Early Detection of Spam Mobile Apps

Suranga Seneviratne[†⋆], Aruna Seneviratne[†⋆], Mohamed Ali Kaafar[†],
Anirban Mahanti[†], Prasant Mohapatra[‡]

[†]NICTA; [⋆]School of EET, UNSW; [‡] Department of CS, University of California, Davis

ABSTRACT

Increased popularity of smartphones has attracted a large number of developers to various smartphone platforms. As a result, app markets are also populated with spam apps, which reduce the users' quality of experience and increase the workload of app market operators. Apps can be "spammy" in multiple ways including not having a specific functionality, unrelated app description or unrelated keywords and publishing similar apps several times and across diverse categories. Market operators maintain anti-spam policies and apps are removed through continuous human intervention. Through a systematic crawl of a popular app market and by identifying a set of removed apps, 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. Finally, we build an *Adaptive Boost* classifier for early identification of spam apps using only the metadata of the apps. Our classifier achieves an accuracy over 95% with precision varying between 85%–95% and recall varying between 38%–98%. By applying the classifier on a set of apps present at the app market during our crawl, we estimate that at least 2.7% of them are spam apps.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Spam, Mobile Apps, Android, Spam Apps

1. INTRODUCTION

Recent years have seen wide adoption of mobile apps, and the number apps that are being offered are growing exponentially. As of mid-2014, Google Play Store and Apple App Store, each hosted approximately 1.2 million apps [48], with around 20,000 new apps being published each month in both of these app markets. These numbers are expected to significantly grow over the next few years [49].

Both Google and Apple have policies governing the publication of apps through their app markets. Google has an explicit spam policy [31] that describes their main reasons for considering an app to be "spammy". These include : i) apps that were automatically generated and published, and as such 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 unintended searches. Overall, spam apps vitiate the app market experience and its usefulness.

At present, Google and Apple take different approaches to the spam detection problem. Google's approach is reactive. Google's app approval process does not check an app against its explicit spam app policy [42], and takes action only on the basis of customer complaints [46]. Such crowdsourced approach can lead to a considerable time lag between app submission and detection of spam behaviour. In contrast, Apple scrutinises the apps submitted for approval, presumably using a manual or semi-manual process, to determine whether the submitted app conforms to Apple's policies. Although this approach is likely to detect spam apps before they are published, it lengthens the app approval process. *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.*

This paper proposes a methodology for automated detection of spam apps at the time of app submission. Our classifier utilises only 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 testing of the app. 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. 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 3 and 4). Using this methodology we found 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.

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741084.

- 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 5). We provide a characterisation of these features highlighting differences between spam and non-spam apps and indicate which features are more discriminative.
- We build an *Adaptive Boost* classifier for early detection of spam apps and show that our classifier can achieve an accuracy over 95% at a precision between 85%–95% and a recall between 38%–98% (Section 6).
- We applied 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 6).

To the best of our knowledge, this is the first work to develop an early detection framework for identifying spam mobile apps and to demonstrate its efficacy using real world datasets. Our work complements prior research on detecting apps seeking *over-permissions* or apps containing *malware* [59, 22, 8, 45, 21].

2. 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 over-privileged 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. [41] 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, and number of words in the page title etc.

Fetterly et al. [17] 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. Gyöngyi et al. [23] propose the use of the link structure in a limited set of manually identified non-spam web pages to iteratively find spam and non-spam 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 et al. [36] use a similar approach. Erdélyi et al. [15] 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 [7]. 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 [44, 50, 38], support vector machines [14, 4, 51], and k nearest neighbour [3]. *Non-content related features* such as *SMTP path* [37] and *user's social network* [43, 11] has also been used in spam email detection.

Spam has also been studied in the context of *SMS* [20, 12], *product reviews* [34, 33, 10], *blog comments* [39], and *social media* [57, 6]. Cormack et al. [12] 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 et al. [34] 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, sales rank as well as *reviewer centric features* such as the average rating given by the reviewer. Wang et al. [57] study detection of spammers in Twitter.

More recently, Malware mobile apps, over-privileged apps (i.e., apps with over-permissions), and similar apps (i.e., *clones:* similar apps by different developers and *rebranding:* similar apps by the same developer) have received attention [59, 22, 8, 45, 21, 56, 13]. Zhou et al. [59] 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. [21] regroup app descriptions using a Latent Dirichlet Allocation and k-means clustering to identify apps that have unexpected API usage characteristics. Viennot et.al [56] 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. [13] clustered apps according to the code level similarity features to identify similar apps.

3. METHODOLOGY

3.1 Dataset

We use apps collected in a previous study [53], 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*[1]. We crawled each app's Google Play Store page through a Java client that uses *jsoup*[2] HTML parser to discover: i) functionally similar apps; ii) other apps by the same developer. Then we collected metadata such as app name, app description, and app category for all the apps we discovered including the seed set. We discarded the apps with a non-English description using the language detection API, "*Detect Language*" [1]. We refer to this final set as the ***Observed Set*** - $\mathbb{O}$.

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 at least one month gap between two consecutive crawls. Figure 1 illustrates data collection process and Table 1 summarises the datasets in use.

We identify temporary versus long-term removal of apps by re-checking 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

[1]www.appbrain.com
[2]www.jsoup.org

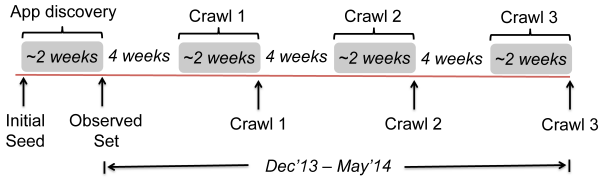


Figure 1: Chronology of the data collection

Table 1: 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

(∼0.02%) identified as removed in ***Crawl 2*** reappeared in ***Crawl 3***. These apps were not included in our analysis.

3.2 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 lead to removal of apps from the market place by consulting numerous market reports [47, 25] as well as by examining the policies of the major app markets [31, 29, 27, 28, 30, 24]. We identified nine key **reasons**, which are summarised in Table 2. 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. Owing to space limitations we do not provide the heuristic checkpoints for removed reasons except for spam. Full list of checkpoints for each reason can be found in our technical report [52]. Section 4 delves into the checkpoints developed for identifying spam apps.

Table 2: Key reasons for removal of apps

Reason	Description
Spam	These apps often have characteristics such as unrelated description, keyword misuse, and multiple instances of the same app. Section 4 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 the 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.

From ***Crawl 1***, we took a random sample of 1500 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 on the level of satisfaction of one or more checkpoints. If a reviewer could 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 for this task. The manual labelling processing took approximately 20 working days (7 hours per day).

Table 3: Reviewer agreement in labelling reason for removal

Reason	3 reviewers agreed	2 reviewers agreed	Total	Percent. (%)
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%
Sub total	**601**	**749**	**1350**	90.00%
Reviewer disagreement	NA	NA	150	10.00%
Total Labelled	**NA**	**NA**	**1500**	100.00%

3.3 Agreement Among the Reviewers

We used majority voting to merge the results of the experts assessment, 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 3 summarises reviewer agreement. For approximately 40% (601 out of 1500) of labelled apps, the three reviewers reached a consensus on the reason for removal and for 90% (1350 out of 1500) of the apps majority of the reviewers agreed on the same reason. For the remaining 10% of apps, reviewers disagreed about the reasons.

Table 3 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 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 release our labelled dataset to the research community [2] to stem further research in spam app detection.

4. 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 un-

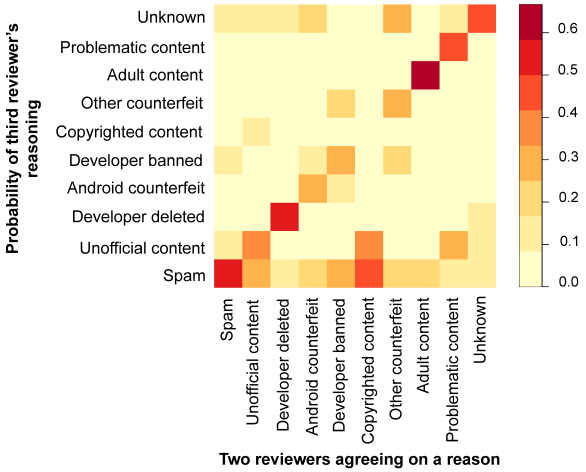


Figure 2: Probability of a third reviewer's judgement when two reviewers already agreed on a reason

ambiguous and suitable for manual labelling. Section 5 maps the defined checkpoints into automated features.

4.1 Spam Checkpoints

Table 4 presents the nine heuristic checkpoints used by the reviewers and those were derived based on Google's spam policy [31]. While we are unaware of Apple's spam policy, we note that Apple's app review guideline [24] includes certain provisions that match our spam checkpoints. For example, term 2.20 in the Apple app review guideline, *"Developers spamming the App Store with many versions of similar apps will be removed from the iOS Developer Program"* maps to our checkpoint S_6. Similarly, term 3.3 *"Apps with names, descriptions, screenshots, or previews not relevant to the content and functionality of the app will be rejected"* 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 4 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.

4.2 Reviewer Agreement on Spam Checkpoints

Table 5 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 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.

Table 5: Checkpoint-wise reviewer agreement for spam

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

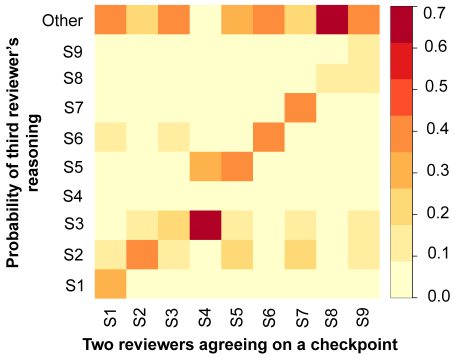


Figure 3: Probability of a third reviewer's judgement when two reviewers already agreed on a checkpoint

The sum of the cases where 3 reviewers agreed, 2 reviewers agreed, and reviewers disagreed shown in Table 5, 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 ($\sim$ 38%), and two reviewers agreed on at least one checkpoint for 296 apps ($\sim$ 54%). For 45 apps ($\sim$ 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 3 cases where 2 reviewers agreed as shown in Table 5. In those 3 cases the other reviewer marked the checkpoint S_3 in 2 cases giving a high probability ($\sim$ 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.

5. FEATURE MAPPING

Checkpoint heuristics for spam (cf. Table 4) need to be mapped into features that can be extracted and used for

Table 4: 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? ***E.g. Signature Capture App (Non - spam) - Description is clear on the functionality of the application*** *"SignIt app allows a user to sign and take notes which can be saved and shared instantly in email or social media."* ***E.g. Manchester (Spam) - Description contains details about Manchester United Football Club without any detail 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 for an English club."*
	S_2	Does the app description contain too much details / incoherent text / unrelated text for an app description? ***E.g. SpeedMoto (Non - spam) - 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 swap 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. More and More maps and motos will coming soon"* ***E.g. Ferrari Wallpapers HD (Spam) - Description starts mentioning app as a wallpaper. However, then it goes into to details 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 race cars before moving into production of street-legal ....."*
	S_3	Does the app description contain a noticeable repetition of words or keywords? ***E.g. English Chinese Dictionary (Non - Spam) - Keywords do not have excessive repetition.*** *"Keywords: ec, dict, translator, learn, translate, lesson, course, grammar, phrases, vocabulary, translation, dict"* ***E.g. 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? ***E.g. FD Call Assistant Free (Non - spam) - All the keywords are related to the fire department.*** *"Keywords: firefighter, fire department, emergency, police, ems, mapping, dispatch, 911"* ***E.g. 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 for other applications from the same developer? ***E.g. Kids Puzzles (Non - spam) - 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. Keywords: kids, puzzle, puzzles, toddler, fun"* ***E.g. 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, PTT Web ....."*
	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 the screen to add water drops on your home screen! Touch the top right corner of the screen to customise the wallpaper <>. 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. This live wallpaper has been tested on latest devices such as Samsung Galaxy S3 and Galaxy Nexus. Please contact us if your device is not supported. Note: If your wallpaper resets to default after reboot, you will need put the app on phone instead of SD card. "*
Identifier	S_7	Does the app identifier make sense and have some relevance to the functionality of the application or does it look like auto generated? ***E.g. Angry Birds Seasons & Candy Crush Saga (Non - spam) - Identifier give an idea about the app.*** *"com.rovio.angrybirdsseasons", "com.king.candycrushsaga"* ***E.g. Game of Thrones FREE Fan App & How To Draw Graffiti (Spam) - Identifiers appear to be auto generated.*** *"com.a1218284518 51a959009786c1a.a10023846a", "com.a13106102865265262e503a24a.a13796080a"*
Reviews	S_8	Do users complain about app being spammy in reviews? ***E.g. Yoga Trainer & Fitness & Google Play 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 app having excessive advertising? ***E.g. Biceps & Triceps Workouts*** *"AVG threat labs" [26] gives a warning about inclusion of malware causing excessive advertising.*

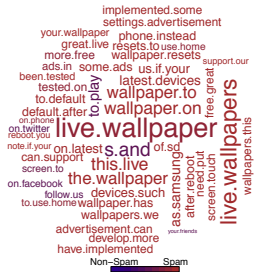


Figure 6: Top-50 bigrams differentiating Spam and Top-1x

automatic spam detection during the app approval process. This section presents this mapping and a characterisation of these features with respect to ***spam*** and ***non-spam*** apps.

An app is considered ***spam*** if two-out-of-three reviewers labelled it as spam. We assumed that top apps with respect to the number of downloads, number of user reviews, and rating, are quite likely to be ***non-spam***. For example, according to this ranking, number one ranked app was *Facebook* and number 100 was *Evernote*[3], number 500 was *Tictoc*[4] and number 1000 was *Saavn*[5]. Thus, for ***non-spam*** apps, 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. We varied k logarithmically between 1 and 32, (i.e., 1x, 2x, 4x,..., 32x) to obtain 6 datasets of non-spam apps. At larger k values it is possible that spam apps are considered to be non-spam, as discussed later in this section.

As noted in Section 4.1, checkpoints S_8 and S_9 are not used to develop features as we intend to enable automated spam detection at the time of developer submission.

5.1 *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 non-spam apps.

First, we manually read the descriptions of top 100 non-spam 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 one or more occurrences of these bigrams whereas 80% of the top 1x apps had one or more of these bi- and tri-grams. The difference in distribution of these bigrams and trigrams between spam and non-spam apps decreases when more and more lower ranked apps are considered in the non-spam 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 non-spam 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 non-spam.

[3]https://evernote.com
[4]http://www.tictoc.net
[5]http://www.saavn.com

Table 6: 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 upper case characters
7	Percentage of punctuations
8	Percentages of numeric characters
9	Percentage of non-alphabet characters
10	Percentage of common English words [9]
11	Percentage of personal pronouns [9]
12	Percentage of emotional words [40]
13	Percentage of misspelled words [58]
14	Percentage of words with alphabet and numeric characters
15	Automatic readability index (AR) [54]
16	Flesch readability score (FR) [18]

5.2 *Checkpoint S_2 - Does the app description contain too much details, incoherent text, or unrelated text?*

We use writing style related "stylometry" features to map this checkpoint to a set of features anticipating spam and non-spam apps might have a different writing style. We used the 16 features listed in Table 6 for this checkpoint.

For characterisation, we select features using the *greedy forward feature selection in wrapper method* [35]. 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 [55]. This process identified the *Total number of words in the description, Total number of sentences in the description, Percentages of numeric characters, Percentage of non-alphabet characters,* and *Percentage of common English words* as the most discriminative features.

Figures 5a shows the CDF of the total number of words in the app description. Spam apps tend to have less wordy app descriptions than non-spam 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 5b presents the CDF of the percentage of common English words [9] in the app description. It illustrates that spam apps typically use fewer common English words compared to non-spam apps.

5.3 *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

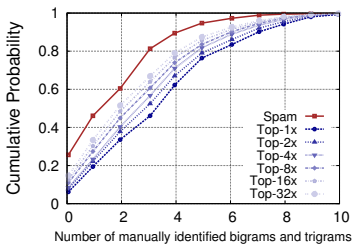
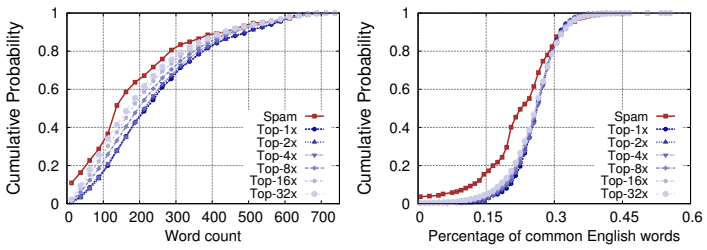


Figure 4: CDF of the frequency of manually identified word-bigrams and word-trigrams



(a) (b)

Figure 5: Example features associated with **Checkpoint** S_2

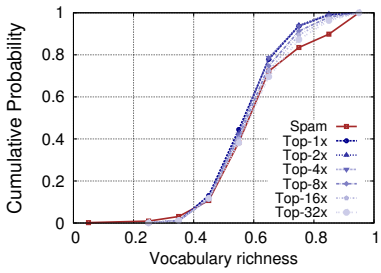
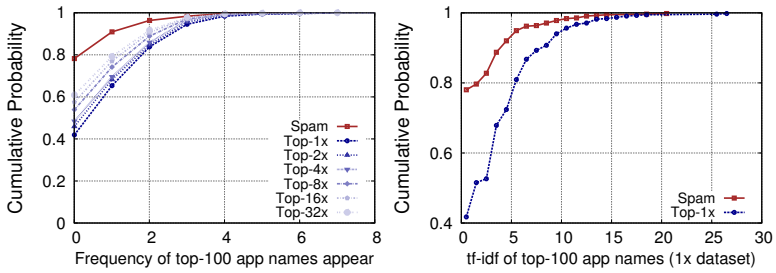


Figure 7: Vocabulary Richness



(a) (b)

Figure 8: Mentioning popular app names

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 non-spam 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 shorter description are more likely to be spam under ***checkpoint*** S_2.

5.4 *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 mentionings of the top 100 popular app names in the app's description.

Figure 8a 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 more than 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 non-spam apps. To summarise, presence of social sharing features or availability of fan pages can be used to identify non spam 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 it. However, if an app refers to another app which 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 $i = 1 : 100$ be the number of times the app name of i^{th} app (n_i) in top-100 apps appear 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 calculation above is dataset dependent. Despite this, as Figures 8b 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 non spam rather than spam. We repeated the same with top-1000 app names and found that the results were similar.

5.5 *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 5). We do not use checkpoint S_5 in our classifier.

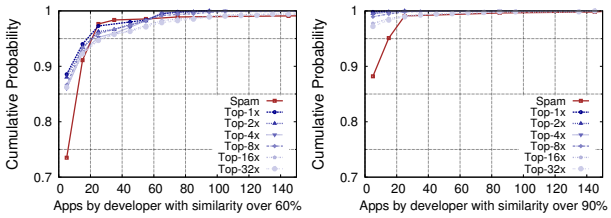
5.6 *Checkpoint S_6 - Does the developer have multiple apps with approximately the same description?*

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 which 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 lower case 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 9a and 9b respectively show the CDF of number of apps with s over 60% and 90% by the same developer.

Figure 9a shows that only about 10%–15% of the non-spam apps have more than 5 other apps from the same developer with over 60% of description similarity. However, approximately 27% of the spam apps have more than 5 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.



(a) Over 60% similarity (b) Over 90% similarity

Figure 9: Similarity with developer's other apps

5.7 *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 auto generated?*

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 [32] and differs from the app name which 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 7 shows the features derived from the *appid*. We applied the feature selection method noted in Section 5.2 to identify the most discriminative features (*Number of words, Average word length, Percentage of bigrams with 2 non-letters*).

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

Table 7: 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

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

5.8 Other Metadata

In addition to the features derived from the checkpoints, we added the metadata related features listed on Table 8.

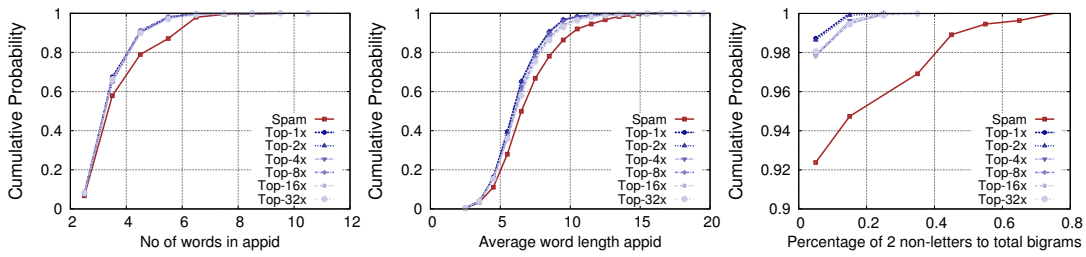
Figure 11 shows the app category distribution in the spam and the top-1x categories. We note that approximately 42% of the spam apps belong to the categories *Personalisation* and *Entertainment*. We found a negligibly small number of spam apps in the categories *Communication, Photography, Racing, Social, Travel,* and *Weather*. Moreover, for categories *Arcade, Tools,* and *Casual*, the percentage of spam is significantly less than the percentage of non-spam. Qualitatively similar observations hold for these other top-kx sets.

Figure 12a 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 non-spam apps had more than 25 characters in their app names. Figures 12b 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. For example if the size of the app is very low those apps are more likely to be non-spam. For example, 30% of the top-kx apps were less than 100KB in size and the corresponding percentage of spam apps 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 9 shows the external information related features. For example, if a link to a developer web site or a privacy policy is given the app is more likely to be non-spam.

Table 8: 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



(a) (b) (c)

Figure 10: Example features associated with **Checkpoint** S_7

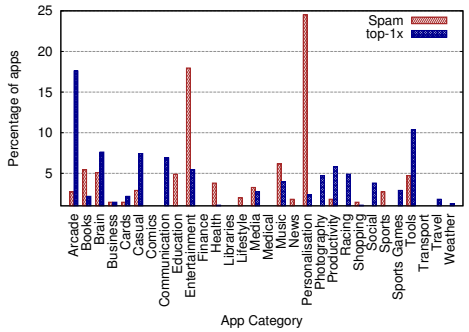
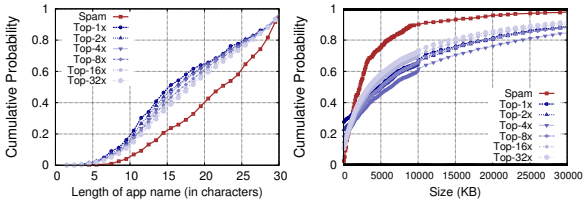


Figure 11: App category



(a) (b)

Figure 12: Features associated with other app metadata

Table 9: Availability of developer's external information

	Spam	top 1x	top 2x	top 4x	top 8x	top 16x	top 32x
Website availa.	57%	93%	94%	93%	91%	89%	86%
Website reacha.	93%	98%	97%	97%	96%	96%	95%
Email availa.	99%	84%	89%	91%	93%	94%	95%
Priv. policy availa.	9%	56%	50%	48%	38%	32%	26%

6. EVALUATION

6.1 Classifier Performance

We build a binary classifier based on the *Adaptive Boost* algorithm [19] to detect whether or not a given app is spam. 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 5. 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 $k = 1, 2, 4, \cdots, 32$.

Some of the features discussed in Section 5 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 non-spam 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[6] and we performed 500 iterations of the algorithm. Table 10 summarises the results. Our classifiers, while varying the value of k, have precision over 85% with recall varying between 38%–98%. Notably, when k is small (e.g., when the total number of non spam 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 larger k values implies inclusion of lower ranked apps as negative (non-spam) 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 non-spam, some spam related features become non-spam features as the number of apps satisfying that feature is high in non-spam. As a result recall drops when k increases.

As 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 non-spam 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 smaller number of negative examples (i.e., $k = 1$ or $k = 2$) can be used. For example, in Table 11 we show the $k = 2$ classifier's performance in the higher order datasets. As can be seen, this classifier identifies nearly 90% of the spam. However, the precision goes down as we traverse down the app ranking because of 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.

6.2 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

[6] *Adaptive Boost* algorithm combines the output of set of weak classifiers to build a strong classifier. Interested readers may refer to [19].

Table 10: 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 11: 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

Table 12: 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 %

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 which were not used for training during the classifier evaluation phase to avoid classifier artificially reporting higher number of spam. Table 12 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 3.1 and *Others* are the apps in $\mathbb{O}$ which 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 for 6 months, and by time we stopped monitoring they were still there in the market.

According to the results, more aggressive classifier ($k = 2$) predicted around 70% of the removed apps and 55% of the other apps to be spam. The conservative classifier ($k = 32$) predicted 6%–12% of the removed apps and approximately 2.7% of the other apps as spam; this can be considered as the lower bound for the percentage of spam apps currently available in Google Play Store.

7. CONCLUSIONS & FUTURE WORK

In this paper, 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 experiment suggests approximately 2.7% of the apps on Google Play Store are potentially spam. We plan to apply our rigorous manual labelling process to a sample of these apps and quantify the accuracy of our predictions. In this work, we used all the features mentioned in Section 5 to build the classifier while highlighting most discriminative features using *forward feature selection* on a manual checkpoint basis. Final weights of each features can be studied further to study the trade off between the number of features selected and the classifier's performance. Another interesting direction is to augment the proposed features with features extracted from the app binary and investigate whether the classifier performance can be enhanced.

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 non-spam apps. Our choices were dictated largely by the time consuming nature of the labelling process as mentioned in Section 3.2. Obviously, the performance of the classifier can be improved by increasing the number of labelled spam apps and further labelling non-spam apps through an extension of our manual effort.

One approach is to rely on crowdsourcing and recruit app-savvy users as reviewers. This in turn would require a major effort of providing individualised guidelines and interactions with the reviewers, and need to deal with assessment inconsistencies due to variable technical expertise. Nonetheless, from an app market provider perspective, this is a plausible option to consider. A framework that can quickly assess whether an app is spam, when the developers submit apps for approval will enable faster approvals whilst reducing the number of spam apps in the market.

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, known as *semi-supervised learning* [5]. The basic idea behind semi-supervised learning is to use "unlabelled data" to improve the classification model that was previously being built using only labelled data. Semi-supervised learning has previously being successfully applied to real-time traffic classification problems [16].

The classification arms race. It is possible that spammers adapt to the spam app detection framework and change their strategies according to our selected features to avoid the detection. The relevant questions in this context are: i) how frequently should the classifier be retrained? and ii) how to 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 non-spam apps. For instance, to avoid the similarity of descriptions of multiple apps, the spammer has to edit the different descriptions of the apps and to customise each of the app description to contain sufficient details and coherent text, etc. An important direction for future work is to study the longevity of our classifier. Additionally, we plan to 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 semi-supervised learning approaches [16].

8. ACKNOWLEDGEMENTS

We thank the anonymous reviewers and our shepherd Lenin Ravindranath Sivalingam for constructive feedback on preparation of the final version of this paper.

9. REFERENCES

[1] Language Detection API. http://detectlanguage.com, 2013.

[2] PrivMetrics. http://privmetrics.org/publications, 2014.

[3] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. D. Spyropoulos, and P. Stamatopoulos. Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. *arXiv preprint cs/0009009*, 2000.

[4] H. B. Aradhye, G. K. Myers, and J. A. Herson. Image analysis for efficient categorization of image-based spam e-mail. In *Proc. of 8th ICDAR*, pages 914–918. IEEE, 2005.

[5] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc. of the 10th KDD*, pages 59–68. ACM, 2004.

[6] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida. Detecting spammers on Twitter. In *Proc. of the 2010 CEAS*.

[7] E. Blanzieri and A. Bryl. A survey of learning-based techniques of email spam filtering. *Artificial Intelligence Review*, 29(1):63–92, 2008.

[8] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for Android. In *Proc. of the 1st SPSM*, pages 15–26. ACM, 2011.

[9] O. Canales, V. Monaco, T. Murphy, E. Zych, et al. A stylometry system for authenticating students taking online tests. Pace University, 2011.

[10] R. Chandy and H. Gu. Identifying spam in the iOS app store. In *Proc. of the 2nd WebQuality*, pages 56–59. ACM, 2012.

[11] P.-A. Chirita, J. Diederich, and W. Nejdl. Mailrank: Using ranking for spam detection. In *Proc. of the 14th CIKM*, 2005.

[12] G. V. Cormack, J. M. Gómez Hidalgo, and E. P. Sánz. Spam filtering for short messages. In *Proc. of the 16th CIKM*, 2007.

[13] J. Crussell, C. Gibler, and H. Chen. Andarwin: Scalable detection of semantically similar Android applications. In *Computer Security–ESORICS 2013*, pages 182–199. Springer.

[14] H. Drucker, S. Wu, and V. N. Vapnik. Support Vector Machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

[15] M. Erdélyi, A. Garzó, and A. A. Benczúr. Web spam classification: a few features worth more. In *Proc. of the 2011 WebQuality*, pages 27–34. ACM, 2011.

[16] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9-12):1194–1213, 2007.

[17] D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *Proc. of the 7th WebDB*, pages 1–6. ACM, 2004.

[18] R. Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221, 1948.

[19] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. of the 13th ICML*, 1996.

[20] J. M. Gómez Hidalgo, G. C. Bringas, E. P. Sánz, and F. C. García. Content based SMS spam filtering. In *Proc. of the 2006 DocEng*, pages 107–114. ACM, 2006.

[21] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proc. of the 36th ICSE*, pages 1025–1035, 2014.

[22] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Riskranker: Scalable and accurate zero-day Android malware detection. In *Proc. of the 10th MobiSys*, pages 281–294. ACM, 2012.

[23] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proc. of the 13th VLDB*, pages 576–587. VLDB Endowment, 2004.

[24] Apple. Inc. App store review guidelines. https://developer.apple.com/app-store/review/, 2014.

[25] Apple. Inc. Common app rejections. https://developer.apple.com/app-store/review/rejections/, 2014.

[26] AVG Threat Labs. Inc. Website safety ratings and reputation. http://www.avgthreatlabs.com/website-safety-reports, 2014.

[27] Google. Inc. Ads. http://developer.android.com/distribute/googleplay/policies/ads.html, 2014.

[28] Google. Inc. Google Play developer program policies. https://play.google.com/about/developer-content-policy.html, 2014.

[29] Google. Inc. Intellectual property. http://developer.android.com/distribute/googleplay/policies/ip.html, 2014.

[30] Google. Inc. Rating your application content for Google Play. https://support.google.com/googleplay/android-developer/answer/188189, 2014.

[31] Google. Inc. Spam. http://developer.android.com/distribute/googleplay/, 2014.

[32] Oracle. Inc. Naming a package. http://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html, 2014.

[33] N. Jindal and B. Liu. Review spam detection. In *Proc. of the 16th WWW*, pages 1189–1190. ACM, 2007.

[34] N. Jindal and B. Liu. Opinion spam and analysis. In *Proc. of the 2008 WSDM*, pages 219–230. ACM, 2008.

[35] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324, 1997.

[36] V. Krishnan and R. Raj. Web spam detection with anti-trust rank. In *Proc. of the 2006 AIRWeb*.

[37] B. Leiba, J. Ossher, V. Rajan, R. Segal, and M. N. Wegman. SMTP path analysis. In *Proc. of the 2005 CEAS*.

[38] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive Bayes-which naive Bayes? In *Proc. of 2006 CEAS*.

[39] G. Mishne, D. Carmel, and R. Lempel. Blocking blog spam with language model disagreement. In *Proc. of the 2005 AIRWeb*.

[40] A. Mukherjee and B. Liu. Improving gender classification of blog authors. In *Proc. of the 2010 EMNLP*, pages 207–217. Association for Computational Linguistics, 2010.

[41] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proc. of the 15th WWW*, pages 83–92. ACM, 2006.

[42] J. Oberheide and C. Miller. Dissecting the Android bouncer. https://jon.oberheide.org/files/summercon12-bouncer.pdf, 2012.

[43] P. Oscar and V. Roychowdbury. Leveraging social networks to fight spam. *IEEE Computer*, 38(4):61–68, 2005.

[44] P. Pantel, D. Lin, et al. Spamcop: A spam classification & organization program. In *Proc. of AAAI-98 Workshop on Learning for Text Categorization*, pages 95–98, 1998.

[45] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using probabilistic generative models for ranking risks of Android apps. In *Proc. of the 19th CCS*, pages 241–252. ACM, 2012.

[46] S. Perez. Developer spams Google Play with ripoffs of well-known apps again. http://techcrunch.com, 2013.

[47] S. Perez. Nearly 60K low-quality apps booted from Google Play Store in February. http://techcrunch.com, 2013.

[48] S. Perez. iTunes App Store now has 1.2 million apps, has seen 75 billion downloads to date. http://techcrunch.com, 2014.

[49] D. Rowinski. Apple iOS App Store adding 20,000 apps a month, hits 40 billion downloads. http://readwrite.com, 2013.

[50] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 workshop*, volume 62, pages 98–105, 1998.

[51] D. Sculley and G. M. Wachman. Relaxed online SVMs for spam filtering. In *Proc. of the 30th SIGIR*. ACM, 2007.

[52] S. Seneviratne, A. Seneviratne, D. Kaafar, A. Mahanti, and P. Mohapatra. Why my app got deleted: Detection of spam mobile apps. Technical report, NICTA, Australia, 2014.

[53] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti. Predicting user traits from a snapshot of apps installed on a smartphone. *ACM SIGMOBILE MC2R*, 18(2):1–8, 2014.

[54] R. Senter and E. Smith. Automated Readability Index. Technical Report AMRL-TR-66-220, Aerospace Medical Research Laboratories, 1967.

[55] I. Soboroff, I. Ounis, J. Lin, and I. Soboroff. Overview of the TREC-2012 microblog track. In *Proc. of the 21st TREC*, 2012.

[56] N. Viennot, E. Garcia, and J. Nieh. A measurement study of Google Play. In *Proc. of the 2014 SIGMETRICS*. ACM, 2014.

[57] A. H. Wang. Don't follow me: Spam detection in Twitter. In *Proc. of the 2010 SECRYPT*, pages 1–10. IEEE, 2010.

[58] Wikipedia. Wikipedia:lists of common misspellings. http://en.wikipedia.org/wiki/, 2014.

[59] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets. In *Proc. of the 2012 NDSS*.