Identifying Spam in Mobile Ad Networks using Latent Class Model

M. Sree Vani
Dept of CSE, MGIT, Gandipet, Hyderabad -500075

Abstract

Smart phone Apps plays a vital role to attract mobile-Advertising. Popular apps can generate millions of dollars in profit and collect valuable personal user information. spam, i.e., fraudulent or invalid tap or click on online ads, where the user has no actual interest in the advertiser's site, results in advertising revenue being misappropriated by spammers. It requires a user touch or click on control ads came from Smartphone-game Apps. It all need the user to tap the screen close to where the ad is displayed .While ad networks take active measures to block click-spam today, but not in mobile advertising. The presence of spam in mobile advertising is largely unknown. In this paper, we take the first systematic look at spam in mobile advertising. We propose a methodology to identify spam Apps in Smartphone-game Apps. We validate our methodology using data from major ad networks. Our findings highlight the severity of the spam in mobile advertising.

Keywords : *Spam, mobile apps, click spam.*

I. INTRODUCTION

Mobile advertisements within the apps are only source of revenue for several mobile app publishers. Maximum of the apps in the major mobile app stores show ads [1]. To embed ads in an app, the app developer typically registers with a third-party mobile ad network such as AdMob [2], iAd [3], Microsoft Mobile Advertising [4] etc. The ad networks supply the developer with an ad control (i.e. library with some visual elements embedded within). The developer includes this ad control in his app, and assigns it some screen real estate. When the app runs, the ad control is loaded, and it fetches ads from the ad network and displays it to the user. Different ad networks use different signals to serve relevant ads. One of the main signals that mobile ad networks use today is the app metadata [24]. As part of the registration process, most ad networks ask the developer to provide metadata information about the app (for e.g. category of the app, link to the app store description etc.). This allows the ad network to serve ads related to the app metadata. Ad networks also receive dynamic signals sent by the ad control every time it fetches a new ad. Depending on the privacy policies and the security architecture of the platform, these signals can include the location, user identity, etc. Note that unlike JavaScript embedded in the browsers, the ad controls are integral parts of the application, and have access to the all the APIs provided by the platform.

A. Background on Mobile Advertising

A typical mobile advertising system has five participants: mobile clients, advertisers, ad servers, ad exchanges and ad networks as Figure 2 shows. A mobile application includes an ad control module (e.g., AdControl for Windows Phones, AdMob for Android) which notifies the associated ad server any time an ad slot becomes available on the client's device. The ad server decides how to monetize the ad slot by displaying an ad. Ads are collected from an ad exchange. Ad exchanges are neutral parties that aggregate ads from different third party ad networks and hold an auction every time a client's ad slot becomes available. The ad networks participating in the exchange estimate their expected revenue from showing an ad in such an ad slot and place a bid on behalf of their customers (i.e., the advertisers). An ad network attempts to maximize its revenue by choosing ads that are most appropriate given the context of the user, in order to maximize the possibility of the user clicking on the ads. The ad network receives information about the user such as his profile, context, and device type from the ad server, through the ad exchange. Ad exchange runs the auction and chooses the winner with the highest bid. Advertisers register with their ad networks by submitting an ad campaign. A campaign typically specifies an advertising budget and a target number of impressions/clicks within a certain deadline (e.g., 50,000 impressions delivered in 2 weeks).

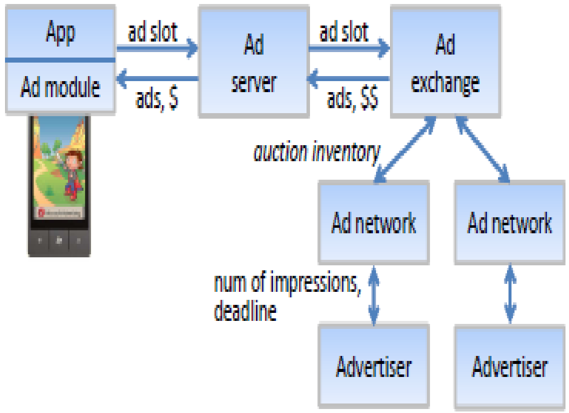


Figure 1. Architecture of a Typical Mobile ad System.

They can also specify a maximum cap on how many times a single client can see a specific ad and how to distribute ads over time (e.g., 150 impressions per hour). The ad server is responsible for tracking which ads are displayed and clicked, and thus determining how much money an advertiser owes. The revenue of an ad slot can be measured in several ways, most often by views (Cost Per Impression) or click-through (Cost Per Click), the former being most common in mobile systems. The ad server receives a premium on the sale of each ad slot, part of which is passed to the developer of the app where the ad was displayed.

B. *Background and Motivation for Spam in Mobile Advertising*

A mobile developer accidentally (or intentionally) places the in-app advertising control close to where the user must tap, or drag on usage of mobile. Given the tiny screen real-estate, the user is prone to mistapping. When he does so, the browser navigates to the ad-click URL. The user may realize his error and switch back to the game. The browser, which in the mean time has already begun fetching the ad landing-page, aborts the attempt. As a result, the user will appear to have spent very little time on the advertiser's page. We saw exactly this behavior on our mobile ads —95% of users spent less than a second as mentioned earlier.

The core issue here is the advertiser being charged despite the user not spending any time on the landing page. It is hard for an ad network to know how long the user spent on the advertiser's site. If it relied on the advertiser to get this information, the advertiser could easily lie to get a discount. Solving this without modifying the browser, and without hurting the user experience is a non-trivial problem. One mitigating approach would be to audit apps that trick users into mistapping on the ad. Doing so would likely spark an arms race for apps intentionally exploiting this loop-hole, but would at least protect advertisers from apps accidentally triggering this. Unfortunately, ad networks are making it harder for advertisers and independent third-parties to identify bad apps.

Having identified the spam problem, in this paper we propose a new framework for spam detection in mobile Apps. We propose a novel set of features particular to the mobile user gaming experience based features. We validate our methodology using data from major ad networks. We demonstrate the effectiveness of our approach via experiments on a datasets consist of all selected game apps crawled from the Apple iOS App Store in 2012. We conduct various experiments with our datasets. We would like to detect as many spam posts as possible while avoiding misclassifying non-spam posts as spam ones. Our approach gives us the best performance in terms of precision and recall, F-1 measure.

The rest of the paper is organized as follows. In Section 2, we review approaches for click spam detection in previous work. In Section 3, we introduce our methodology for spam detection in Mobile Apps. Section 4 presents novel set of features used in the classifier. Section 5 describes our experiment setup and shows experimental results. Finally, our conclusions and future directions are presented in Section 6.

II. RELATED WORK

Existing works on ad fraud mainly focus on the click-spam behaviors, characterizing the features of click-spam, either targeting specific attacks [5, 6, 16, 18], or taking a broader view [7]. Some work has examined other elements of the click-spam ecosystem: the quality of purchased traffic [19, 20], and the spam profit model [12, 13]. Very little work exists in exploring clickspam in mobile apps. From the controlled experiment, authors in [7] observed that around one third of the mobile ad clicks may constitute click-spam. A contemporaneous paper [9] claimed that they are not aware of any mobile malware in the wild that performs advertising click fraud. DECAF focuses on detecting violations to ad network terms and conditions, and even before potentially fraudulent clicks has been generated. With regard to detection, most existing works focus on bot-driven click spam, either by analyzing search engine query logs to identify outliers in query distributions [52], characterizing networking traffic to infer coalitions made by a group of bot-driven fraudsters [14, 15], or authenticating normal user clicks to filter out bot-driven clicks [10, 11, 49]. A recent work, Viceroi [8], designed a more general framework that is possible to detect not only bot-driven spam, but also some non-bot driven ones (like search-hijacking).To the best of our knowledge, ours is the first work to detect touch spam in mobile apps.

III. METHODOLOGY

This section presents an overview of our approach for spam detection in mobile Apps. We propose a simple latent class model to capture the relationship among the user gaming experience, app, user, and developer. Figure 2 shows the structure of the graphical model. For each entity of interest, we assign it a feature node and a latent node which represents the latent class. The latent class model assumes that the feature is generated from the unobserved class, and is independent of other nodes given the class. Furthermore, we assume that the developer only directly affects the app.

All latent class variables in this model are chosen to be binary. Ia indicates good or bad apps, Id indicates good or bad developers, Iu indicates normal or malicious users, and Ie indicates truthful or spam experiences. Table 1 summarizes the features and the conditional probability model at each node.

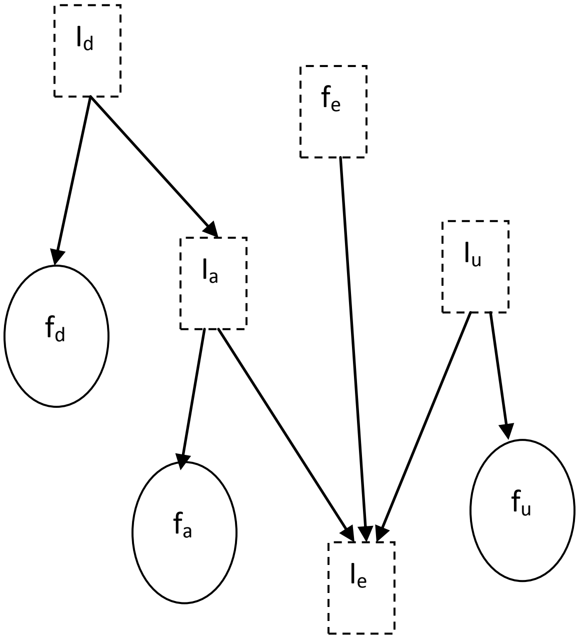


Figure 2: Latent Class Model for user Experience, User, App, and Developer.

Table 1: Features and CPD

Node	Features	CPD
Fu	User-avg-rating,user-num-rev	Conditional Gaussian
Fa	App-avg-rating,app-num-rev	C nditional Guassian
Fe	I(stars<=2) I(stars<=3) I(stars<=4)	NA
Fd	Dev-num-app,deb-avg-rating	Conditional Guassian
Ia,Id,Iu,Ie	Binary class indicator	CPT

IV. FEATURES

To build classifiers, we extract numerous features from the App page. We first use an App features. Then we extract App Developer features from App. Then we do deeper analysis to extract more mobile ad-control location based features including ones using external sources of information.

A. *App Features*

Developers of spam apps (malicious developers) are primarily interested in gaining monetary profit or leaching valuable user data, such as address book contacts. Popular, seemingly legitimate apps can leak user data quietly [22, 23], so it is feasible that spam apps would attempt to do the same. In the App Store, each app has its own webpage, which displays app price, screenshots, description, ratings and text reviews left by users who downloaded the app, and related metadata. Ratings are integer stars in the range 1-5. Similar to other online shopping platforms, positive reviews are crucial for convincing potential customers to purchase the app. We extract App features from above information like App ID, Developer ID, Price, Category ID, app popularity, Release Date, Current Version.

B. *Developer Features*

For malicious game App developers, spamming the App Store can be beneficial and is not difficult. A mobile game developer accidentally (or intentionally) places the in-app advertising control close to where the user must swipe or tap, or drag things to, in order to succeed in the game. Given the tiny screen real-estate, the user is prone to miss tapping. When he does so, the browser navigates to the ad-click URL. Mobile app developers are incentivized to commit such fraud since ad networks pay app publishers based on impression count [25, 24, 26]. We extract Developer features like Developer ID, Number of Apps, Avg App Rating, Avg Number of App Versions, Avg Review Helpfulness, and Proportion of Free Apps.

C. User Experience Based Features

Ad networks usually impose strict guidelines to advertisers on how ad controls should be used in apps, documented in lengthy Publisher Terms and Conditions. Based on these guidelines, we extract the features relate to how and where the ad control is placed from user exeriences. Ad networks impose placement restrictions to prevent impression or click inflation, while the advertiser may restrict what kinds of content (i.e., ad context) the ads are placed with. For instance, Microsoft Mobile Advertising stipulates that a publisher must not "edit, resize, modify, filter, obscure, hide, make transparent, or reorder any advertising" and must not "include any Ad Inventory or display any ads ... that includes materials or links to materials that are unlawful (including the sale of counterfeit goods or copyright piracy), obscene,." [28]. Similarly, Google AdMob's terms dictate that "Ads should not be placed very close to or underneath buttons or any other object which users may accidentally click while interacting with your application" and "Ads should not be placed in areas where users will randomly click or place their fingers on the screen" [27].Violators may manipulate the UI layout to inflate impressions, or increase none ad screen real estate. From a large dataset of apps, we extract the following features of misplacement of Ad-control which are mainly leads to ad touch spam and how these are vary with app rating, the category of the app, and other factors.

Number of Ad-controls An app page may contains too many ads , while Microsoft Advertising allows at most 1 ad per phone screen and 3 ads per tablet screen [28].Therefore, if any app contains the number of viewable ads in a screen is more than k, the maximum allowed number of ads then it is a violator.
Visibility of Ad-controls Hiding the Ads behind other controls (e.g., buttons or images) or placed outside the screen also violates the terms and conditions in [28, 27]. Developers often use this trick to give users the feel of an "ad-free app", or to accommodate many ads in a page when ad networks visually inspect for ad count violations.

We extract this feature from app page, if any ad in the given page is (partially) hidden or unviewable. For each ad, the detector first finds non-ad GUI elements that overlap with the Ad. Then it checks if any of these non-ad elements is rendered above the Ad. To get this we are traversed depth-first order of the DOM tree of app page.

Size of Ad-control, Changing size of Ads too small for users to read, violates the terms and conditions. We extract this feature from app page if any ad in the given page is smaller than the minimal valid size required by the ad network.

V. EXPERIMENTS

A. Datasets

The datasets consist of all selected users experiences for selected apps crawled from the Apple iOS App Store in 2012. From this data, we computed metadata for apps, developers, and users who post their gaming experiences. We obtained two datasets: the Top gameApps(TGA) dataset containing user gaming experiences and game App and the Entertainment & Gaming(E&G) dataset containing user gaming experiences and metadata for all apps in the Entertainment and Gaming categories. In addition, we created a third dataset, Labeled TGA, which contains a randomly chosen subset of apps from TGA having more than twenty user gaming experiences with app spam binary labels acquired through manual inspection. The size of each dataset is shown in Table 2.

Figure 3 shows CCDFs for the TGA and E&G datasets. In Figure 3a, we observe that the CCDFs go at integer rating values, with noticeable drops immediately before these values. As expected, the TGA CCDF remains higher than the E&G CCDF at all rating levels in this plot, with the difference increasing with rating. This is because the TGA dataset contains many of the best apps in the entire store, while the E&G dataset simply contains all apps from two categories.

Table 2: Sizes of Top Apps (TGA), Entertainment &Gaming (E&G), and Labeled E&G datasets.

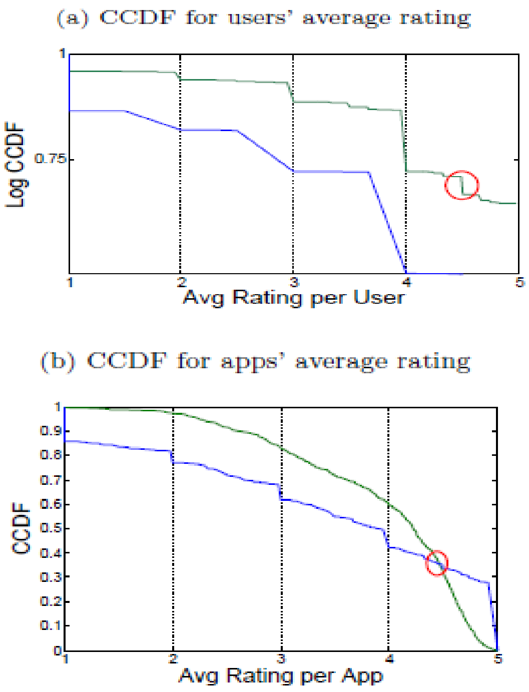
	TGA	E&G	Labeled E&G
#Apps	690	2,400	114
#user gaming Exeriences	4,416,800	35,035	33,130
#Users	2,217,500	32,700	32,900
#Develoers	350	2000	100

Also, the drop near 4.5 in the TA CCDF (circled in red) indicates that many users would find an intermediate rating between 4 and 5 useful for delineating the very best apps. Figure 3b conforms this near the 4.5 rating level and also shows how the best apps differ from the rest.

B. Results

As a baseline method to classify app spam, a pruned Decision Tree was trained on app and developer features from the Labeled E&G dataset. In addition to the decision tree, we tested our method. For simplicity and interpretability, we choose the two most common used features for each observed node. We choose Linear Gaussian to model the conditional probability of a feature node given its latent class. We also simplify the user experience feature to be a class indicator of high, middle and low. Hence, it is convenient for us to put priors on the CPT based on heuristics such as if conditioning on a dishonest user, a high quality app, and a low experience, the app is more likely to be spam.

Figure 3: CCDFs from the E&G (blue) and TGA (green) Datasets.


(a) CCDF for users' average rating

(b) CCDF for apps' average rating

Table 4: Learned Parameters on Latent Nodes. The first Column Contains the Value of the Variable.

	$P(I_u)$	$P(I_d)$	$P(I_a /I_d =0)$	$P(I_a /I_d =1)$
0	0.11	0.13	0.92	0.17
1	0.87	0.87	0.09	0.85

The unsupervised learning is run on the E&G dataset. The goal is to cluster reviews using the latent node Ir. We start with a uniform prior for Ia, Id, Iu. We set a prior on P (Ie/Ia, Iu, fe) to encode common beliefs on a user gaming experiences truthfulness based on user's honesty, user experiences rating, and app's quality. We run Expectation Maximization [3] for 6 iterations with a Junction Tree inference algorithm provided by the Bayesian Network Toolbox (BNT) [8]. Note that, although our goal is to cluster spam experiences, having other latent nodes in the model provides a clustering on the users, apps, and developers as a free byproduct. Tables 4 and 5 show the parameters learned from EM.

Table 5: Learned Parameters on Feature Nodes.

Id	$\mu(f_d)$	$\sigma^2(f_d)$
0	(2.5,1.5)	(0.6,0.7)
1	(4.2,1)	(0.01,0.18)

(a) parameters of fd/Id

Ia	$\mu(f_a)$	$\sigma^2(f_a)$
0	(3.3,71)	(1.2,6701)
1	(4.2,432)	(0.01,90500)

(b) parameters of fa/Ia

Iu	$\mu(f_u)$	$\sigma^2(f_u)$
0	(3.9,1)	(0.01,2.13)
1	(4.0,1.5)	(0.51,1.47)

(c) parameters of fu/Iu

In Table 5a and 5b, the conditional mean of average rating of apps and developers agrees with the intuition that higher quality apps and developers receive higher ratings. Apps from class 1 (good quality) receive more reviews than apps from class 0, and the variance is much higher in class 1. However, we notice that the number of apps feature for developer is similar for both classes, because most of the developers have only 1 or 2 apps in this dataset. Also, the parameters of user features are similar for both classes because most of the users only posted one experience. Therefore average rating of users does not provide enough information for clustering users. The marginal probability of latent class in Table 4 shows that the prior belief on users, apps, and developers is heavily favored toward 1 (good class).

VI. CONCLUSION

In this paper we propose an approach for spam detection in Mobile game apps. To the best of our knowledge, our work is the first attempt for spam detection in this important domain. First, we propose a new framework for spam detection. Second, we propose a novel set of features particular to the mobile user gaming experience based features that discriminate spam from nonspam. Third, we demonstrate the effectiveness of our approach via experiments on a datasets consist of all selected game apps crawled from the Apple iOS App Store. We propose a latent class model with interpretable structure and low complexity. On the labeled data set, even though we use the simple Linear Gaussian parameterization, it still achieves significantly higher accuracy than a baseline Decision Tree. On the unlabeled data set, it succeeds in clustering the apps and user experiences into well separated groups. Future work could explore extending our Latent Class graphical model to adopt more features with a different parameterization.

REFERENCES

[1] S.Ganov, C. Killmar, S. Khurshid, and D. Perry. Event listener analysis and symbolic execution for testing gui applications. In ICFEM, 2009.

[2] Google admob. http://www.google.com/ads/admob/.

[3] iad app network. http://developer.apple.com/support/appstore/iad-app-network/.

[4] Microsoft advertising. http://advertising.microsoft.com/en-us/splitter.

[5] S.Alrwais, A. Gerber, C. Dunn, O. Spatscheck,M. Gupta, and E. Osterweil. Dissecting ghost clicks: Ad fraud via misdirected human clicks. In ACSAC, 2012.

[6] T.Blizard and N. Livic. Click-fraud monetizing malware: A survey and case study. In MALWARE,2012.

[7] P Chia, Y. Yamamoto, and N. Asokan. Is this app safe? a large scale study on application permissions and risk signals. In WWW, 2012.

[8] [8] V. Dave, S. Guha, and Y. Zhang. Measuring and fingerprinting click-spam in ad networks. In ACM SIGCOMM, 2012.

[9] C.Cadar D. Dunbar and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In USENIX OSDI, 2008.

[10] P.Gilbert, B. Chun, L. Cox, and J. Jung. Vision:automated security validation of mobile apps at app markets. In MCS, 2011.

[11] H.Haddadi. Fighting online click-fraud using bluff ads. ACM Computer Communication Review, 40(2):21–25, 2010.14

[12] C.Hu and I. Neamtiu. Automating gui testing for android applications. In AST, 2011.

[13] A.MacHiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for android apps. In FSE, 2013.

[14] A.Mesbah and A. van Deursen. Invariant-based automatic testing of ajax user interfaces. In ICSE, 2009.

[15] Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. ACM Transactions on the Web, 6(1):1–30, 2012.

[16] A.Metwally, D. Agrawal, and A. El Abbadi. Detectives:Detecting coalition hit inflation attacks in advertising networks streams. In WWW, 2007.

[17] A.Metwally, F. Emekci, D. Agrawal, and A. El Abbadi.Sleuth: Single-publisher attack detection using correlation hunting. In PVLDB, 2008.

[18] B Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson. What's clicking what? techniques and innovations of today's clickbots. In DIMVA, 2011.

[19] L.Ravindranath, J. Padhye, S. Agarwal, R. Mahajan,I. Obermiller, and S. Shayandeh. Appinsight: mobile app performance monitoring in the wild. In USENIX OSDI, 2012.

[20] W.Yang, M. Prasad, and T. Xie. A grey-box approach for automated gui-model generation of mobile applications. In FASE, 2013.

[21] M.Najork. Web spam detection. In L. Liu and M. T.□Ozsu, editors, Encyclopedia of Database Systems, pages 3520{3523. Springer US, 2009.

[22] Nick Bilton. Disruptions: So Many Apologies, So Much Data Mining.

[23] http://bits.blogs.nytimes.com/2012/02/12/disruptions-so-many-apologies-so-much-data-mining, 2012.

[24] Peter Gilbert, Byung-Gon Chun, Landon P Cox, and Jaeyeon Jung. Vision: automated security validation

[25] of mobile apps at app markets. In Proceedings of the second international workshop on Mobile cloud

[26] computing and services - MCS '11, page 21, New York, New York, USA, 2011. ACM Press.

[27] Google admob: What's the difference between estimated and finalized earnings? http://support.google.com/adsense/answer/168408/.

[28] Microsoft advertising: Build your business. http://advertising.microsoft.com/en-us/splitter.

[29] iad app network. http://developer.apple.com/support/appstore/iad-app-network/.

[30] Admob publisher guidelines and policies. http://support.google.com/admob/answer/1307237?hl=en&ref topic=1307235.

[31] Microsoft pubcenter publisher terms and conditions.http://pubcenter.microsoft.com/StaticHTML/TC/TC en.html.

[32] L. Breiman. Bagging predictors. Machine Learning,24(2):123{140, 1996.

[33] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In European Conference on Computational Learning Theory, pages 23{37, 1995.

[34] J.R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco,CA, USA, 1993.