Spam Mobile Apps: Characteristics, Detection, and in the Wild Analysis

SURANGA SENEVIRATNE, Data61, CSIRO
ARUNA SENEVIRATNE, Data61, CSIRO & University of New South Wales
MOHAMED ALI KAAFAR and ANIRBAN MAHANTI, Data61, CSIRO
PRASANT MOHAPATRA, Department of CS, University of California, Davis

The increased popularity of smartphones has attracted a large number of developers to offer various applications for the different smartphone platforms via the respective app markets. One consequence of this popularity is that the app markets are also becoming populated with spam apps. These spam apps reduce the users' quality of experience and increase the workload of app market operators to identify these apps and remove them. Spam apps can come in many forms such as apps not having a specific functionality, those having unrelated app descriptions or unrelated keywords, or similar apps being made available several times and across diverse categories. Market operators maintain antispam policies and apps are removed through continuous monitoring. Through a systematic crawl of a popular app market and by identifying apps that were removed over a period of time, we propose a method to detect spam apps solely using app metadata available at the time of publication. We first propose a methodology to manually label a sample of removed apps, according to a set of checkpoint heuristics that reveal the reasons behind removal. This analysis suggests that approximately 35% of the apps being removed are very likely to be spam apps. We then map the identified heuristics to several quantifiable features and show how distinguishing these features are for spam apps. We build an *Adaptive Boost* classifier for early identification of spam apps using only the metadata of the apps. Our classifier achieves an accuracy of over 95% with precision varying between 85% and 95% and recall varying between 38% and 98%. We further show that a limited number of features, in the range of 10–30, generated from app metadata is sufficient to achieve a satisfactory level of performance. On a set of 180,627 apps that were present at the app market during our crawl, our classifier predicts 2.7% of the apps as potential spam. Finally, we perform additional manual verification and show that human reviewers agree with 82% of our classifier predictions.

Categories and Subject Descriptors: H4 [**Information Systems Applications**]: Miscellaneous

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: Spam, mobile apps, android, spam apps

ACM Reference Format:
Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. 2017. Spam mobile apps: Characteristics, detection, and in the wild analysis. ACM Trans. Web 11, 1, Article 4 (March 2017), 29 pages.
DOI: http://dx.doi.org/10.1145/3007901

This submission is an extension of the conference paper, *"Early Detection of Spam Mobile Apps"* appeared in *Proceedings of the* 24th *WWW* (2015) [Seneviratne et al. 2015].

Authors' addresses: S. Seneviratne, A. Seneviratne, M. A. Kaafar, and A. Mahanti, 13, Garden Street, Eveleigh, NSW, Australia; emails: {suranga.seneviratne, aruna.seneviratne, dali.kaafar}@data61.csiro.au, anirban.mahanti@gmail.com; P. Mohapatra, Department of Computer Science, 2063 Kemper Hall, University of California, Davis, CA 95616; email: pmohapatra@ucdavis.edu.

1. INTRODUCTION

Recent years have seen the wide adoption of mobile apps, and the number apps that are being offered is growing exponentially. As of mid-2016, Google Play Store is reported to host approximately 2.2 million apps and Apple App Store hosts approximately 2 million apps [Statista, Inc. 2016]. During the first 5 months of 2016, Google Play Store added on average 50,000 new apps per month and the corresponding value for Apple App Store was 43,000 [AppBrain, Inc. 2016; PocketGamer.biz 2016]. These numbers are expected to grow significantly over the next few years [App Annie 2016].

Both Google and Apple have policies governing the publication (offering) of apps through their app markets. Google has an explicit spam policy [Google 2016c] that describes the criteria for classifying an application as spam. These include (i) apps that were automatically generated and as a result do not have any specific functionality or a meaningful description; (ii) multiple instances of the same app being published to obtain increased visibility in the app market; and (iii) apps that make excessive use of unrelated keywords to attract users through unintended searches. Overall, spam apps vitiate the app market experience and its usefulness as these apps will occupy search results, app recommendations, and top charts. Moreover, such apps might mislead users to install (or buy) apps they did not intend to install. Also, spam apps add additional work to the app market operators to remove those apps from the app markets.

At present, Google and Apple take different approaches to detecting spam applications. Google's approach is reactive. The app approval process does not check an app against Google's explicit spam app policy [Oberheide and Miller 2012], and takes action only on the basis of customer complaints [Perez 2013a]. Such crowdsourced approach can lead to a considerable time lag between app submission and detection of spam apps. As a result, by the time an app is taken down from the app market some damage might have already happened, such as some users installing the app and spam apps occupying search results and top charts.

In contrast, Apple scrutinises the apps submitted for approval to determine whether the submitted app conforms to Apple's policies. The process used for detection is unknown. Although this approach is likely to detect spam apps before they are published, it lengthens the app approval process. Nonetheless, recent market reports indicate that spam apps also make it into Apple App Store as well [Tecno Buffalo 2016; Farooqui 2016; Perez 2016]. Thus, with the ever increasing number of apps being submitted daily for approval, the app market operators need to be able to detect spam apps quickly and accurately [Perez 2013b].

In this article, we propose a methodology for spam app detection and classification at the time of app submission without the need for any human intervention such as manual inspection of the metadata or manual testing of the app. The proposed scheme utilises only features that can be derived from an app's metadata available during the publication approval process. We validate the app classifier, by applying it to a large dataset of apps collected between December 2013 and May 2014, by crawling and identifying apps that were removed from Google Play Store. We make the following contributions:

- We develop a manual app classification methodology based on a set of heuristic checkpoints that can be used to identify reasons behind an app's removal (Sections 2 and 3). Using this methodology shows that approximately 35% of the apps that were removed are spam apps; problematic content and counterfeits were the other key reasons for removal of apps.
- We present a mapping of our proposed spam checkpoints to one or more quantifiable features that can be used to train a learning model (Section 4). We provide a characterisation of these features highlighting the differences between spam and nonspam apps and indicate which features are more discriminative.

—We build an *Adaptive Boost* classifier for the detection of spam apps and show that our classifier can achieve an accuracy over 95% at a precision between 85% and 95% and a recall between 38% and 98% on the testing set. We perform further manual verification and show that 82% of our classifier predictions were agreed upon by human reviewers and apps that were predicted as spam by our classifier had 20% higher likelihood of being removed from Google Play Store (Section 5).
—We provide a feature engineering analysis and show that a limited number of features in the range of 10–30 is sufficient to a reach a satisfactory level of performance with respect to precision and recall (Section 5).
—We apply our classifier to over 180,000 apps available in Google Play Store and show that approximately 2.7% of them are potentially spam apps (Section 5).

This article extends our previous work [Seneviratne et al. 2015], that proposed a classifier capable of detecting spam mobile apps solely based on features derived from the app metadata available during the time of publication. In this extension, we complement our previous study by investigating the relative importance of features using several feature selection algorithms. We also provide additional analysis on the performance of the classifier. More specifically, we first revisit the pages of the apps that were predicted as spam and check whether they have been removed from Google Play Store. Second, we manually label a subset of apps for which we have the classifier's predicted label to check whether the human reviewers agree with the predictions.

2. METHODOLOGY

2.1. Dataset

We used periodic crawls of the Google Play Store, to develop a dataset as described next. The crawls were done using a Java client that uses *jsoup*[1] HTML parser to determine (i) functionally similar apps and (ii) other apps by the same developer. Then we recorded app metadata such as the name, description, and category for all the apps we discovered during each crawl. We then discarded the apps with a non-English description using the language detection Application Programming Interface (API), "*Detect Language*."[2] We refer to this final set as the ***Observed Set*** - $\mathbb{O}$.

We use apps collected in a previous study [Seneviratne et al. 2014b], as a seed for collecting a new dataset. This initial seed contained 94,782 apps and was curated from the lists of apps obtained from approximately 10,000 smartphone users. The user base consisted of volunteers, Amazon mechanical turk users, and users who published their lists of apps in *social app discovery* sites such as *Appbrain*.[3]

After identifying the ***Observed Set*** apps, we revisited Google Play Store to check the availability of each app. The subset of apps that were unavailable at the time of this second crawl is referred to as ***Crawl 1*** - $\mathbb{C}_1$. This process was repeated two times, ***Crawl 2*** - $\mathbb{C}_2$ and ***Crawl 3*** - $\mathbb{C}_3$ with a 1 month gap between two consecutive crawls. Figure 1 illustrates the data collection process and Table I summarises the datasets in use.

We identify temporary versus long-term removal of apps by rechecking the status of apps deemed to have been removed during an earlier crawl. For instance, all apps in ***Crawl 1*** were checked again during ***Crawl 2***. We found that only 85 (~0.13%) apps identified as removed in ***Crawl 1*** reappeared in ***Crawl 2***. Similarly, only 153 apps (~0.02%) identified as removed in ***Crawl 2*** reappeared in ***Crawl 3***. These apps were not included in our analysis.

[1]www.jsoup.org.
[2]http://detectlanguage.com.
[3]www.appbrain.com.

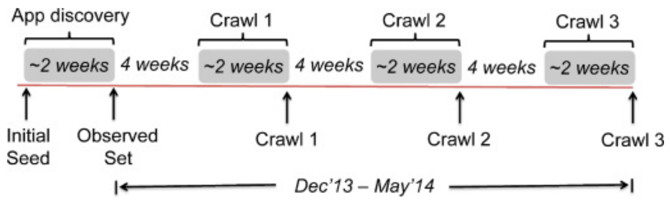


Fig. 1. Chronology of the data collection.

Table I. Summary of the Dataset

Set	Number of apps
Observed set ($\mathbb{O}$)	232,906
Crawl 1 ($\mathbb{C}_1$)	6,566
Crawl 2 ($\mathbb{C}_2$)	9,184
Crawl 3 ($\mathbb{C}_3$)	18,897

Table II. Key Reasons for Removal of Apps

Reason	Description
Spam	These apps often have characteristics such as unrelated descriptions, keyword misuse, and multiple instances of the same app. Section 3 presents details on spam app characteristics.
Unofficial content	Apps that provide unofficial interfaces to popular websites or services (e.g., an app providing an interface to a popular online shopping site without any official affiliation).
Copyrighted content	Apps illegally distributing copyrighted content.
Adult content	Apps with explicit sexual content.
Problematic content	Apps with illegal or problematic content (e.g., hate speech and drug related).
Android counterfeit	Apps pretending to be another popular app in Google Play Store.
Other counterfeit	A counterfeit app, for which the original app comes from a different source than Google Play Store (e.g., Apple App Store).
Developer deleted	Apps that were removed by the developer.
Developer banned	Developer's other apps were removed due to various reasons and Google decides to ban the developer. Thus, all of his apps get removed.

2.2. App Labelling Process

For a subset of the removed apps in our dataset, our goal was to manually identify the reasons behind their removal.

We identified factors that led to the removal of apps from the market place by consulting numerous market reports [Perez 2013b; Apple 2014] as well as by examining the policies of the major app markets [Google 2014, 2016a, 2016b, 2016c; Apple 2016]. We identified nine key ***reasons***, which are summarised in Table II.

For each of these reasons, we formulated a set of heuristic checkpoints that can be used to manually label whether or not an app is likely to be removed by observing only the metadata of apps. The checkpoints do not require one to install the apps in a smartphone or a virtual device. As an example, we show the checkpoints used to label ***copyrighted content*** apps in Table III and Section 3 delves into the checkpoints developed for identifying spam apps. A full list of checkpoints for each reason can be found in our technical report [Seneviratne et al. 2014a].

From ***Crawl 1***, we took a random sample of 1,500 apps and asked three independent reviewers to identify the ***highest likely*** reason behind the removal of each app using the heuristic checkpoints that we developed as a guideline. A reviewer's selection of a reason for app removal is subjective and it is based upon their judgement as to whether one or more checkpoints conditions have been satisfied. If a reviewer could

Table III. Checklist used for Manual Labelling of Copyrighted Content Apps

Attribute	ID	Description and Examples
Description	R_1	Does the app description indicate potential copyright infringement or a trademark infringement (unauthorized use of registered trademarks)? ***For example, WP8 Surface Wallpaper HD - App description indicates potential copyright infringement.*** *".... Windows Phone 8 & Surface Tablet Wallpaper HD Official Original From Microsoft Device. ...."* ***For example, GHOST IN THE SHELL SAC+2ndGIG - App description indicates potential copyright infringement.*** *".... This app includes all episodes of GHOST IN THE SHELL SAC VIDEO Please ENJOY!!!. ...."*
	R_2	Does the app provide copyrighted material at a price? ***For example, Dungeon Keeper 2 Soundboard - According to app description it provides sounds from a PC game and the app is priced at AU$7.44. The developer has no affiliation to original content owner.*** *"A lot of sounds from the old but good Dungeon Keeper 2 PC game"* ***For example, Pin Search For BBM - According to app description it provides contents from a movie and a book and the app is priced at AU$1.49. The developer has no affiliation to original content owner.*** *".... MOVIE AND BOOK BY CHUCK PALAHNIUK, THIS FIGHT CLUB SOUNDBOARD BRINGS YOU YOUR FAVORITE QUOTES FROM THE MOVIE RIGHT TO YOUR PHONE WITH JUST THE TOUCH OF A FINGER! ...."*
	R_3	Does the app description contain disclaimers related to copyrights? ***For example, Love Stories - App description contains a disclaimer on potential copyright infrigements.*** *".... Smart Applications does not support plagiarism in any form. Author must note that the story should not be copied from anywhere; it should be an original piece of work. If found otherwise, we reserve the right to delete it. ...."*
Reviews	R_4	Do the users mention about copyright infringements? ***For example, Astro Soldier War Of The Stars - Users mention about copyright infringements.*** *".... You stole copyrighted material from TimeGate Studios. ....", "..... Your so icon is artwork from Section 8, one of their video games ....."* ***For eample, Harlem Shake Creator Pro - Users mention about copyright infringements.*** *"..... Want my money back!! Payed for this app so that my kids could save and share their video, then Facebook removed it on account of copyright infringement!! ....."*

not conclusively determine the reasons behind a removal, she labelled those apps as ***unknown***.

The reviewers were Android app developers and worked full time for 1.5 months at NICTA (now known as Data61) for this task. The manual labelling process took approximately 20 working days (7 hours per day).

2.3. Agreement Among the Reviewers

We used majority voting to merge the results of the expert's assessments, to arrive at the reason behind the app removal in our labelled dataset ($\mathbb{L}$). We decided not to crowdsource the labelling task to avoid issues with training and expertise.

Table IV summarises reviewer assessments. For approximately 40% (601 out of 1,500) of labelled apps, the three reviewers reached a consensus on the reason for removal and for 90% (1,350 out of 1,500) of the apps the majority of the reviewers agreed the reason for removal was the same. For the remaining 10% of apps, reviewers disagreed about the reasons. We calculated the Fleiss' Kappa coefficient to quantify the inter-rater agreement and the value was 0.455.

Table IV. Reviewer Agreement in Labelling Reason for Removal

Reason	**3 reviewers agreed**	**2 reviewers agreed**	**Total**	**Percentage (%)**
Spam	292	259	551	36.73%
Unofficial content	65	127	192	12.80%
Developer deleted	68	56	124	8.27%
Android counterfeit	27	61	88	5.87%
Developer banned	24	54	78	5.20%
Copyrighted content	2	34	36	2.40%
Other counterfeit	11	23	34	2.27%
Adult content	8	4	12	0.80%
Problematic content	3	4	7	0.47%
Unknown	101	127	228	15.20%
Subtotal	**601**	**749**	**1350**	90.00%
Reviewer disagreement	NA	NA	150	10.00%
Total Labelled	**NA**	**NA**	**1500**	100.00%

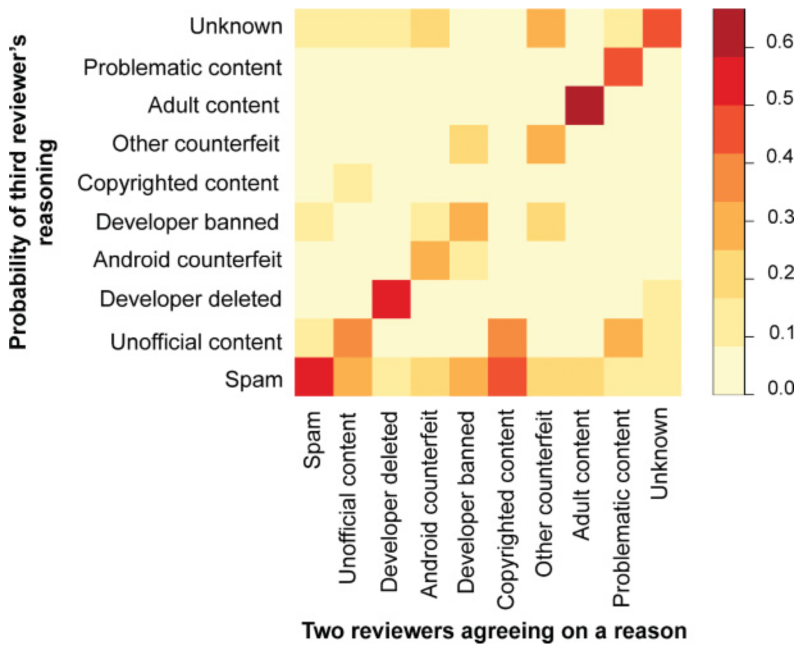


Fig. 2. Probability of a third reviewer's judgement when two reviewers already agreed on a reason.

Table IV also shows the composition of labelled dataset ($\mathbb{L}$) after majority voting-based label merging. We observe that *spam* is the main reason for app removal, accounting for approximately 37% of the removals, followed by *unofficial content* accounting for approximately 13% of the removals. Around 15% of the apps were labelled as *unknown*.

Figure 2 shows the conditional probability of the third reviewer's reasoning, given that the other two reviewers are in agreement. There is over 50% probability of the third reviewer's judgement of an app being spam, when two reviewers had already judged the app to be spam. Other reasons showing such high probability are *developer deleted* and *adult content* apps.

Our analysis through the manual labelling process shows that the main reason behind app removal is them being *spam* apps. Furthermore, the reviewer agreement was high (more than 50%) when manually labelling *spam* apps indicating spam apps stand out clearly when looking at removed apps.

We have released the labelled dataset to the research community to facilitate further research in spam app detection.[4]

3. MANUAL LABELLING OF SPAM APPS

This section introduces our heuristic checkpoints, which are used to manually label the spam apps. We also provide a characterisation of the reviewer agreement related to nine manual spam checkpoints and show that checkpoints are unambiguous and suitable for manual labelling. Section 4 maps the defined checkpoints into automated features.

3.1. Spam Checkpoints

Table V presents the nine heuristic checkpoints used by the reviewers and those were derived based on Google's spam policy [Google 2016c]. While we are unaware of Apple's spam policy, we note that Apple's app review guideline [Apple 2016] includes certain provisions that match our spam checkpoints. For example, term 4.3 in the Apple app review guideline, *"Don't create multiple Bundle IDs of the same app / Spamming the store may lead to your removal from the Developer Program."* maps to our checkpoint S_6. Similarly, term 2.3.7 *"Don't try to pack any of your metadata with trademarked terms, popular app names, or other irrelevant phrases just to game the system"* is similar to our checkpoint S_2.

Reviewers were asked not to deem the apps as spam when they simply observe an app satisfying a single checkpoint, but to consider making their decision based on matches with multiple checkpoints according to their domain expertise.

Note that Table V includes all checkpoints relevant to spam, including those that are considered only for manual labelling and not for automated labelling. In particular, checkpoints S_8 and S_9 are only used for manual labelling as features corresponding to these checkpoints are either not available at the time of app submission or require significant dynamic analysis for feature discovery. For instance, we do not use user "reviews" of the app (cf. checkpoint S_8), as user reviews are not available prior to the app's approval. Similarly, checking for excessive advertising (cf. checkpoint S_9) was also used only for manual labelling, as it requires dynamic analysis by executing it in a controlled environment.

3.2. Reviewer Agreement on Spam Checkpoints

Table VI shows how often the reviewers agreed upon each checkpoint. Checkpoints S_2 and S_6 led to the highest number of 3-reviewers agreements and 2-reviewer agreements. Checkpoints S_1 and S_3 have a moderate number of 3-reviewer agreements while having a high number of 2-reviewer agreements. The table also suggests that checkpoints S_1, S_2, S_3, and S_6 are the most dominant checkpoints.

The sum of the cases where three reviewers agreed, two reviewers agreed, and reviewers disagreed shown in Table VI, is more than the total number of spam apps identified by merging the reviewer agreement (i.e., 551 apps) because one reviewer might mark an app as satisfying multiple checkpoints. For example, consider a scenario where reviewer-1 labels the app as satisfying checkpoints S_1, S_2 and reviewer-2 labels as S_1, S_2 and reviewer-3 labels only as S_1. This will cause one scenario of 3-reviewer agreement (for S_1) and another scenario of 2-reviewer agreement (for S_2).

Out of the 551 spam apps, three reviewers agreed on at least one checkpoint for 210 apps (∼38%), and two reviewers agreed on at least one checkpoint for 296 apps

[4]http://www.privmetrics.org/publications/www15.

Table V. Checkpoints used for Labelling of Spam Apps

Attribute	ID	Description and Examples
Description	S_1	Does the app description describe the app function clearly and concisely? ***For example, Signature Capture App (Nonspam) - Description is clear on the functionality.*** *"SignIt app allows a user to sign and take notes which can be saved and shared instantly in email."* ***For example, Manchester (Spam) - Description contains details about Manchester United Football Club without any details on the functionality of the app.*** *"Manchester United Football Club is an English professional football club, based in Old Trafford, Greater Manchester, that plays in the Premier League. ..... In 1998-99, the club won a continental treble of the Premier League, the FA Cup and the UEFA Champions League, an unprecedented feat ....."*
	S_2	Does the app description contain too much detail/incoherent/unrelated text for an app description? ***For example, SpeedMoto (Nonspam) - Description is clear and concise about the functionality and usage.*** *"SpeedMoto is a 3d moto racing game with simple control and excellent graphic effect. Just swipe your phone to control moto direction. Tap the screen to accelerate the moto. In this game you can ride the motorcycle shuttle in outskirts, forest, snow mountain, bridge."* ***For example, Ferrari Wallpapers HD (Spam) - Description starts mentioning app as a wallpaper. However, then it goes into detail about Ferrari.*** *"*HD WALLPAPERS *EASY TO SAVE *EASY TO SET WALLPAPER *INCLUDES ADS FOR ESTABLISHING THIS APP FREE TO YOU THANKS FOR UNDERSTANDING AND DOWNLOADING =) Ferrari S.p.A. is an Italian sports car manufacturer based in Maranello, Italy. Founded by Enzo Ferrari in 1929, as Scuderia Ferrari, the company sponsored drivers and manufactured ....."*
	S_3	Does the app description contain a noticeable repetition of words or keywords? ***For example, English Chinese Dictionary (Nonspam) - Keywords do not have excessive repetition.*** *"Keywords: ec, dict, translator, learn, translate, lesson, course, grammar, phrases, vocabulary"* ***For example, Best live HD TV no ads (Spam) - Excessive repetition of words.*** *"Keywords: live tv for free mobile tv tuner tv mobile android tv on line windows mobile tv verizon mobile tv tv streaming live tv for mobile phone mobile tv stream mobile tv phone mobile phone tv rogers mobile tv live mobile tv channels sony mobile tv free download mobile tv dstv mobile tv mobile tv....."*
	S_4	Does the app description contain unrelated keywords or references? ***For example, FD Call Assistant Free (Nonspam) - All the keywords are related to the fire department.*** *"Keywords: firefighter, fire department, emergency, police, ems, mapping, dispatch, 911"* ***For example, Diamond Eggs (Spam) - Reference to popular games Bejeweled Blitz and Diamond Blast without any reason.*** *"Keywords : bejeweled, bejeweled blitz, gems twist, enjoy games, brain games, diamond, diamond blast, diamond cash, diamond gems, Eggs, jewels, jewels star"*
	S_5	Does the app description contain excessive references to other applications from the same developer? ***For example, Kids Puzzles (Nonspam) - Description does not contain references to developer's other apps.*** *"This kids game has 55 puzzles. Easy to build puzzles. Shapes Animals Nature and more... With sound telling your child what the puzzle is. Will be adding new puzzles very soon."* ***For example, Diamond Snaker (Spam) - Excessive references to developer's other applications.*** *"If you like it, you can try our other apps (Money Exchange, Color Blocks, Chinese Chess Puzzel ....."*
	S_6	Does the developer have multiple apps with approximately the same description? ***The developer "Universal App" has 16 apps having the following description, with each time XXXX term is replaced with some other term.*** *"Get XXXX live wallpaper on your devices! Download the free XXXX live wallpaper featuring amazing animation. Now with "Water Droplet", "Photo Cube", "3D Photo Gallery" effect! Touch or tap ..... To Use: Home -> Menu -> Wallpaper -> Live Wallpaper -> XXXX 3D Live Wallpaper. To develop more free great live wallpapers, we have implemented some ads in settings. Advertisement can support our develop more free great live wallpapers... "*
Identifier	S_7	Does the app identifier make sense and have some relevance to the functionality of the application or does it look like it is autogenerated? ***For example, Angry Birds & Candy Crush Saga (Nonspam) - Identifiers give an idea about the app.*** *"com.rovio.angrybirds", "com.king.candycrushsaga"* ***For example, Game of Thrones FREE Fan App & How To Draw Graffiti (Spam) - Identifiers appear to be autogenerated.*** *"com.a121828451851a959009786c1a.a10023846a", "com.a13106102865265262e503a24a.a13796080a"*
Reviews	S_8	Do users complain about the app being spammy in reviews? ***For example, Yoga Trainer & History Cleaner (Spam) - Users complain about app being spammy.*** *"Junk spam app Avoid", "More like a spam trojan! Download if you like, but this is straight garbage!!"*
Adware	S_9	Do the online APK checking tools highlight apps having excessive advertising? ***For example, Biceps & Triceps Workouts*** *"AVG threat labs" [AVG 2014] gives a warning about inclusion of adware.*

Table VI. Checkpoint-wise Reviewer Agreement for Spam

Checkpoint	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**
Three reviewers agreed	22	63	20	0	4	89	11	3	3
Two reviewers agreed	52	81	75	3	6	115	11	26	15
Disagreed	45								

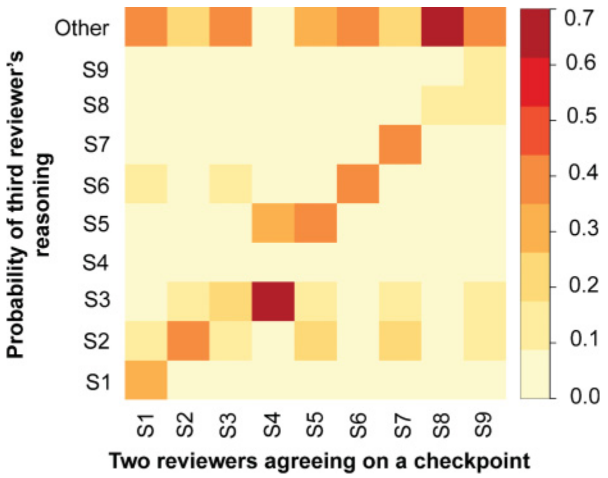


Fig. 3. Probability of a third reviewer's judgement when two reviewers already agreed on a checkpoint.

(∼54%). For 45 apps (∼8%), the three reviewers gave different checkpoints (i.e., when the reviewers assess an app as spam, over 90% of the time they also agreed on at least one checkpoint).

Figure 3 illustrates the conditional probability of the third reviewer's checkpoint selection given two reviewers have already agreed on a checkpoint. We observe that for checkpoints S_1, S_2, S_5, S_6, S_7, and S_8, there is a high probability that the third reviewer indicates the same checkpoint when two of the reviewers already agreed on a checkpoint. There is, however, a noticeable anomaly for checkpoint S_4, for which it seems that reviewers are getting confused with S_3. This is due to the lower frequency of occurrences of checkpoint S_4. There were no 3-reviewer agreements for S_4 and there were only three cases where two reviewers agreed as shown in Table VI. In those three cases the other reviewer marked the checkpoint S_3 in two cases giving a high probability (∼67%) for S_4 getting confused with S_3. Note that *Other* refers to all the checkpoints associated with reasons for app removals other than spam; the probability of *Others* being chosen is high because of the accumulated sum of probabilities of checkpoints associated with other reasons.

4. FEATURE MAPPING

Checkpoint heuristics for spam (cf. Table V) need to be mapped into features that can be extracted and used for automatic spam detection during the app approval process. This section presents this mapping and a characterisation of these features with respect to ***spam*** and ***nonspam*** apps.

An app is considered ***spam*** if two out of three reviewers labelled it as spam. We assumed top apps found by ranking apps with respect to the number of downloads, number of user reviews, are quite likely to be ***nonspam***. For example, according to this

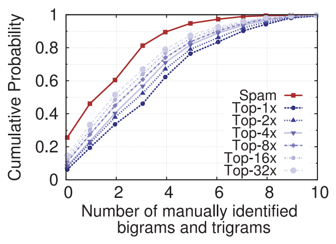


Fig. 4. Frequency of manually identified word n-grams.

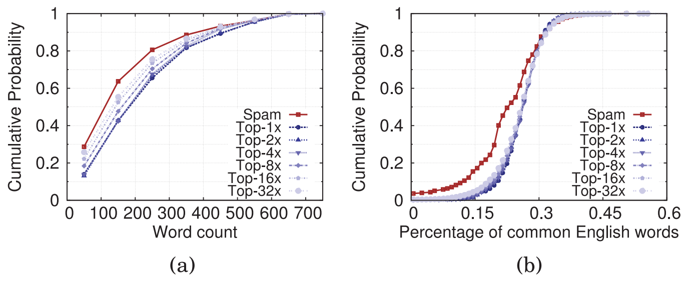


Fig. 5. Example features for **Checkpoint** S_2.

ranking, the number one ranked app was *Facebook* and number 100 was *Evernote*,[5] number 500 was *Tictoc*,[6] and number 1,000 was *Saavn*,[7] which are all very popular and legitimate apps. Thus, we selected top k times the number of labelled spam apps (551) from the set $\mathbb{O}$, except all removed apps, after ranking them according to total number of downloads, total number of user reviews, and average rating as ***nonspam*** apps. We varied k logarithmically between 1 and 32, (i.e., 1x, 2x, 4x, . . . ,32x) to obtain six datasets of nonspam apps. At larger k values it is possible that spam apps are considered to be nonspam, as discussed later in this section.

As noted in Section 3.1, checkpoints S_8 and S_9 are not used to develop features since we intend to enable automated spam detection at the time of developer submission.

4.1. *Checkpoint S_1 - Does the App Description Describe the App Function Clearly and Concisely?*

We automate this heuristic by identifying word-bigrams and word-trigrams that are used to describe a functionality and are popular among either spam or nonspam apps.

First, we manually read the descriptions of the top-100 nonspam apps and identified 100 word bigrams and word trigrams that describe app functionality, such as *"this app,"* *"allows you,"* and *"app for."* Figure 4 shows the *Cumulative Distribution Function (CDF)* of the number of bigrams and trigrams observed in app descriptions in each dataset. There is a high probability of spam apps not having these features in their app descriptions. For example, 50% of the spam apps had more than one occurrence of these bigrams, whereas 80% of the top-1x apps had more than one of these bigrams and trigrams. The difference in distribution of these bigrams and trigrams between spam and nonspam apps decreases when more and more lower ranked apps are considered in the nonspam category. This finding is consistent across some of the features we discuss in subsequent sections and we believe this indicates lower ranked apps can potentially include spams that are as yet not identified by Google.

Second, as manual identification of bigrams and trigrams is not exhaustive, we identified the word-bigrams and word-trigrams that appear in at least 10% of the spam and 10% of the nonspam apps. We used each such word-bigram and word-trigram as features and the frequency of occurrence as the value of the feature. Figure 6 visualises top-50 bigrams (when ranked according to *information gain*); the size of the bigrams proportional to the logarithm of the *normalised information gain*. Note that bigrams such as *"live wallpaper," "wallpaper on,"* and *"some ads"* are mostly present in spam, whereas bigrams such as *"follow us," "to play,"* and *"on twitter"* are popular in nonspam.

[5]https://evernote.com.
[6]http://www.tictoc.net.
[7]http://www.saavn.com.

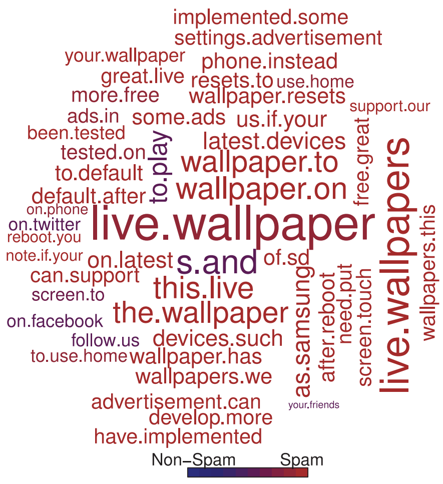


Fig. 6. Top-50 bigrams differentiating spam and top-1x.

Table VII. Features Associated with **Checkpoint** S_2

	Feature
1	Total number of characters in the description
2	Total number of words in the description
3	Total number of sentences in the description
4	Average word length
5	Average sentence length
6	Percentage of uppercase characters
7	Percentage of punctuations
8	Percentage of numeric characters
9	Percentage of nonalphabet characters
10	Percentage of common English words [Canales et al. 2011]
11	Percentage of personal pronouns [Canales et al. 2011]
12	Percentage of emotional words [Mukherjee and Liu 2010]
13	Percentage of misspelled words [Wikipedia 2014]
14	Percentage of words with alphabet and numeric characters
15	Automatic Readability (AR) index [Senter and Smith 1967]
16	Flesch Readability (FR) score [Flesch 1948]

4.2. *Checkpoint S_2 - Does the App Description Contain Too Much Detail, Incoherent Text, or Unrelated Text?*

We use writing style related "stylometry" features to map this checkpoint to a set of features anticipating spam and nonspam apps might have a different writing style. We used the 16 features listed in Table VII for this checkpoint. The feature list was complied by combining the features used by Canales et al. [2011] in authorship identification in online exams and features used by Mukherjee and Liu [2010] in gender classification in blog authors. Further details about the features can be found in our technical report [Seneviratne et al. 2014a].

For characterisation, we select features using the *greedy forward feature selection in wrapper method* [Kohavi and John 1997]. We use a decision tree classifier with maximum depth 10 and the feature subset was selected such that the classifier optimises the performance metric *asymmetric F-Measure*, $F_\beta = (1+\beta^2).\frac{precision.recall}{(\beta^2.precision)+recall}$, with $\beta = 0.5$. This metric was used since in spam app detection, precision is more important than recall [Soboroff et al. 2012]. This process identified the *total number of words*

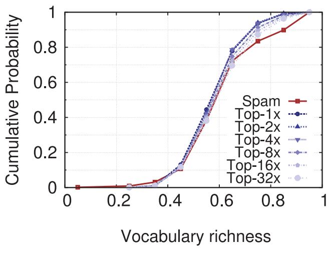


Fig. 7. Vocabulary richness.

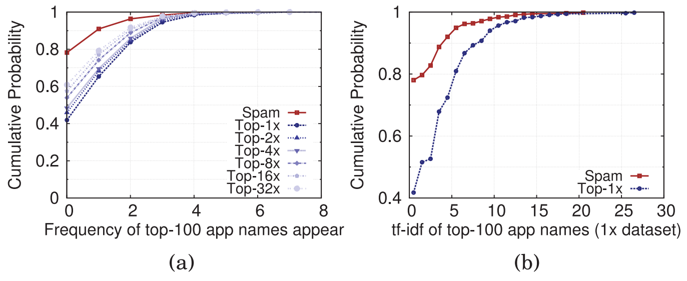


Fig. 8. Mentioning popular app names.

in the description, total number of sentences in the description, percentages of numeric characters, percentage of nonalphabet characters, and *percentage of common English words* as the most discriminative features.

Figure 5(a) shows the CDF of the total number of words in the app description. Spam apps tend to have less wordy app descriptions than nonspam apps. For instance, nearly 30% of the spam apps have less than 100 words, whereas approximately only 15% top-1x apps and top-2x apps have less than 100 words. As we inject more and more apps of lower popularity the difference diminishes. Figure 5(b) presents the CDF of the percentage of common English words [Canales et al. 2011] in the app description. It illustrates that spam apps typically use fewer common English words compared to nonspam apps.

4.3. *Checkpoint S_3 - Does the App Description Contain a Noticeable Repetition of Words or Keywords?*

We quantify this checkpoint using the *Vocabulary Richness* (VR)=$\frac{\text{Number of unique words}}{\text{Total number of words}}$ metric. Figure 7 shows the CDF of the VR measure. We expected spam apps to have low VR due to repetition of keywords. However, we observe this only in a limited number of cases. According to Figure 7, if VR is less than 0.3, an app is only marginally more likely to be spam. Perhaps the most surprising finding is that the apps with VR close to 1 are more likely to be spam. 10% of the spam apps had VR over 0.9 and none of the nonspam apps had such high VR values. When we manually checked these spam apps we found that they had terse app descriptions. We showed earlier that apps with a shorter description are more likely to be spam under ***checkpoint S_2***.

4.4. *Checkpoint S_4 - Does the App Description Contain Unrelated Keywords or References?*

Inserting unrelated keywords such that an app would appear in popular search results is a common spamming technique. Since the topics of the apps can vary significantly, it is difficult to find when the terms included in the description are actually related to the functionality. In this study, we focussed on a specific strategy the developers might use, that is, mentioning the names of the popular applications with the hope of appearing in the search results for these applications. For each app we calculated the number of mentions of the top-100 popular app names in the app's description.

Figure 8(a) shows the distribution of the number of times the name of top-100 apps appear in the description. Somewhat surprisingly, we found that only 20% of the spam apps had at least one mention of popular apps, whereas 40%–60% of the top-kx apps had more than a single mention of popular apps. On investigation, we find that many top-kx apps have social networking interfaces and fan pages and as a result they mention the names of popular social networking apps. This is evident in Figure 6 as well where *Twitter* or *Facebook* are mentioned mostly in nonspam apps. To summarise, the presence of social sharing features or the availability of fan pages can be used to identify nonspam apps.

Next, we consider the sum of the *tf-idf* weights of the top-100 app names. This feature attempts to reduce the effect of highly popular apps. For example, *Facebook* can have a high frequency because of its sharing capability and many apps might refer to it. However, if an app refers to another app that is not as popular as *Facebook*, such as reference to popular games, it indicates a likelihood of it being a spam app. To discount the effect of the commonly available app names, we used the sum of *tf-idf* weights as a feature. For a given dataset let tf_{ik} where $i = 1 : 100$ is the number of times the app name of i^{th} app (n_i) in top-100 apps appears in an app description of app k (a_k). Then we define IDF_i for i^{th} app in top-100 apps as $IDF_i = \frac{N}{log(1+|\{n_i \in a_k\}|)}$ $\forall k$. Then feature $tf-idf(k)$ for k^{th} app is $tf-idf(k) = \sum_{i=1}^{i=100} tf_{ik} \, . \, IDF_i$.

The preceding calculation is dataset dependent. Despite this, as Figure 8(b) shows, the CDF of top-1x dataset and the results still indicates that if popular app names are found in the app description, the app tends to be nonspam rather than spam. We repeated the same with top-1000 app names and found that the results were similar.

4.5. Checkpoint S_5 - Does the App Description Contain Excessive References to Other Applications from the Same Developer?

We use the *number of times a developer's other app names appear* as the feature corresponding to this checkpoint. However, none of the cases marked by the reviewers as matching checkpoint S_5 satisfied this feature because the description contained links to the applications rather than the app names and only 10 spam apps satisfied this feature (cf. Table VI). We do not use checkpoint S_5 in our classifier.

4.6. Checkpoint S_6 - Does the Developer Have Multiple Apps with Approximately the Same Description?

For this checkpoint, for each app we considered the following features: (i) the total number of other apps the developer has, (ii) the total number of apps with an English language description that can be used to measure descriptions similarity, and (iii) the number of other apps from the same developer having a description *cosine similarity (s)*, of over 60%, 70%, 80%, and 90%.

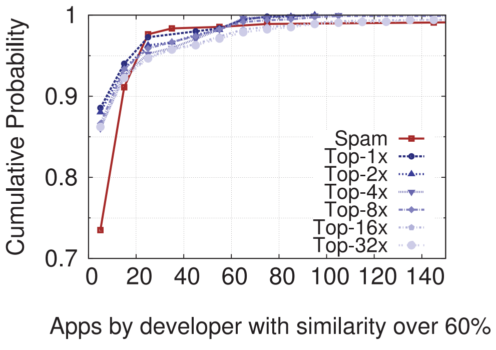
To calculate the *cosine similarity* we first preprocess the app description text by converting the characters to lowercase and removing punctuation symbols. Then we represent each document as a word frequency vector and calculate the cosine similarity between the two documents a and b as $Cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}.\mathbf{b}}{||\mathbf{a}|| \, ||\mathbf{b}||}$.

We observe that the features based on the similarity between app descriptions are the most discriminative. As examples, Figures 9(a) and 9(b), respectively, show the CDF of the number of apps with s over 60% and 90% by the same developer.

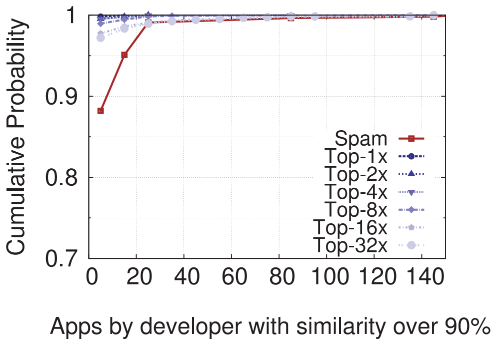
Figure 9(a) shows that only about 10%–15% of the nonspam apps have more than five other apps from the same developer with over 60% of description similarity. However, approximately 27% of the spam apps have more than five apps with over 60% of description similarity. This difference becomes more evident when the number of apps from the same developer with over 90% description similarity is considered, indicating that spam apps tend to have multiple clones with similar app descriptions.

4.7. Checkpoint S_7 - Does the App Identifier (Appid) Make Sense and have Some Relevance to the Functionality of the Application or Does It Appear to be Autogenerated?

Every Android app has an app identifier (*appid*), which is used to uniquely identify the app in Google Play Store. *Appid* follows the Java package naming convention



(a) Over 60% similarity

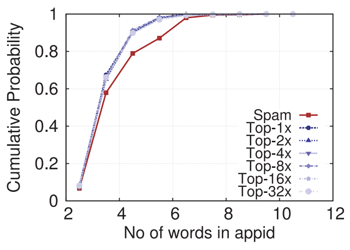


(b) Over 90% similarity

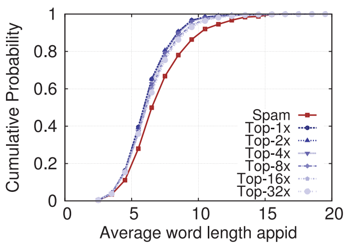
Fig. 9. Similarity with developer's other apps.

Table VIII. Features Associated with **Checkpoint** S_7

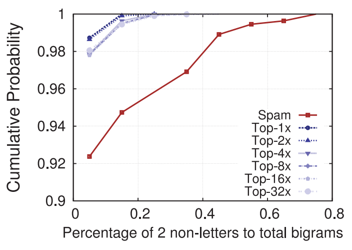
	Feature
1	Number of characters
2	Number of words
3	Average word length
4	Percentage of of non-letter characters to total characters
5	Percentage of upper case characters to total letter characters
6	Presence of parts of *app name* in *appid*
7	Percentage of bigrams with 1 non-letter to total bigrams
8	Percentage of bigrams with 2 non-letters to total bigrams
9	Percentage of bigrams with 1 or 2 non-letters to total bigrams
10	Percentage of trigrams with 1 non-letter to total trigrams
11	Percentage of trigrams with 2 non-letters to total trigrams
12	Percentage of trigrams with 3 non-letters to total trigrams
13	Percentage of trigrams with 1, 2 or 3 non-letters to total trigrams



(a)



(b)



(c)

Fig. 10. Example features associated with **Checkpoint** S_7.

[Oracle 2014] and differs from the app name that is visible to the users. For example, for the Facebook Android app, the app name is *Facebook*, whereas the *appid* is *com.facebook.katana*.

Table VIII shows the features derived from the *appid*. We applied the feature selection method noted in Section 4.2 to identify the most discriminative features (*number of words, average word length, percentage of bigrams with two nonletters*).

In Figure 10(a) we show the CDF of the number of words in the appid. The spam apps tend to have more words compared to nonspam apps. For example, 15% of the spam apps had more than five words in the appid, whereas only 5% of the nonspam had the same. Figure 10(b) shows the CDF of the average word length of a word in

Table IX. Features Associated with Other App Metadata

	Feature		Feature
1	App category	**5**	Developer's website available
2	Price	**6**	Developer's website reachable
3	Length of app name	**7**	Developer's email available
4	Size of the app (KB)	**8**	Privacy policy available

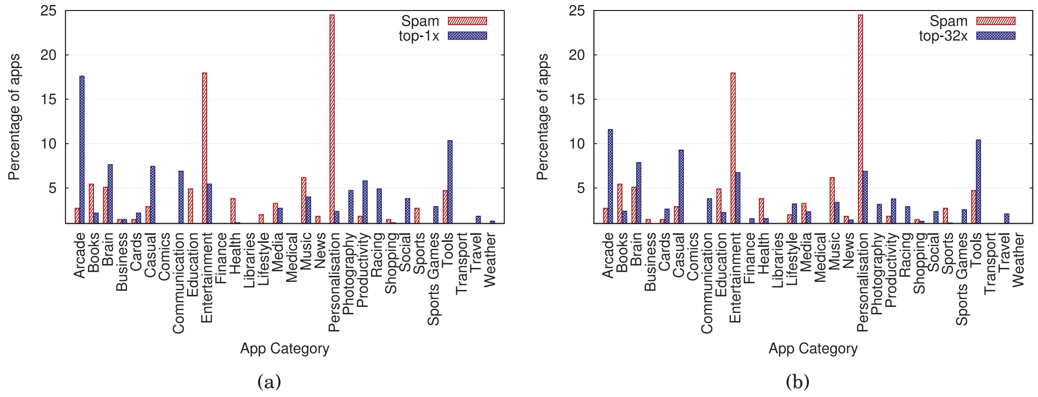


Fig. 11. App category.

appid. For 10% of the spam apps the average word length is higher than 10 and it was so only for 2%–3% of the nonspam apps.

Figure 10(c) shows the percentage of nonletter bigrams (e.g., *"01," "8"*) among all character bigrams that can be generated from the appid. None of the nonspam apps had more than 20% of nonletter bigrams in the appid, whereas about 5% of the spam apps had more than 20% of nonletter bigrams. Therefore, if an appid contains more than 20% of nonletter bigrams out of all possible bigrams that can be generated, that app is more likely to be spam than nonspam.

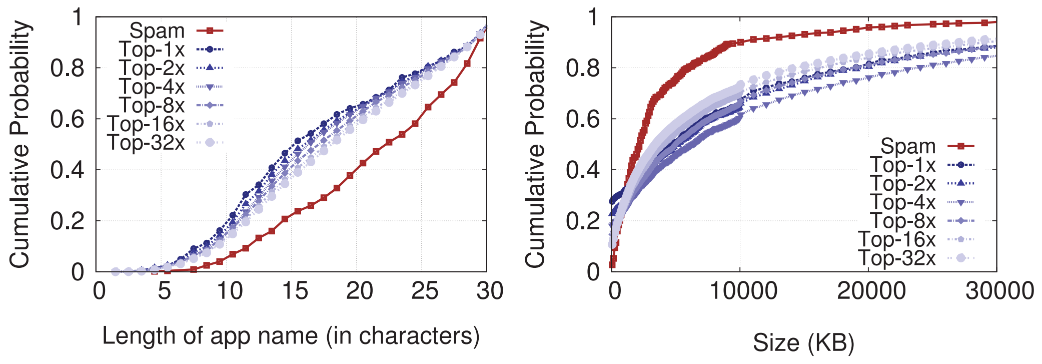
4.8. Other Metadata

In addition to the features derived from the checkpoints, we added the metadata related features listed in Table IX.

Figures 11(a) and 11(b) show category-wise app distribution of spam apps against top-1x and top-32x app categories. We note that approximately 42% of the spam apps belong to the categories *Personalisation* and *Entertainment*, whilst the corresponding values for top-1x and top-32x sets are approximately 8% and 14%, respectively. Manual inspection of spam apps in the *Personalisation* category showed that the majority are wallpaper and the spam apps in the *Entertainment* category are Youtube Playlists.

We found a negligibly small number of spam apps in the categories *Communication, Photography, Racing, Social, Travel,* and *Weather*. Moreover, for categories *Arcade, Tools,* and *Casual*, the percentage of spam is significantly less than the percentage of nonspam. Qualitatively similar observations hold for these other top-kx sets.

Figure 12(a) shows the CDF of the length of the app name. Spam apps tend to have longer names. For example, 40% of the spam apps had more than 25 characters in the app name. Only 20% of the nonspam apps had more than 25 characters in their app names. Figure 12(b) shows the CDF of the size of the app in kilobytes. Spam apps tend to have a high probability of having a size in some specific ranges. If the size of the app is very low, those apps are more likely to be nonspam. For example, 30% of the top-kx apps were less than 100KB in size and the corresponding percentage of spam apps



(a) (b)

Fig. 12. Features associated with other app metadata.

Table X. Availability of Developer's External Information

	Spam	Top 1x	Top 2x	Top 4x	Top 8x	Top 16x	Top 32x
Website available	57%	93%	94%	93%	91%	89%	86%
Website reachable	93%	98%	97%	97%	96%	96%	95%
Email available	99%	84%	89%	91%	93%	94%	95%
Privacy policy available	9%	56%	50%	48%	38%	32%	26%

is almost zero. Almost all the spam apps were having sizes less than 30MB, whereas 10%–15% of the top-kx apps were more than 30MB in size. CDF of the spam app shows a sharp rise between 1000KB and 3000KB indicating there are more apps in that size range.

Table X shows the external information related features. For example, if a link to a developer website or a privacy policy is given the app is more likely to be nonspam.

5. EVALUATION

5.1. Classifier Performance

We build a binary classifier based on the *Adaptive Boost* algorithm [Freund and Schapire 1996] to detect whether or not a given app is spam. The Adaptive Boost classifier was selected according to the performance results by testing the number of off-the-shelf classifiers. We use the spam apps as *positives* and apps from a top-kx set as *negatives*. Each app is represented by a vector of all the features listed in Section 4. The classifier is trained using 80% of the data and the remaining 20% of the data is used for testing. We repeat this experiment for $k = 1, 2, 4, \ldots, 32$.

Some of the features discussed in Section 4 depend on the other apps in the dataset rather than on the metadata of the individual apps. For such cases we calculate the features based only on the training set to avoid any information propagation from the training to the testing set that would artificially inflate the performance of the classifier. For example, when extracting the bigrams (and trigrams) that are popular in at least 10% of the apps, only spam and nonspam apps from the training set are used. This vocabulary is then used to generate feature values for individual apps in both training and testing sets.

Decision trees with a maximum depth of 5 are used as the weak classifiers in the *Adaptive Boost* classifier. We select decision trees as weak classifiers based on the observations made in multiple previous works that showed *Adaptive Boost* in combination with decision trees provides more accurate results [Freund and Schapire 1996; Quinlan 1996; Margineantu and Dietterich 1997]. Through experimenting, we select

Table XI. Classifier Performance

k	**Precision**	**Recall**	**Accuracy**	$F_{0.5}$
1	0.9310	0.9818	0.9545	0.9408
2	**0.9533**	**0.9273**	**0.9606**	**0.9480**
4	0.9126	0.8545	0.9545	0.9004
8	0.9405	0.7182	0.9636	0.8857
16	0.8833	0.4818	0.9658	0.7571
32	0.8571	0.3818	0.9793	0.6863

Table XII. Classifier Performance: $k = 2$ Model

k	**Precision**	**Recall**	**Accuracy**	$F_{0.5}$
4	0.8080	0.9182	0.9400	0.8279
8	0.5549	0.9182	0.9091	0.6026
16	0.2730	0.9182	0.8513	0.3176
32	0.1164	0.9182	0.7862	0.1410

500 as the number of iterations. Table XI summarises the results. Our classifiers, while varying the value of k, have precision over 85% with recall varying between 38% and 98%. Notably, when k is small (e.g., when the total number of nonspam apps represents $\leq$2x the number of spam apps) the classifier achieves up to 95% accuracy.

Recall drops when we increase the number of negative examples because a larger k value implies inclusion of lower ranked apps as negative (nonspam) examples. Some of these lower ranked apps, however, exhibit some spam features and some may indeed be spam (not yet been detected as spam). As we have only a small number of spam apps in the training set, when unlabelled spam apps are added as nonspam, some spam related features become nonspam features as the number of apps satisfying that feature is high in nonspam. As a result, recall drops when k increases. Another potential reason for this observation is the classifier's behaviour under *class imbalance* [Japkowicz and Stephen 2002]. Since the classifier tries to improve the overall accuracy of predictions, as the imbalance in classes increases (i.e., as k increases) the classifier might tend to mark more samples as nonspam.

When the number of negative examples increases, the classifier becomes more conservative and correctly identifies a relatively small portion of spam apps. Nonetheless, even at $k = 32$ we achieve a precision over 85%. This is particularly helpful in spam detection, as marking a nonspam as spam can be more expensive than missing a spam.

Additionally, if the objective is to build an aggressive classifier that identifies as much spam as possible so that app market operators can flag these apps to make a decision later, a classifier built using a smaller number of negative examples (i.e., $k = 1$ or $k = 2$) can be used. For example, in Table XII we show the $k = 2$ classifier's performance in the higher order datasets. As can be seen, this classifier identifies nearly 92% of the spam. However, the precision goes down as we traverse down the app ranking because of an increased number of false positives. Nonetheless, in reality some of these apps may actually be spam and as a result may not exactly be false positives.

5.2. Feature Significance Analysis

We checked whether a limited number of features are sufficient to achieve a satisfying level of performance. We first analyse the relative feature weights calculated based on the trained *Adaptive Boosting* models. Afterwards we used two commonly used feature selection methods, *forward feature selection* and *entropy based feature selection*, implemented in *FSelector* package[8] in *R* to identify a subset of features. For each feature

[8]https://cran.r-project.org/web/packages/FSelector/FSelector.pdf.

Table XIII. Significant Features According to the Average Relative Feature Importance Over All the Classifiers

	Feature	Feature Importance	Relative Feature Importance
1	Total number of words in the description	0.0376	1.0000
2	Total number of characters in the description	0.0371	0.9884
3	Percentage of nonalphabet characters	0.0354	0.9448
4	Total number of sentences in the description	0.0354	0.9431
5	Automatic readability index	0.0352	0.9374
6	Average sentence length	0.0351	0.9346
7	Average word length	0.0345	0.9196
8	Percentage of punctuations	0.0342	0.9113
9	Flesch score	0.0341	0.9072
10	Vocabulary richness	0.0337	0.8967

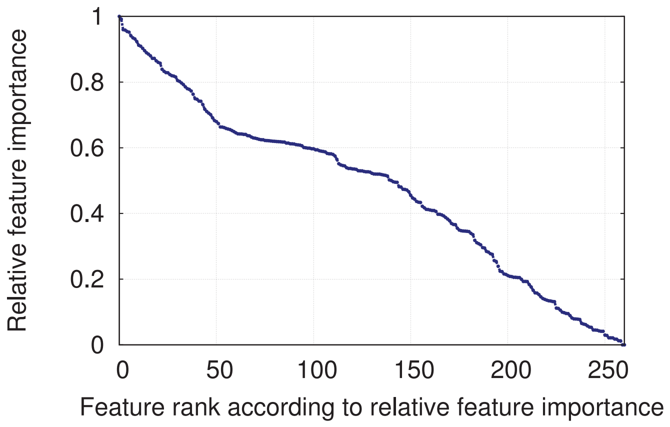


Fig. 13. Values of relative feature importance for all the features.

selection scenario we evaluated the performance of the classifier using only the subset of features.

Adaboost Feature Weights: We calculated the *feature importance* based on the trained *Adaptive Boosting* models. *Feature importance* for a single decision tree as defined by Breiman et al. [1984], is a measure based on the number of times a feature appears in an internal tree node weighted by the squared improvement to the model by introducing the node. For additive tree models such as *Adaptive Boosting* the *feature importance measure* is generalised by averaging over all the trees in the model and it is usually reported as a relative value by normalising to the range between zero and one [Friedman et al. 2001]. We used the *ada* package[9] in *R* to calculate the *relative feature importance* values.

As we have six classifiers (one for each *kx* dataset), to aggregate the *relative feature importance*, we ranked the features according to the average of the *relative feature importance* across all the classifiers. Table XIII shows the top-10 features and their average *relative feature importance* values. All the features in the top-10 are from the checkpoint S_6. Figure 13 further shows the *relative feature importance* values for all the features when they are ranked according to the decreasing order of the same.

[9]https://cran.r-project.org/web/packages/ada/ada.pdf.

Table XIV. Classifier Performance with Selected Features

k	No. of features	Precision	Recall	Accuracy	$F_{0.5}$
1	9	0.9099	0.9182	0.9136	0.9116
2	8	0.9208	0.8455	0.9242	0.9047
4	10	0.8866	0.7818	0.9364	0.8635
8	13	0.8592	0.5546	0.9404	0.7741
16	3	0.8333	0.1364	0.9476	0.4121
32	2	0.7895	0.1364	0.9727	0.4032

Forward Feature Selection: We then used the same greedy forward feature selection method mentioned in Section 4 for each k value. The difference here is that we try to identify most significant features among all the features corresponding to manual heuristics, whereas in Section 4 we identified significant features corresponding to each manual checkpoint separately.

In Table XIV we show the number of features identified for each k value and the performance of the classifier only using these features. The results show that a limited number of features can yield performance close to the use of all features for some k values. For example, for $k = 1,\ 2,$ and 4, use of 8 to 10 features results in a $F_{0.5}$ performance loss of only 3%, approximately. Moreover, when $k = 16$ and $k = 32$, three and two features alone achieve a precision of 78% and a recall of 13%. Those features were *Percentage of bigrams with two nonletters to total bigrams in appid, Bigram "wallpaper to," Bigram "screen to,"* and *Bigram "live wallpapers."*

Table XV lists all the features that were identified for all the k values. Altogether there were 29 distinct features. In Table XVI we show the performance of the classifier when only these 29 features were used. The results are close to the use of all the features and differs only by 8% in $F_{0.5}$ at maximum. This suggests that from all the features considered these are the features that mostly influence the performance of the classifier.

Entropy Based Feature Selection: We used *information gain* to rank all the features. The top-10 features with highest information gain are shown in Table XVII. Then, we progressively trained and tested the classifiers by adding features according to their values of information gain. Figures 14(a) and 14(b) show how precision and recall increases with the progressive addition of features. As can be seen from Figure 14(a), for all k values apart from 32, precision reaches approximately its maximum value when the number of selected features is close to 25. Similar observation holds for recall as illustrated in Figure 14(b) for k values up to 8.

5.3. Applying the Classifier in the Wild

To estimate the volume of potential spam that might be in Google Play Store, we used two of our classifiers to predict whether or not an app in our dataset was spam. We selected a conservative classifier ($k = 32$) and an aggressive classifier ($k = 2$) to obtain a lower and upper bound.

We did this prediction only for the apps that were not used for training during the classifier evaluation phase to avoid classifier artificially reporting higher number of spam. Table XVIII shows the results. $\mathbb{C}_1$, $\mathbb{C}_2$, and $\mathbb{C}_3$ are three sets of removed apps we identified as described in Section 2.1 and *Others* are the apps in $\mathbb{O}$ that were neither removed nor belong up to top-32x. Thus, *Others* apps represent the average apps in Google Play Store and we know that they were there in the market for at least 6 months, and by the time we stopped monitoring they were still there in the market.

According to the results, the more aggressive classifier ($k = 2$) predicted around 70% of the removed apps and 55% of the other apps to be spam. The conservative classifier

Table XV. Significant Features According to Forward Feature Selection

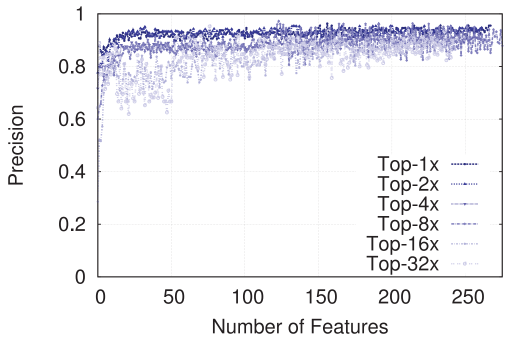
	Feature
1	Frequency of manually identified word-bigrams and word-trigrams
2	Bigram “the screen”
3	Bigram “wallpaper to”
4	Bigram “will be”
5	Bigram “you want”
6	Bigram “screen to”
7	Bigram “live wallpapers”
8	Bigram “app is”
9	Bigram “as samsung”
10	Total number of characters in the description
11	Total number of words in the description
12	Percentage of common English words
13	Percentage of emotional words
14	Percentage of nonalphabet characters
15	Percentages of numeric characters
16	Percentage of punctuations
17	Percentage of trigrams with one, two, or three nonletters to total trigrams in app id
18	Percentage of bigrams with two nonletters to total bigrams in app id
19	Mentioning popular app names top-100 TF-IDF
20	Mentioning popular app names top-1000 TF-IDF
21	Similarity with developer’s other apps (over 60%)
22	Similarity with developer’s other apps (over 70%)
23	Price
24	Size
25	Category
26	Length of the app name
27	Developer’s website available
28	Developer’s website reachable
29	Developer’s email address available

Table XVI. Classifier Performance With 29 Selected Features

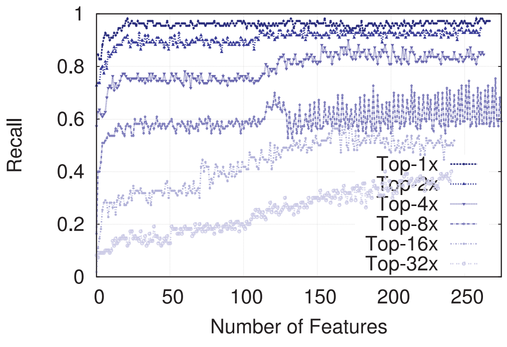
k	**Precision**	**Recall**	**Accuracy**	$F_{0.5}$
1	0.8983	0.9636	0.9273	0.9107
2	0.9143	0.8727	0.9303	0.9057
4	0.8800	0.8000	0.9382	0.8627
8	0.8816	0.6091	0.9475	0.8092
16	0.8393	0.4273	0.9615	0.7036
32	0.8788	0.2636	0.9766	0.5992

Table XVII. Significant Features According to Entropy Based Feature Selection (1x Dataset)

	Feature	**Information Gain**
1	Size	0.2154
2	Category	0.2056
3	Mentioning popular app names top-1000 TF-IDF	0.1764
4	Mentioning popular app names top-1000	0.1573
5	Privacy policy available	0.1429
6	Developer’s website available	0.0992
7	Developer’s website reachable	0.0992
8	Number of apps by the developer	0.0904
9	Frequency of manually identified word-bigrams and word-trigrams	0.0880
10	Mentioning popular app names top-100 TF-IDF	0.0831



(a)



(b)

Fig. 14. Performance against the number of features.

Table XVIII. Predictions on Spam Apps in Google Play Store

Dataset	**Size**	$k = 2$	$k = 32$
Crawl 1 ($\mathbb{C}_1$)	6,566	70.37%	12.89%
Crawl 2 ($\mathbb{C}_2$)	9,184	73.14%	6.57%
Crawl 3 ($\mathbb{C}_3$)	18,897	72.99%	6.49%
Others	180,627	54.59%	2.69%

Table XIX. Manual Validation of the Predictions

		Prediction	
		Spam	**Nonspam**
Manual Labelling	**Spam**	82.47%	32.58%
	Legit	5.98%	29.69%
	Other	11.55%	37.73%

($k = 32$) predicted 6%–12% of the removed apps and approximately 2.7% of the other apps as spam.

As an indicator of the extent to which our predictions were correct, we perform two additional analyses.

5.4. Manual Labelling of a Sample of Predictions

We gave the same reviewers a set of apps that contained 10% of the spam apps that were labelled by the $k = 32$ classifier (485) and another 485 of nonspam apps. Reviewers were asked to check the apps according to the guidelines provided in Section 2.2 and it was mentioned to them that in this occasion the set might contain apps that are legitimate (Legit).

Table XIX shows the results of this manual labelling process. As can be seen, human reviewers marked approximately 82% of the spam predictions made by the classifier as spam and only 6% of the spam predictions are labelled as legitimate. Reviewers also labelled approximately 32% of the apps that were predicted as nonspam also as spam. The reason behind this is that *Others* set contains the apps that are not in top app rankings (beyond top-32k $\sim$ 17,632) and can potentially include spam apps or apps showing some marginal spam characteristics.

The results of the manual labelling process is in agreement with the results of Table XI. The precision calculated according to the human reviewing (82%) is close to the precision obtained on the testing set (86%).

Table XX. Our Predictions Against Google App Removals

k	Prediction	Removed	Not Removed
$k = 2$	Spam	24.47%	75.53%
	Nonspam	16.97%	83.03%
$k = 32$	Spam	**40.14%**	59.86%
	Nonspam	20.55%	**79.45%**

5.5. A 1 Year Later Recrawl

We did another crawl 1 year after Crawl-3 (in May '15), targeting the *Others* set, to check our predictions against app removals. Table XX shows the results of the crawl. As can be seen, at $k = 2$, 24% apps predicted as spam got removed from the Play Store. However, approximately 17% of the apps that were predicted as nonspam was also removed from the Play Store resulting in only a marginal performance. $k = 32$ classifier's performance was significantly better as 40% of the predicted spam apps were removed, compared to the 21% of the predicted nonspam apps that were removed.

To elaborate further, over the year, 21% apps were removed from the app market for various reasons. This means a random prediction will give a 21% accuracy. However, in the set predicted as spam using the proposed classifier, 40% of apps were removed, which indicates the classifier is performing twice as well as a random classifier. Moreover, it is unlikely that Google removes all possible spam apps during a year and thus a number of spam apps can still remain in the market. Thus, the actual classifier performance can be even higher.

Over this crawl, we found that approximately 19% of the apps were removed and the percentage of removal in our nonspam predicted datasets is close to that. This further shows that our spam predictions are meaningful and perform better than a random prediction.

6. RELATED WORK

In this section, we discuss related work in spam detection for web pages, email, and SMS, and the detection of malware apps and overprivileged apps.

Web spam refers to the publication of web pages that are specifically designed to influence search engine results. Using a set of manually classified samples of web pages obtained from *"MSN Search,"* Ntoulas et al. [2006] characterise the web page features that can be used to classify a web page as spam or not. The features include top-level domain, language of the web page, number of words in the web page, number of words in the page title, etc.

Fetterly et al. [2004] characterise the features that can potentially be used to identify *web spam* through statistical analysis. The authors analyse features such as *URL properties*, *host name resolutions*, and *linkage properties* and find that outliers within each of the properties considered are spam. Castillo et al. [2007] describe a cost-sensitive bagging classifier with decision trees, which when used individually with *link-based features* and *content-based features* can classify web pages into spam and nonspam. Gyöngyi et al. [2004] propose the use of the link structure in a limited set of manually identified nonspam web pages to iteratively find spam and nonspam web pages, and show that a significant fraction of the *web spam* can be filtered using only a seed set of less than 200 sites. Krishnan and Raj [2006] use a similar approach. Erdélyi et al. [2011] show that a computationally inexpensive feature subset, such as the number of words in a page and the average word length, is sufficient to detect web spam.

Detection of *email spam* has received considerable attention [Blanzieri and Bryl 2008]. Various *content related features* of mail messages such as *email header, text in the email body*, and *graphical elements* have been used together with machine learning techniques, such as naive Bayesian [Pantel and Lin 1998; Sahami et al. 1998; Metsis et al. 2006], support vector machines [Drucker et al. 1999; Aradhye et al. 2005; Sculley and Wachman 2007], and *k* nearest neighbour [Androutsopoulos et al. 2000]. *Noncontent related features* such as *SMTP path* [Leiba et al. 2005] and *user's social network* [Oscar and Roychowdbury 2005; Chirita et al. 2005] has also been used in spam email detection.

Spam has also been studied in the context of *SMS* [Gómez Hidalgo et al. 2006; Cormack et al. 2007], *product reviews* [Jindal and Liu 2007, 2008; Chandy and Gu 2012], *blog comments* [Mishne et al. 2005], and *social media* [Wang 2010; Benevenuto et al. 2010]. Gómez Hidalgo et al. [2006] use content-based features such as words, lowercased words, and character and word bigrams and trigrams to train various classifiers including naive Bayesian and decision trees. Cormack et al. [2007] show that due to the limited size of SMS messages, bag of words or word bigram based spam classifiers do not perform well, and their performance can be improved by expanding the set of features to include orthogonal sparse word bigrams, and character bigrams and trigrams. Jindal and Liu [2008] identify spam product reviews using *review centric features* such as the number of feedback reports, textual features of the reviews, and *product centric features* such as price and sales rank, as well as *reviewer centric features* such as the average rating given by the reviewer. Wang [2010] study detection of spammers in Twitter.

In contrast to aforementioned spam such as web spam, email spam, or SMS spam, spamming in mobile app markets is a new problem and not as well established as other forms. To the best of our knowledge, our previous work [Seneviratne et al. 2015] is the first attempt at characterising spam mobile apps and developing an automated means of detecting them when they are submitted for publication approval. As mentioned in the Introduction, this work is an extension of our previous work with further analysis of significant features that defines the performance of the classifier and further performance evaluation with manual labelling and validation by revisiting the app market again for the predicted apps.

More recently, *malware mobile apps* [Zhou et al. 2012; Grace et al. 2012; Burguera et al. 2011; Yang et al. 2015; Feng et al. 2014], *overprivileged apps* (i.e., apps with overpermissions) [Peng et al. 2012; Gorla et al. 2014; Avdiienko et al. 2015], and *similar apps* (i.e., *clones:* similar apps by different developers and *rebranding:* similar apps by the same developer) [Viennot et al. 2014; Crussell et al. 2013; Chen et al. 2014] have received attention. Zhou et al. [2012] propose *DroidRanger*, which uses permission-based behavioural fingerprinting to detect new samples of known Android malware families and heuristics-based filtering to identify inherent behaviours of unknown malware families. Gorla et al. [2014] regroup app descriptions using a Latent Dirichlet Allocation and k-means clustering to identify apps that have unexpected API usage characteristics. Viennot et al. [2014] clustered apps based on the Jaccard Similarity of app resources such as images and layout XMLs, to identify similar apps and used developer information such as the name and the certificate included in the app to differentiate clones from rebranding. Crussell et al. [2013] clustered apps according to the code level similarity features to identify similar apps. Spam apps are not necessarily malware and the objective of spam is to attract keyword search results or obtain more visibility for apps by unfair means, compared to malicious intentions such as personal data collection or ransomware.

Several studies use *app review mining* as a means of *identifying bugs*, *new feature requirements*, and *user complaints and praises* [Iacob and Harrison 2013; Maalej and Nabil 2015; Panichella et al. 2015; Di Sorbo et al. 2016; Guzman and Maalej 2014; Fu et al. 2013]. As mentioned in Section 3, app reviews can be helpful in identifying spam apps. However, as the reviews are not available when the developer submits the app for approval, they cannot be used in a solution for early identification of potential spam apps. Factors affecting app popularity such as reviews, price, and app category have also been studied [Harman et al. 2012; Petsas et al. 2013]. In our work, we used the features that are available during the time of publication and as our analysis showed, detection of spam apps requires delving into text description based features.

7. DISCUSSION

In this section, we discuss the potential end users of the proposed methodology, limitations, and possible future extensions.

Applicability. Our proposed classifier is relevant to app market operators such as Google Play Store and Apple App Store to enable faster app approval times. As mentioned in the Introduction, spam apps are increasingly becoming an issue in app markets [Perez 2013a, 2013b] and it is essential to come up with automated checks for apps in order to achieve a faster approval process whilst ensuing users' quality of experience. As of now, while major app markets have spam policies [Google 2016c; Apple 2016],[10] enforcement of these policies is complaint-driven and/or semimanual. As a result, spam apps enter the app markets before they are periodically removed. For example, more recently a number of spam apps made it to the feature charts of Apple App Store before they were taken down [Tecno Buffalo 2016; Farooqui 2016; Perez 2016]. We believe a mechanism similar to the proposed classifier will enable app market operators to automate the vetting of new apps that will be much faster than if it were to be done manually.

The Manual Labelling Challenge. In this work, we used a relatively small set (551) of labelled spam apps and used the top apps from the market place as proxies for nonspam apps. Our choices were dictated largely by the time-consuming nature of the labelling process as mentioned in Section 2.2. Obviously, the performance of the classifier can be improved by increasing the number of labelled spam apps and further labelling nonspam apps through an extension of our manual effort. Moreover, having more than three reviewers will enhance the quality of the manual labelling process as more judgements will reduce the effect of noise.

One approach is to rely on crowdsourcing and recruit app-savvy users as reviewers. This would require a major effort of providing individualised guidelines and interactions with the reviewers, and a need to deal with assessment inconsistencies due to the variability in technical expertise. Nonetheless, from an app market provider perspective, this is a plausible option to consider. Alternatively, hybrid schemes can be developed where apps are flagged during the approval process and removed based on a limited number of customer complaints. Another potential direction is to consider a mix of supervised and unsupervised learning, namely, *semisupervised learning* [Basu et al. 2004]. The basic idea behind semisupervised learning is to use "unlabelled data" to improve the classification model that was previously being built using only labelled data. Semisupervised learning has previously being successfully applied to real-time traffic classification problems [Erman et al. 2007].

Improving Classifier's Performance. In addition to increasing the number of labelled samples, a few other options can be explored to further improve the performance of the classifier. For example, to alleviate the class imbalance effect that happens as *k* increases, modified versions of *Adaptive Boosting* such as *RUSBoost* [Seiffert et al.

[10]As mentioned in Section 3, Apple has a single policy for app developers and only some sections are applicable for spam.

2010] and *SMOTEBoost* [Chawla et al. 2003] can be tested. Also, a classifier's performance can be further evaluated by adjusting the classifier threshold value for each k value and checking the trade-off between precision and recall.

Additional Features. The proposed classifier in this article used only the app metadata to make a decision at the time of publication. The feature set can be augmented by adding features related to the source code obtained by decompiling the applications following the intuition that some spam apps might show a high level of similarity in the source code as they might have been generated in bulk, potentially using automated techniques. However, such features need to consider the fact that there is a significant amount of code level redundancy between apps due to the third-party Software Development Kits (SDKs) such as advertising and analytics libraries and open source utility libraries [Viennot et al. 2014]. Also, code obfuscation and encryption methods are becoming increasingly popular and the use of such methods eliminate the possibility of source code inspection [Zhang et al. 2015]. Another set of features can be added based on user reviews if the app store wants to make a decision on the app after publishing it in the market. Similarly, availability of app updates and changes in app metadata can also be leveraged to generate features in such a setting as most of the legitimate apps are likely to be updated based on user feedback, whereas updates are unlikely for spam.

The classification arms race. It is possible that spammers adapt to the spam app detection framework and change their strategies according to the selected features to avoid the detection. The relevant questions in this context are the following: (i) how frequently should the classifier be retrained? and (ii) how can one detect when retraining is required? We believe that spam app developers will find it challenging and have significant cost overheads to adapt their apps to avoid features that allow discriminating between spam and nonspam apps. For instance, to avoid the similarity of descriptions of multiple apps, the spammer has to edit the different descriptions of the apps and customise each of the app descriptions to contain sufficient details and coherent text, etc. An important direction for future work is to study the longevity of our classifier and investigate how the process of identifying retraining requirements can be automated. Prior work on traffic classification suggests that automated identification of retraining points is possible using semisupervised learning approaches [Erman et al. 2007].

Other Problematic Apps. Our manual app analysis process revealed multiple other types of problematic apps in app markets such as counterfeits, unofficial content apps, and apps violating copyrights. While these apps are not as common as spam apps, it is necessary to detect them and remove them from app markets for a healthier app market ecosystem. Thus, another interesting research direction is to come up with efficient prediction models to detect other types of problematic apps. The category that also requires immediate attention is counterfeits. According to our labelling, counterfeits contribute to approximately 8% of the app removals. Counterfeits represent a hindrance to the evolution of the app ecosystem with potential security risks, ultimately disrupting the economy and interests of legitimate and successful apps.

8. CONCLUSION

In this article, we propose a classifier for automated detection of spam apps at the time of app submission. Our app classifier utilises only those features that can be derived from an app's metadata available during the publication approval process. It does not require any human intervention such as manual inspection of the metadata or manual app testing. We validate our app classifier, by applying it to a large dataset of apps collected between December 2013 and May 2014, by crawling and identifying apps that were removed from Google Play Store. Our results show that it is possible to automate the process of detecting spam apps solely based on apps' metadata available at the time of publication and achieve both high precision and recall.

Our predictions on a set of 180,627 apps that were present in Google Play Store during our crawl, suggest that approximately 2.7% of the apps can be potential spam. We provide additional analysis of our predictions by performing a new crawl as well as applying our rigorous manual labelling process to a sample of the apps for which we made predictions. This analysis shows human reviewers agreed with 82% of the classifier's spam predictions and apps that were predicted as spam by our classifier had 20% higher likelihood of being removed from Google Play Store. We also showed that a smaller number of features in the range of 10–30 is sufficient to reach a satisfactory level of performance with respect to precision and recall.

REFERENCES

Ion Androutsopoulos, Georgios Paliouras, Vangelis Karkaletsis, Georgios Sakkis, Constantine D. Spyropoulos, and Panagiotis Stamatopoulos. 2000. Learning to filter spam e-mail: A comparison of a naive Bayesian and a memory-based approach. *arXiv preprint cs/0009009* (2000).

App Annie. 2016. App Forecast: Over $100 Billion In Revenue by 2020. Retrieved from http://blog.appannie.com/app-annie-releases-inaugural-mobile-app-forecast/.

AppBrain, Inc. 2016. New Android apps per month. Retrieved from http://www.appbrain.com/stats/number-of-android-apps.

Apple. 2014. Common App Rejections. Retrieved from https://developer.apple.com/app-store/review/rejections/.

Apple. 2016. App Store Review Guidelines. Retrieved from https://developer.apple.com/app-store/review/guidelines/.

Hrishikesh B. Aradhye, Gregory K. Myers, and James A. Herson. 2005. Image analysis for efficient categorization of image-based spam e-mail. In *Proceedings of the 8th International Conference on Document Analysis and Recognition*. IEEE, 914–918.

Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Andreas Zeller, Steven Arzt, Siegfried Rasthofer, and Eric Bodden. 2015. Mining apps for abnormal usage of sensitive data. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 426–436.

AVG. 2014. Website Safety Ratings and Reputation. Retrieved from http://www.avgthreatlabs.com/website-safety-reports/app.

Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. 2004. A probabilistic framework for semi-supervised clustering. In *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining*. ACM, 59–68.

Fabrıcio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgılio Almeida. 2010. Detecting spammers on twitter. In *Proceedings of the 7th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*.

Enrico Blanzieri and Anton Bryl. 2008. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review* 29, 1 (2008), 63–92.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. 1984. *Classification and Regression Trees*. CRC Press.

Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 15–26.

Omar Canales, Vinnie Monaco, Thomas Murphy, Edyta Zych, John Stewart, Charles Tappert, Alex Castro, Ola Sotoye, Linda Torres, and Greg Truley. 2011. A stylometry system for authenticating students taking online tests. In *Proceedings of the Student-Faculty CSIS Research Day* (2011).

Carlos Castillo, Debora Donato, Aristides Gionis, Vanessa Murdock, and Fabrizio Silvestri. 2007. Know your neighbors: Web spam detection using the web topology. In *Proceedings of the 30th Annual International Conference on Research and Development in Information Retrieval*. ACM, 423–430.

Rishi Chandy and Haijie Gu. 2012. Identifying spam in the iOS app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*. ACM, 56–59.

Nitesh V. Chawla, Aleksandar Lazarevic, Lawrence O. Hall, and Kevin W. Bowyer. 2003. SMOTEBoost: Improving prediction of the minority class in boosting. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 107–119.

Kai Chen, Peng Liu, and Yingjun Zhang. 2014. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 175–186.

Paul-Alexandru Chirita, Jörg Diederich, and Wolfgang Nejdl. 2005. MailRank: Using ranking for spam detection. In *Proceedings of the 14th International Conference on Information and Knowledge Management*. ACM, 373–380.

Gordon V. Cormack, José María Gómez Hidalgo, and Enrique Puertas Sánz. 2007. Spam filtering for short messages. In *Proceedings of the 16th Conference on Information and Knowledge Management*. ACM, 313–320.

Jonathan Crussell, Clint Gibler, and Hao Chen. 2013. AnDarwin: Scalable detection of semantically similar android applications. In *Computer Security–ESORICS 2013*. Springer, 182–199.

Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald Gall. 2016. What would users change in my app? Summarizing app reviews for recommending software changes. In *Proceedings of the 2016 ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*.

Harris Drucker, S. Wu, and Vladimir N. Vapnik. 1999. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks* 10, 5 (1999), 1048–1054.

Miklós Erdélyi, András Garzó, and András A. Benczúr. 2011. Web spam classification: A few features worth more. In *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality*. ACM, 27–34.

Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. 2007. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation* 64, 9–12 (Oct. 2007), 1194–1213.

Adnan Farooqui. 2016. Apple Promises To Clamp Down On Spam Apps. Retrieved from http://www.ubergizmo.com/2016/03/apple-promises-to-clamp-down-on-spam-apps/.

Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 576–587.

Dennis Fetterly, Mark Manasse, and Marc Najork. 2004. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *Proceedings of the 7th International Workshop on the Web and Databases*. ACM, 1–6.

Rudolph Flesch. 1948. A new readability yardstick. *Journal of Applied Psychology* 32, 3 (1948), 221.

Yoav Freund and Robert E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, Vol. 96. Morgan Kaufmann, 148–156.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer Series in Statistics. Springer, Berlin. 367–370.

Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1276–1284.

José María Gómez Hidalgo, Guillermo Cajigas Bringas, Enrique Puertas Sánz, and Francisco Carrero García. 2006. Content based SMS spam filtering. In *Proceedings of the 2006 Symposium on Document Engineering*. ACM, 107–114.

Google. 2014. Rating your application content for Google Play. Retrieved from https://support.google.com/googleplay/android-developer/answer/188189.

Google. 2016a. Google Play Developer Policy Center. Retrieved from https://play.google.com/about/developer-content-policy-print/.

Google. 2016b. Impersonation and Intellectual Property. Retrieved from https://play.google.com/about/ip-deception-spam/impersonation-ip/.

Google. 2016c. Spam. Retrieved from https://play.google.com/about/ip-deception-spam/spam.

Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*. 1025–1035.

Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. Riskranker: Scalable and accurate zero-day android malware detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. ACM, 281–294.

Emitza Guzman and Walid Maalej. 2014. How do users like this feature? A fine grained sentiment analysis of app reviews. In *Proceedings of the 2014 IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, 153–162.

Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. 2004. Combating web spam with trustrank. In *Proceedings of the 13th International Conference on Very Large Databases*. VLDB Endowment, 576–587.

Mark Harman, Yue Jia, and Yuanyuan Zhang. 2012. App store mining and analysis: MSR for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 108–111.

Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 2013 10th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 41–44.

Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intelligent Data Analysis* 6, 5 (2002), 429–449.

Nitin Jindal and Bing Liu. 2007. Review spam detection. In *Proceedings of the 16th International Conference on World Wide Web*. ACM, 1189–1190.

Nitin Jindal and Bing Liu. 2008. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 219–230.

Ron Kohavi and George H. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence* 97, 1 (1997), 273–324.

Vijay Krishnan and Rashmi Raj. 2006. Web spam detection with anti-trust rank. In *Proceedings of the 2nd International Workshop on Adversarial Information Retrieval on the Web*, Vol. 6. 37–40.

Barry Leiba, Joel Ossher, V. T. Rajan, Richard Segal, and Mark N. Wegman. 2005. SMTP path analysis. In *Proceedings of the 2nd Conference on Email and Anti-Spam*.

Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proceedings of the 2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE, 116–125.

Dragos D. Margineantu and Thomas G. Dietterich. 1997. Pruning adaptive boosting. In *ICML*, Vol. 97. 211–218.

Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. 2006. Spam filtering with naive Bayes—Which naive Bayes? In *Proceedings of 3rd Conference on Email and Anti-Spam*. 27–28.

Gilad Mishne, David Carmel, and Ronny Lempel. 2005. Blocking blog spam with language model disagreement. In *Proceedings of the 1st International Workshop on Adversarial Information Retrieval on the Web*, Vol. 5. 1–6.

Arjun Mukherjee and Bing Liu. 2010. Improving gender classification of blog authors. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 207–217.

Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. 2006. Detecting spam web pages through content analysis. In *Proceedings of the 15th International Conference on World Wide Web*. ACM, 83–92.

Jon Oberheide and Charlie Miller. 2012. Dissecting the Android bouncer. Retrieved from https://jon.oberheide.org/files/summercon12-bouncer.pdf.

Oracle. 2014. Naming a Package. Retrieved from http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html.

Boykin P. Oscar and Vwani P. Roychowdbury. 2005. Leveraging social networks to fight spam. *IEEE Computer* 38, 4 (2005), 61–68.

Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2015. How can I improve my app? Classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 281–290.

Patrick Pantel and Dekang Lin. 1998. Spamcop: A spam classification & organization program. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*. 95–98.

Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the Conference on Computer and Communications Security*. ACM, 241–252.

Sarah Perez. 2013a. Developer Spams Google Play With Ripoffs of Well-Known Apps Again. Retrieved from http://techcrunch.com.

Sarah Perez. 2013b. Nearly 60K Low-Quality Apps Booted From Google Play Store in February, Points To Increased Spam-Fighting. (2013). http://tcrn.ch/14SwCQj.

Sarah Perez. 2016. Apple's Phil Schiller promises to address the issue of spammy apps being featured in the App Store. Retrieved from https://techcrunch.com/2016/03/14/apples-phil-schiller-promises-to-address-the-issue-of-spammy-apps-being-featured-in-the-app-store/.

Thanasis Petsas, Antonis Papadogiannakis, Michalis Polychronakis, Evangelos P. Markatos, and Thomas Karagiannis. 2013. Rise of the planet of the apps: A systematic study of the mobile app ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*. ACM, 277–290.

PocketGamer.biz. 2016. Count of Application Submissions. Retrieved from http://www.pocketgamer.biz/metrics/app-store/submissions/.

J. R. Quinlan. 1996. Bagging, boosting, and C4.S. In *Proceedings of the 13th National Conference on Artificial Intelligence - Volume 1*. AAAI Press, 725–730.

Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. 1998. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Vol. 62. 98–105.

David Sculley and Gabriel M. Wachman. 2007. Relaxed online SVMs for spam filtering. In *Proceedings of the 30th Annual International Conference on Research and Development in Information Retrieval*. ACM, 415–422.

Chris Seiffert, Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. 2010. RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 40, 1 (2010), 185–197.

Suranga Seneviratne, Aruna Seneviratne, Dali Kaafar, Anirban Mahanti, and Prasant Mohapatra. 2014a. *Why My App Got Deleted: Detection of Spam Mobile Apps*. Technical Report. NICTA, Australia.

Suranga Seneviratne, Aruna Seneviratne, Mohamed Ali Kaafar, Anirban Mahanti, and Prasant Mohapatra. 2015. Early detection of spam mobile apps. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. International World Wide Web Conferences Steering Committee, 949–959.

Suranga Seneviratne, Aruna Seneviratne, Prasant Mohapatra, and Anirban Mahanti. 2014b. Predicting user traits from a snapshot of apps installed on a smartphone. *ACM SIGMOBILE Mobile Computing and Communications Review* 18, 2 (2014), 1–8.

R. J. Senter and E. A. Smith. 1967. *Automated Readability Index*. Technical Report AMRL-TR-66-220. Aerospace Medical Research Laboratories.

Ian Soboroff, Iadh Ounis, J. Lin, and I. Soboroff. 2012. Overview of the TREC-2012 microblog track. In *Proceedings of the 21st Text Retrieval Conference*.

Statista, Inc. 2016. Number of apps available in leading app stores as of June 2016. Retrieved from http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

Tecno Buffalo. 2016. Apple exec responds flood of spam apps in App Store. Retrieved from http://www.technobuffalo.com/2016/03/14/apple-exec-responds-flood-of-spam-apps-in-app-store/.

Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A measurement study of google play. In *Proceedings of the 2014 International Conference on Measurement and Modeling of Computer Systems*. ACM, 221–233.

Alex Hai Wang. 2010. Don't follow me: Spam detection in twitter. In *Proceedings of the 2010 International Conference on Security and Cryptography*. IEEE, 1–10.

Wikipedia. 2014. Wikipedia: Lists of common misspellings. Retrieved from http://en.wikipedia.org/wiki/.

Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie, and William Enck. 2015. Appcontext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 303–313.

Yueqian Zhang, Xiapu Luo, and Haoyang Yin. 2015. Dexhunter: Toward extracting hidden code from packed android applications. In *Computer Security–ESORICS 2015*. Springer, 293–311.

Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. 2012. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *Proceedings of the 2012 Network and Distributed System Security Symposium*. The Internet Society.

Received November 2015; revised November 2016; accepted November 2016