraction of app reviews mention application permissions [16].

Supervised classification – The final step of our data collection process is a supervised classification procedure. First, we manually classified a sample of 1,000 randomly extracted potential permission-related reviews to build a ground truth for a subsequent automatic classification procedure. This resulted in a set of 780 classified permission-related reviews (the others were discarded as, despite containing selected keywords, clearly do not deal with Android permissions, e.g., "*Simple enough for beginners. Includes features like backup, paper wallet and privacy settings.*"). Complete details about the manual classification procedure are provided in Section 4.1. Secondly, we extended the classification to the remaining 5,345 potential permission-related reviews leveraging an automatic classification pipeline built on top of established machine learning and natural language process techniques, described in Section 4.2. The result of this step is a set of 3,574 permission related reviews, which form the objects of our study.

3.3 Variables

The main focus of RQ1 is on the configuration of the classification pipeline. The independent variables are three, each of them mapping to a specific phase of the pipeline: preprocessing, features extraction, and classification. The specific levels for these variables are described in details in Section 4. For what concerns the dependent variables, we measure the *accuracy* of our classification pipeline by relying on two key metrics commonly used in the field of automatic classification: *precision* and *recall* [17]. $Precision_c$ is the fraction of reviews that are classified correctly to belong to class c. $Recall_c$ is the fraction of reviews of class c which are classified correctly. We used the following formulas to calculate them:

[2]https://apkpure.com
[3]https://github.com/facundoolano/google-play-scraper
[4]http://developer.android.com/guide/topics/manifest/uses-sdk-element.html

$$Precision_c = \frac{TP_c}{(TP_c + FP_c)} \quad Recall_c = \frac{TP_c}{(TP_c + FN_c)} \qquad (1)$$

TP_c stands for *True Positives*, the number of reviews classified as of class c that actually belong to class c. FP_c stands for *False Positives*, the number of reviews *not* belonging to class c mistakenly classified as belonging to class c. FN_c stands for *False Negatives*, the number of reviews mistakenly classified as *not* belonging to class c even though they actually belong to class c. Considering precision and recall, we calculated one further metric, the F_1-*Score*, defined as the harmonic mean of the two, and provides a single measure of *accuracy* [17].

When addressing RQ2, for each app a in our dataset, we considered the following variables: the number of its permission-related reviews (R_a^p), the total number of reviews it received in the Google Play Store (R_a), and its category in the Google Play Store (cat_a).

Concerning RQ3, we considered the different categories of reviews concerning the run-time permission system, as extracted by our classification pipeline (see Section 4).

3.4 Execution

Concerning RQ1, we implemented our classification pipeline in Python, by relying on the Scikit-learn [18] and Nltk [19] Python modules version 0.19.01 and 3.2.2 respectively, with default parameters. We executed all the meaningful (possible) configurations for our classification pipeline, investigating on the impact that each single component of the pipeline had on its overall accuracy. Then, considering the limited availability of training data, we decided to evaluate each configuration by adopting the *k-fold cross-validation method* [20], with k equal to 10. According to this method, we split the available training data into k equal sized partitions. A single partition was retained as the validation data for testing; the remaining $k - 1$ partitions were used as training data. The training-validation process was repeated k times, with each of the k partitions being used exactly once, as per the validation data. At each iteration, we computed both the precision metric and the recall metric. In order to further account for the possible variability of the results that might occur due to the selection of the random partitions, we *repeated the whole process 100 times for each configuration of our classification pipeline*. Then, at the end of all iterations, we performed the average to produce a single estimation. The whole experiment was performed on a Ubuntu Linux virtual machine equipped with 8GB RAM and an Intel Xeon CPU E5630 Processor.

We extracted the values of the variables for answering RQ2 and RQ3 by relying on the data we mined when building the dataset and on the semi-automated process for identifying permission-related reviews. There, R_a^p is the count of permission-related reviews for each specific app category (as defined in the Google Play Store). We approached automatic classification of user reviews with a dual goal in mind: (i) we wanted to automate a process that would be unfeasible to perform manually; (ii) we still wanted to retain a satisfactory level of accuracy. R_a and cat_a are extracted when building the dataset (see previous section).

3.5 Study Replicability

The source code of the classification pipeline, the source code developed for the experiments, and the raw data with the results of both manual and automatic processes are publicly available in the on-line replication package[5].

4 THE CLASSIFICATION PIPELINE

Given the number of items to classify (i.e., 6,435 potential permission-related user reviews), it would be unfeasible to perform a complete manual analysis. Therefore, we resorted to a two steps semi-automated classification pipeline. First, we manually analyzed a sample of the data and built a taxonomy of end users' comments, grouping together permission-related reviews into relevant groups with descriptive labels (see Section 4.1). Then, we trained a machine-learning classifier with the manually-built sample, and we used the trained classified on all the remaining reviews (see Section 4.2).

4.1 Manual Analysis

To perform the initial manual analysis, we randomly selected a sample of 1,000 reviews from the permission-related set of our dataset. Such a sample allows us to achieve a confidence level higher than 95% and a 5% confidence interval. We further divided the sample into two equal parts and assigned each to a different researcher. To reduce bias, each researcher independently surveyed the reviews in his sample, grouping together reviews containing similar concerns, and determined an informative label for each group found. After completing the analysis, the two researchers discussed together the identified groups, aligned the labels with each other, and solved all the cases in which there was a disagreement. During this revision step, a total of 48 reviews were reclassified: one category identified by one researcher was divided into more finer-grade ones detected by the other; another category was discarded (it grouped user complaints about apps performing actions without their knowledge but not related to the Android permission system, e.g., *"Sends requests to Facebook friends without permission."*). The resulting categories are discussed in Section 7, whereas the manually-classified reviews are available in the replication package of this study.

4.2 Automatic Classification

The structure of our classification pipeline is outlined in Figure 2. A detailed description follows.

The main input of our classification pipeline is composed of the raw text of end users' reviews. Also, since the quality of the results may potentially be improved by feeding the classification pipeline with additional data, we included also user ratings as part of the input.

In the following, we illustrate the design of our classification pipeline and the main components that constitute it.

Preprocessing – The quality of the results can be affected by the preprocessing steps performed on the inputs. Therefore, we experimented with different preprocessing steps commonly performed in the field of Natural Language Processing, whose purpose is to refine the data fed to the subsequent classification step. We experimented the following techniques:

[5] http://cs.gssi.it/MobileSoft2018ReplicationPackage

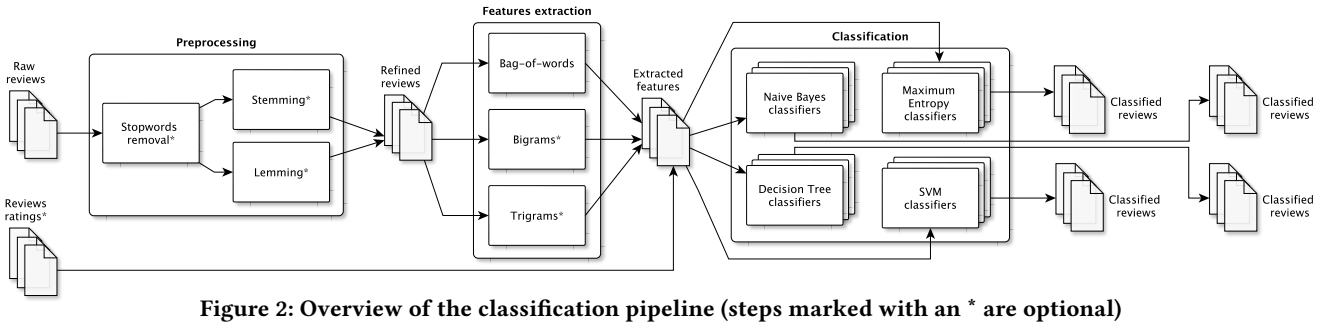

Figure 2: Overview of the classification pipeline (steps marked with an * are optional)

- *Removal of stopwords* – words commonly used in the English language, such as "as", "can", "it", "so", which do not greatly affect the semantics of a sentence. Removing stopwords from reviews potentially removes noise from input data, thus allowing classifiers to focus on more relevant words. We experimented with the default list used by Scikit-learn[6].
- *Stemming* – the process of reducing inflected or derived words to their root form. For instance, the words "connections", "connective" and "connected" are all reduced to the same base word "connect".
- *Lemmatization* – a process that reduces a word to its canonical form named *lemma*. Unlike stemming, lemmatization is performed with the aid of dictionaries and takes the linguistic context of the term into consideration. For instance, lemmatization correctly identifies "good" as the lemma of the word "better".

Reviews representation – We selected the popular *bag-of-words* model [21] to represent reviews in our pipeline. Bag-of-words is a simplifying representation in which a document is represented as the multiset of its words, disregarding grammar and word order, but keeping multiplicity. From this representation classification algorithms can learn the review type based on the terms existence and frequency. However, common words like "the", "a", "to" are almost always the terms with highest frequency in documents. To address this problem we employed *tf-idf* (term frequency-inverse document frequency) normalization [21], that weights with diminishing importance terms that occur in the majority of documents.

One of the known disadvantages of bag-of-words representation is that spatial information about words sequences is not preserved. Hence, we also experimented with *n-grams* [21], another commonly used document representation model. An n-gram is a sequence of n contiguous words extracted from document text. We consider n-grams of length 2 and 3, namely *bi-grams* and *tri-grams* respectively.

Classification – Since, potentially, a review can contain multiple users' comments, it can correctly be classified as belonging to multiple categories. Therefore, our case falls into the problem of *multi-label classification*. This poses the question of whether to train a single *multi-label classifier* or multiple *binary classifiers*, one for each category. A binary classifier is a classifier trained for the task of deciding whether an input instance belongs to a given class or not. A multi-label classifier instead is trained to assign a given instance to one of multiple classes, greater than two, with no restriction on how many of the classes the instance can be assigned to. We decided to rely on the former for two main reasons: (i) previous research provided evidence that multiple binary classifiers *perform better* than multi-label ones for user reviews [22]; (ii) training multiple binary classifiers gives us more *flexibility*, potentially allowing us to choose a different classifier for each taxonomy category.

We included four commonly used binary classification techniques known to perform well on textual inputs in our experimentation [23]:

- *Naive Bayes* [24], a popular algorithm for binary classifiers based on applying Bayes' theorem with strong independence assumptions between features. It is computationally efficient and achieves good predictive performance in many real-world applications.
- *Decision tree learning* [25], which iteratively constructs a decision tree to be used as a classifier. A decision tree is a tree-shaped graph in which each non-leaf node denotes a test on a specific feature, each branch represents the outcome of a test, and each leaf holds a class label. Decision trees are simple to interpret and mirror human decision making more closely than other approaches. For our experimentation we adopted a *CART* tree [26].
- *Maximum entropy* [27] (also known as MaxEnt or multinomial logistic regression), a probabilistic classifier based on the principle of maximum entropy. It does not assume conditional independence of features.
- *Support vector machines* (SVM) [28], a non-probabilistic binary classifier. SVMs plot each data item as a point in n-dimensional space and finds the hyper-plane that maximizes the gap among the two different classes.

5 RQ1 - HOW ACCURATE IS AN AUTOMATED APPROACH IN CLASSIFYING USER REVIEWS VIA DIFFERENT COMBINATIONS OF MACHINE LEARNING TECHNIQUES?

As discussed in Section 3.4, we executed a large number of runs of our classification pipeline, exploring different combinations of: reviews representations, preprocessing steps, and classification algorithms. For reasons of space, and for a clearer exposition, we report on the most indicative results only. The complete results of all

[6]https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py

the performed runs are available in the replication package, together with a complete implementation of our classification pipeline.

Baseline: No preprocessing – bag-of-words representation. Table 1 summarizes the results of the first combination we explored. We provided as inputs to the classification algorithms the raw text of the reviews in the bag-of-words representation and without any preprocessing. This is a basic configuration, which allowed us to assess the baseline performance of the different algorithms for our problem.

Decision Tree performed the worst, achieving an F_1-Score of 0.635, well below the 0.75 threshold reached by the other algorithms. Still, despite achieving a similar F_1-Score, we can notice a difference in the performances of the remaining algorithms: Naive Bayes attained an high recall (the highest for 9 categories out of 10, with an average of 0.859) at the cost of a lower precision, while the opposite is true for Maximum Entropy and SVM, as both attained an higher precision (0.735 on average for both) at the cost of a lower recall.

Table 1: Baseline precision, recall and F_1−Score for all algorithms

	PP	PC	MP	TMP	UP	PB	FU	RPR	SP	BRT	Avg
Precision											
Naive Bayes	0.786	0.664	0.790	0.671	0.676	0.753	0.608	0.635	0.640	0.615	0.684
Decision Tree	0.693	0.612	0.716	0.641	0.731	0.679	0.597	0.620	0.587	0.529	0.640
Max Entropy	**0.788**	**0.686**	**0.833**	**0.733**	**0.811**	0.788	0.675	0.696	0.700	0.638	**0.735**
SVM	0.766	0.681	0.821	0.712	0.791	**0.792**	**0.680**	**0.702**	**0.747**	**0.653**	**0.735**
Recall											
Naive Bayes	0.732	**0.777**	**0.897**	**0.873**	**0.903**	**0.915**	**0.895**	**0.887**	**0.910**	**0.799**	**0.859**
Decision Tree	0.714	0.595	0.735	0.649	0.743	0.670	0.582	0.577	0.554	0.488	0.631
Max Entropy	0.751	0.723	0.865	0.747	0.775	0.849	0.779	0.812	0.868	0.713	0.788
SVM	**0.754**	0.714	0.841	0.737	0.773	0.833	0.768	0.794	0.890	0.732	0.784
F-Score											
Naive Bayes	0.758	**0.716**	0.840	**0.759**	0.773	**0.826**	**0.724**	0.740	0.751	**0.695**	0.758
Decision Tree	0.703	0.603	0.725	0.644	0.737	0.675	0.589	0.597	0.570	0.508	0.635
Max Entropy	**0.769**	0.704	**0.848**	0.740	**0.793**	0.818	0.723	**0.750**	0.775	0.673	**0.759**
SVM	0.760	0.697	0.831	0.725	0.782	0.812	0.721	0.745	**0.812**	0.690	0.757

No preprocessing – different review representations. Going further, for our second set of runs, we aimed at evaluating which reviews representation, or combination of, leads to a better classification accuracy. Hence, we performed several runs changing the reviews representation while still not performing any preprocessing or introducing the score metadata.

Results highlight that adopting bigrams or trigrams does not improve the accuracy for any of the algorithms. We conjecture that this lack of improvement over the bag-of-words representation is due to the short length of reviews (in the sample we used for the manual analysis, the average length is 204 characters, with a standard deviation of 128) from which only a relatively low number of features can be extracted when choosing n-grams over bag-of-words. However the quality of the results improves when combining bigrams with bag-of-words. With this combined representation, Naive Bayes is outperformed by both Max Entropy and SVM, the latter achieving the best F_1-Score of 0.773. We conjecture that this improvement is due to the ability of this combined representation to extract a sufficient number of features, while preserving spatial information that helps when having to discern different usages of frequent words, e.g., the bigrams "excessive permissions" and "no permissions" have a very different meaning despite both containing the term "permission". Additionally including trigrams in this combined representation does not seem to lead to a further improvement.

With preprocessing – combined review representations. For our next set of runs, we introduced different preprocessing steps in the classification pipeline, while still maintaining the combined representation of bigrams and bag-of-words, which resulted in the best quality of results in the previous case.

The results of this new set of runs evidence that stemming, stopwords removal, and the combination of the two slightly improve the accuracy of the Naive Bayes and Decision Tree algorithms, with the former achieving an F_1-Score of 0.778 (the highest so far). SVM and Maximum Entropy are instead negatively affected by this preprocessing method. Usage of lemmatization, both alone and combined with stopwords removal, instead improves average accuracy *for all algorithms*. Whit lemmatization, SVM also achieves an F_1-Score of 0.778, tying for the best performance so far.

With preprocessing – with user ratings. In this case, we consider also user ratings as input to the classification pipeline. We performed both stopwords removal and lemmatization since they gave the best results in the previous set of runs. Table 2 shows the results we obtained in this set of runs.

Table 2: Final precision, recall and F_1−Score for all algorithms

	PP	PC	MP	TMP	UP	PB	FU	RPR	SP	BRT	Avg
Precision											
Naive Bayes	0.604	**0.911**	0.578	**0.868**	**0.892**	**0.820**	**0.833**	**0.784**	0.611	**0.667**	**0.757**
Decision Tree	**0.762**	0.691	0.791	0.664	0.739	0.699	0.641	0.661	0.648	0.542	0.684
Max Entropy	0.740	0.663	0.737	0.676	0.686	0.731	0.680	0.624	0.627	0.648	0.681
SVM	0.756	0.718	**0.799**	0.733	0.761	0.790	0.686	0.667	**0.681**	0.642	0.723
Recall											
Naive Bayes	**0.997**	0.278	**0.999**	0.451	0.615	0.702	0.240	0.432	0.745	0.317	0.578
Decision Tree	0.764	0.654	0.797	0.652	0.758	0.710	0.601	0.644	0.711	0.487	0.678
Max Entropy	0.923	**0.779**	0.914	0.820	0.824	0.816	0.868	0.747	0.828	0.745	0.826
SVM	0.885	0.774	0.916	**0.834**	**0.843**	**0.861**	**0.902**	**0.819**	**0.916**	**0.807**	**0.856**
F-Score											
Naive Bayes	0.752	0.426	0.732	0.593	0.728	0.757	0.373	0.557	0.672	0.429	0.655
Decision Tree	0.763	0.672	0.794	0.658	0.748	0.705	0.620	0.652	0.678	0.513	0.681
Max Entropy	**0.822**	0.717	0.816	0.741	0.749	0.771	0.763	0.680	0.713	0.693	0.747
SVM	0.815	**0.745**	**0.854**	**0.780**	**0.800**	**0.824**	**0.780**	**0.735**	**0.781**	**0.715**	**0.787**

In this new configuration, the Naive Bayes algorithm achieved the highest average precision, with a value of 0.757, but it came at the cost of recall, for which it performed worst with a value of 0.578. Differently, Max Entropy and SVM have an increased F_1-Score, while still retaining a reasonable balance among precision and recall. SVM, with an average F_1-Score of 0.787, is the configuration that attained the best performance. We statistically tested this result by applying the Wilcoxon rank-sum test [29] for pairwise data comparison under the alternative hypothesis that the samples do not have equal medians. The performed test confirmed that this last configuration performs statistically better than the others (higher p-value = $2.2e^{-16}$). We believe that this improvement comes from the synergistic effect of all the techniques that we selected during the exploration. Based on this result, we adopted the **lemmatization+bigrams+bag-of-words+SVM** configuration to answer RQ1 and RQ2.

6 RQ2 - TO WHAT EXTENT APP REVIEWS EXPRESS CONCERNS ABOUT THE ANDROID RUN-TIME PERMISSION SYSTEM?

We identified a total of 3,574 reviews that discuss the Android run-time permission system. Even if they amount for a very small portion of the 18,326,624 reviews we started from, these belong to a total of 1,278 unique apps, equal to the 23% of the collected apps that employ the run-time permission system, and 8.6% of the total amount of all apps of our dataset.

Looking at the frequencies of permission-related reviews among app categories (summarized in Table 3), we can observe that permission-related reviews occur in almost all categories, with the exception of *Libraries and demo* and *Events* (notice that both categories were recently introduced in the Google Play Store and thus contain a smaller amount of apps). This observation may be an indication that permission-related reviews are somehow orthogonal across apps, independently of the specific context and permissions requirements. We can also notice that, even if still limited in numbers, categories *Productivity* and *Tools* contain more permission-related reviews than others. The opposite is happening for other categories, like *Games* and *Photography*, which have a lower amount, despite a high number of reviews.

Table 3: Distribution of apps, reviews, and permission-related reviews across categories

Category (cat_a)	Apps	Reviews (R^a)	Permission-related reviews (R^a_p)
Productivity	483	982,607	453 (0.0046‰)
Tools	490	1,343,676	372 (0.0028‰)
Communication	471	953,134	287 (0.0030‰)
Health and fitness	480	678,978	276 (0.0041‰)
Entertainment	498	1,039,598	234 (0.0023‰)
Shopping	484	631,367	216 (0.0034‰)
Business	495	482,456	209 (0.0043‰)
Lifestyle	494	713,815	201 (0.0028‰)
Social	491	747,139	176 (0.0023‰)
Android Wear	500	851,086	176 (0.0201‰)
News and magazine	491	419,504	149 (0.0035‰)
Travel and local	481	435,270	138 (0.0032‰)
Media and video	486	729,509	131 (0.0018‰)
Transportation	492	249,239	119 (0.0048‰)
Finance	493	523,841	102 (0.0020‰)
Music and audio	485	830,703	101 (0.0012‰)
Weather	484	237,229	99 (0,0041‰)
Personalization	497	713,124	99 (0.0014‰)
Education	500	675,976	81 (0.0012‰)
Photography	496	1,002,740	78 (0.0008‰)
Sports	488	331,062	78 (0.0023‰)
Games	444	1,654,846	76 (0.0004‰)
Books and reference	495	551,705	67 (0.0012‰)
Family	418	848,046	58 (0.0007‰)
Medical	497	195,632	47 (0,0024‰)
Food and drink	498	115,387	35 (0.0030‰)
Auto and vehicles	493	54,428	31 (0.0057‰)
House and home	292	57,990	23 (0.0040‰)
Comics	500	116,881	19 (0.0016‰)
Dating	247	61,489	19 (0.0031‰)
Beauty	95	6,760	11 (0.00162‰)
Art and design	492	44,323	4 (0.0090‰)
Parenting	185	26,667	2 (0.0001‰)
Libraries and demo	137	17,615	0 (0‰)
Events	52	2,748	0 (0‰)
Sum total	15,124	18,326,624	3,574 (0.0019‰)

7 RQ3 - WHAT ARE THE MAIN CONCERNS ABOUT THE ANDROID RUN-TIME PERMISSIONS SYSTEM IN APP REVIEWS?

The manual analysis of the 1,000 reviews led to the definition of a taxonomy composed of 10 categories of recurring users's concerns. In addition, we identified two macro-categories: positive opinions (in the following marked with a + sign), and negative ones (marked with a − sign). In the following we describe each category.

+ **Permission Praise (PP) –** The reviewer expresses some praises for the app handling of permissions, but it does not delve into details. A clarifying example is the following: "*Cool game -clean code and permission friendly app.*"
+ **Minimal Permissions (MP) –** The reviewer feels that the app asks only for permissions strictly needed in order to carry on with its advertised functionalities. An example of this kind of reviews is: "*I've tried others, this is the simplest, easiest to use. Does not ask for permissions!!*"
− **Permissions Complaint (PC) –** The reviewer expresses some complaints about the app handling of permissions, but does not provide details about it. An example: "*Used to love it. Don't like the new permissions.*"
− **Too Many Permissions (TMP) –** The reviewer complains about the excessive amount of permissions requested by the app. An illustrative case: "*This app demands access to contacts, the camera, and SMS. Too many permissions for a glorified chat app. A real shame.*"
− **Unclear Permissions (UP) –** The app does not explain why some of the requested permissions are needed or the provided explanation is not convincing. An example is the following: "*Why on earth does it need that permission?*"
− **Permission-related Bug (PB) –** The app contains some bug related to permission requests that prevents it's correct functioning. An instance: "*Whenever I tried to measure my bp, it gave error that doesn't have permission to access camera but actually it has access to camera.*"
− **Repeated Permission Requests (RPR) –** When some permission are denied, the app repeatedly keeps requesting them, rendering usage of the app itself impossible or burdensome. An example of this kind of reviews is: "*App sounds like a great idea. However you cannot go without swiping one app with location denied before it asks for a permission again. Im not searching for jobs in my area I don't want the location on. [...]*"
− **Settings Permissions (SP) –** Despite adopting an Android API version greater or equal to 23, the app does not perform run-time permission requests properly. Thus, in order to properly use the app functionalities, the review author was forced to manually grant permissions to it through the device system settings. One example is the following: "*Nexus 5, had to manually give the app permissions to access GPS. Seems to be working now.*"
− **Bad Request Timing (BRT) –** According to the review author, run-time permission requests are performed at a wrong time during app execution. One example of such reviews is the following: "*Strange that you ask for all the permissions up front now, instead of the Marshmallow approach of asking for permissions when people actually invoke the corresponding functionality [...]*"
− **Functionality Unavailable (FU) –** Without granting one or more permissions, usage of some key app functionality is impossible. A clarifying example is the following: "*The app requests permissions per API 23, but if you refuse to grant them, the app shuts down and refuses to run [...]*"

Notice that a review can potentially be assigned to more than one category as not-all categories are mutually exclusives (e.g., *Settings permissions* and *Permission-related bug* are non-exclusive, as the user might be forced to assign permissions from the device settings to circumvent a bug). A breakdown of the amount of reviews classified in each category is provided in Table 4.

The results of the **automatic classification** are summarized in Table 4. Using our classification pipeline, we have been able to

Table 4: Breakdown of classification results

Category	Manual Classification	Automatic Classification	Total
Permission praise (+)	128	572	700
Minimal permissions (+)	121	400	521
Permission complaint (-)	100	386	486
Too many permissions (-)	151	594	745
Unclear permissions (-)	175	739	914
Permission-related bug (-)	120	423	543
Functionality unavailable (-)	41	45	86
Repeated permission requests (-)	28	67	95
Settings permission (-)	20	22	42
Bad request timing (-)	21	3	24
Sum total	905	3,251	4,156

assign a total of 3,251 labels distributed among all categories of the manually extracted taxonomy. The total amount of assigned labels from the combination of manual and automatic classification is 4,156 (they are distributed over 3,574 unique reviews). The majority of the classified reviews belong to categories *Unclear permissions* (914), *Too many permissions* (745), *Permissions praise* (700), *Permission-related bug* (543) and *Minimal permissions* (521). A smaller amount has been identified for categories *Repeated permission request* (95), *Functionality unavailable* (86), *Setting permissions* (42) and *Bad request timing* (21), presumably since, for these categories the amount of reviews identified during the manual analysis was insufficient for proper training of the classifier.

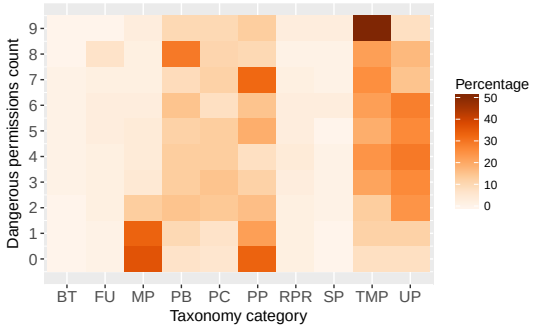

Figure 3: Permission-related reviews by number of requested permissions

The heatmap in Figure 3 provides an overview of how the frequency of these permission-related reviews is distributed among apps, grouping the latter according to the number of requested permissions. Here, we focused only on dangerous permissions [7], the only ones that must be granted at run-time after the changes introduces by Android 6. We can notice that, for apps that only request one or two dangerous permissions the majority of permission-related reviews belongs to categories *Minimal Permissions* (MP) and *Permission Praise* (PP) thus suggesting that privacy-aware users notice, and appreciate, the low amount of privileges required by these apps. Unsurprisingly instead, for app that request all 9 dangerous permissions, the majority of permission-related reviews belong to category *Too Many Permissions* (TMP). Furthermore, focusing on apps that request between 3 and 6 dangerous permissions, we notice that the majority of permission-related reviews for these apps belong to categories *Too Many Permissions* (TMP) and *Unclear Permissions* (UP) but does not significantly increase, as one would expect, as the number of requested permissions increases. We may conjecture that users are keen to form a binary opinion about the number of permissions requested by an app: they appreciate it when a low amount of permissions is requested and they are disappointed when it is higher than they expect.

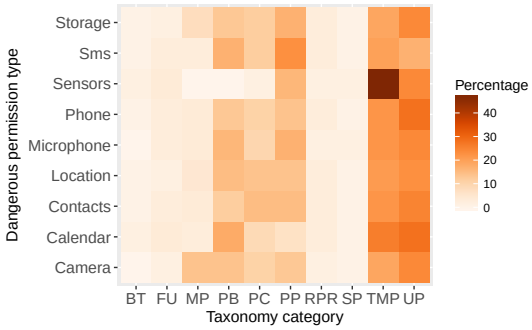

Figure 4: Permission-related reviews by requested permission

Figure 4 show the distribution of categories of permission-related reviews across the types of requested permission. Praises and complaints about permissions expressed by users in app reviews are distributed evenly, independently of the type of requested permission, with the sole exception of the *Sensors* permission, for which users seem to be concerned more when too many permissions are requested.

8 DISCUSSION

Overall we identified 3,574 reviews discussing the run-time permission system. These reviews belong to a total of 1,278 unique apps, which are 23% of the collected apps that employ the Android run-time permission system, and indeed the 8.6% of the total amount of apps collected from the store. Considering these numbers, we can infer that:

> The number of users having concerns w.r.t. the Android run-time permission system is quite limited, even though problems pertaining to permissions are widespread among apps.

Going one step further, when focusing on the macro-categories of positive and negative opinions, we can notice that the majority of classified reviews belong to the latter and amount to 70.6% of the total. Negatives opinions appear in a total amount of 2,459 reviews, as opposed to 1,185 reviews containing positive ones (note that only 70 reviews contain both positive and negative opinions).

The distribution of ratings, shown in Figure 5, is skewed towards the maximum score (i.e., 5 stars) for positive categories; instead the distribution of ratings of negative categories is skewed towards the lower end. We verified for statistical significance of these differences in ratings by performing the two-tailed Mann-Whitney U-test among each possible pair positive-negative categories, obtaining always a p-value below $2.2e^{-16}$. Henceforth, we can deduce that:

> For privacy-aware users, negative concerns on how permissions are handled are correlated with negative concerns about the whole app, thus confirming the importance of users' judgments about permissions.

We identified a total of 972 apps whose reviews contain negative opinions. Analyzing the number of identified reviews, displayed in Table 4, negative categories with the highest count are *Unclear permissions* (914), *Too many permissions* (745) and *Permission-related bug* (543). The high cardinality of the first category evidences one of the problems of the run-time permission system:

> Users still do not understand why they are being asked for a permission. More work is required to fully point out the reasons, which could derive from poor design of the run-time permission system itself or from developers that do not utilize it properly, often requesting permissions without providing an explanation.

App `com.chopracenter.meditationexperience` is a clear example of the latter, one of the apps with the highest amount of this kind of reviews. One of them points out: "*Why does the app now need to use my permission to use my phone and my contacts? Seems unnecessary.*". Furthermore, the high number of reviews belonging to the category *Too many permissions* leads us to believe that:

> The problem of *permission creep* [30, 31], i.e., apps routinely requesting more permissions than needed to carry on with advertised functionalities, is still present under the new run-time permission system.

This issue is clear when looking at the reviews of `com.lge.app1`, an app having numerous reviews of this kind: "*Update app to use less permissions and I'll reconsider. No way it needs to make phone calls or see my calendar to function with the TV.*". The high count of reviews classified under *Permission-related bug* evidences that:

> The introduction of the new run-time permission system also introduced a new class of bugs that didn't exist before, in the old install-time permission system. These bugs derive from the fact that developers do not always correctly perform permission requests.

We believe that this misusage of permission requests by developers is partially due to a lack of understanding of the inner workings of the run-time permission system, and partially because it is hard for developers to foresee all points where a permission must be requested at run time. Indeed, Android apps often have a complex event-driven control-flow with multiple entry points [32]. For instance, one of the apps with a high amount of this kind of reviews

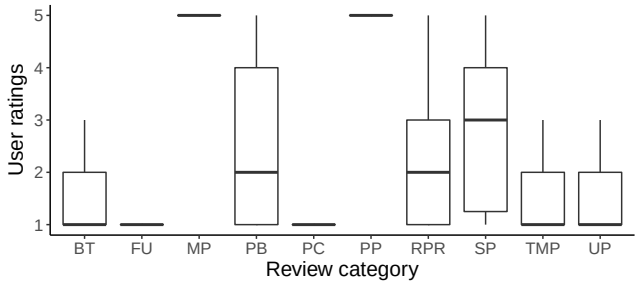

Figure 5: User ratings across reviews categories (outlayers not shown)

is `com.getittechnologies.getit`, where users highlight that the developer never requested the required permission: "*It can't access my GPS location but I don't think it ever requested permission to do so*".

Even if not among the most numerous, the categories *Repeated permissions requests* and *Functionality unavailable* can be an indication of the fact that:

> Some developers do not adhere to the guidelines provided by Google [8, 33]. Rather, they attempt at coercing users to provide some permissions, by constantly repeating permission requests or completely blocking access to specific features when permissions are not granted.

One example of the former case is `com.htc.Weather`: "*Annoying that it keeps asking permission of location [...]*". An instance of the latter case is `com.dteenergy.mydte`: "*I can't effectively use it because I denied accessing my contacts and files. There is no reason to allow that [...] "*.

From a more technical standpoint, our analysis shows that:

> Well-established, off-the-shelf machine learning techniques, combined with basic NLP preprocessing, can be profitably used to derive interesting insights from app store reviews, even on specific topics when sufficient amounts of data are considered.

9 THREATS TO VALIDITY

In the following we discuss the threats to validity of our study according to the Cook and Campbell categorization [34].

Internal validity refers to the causality relationship between treatment and outcome [9]. In our study we classified user reviews dealing with Android run-time permissions relying on machine learning techniques. This kind of analysis is non-deterministic as different runs of the analysis may lead to different results. To mitigate this potential treat, we repeated the experiments multiple times to correctly assess the accuracy of the different combinations of applied techniques. Moreover, we manually created training data for the machine learning models, following the procedure described in Section 4.1. To ensure that the final model used for classification does not show abnormal behavior, we manually analyzed 100 classification results to exclude the presence of glaring anomalies.

External validity deals with the generalizability of obtained results [9]. To ensure that our subjects are representative of the population of Android apps, we downloaded the top 15,124 apps in the United States across all categories of the Google Play Store, as ranked by App-Annie. Since the apps are the top ranking apps of all categories, we can expect that they have a high number of users because they are ranked using a combination of number of downloads and aggregate user ratings. Also, by considering the most popular free apps per category, we increase the chance to include apps with a more active user base in terms of quality and number of reviews. Free apps represent 75% of all Google Play Store apps and they are downloaded more often [35].

Construct validity deals with the relation between theory and observation [9]. The goal of our study is to analyse user reviews dealing with the Android run-time permission system, with the ultimate goal of identifying its recurrent issues from the end user perspective. Even when focusing on issues potentially noticeable

by end users, only a minority of them show sufficient awareness. Hence, the proposed approach might not discover the more subtle issues. We mitigate this threat by considering an initially large set of reviews. Moreover, as we employed keyword-based filtering, we might have missed an amount of reviews that reference permissions using keywords not in our list. We mitigated this threat by adopting a list of highly pertinent neuter keywords, thus, albeit in lower number, the selected reviews are highly on target for our purposes.

Conclusion validity deals with the statistical correctness and significance [9]. In this study we assumed that user reviews are a reliable source for inferring user concerns about the Android run-time permission system. However, there may be other factors that potentially may affect users judgment. To mitigate this threat, while performing the initial manual analysis we also ensured that many permissions-related reviews bring meaningful insights on the subject. Numerous examples of purposeful reviews have been reported in the paper to highlight the usefulness of classified concerns. The full set of classified user reviews is publicly available in the replication package of this study.

10 RELATED WORK

In the literature, few studies have focused on the Android run-time permission model. Panagiotis et al. analyzed users' adaptation to the new run-time permission model [36]. They gathered anonymous data from 50 participants. Their analysis indicates that users adapted positively to the new model. In [37], the same authors reported that users are willing to permit access to the devices' storage, and are more likely to grant access to some resources for specific app categories. They suggest that users should be informed about the resources an app needs to provide its core functionalities before installation.

As of today, studies about opinion mining from app reviews are quite common [38], but those that focus on permissions are scarce. Ha et al. investigated on whether users were discussing privacy and security risks of an app in their reviews [16]. After manually analyzing 556 reviews belonging to 59 different apps, they noticed that only 1% of collected reviews mentioned permissions. Khalid et al. investigated on the most common kinds of complaints in iOS app reviews [39, 40]. By manually analyzing 6,390 app reviews, they reported that privacy and ethics related complaints amount to only 1.19% of the total, but are among the ones that bother users' the most. We conducted a semi-automated analysis on a much larger scale.

Automatic classification of app user reviews has been performed in the past for a variety of reasons. For example, Maalej et al. investigated the usage of several machine learning and NLP techniques to automatically classify user reviews into four categories: bug reports, feature requests, user experiences, and ratings [22]. In their experiment, they achieved 80% of precision and recall user experiences. Our results are comparable, when considering that we focus on much narrower categories.

McIlroy et al. investigate the multi-faceted nature of user reviews published on app stores [41]. They discovered that up to 30% of reviews raise various types of issues and propose an approach for automated multi-label classification, achieving an F_1-Score of 45% for reviews that express privacy or ethical issues. In our work, we focused more closely on run-time permissions and leveraged multiple binary classifiers, thus achieving a much higher average F_1-Score.

11 FUTURE RESEARCH CHALLENGES

Link permissions to functionalities – In Section 8 it was evidenced that: *(i)* users still tend to not understand why they are being asked for permissions, and *(ii)* developers routinely request more permissions than needed. We believe that one possible way to address these problems is to design new permission systems in which permissions are granted to individual app functionalities, rather than to the full app as a whole. With such systems, users may better understand why a given permission is requested. The first stepping stone in this direction is to devise an effective way to *logically link permissions to functionalities.* In this context, information coming from software repositories (e.g., GitHub) may play a key role since it empirically emerged that developing and updating app functionalities are the most frequent activities reported by developers in their commit messages [42].

Better support for developers – A numerous category of users perceptions is about *Permission-related bugs*. Indeed, the shift to run-time permissions burdened developers with one additional task, as now they have to preemptively ask for allowance of permissions before accessing restricted parts of the platform. Failure to properly do so renders functionalities or, in worst cases, the whole app unreachable and/or unusable. In order to properly deal with this additional task, developers should not only be further educated, but also empowered with tools that assist them in correctly handling permissions and suitably positioning the corresponding requests.

Impact of run-time permission requests on user experience – Developers are faced with one additional challenge when dealing with run-time permissions: on the one hand, they must promptly and precisely inform users about needed sensitive data; on the other hand, they must carefully plan the usage of notification dialogs, so as to avoid disrupting the user experience (in the literature this phenomenon is called *warning fatigue* [43]). When using run-time permissions, there can be cases in which introducing new functionalities that require additional permissions may even be detrimental to the app success. Hence, a further open research area is understanding and quantifying the effects of run-time permission requests on the overall user experience. We are currently investigating, leveraging static analysis techniques, how performing permission requests at different phases of program execution impacts user experience.

12 CONCLUSION

We conducted a large-scale empirical study to investigate users' perceptions about the run-time permission system offered by Android. We inspected over 4.3 million user reviews out of 5,572 apps within the Google Play Store.

The reviews were classified and analyzed by employing a classification pipeline based on machine learning and natural language processing techniques. Its accuracy has been empirically evaluated with promising results.

Future work involves pursuing the future research directions discussed in Section 11, extending the developed classification pipeline, and continuously evaluating its accuracy when including new preprocessing, review representations, and classification components.

13 REFERENCES

[1] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In International Conference on Trust and Trustworthy Computing, pages 291–307. Springer, 2012.

[2] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In Proceedings of the Eighth Symposium on Usable Privacy and Security, pages 3:1–3:14, 2012.

[3] Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In Proceedings of the 16th International Conference on Financial Cryptography and Data Security, 2012.

[4] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pages 3393–3402, 2013.

[5] Requesting permissions at run time. https://developer.android.com/training/permissions/requesting.html#perm-check. Online; accessed 12-January-2017.

[6] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach (t). In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pages 749–759. IEEE, 2015.

[7] Normal and dangerous permissions. https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous. Online; accessed 12-January-2017.

[8] Requesting permissions at run time. https://developer.android.com/training/permissions/requesting.html. Online; accessed 24-October-2017.

[9] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén. Experimentation in Software Engineering - An Introduction. Kluwer Academic Publishers, 2012.

[10] Forrest Shull, Janice Singer, and Dag IK Sjøberg. Guide to advanced empirical software engineering, volume 93. Springer, 2008.

[11] Distribution of free and paid android apps in the google play store from 3rd quarter 2017 to 4th quarter 2017. https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/. Accessed: 2018-03-09.

[12] William Martin, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. The app sampling problem for app store mining. In Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, pages 123–133. IEEE, 2015.

[13] Phong Minh Vu, Hung Viet Pham, Tam The Nguyen, and Tung Thanh Nguyen. Tool support for analyzing mobile app reviews. In Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, pages 789–794. IEEE, 2015.

[14] Vitalii Avdiienko, Konstantin Kuznetsov, Paolo Calciati, Juan Carlos Caiza Román, Alessandra Gorla, and Andreas Zeller. Calappa: a toolchain for mining android applications. In Proceedings of the International Workshop on App Market Analytics, pages 22–25. ACM, 2016.

[15] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. Arminer: mining informative reviews for developers from mobile app marketplace. In Proceedings of the 36th International Conference on Software Engineering, pages 767–778. ACM, 2014.

[16] Elizabeth Ha and David Wagner. Do android users write about electric sheep? examining consumer reviews in google play. In Consumer Communications and Networking Conference (CCNC), 2013 IEEE, pages 149–157. IEEE, 2013.

[17] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. Modern information retrieval, volume 463. 1999.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[19] Steven Bird. Nltk: the natural language toolkit. In Proceedings of the COLING/ACL on Interactive presentation sessions, pages 69–72. Association for Computational Linguistics, 2006.

[20] Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. Journal of the royal statistical society. Series B (Methodological), pages 111–147, 1974.

[21] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

[22] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? on automatically classifying app reviews. In Requirements Engineering Conference (RE), 2015 IEEE 23rd International, pages 116–125. IEEE, 2015.

[23] Charu C Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. Mining text data, pages 163–222, 2012.

[24] Steven Bird, Ewan Klein, and Edward Loper. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.", 2009.

[25] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. An introduction to statistical learning, volume 112. Springer, 2013.

[26] L Breiman, JH Friedman, RA Olshen, and CJ Stone. Classification and regression trees. monterey, calif., usa: Wadsworth, 1984.

[27] Luis Torgo. Data mining with R: learning with case studies. CRC press, 2016.

[28] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. Machine learning: ECML-98, pages 137–142, 1998.

[29] Gregory W Corder and Dale I Foreman. Nonparametric statistics: A step-by-step approach. John Wiley & Sons, 2014.

[30] Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. Mining apps for abnormal usage of sensitive data. In Proceedings of the 37th International Conference on Software Engineering-Volume 1, pages 426–436. IEEE Press, 2015.

[31] Timothy Vidas, Nicolas Christin, and Lorrie Cranor. Curbing android permission creep. In Proceedings of the Web, volume 2, pages 91–96, 2011.

[32] Shengqian Yang, Dacong Yan, Haowei Wu, Yan Wang, and Atanas Rountev. Static control-flow analysis of user-driven callbacks in android applications. In Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, volume 1, pages 89–99. IEEE, 2015.

[33] Permissions usage notes. https://developer.android.com/training/permissions/usage-notes.html. Online; accessed 24-October-2017.

[34] Thomas D Cook, Donald Thomas Campbell, and Arles Day. Quasi-experimentation: Design & analysis issues for field settings, volume 351. Houghton Mifflin Boston, 1979.

[35] Inc Gartner. Gartner says free apps will account for nearly 90 percent of total mobile app store downloads in 2012, 2012. http://www.gartner.com/newsroom/id/2153215.

[36] Panagiotis Andriotis, Martina Angela Sasse, and Gianluca Stringhini. Permissions snapshots: Assessing users' adaptation to the android runtime permission model. In Information Forensics and Security (WIFS), 2016 IEEE International Workshop on, pages 1–6. IEEE, 2016.

[37] Panagiotis Andriotis, Shancang Li, Theodoros Spyridopoulos, and Gianluca Stringhini. A comparative study of android users? privacy preferences under the runtime permission model. In International Conference on Human Aspects of Information Security, Privacy, and Trust, pages 604–622. Springer, 2017.

[38] Necmiye Genc-Nayebi and Alain Abran. A systematic literature review: Opinion mining studies from mobile app store user reviews. Journal of Systems and Software, 125:207–219, 2017.

[39] Hammad Khalid. On identifying user complaints of ios apps. In Proceedings of the 2013 International Conference on Software Engineering, pages 1474–1476. IEEE Press, 2013.

[40] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. What do mobile app users complain about? IEEE Software, 32(3):70–77, 2015.

[41] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E Hassan. Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews. Empirical Software Engineering, 21(3):1067–1106, 2016.

[42] Luca Pascarella, Franz-Xaver Geiger, Fabio Palomba, Dario Di Nucci, Ivano Malavolta, and Alberto Bacchelli. Self-Reported Activities of Android Developers. In 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems, New York, NY, May 2018. ACM.

[43] Sara Motiee, Kirstie Hawkey, and Konstantin Beznosov. Do windows users follow the principle of least privilege?: investigating user account control practices. In Proceedings of the Sixth Symposium on Usable Privacy and Security, page 1. ACM, 2010.