An Ensemble Learning Approach to Detect Touch-Spam in Mobile Ad-Networks

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

Abstract

A touch user interface (TUI) is a computer-pointing technology based upon the sense of touch (haptics). Touch-spam is a type of fraud that occurs over TUI gadgets ex. Smart phones, tablets, phablets, touch laptops etc. It actually happens in TUI applications when a person, automated script, computer program or robotic action imitates a legitimate user of a TUI application touching on an advertisement (ad), for the purpose of generating a charge per touch without having actual interest in the target of the ad's popup. Touch-spam is becoming an issue due to the advertising networks being a key beneficiary of this spam. In present days, smart phone gaming applications (apps) are playing a vital role to attract mobile-advertisements (ads) since their pocket portability and other versatile features. Popular apps are able to read the user personalized data to process user interests helping to generate customized ads. Touch-spam in smart phone apps is a fraudulent or invalid tap or touch on online ads, where the user has no actual interest in the advertiser's site. It requires a user touch on online ads that pop-up dynamically in smart phone gaming apps. It all need the user to tap the screen close to where the ad is displayed .While the ad networks continue taking active measures to block click-spam today, the touch-spam still creeping under the TUI. It is being used by spammers to misappropriate the advertising revenue. The presence of touch-spam is largely unknown. In this paper, we take the first systematic look at touch-spam. We propose an ensemble learning approach to identify touch-spam Apps in Smartphone-game Apps. We validate our methodology using data from major ad networks. Our findings highlight the severity of the touch-spam problem.

Keywords : *Spam, mobile apps, touch spam, 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).

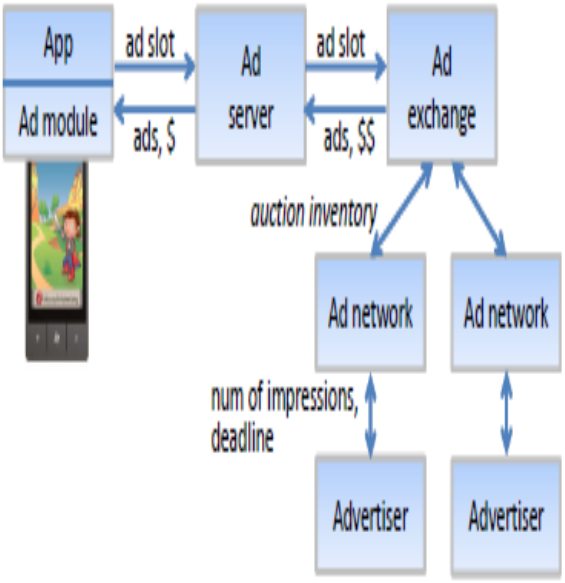


Figure 1:An Architecture of Mobile Ad Network

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).

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.

B. Background and Motivation for Touch Spam

A mobile game developer accidentally (or intentionally) places the in-app advertising control close to where the user must tap, or drag things to, in order to succeed in the game. 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 games and 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. The most beneficiary is app, because the app made money from the ad network. Our studies shows that at least 2% of clicks on control ads came from smartphone games that all require the user to tap the screen close to where the ad is displayed. One such example is the Ant-smasher iPhone app where ants randomly walk around the screen up to (and under) where the ad is shown in the game, and the user must tap the ant before it disappears from the screen to progress in the game.

Figure 2: Illustration of Click-Spam

Having identified the touch-spam problem, in this paper we propose a new framework for touch-spam detection in mobile game Apps. We propose a novel set of features particular to the mobile advertisement-control location 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 dataset using Weka, We would like to detect as many spam posts as possible while avoiding misclassifying non-spam posts as spam ones. Classifiers built using J48, an implementation of C4.5 [24] decision tree learning

algorithm, give 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 overall framework for touch-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 have 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. FRAMEWORK

This section presents an overview of our approach for touch-spam detection in mobile Apps. The overall framework is described in Figure 2. First, we extract the App features from Apps. Given the information of App, we extract App developer features. These features traditionally used in previous work for spam detection in IOS App store [99(identifying sam ios store]. we propose a novel set of features particular to the mobile advertisement-control location based , e.g. Ad-control is located at underneath buttons or any other object which users may accidentally click while interacting with your application or users will randomly click or place their fingers on the screen. These features are also defined Using external resources collected from Mobile game user experiences. Finally, given a feature vector for each App, we transform the spam detection problem into a classification problem for which we can use many

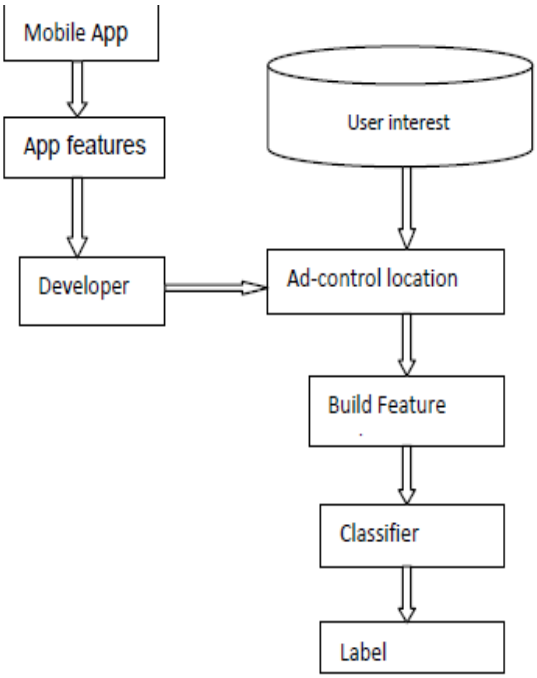


Figure 3:Framework for Classification of Spam Apps

well established tools and techniques to solve. Like some previous works in detecting spam content on web, we apply a decision tree classifier to classify the App into spam or non-spam category [21].

IV. FEATURES

To build classifiers, we extract numerous Group 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. *Ad-control Location 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. 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.

Misplace of Ad-control on tiny screen Ads are partially hidden or placed next to actionable control such as buttons to capture accidental clicks. We extract this feature from app if the distance between an ad control and a clickable non-ad element is below a predefined threshold or if an ad control partially covers a clickable non-ad control.

User interest We collect information from server logs. We extract this feature from app if user interest ratio is below a predefined threshold.

$$user\ interest = \frac{number\ of\ times\ Ads\ are\ visited\ by\ user > 1sec}{number\ of\ clicks\ on\ Ads\ by\ users}$$

V. EXPERIMENTS

The datasets consist of all selected game apps crawled from the Apple iOS App Store in 2012. We collected 13,267 top free game apps from App Store. From this data, we computed metadata for apps, developers, and users who post reviews .We divide the whole data sets into two parts as training dataset and test dataset. We then went for manual labeling of training datasets. To label apps as spam or non spam, we invited some volunteers who had experience with mobile game playing to participate. After labeling process, we have 81 spam posts (17%) and 401 non spam posts (83 %.).

A. *Results*

We use Recall, Precision and F-Measure to measure the performance of classifier. Assume that we have the D classifier. We use D to classify a set of input pages. The possible output of D can be represented using a confusion matrix as in Table1.

Table 1:Confusion Matrix

spam	non-spam	classified as
a	b	spam
c	d	non-spam

The following formulas are used to calculate Recall, Precision, and F-Measure.

$$\text{Recall: } R = \frac{a}{a \pm b} \quad (1)$$

$$\text{Precision: } P = \frac{a}{a \pm c} \quad (2)$$

$$\text{F-Measure: } F - Measure = 2\frac{PR}{P \pm R} \quad (3)$$

We conduct various experiments with our dataset using Weka, a popular machine learning software suite implementing commonly used machine learning algorithms. We would like to detect as many spam touches as possible while avoiding misclassifying non-spam touches as spam ones. Classifiers built using J48, a decision tree learning algorithm, an implementation of C4.5 [31] give us the best performance with our experiments. Table 2 summarizes our experiment results.

To improve the performance of our classifier, we try to use boosting [30] and bagging [29] techniques. However, our experiment results show that boosting and bagging cannot help much. From the experimental results, we can see that the Ad-control location features help improve classifier's performance significantly in all True Positive rate, False Positive Rate, and F-Measure.

Figure 4 plots Precision-Recall curve as the threshold parameter is varied. Recall (same as true-positive rate) tracks what fraction of click-spam. Precision tracks the fraction of true positives i.e., the more false positives we admit for a given recall, the lower the precision.

Table 2: Performance Measures with Different Ensemble Algorithms

Algorithm	Precision	Recall	F-Measure	ROC
Bagging FT Tree	0.968	0.969	0.977	0.930
Bagging RandomForest	0.998	0.998	0.997	0.935
Adaboost REPTree	0.987	0.987	0.980	0.933
Adaboost LADTree	0.976	0.970	0.977	0.933
Stacking NBTree	0.968	0.969	0.977	0.922

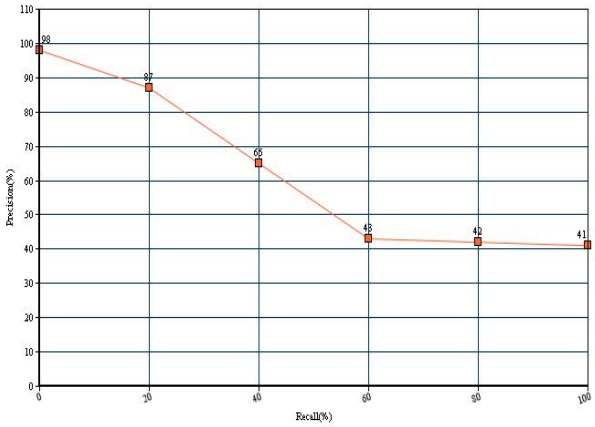


Figure 4: Precision-Recall Curve

The performance of the models was evaluated based on the average precision as shown in Table 3.

$$AP = \frac{1}{m}\sum_{i=1}^{k} precision(i)$$

Where Precision(i) denotes the precision at cutoff i in the publisher list, i.e., the fraction of correct fraud prediction up to the position i, and m is the number of actual fraud publishers. Note that, when the

ith prediction is incorrect, Precision (i) = 0. In our experimental setting, we set k=260 for evaluation.

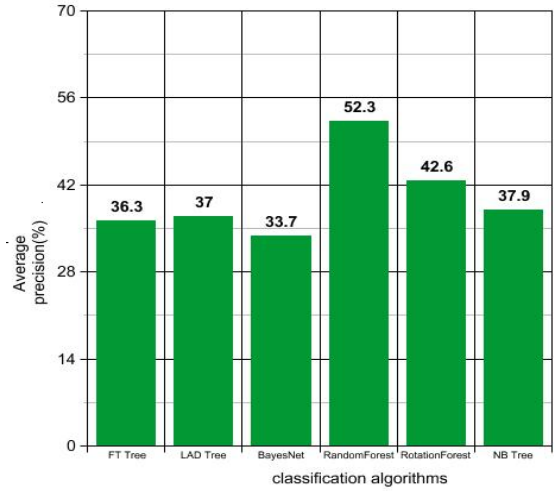
Average precision works on the prediction ranking rather than actual number of prediction to OK and Fraud classes.

Table 3: Performance for Different Ensemble Classification Algorithms on the New set of Features

Method	Average Precision
Bagging FT Tree	36.3%
Bagging RandomForest	52.3%
Adaboost REPTree	35.8%
Adaboost LADTree	37%
Stacking NBTree	37.9%
RandomsubSpace	38.9%
RotationForest	42.9%
BayesNet	33.7%
RPROP	48.3%

The Random Forest classification algorithm used with the new set of features for classifying spam web pages showed that around 52.3 percent of the publishers are involved in spam as shown in Figure 5.

Figure



5: Average precision of the Different CLassification Algorithms

VI. CONCLUSION

In this paper we propose an approach for touch-spam detection in Mobile game apps. To the best of our knowledge, our work is the first attempt for touch-spam detection in this important domain. First, we propose a new framework for touch-spam detection. Second, we propose a novel set of features particular to the mobile advertisement-control location 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 conduct various experiments with our dataset using Weka, Classifiers built using J48, an implementation of C4.5 decision tree learning algorithm, give us the best performance in terms of precision and recall, F-1 measure. For future work, we plan to extend the experimental dataset. Identifying good instances for judgment is itself an interesting problem. Since this is a highly imbalanced classification problem, if we randomly pick a sample of instances for training datasets, there are very few spam instances in the sample. To overcome this, we plan to use active learning technique to pick training dataset instances that are likely to be spam.

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/enus/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] 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. Conference [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.
[15] A.Metwally, D. Agrawal, and A. El Abbadi. Detectives:Detecting coalition hit inflation attacks in advertising networks streams. In WWW, 2007.
[16] A.Metwally, F. Emekci, D. Agrawal, and A. El Abbadi.Sleuth: Single-publisher attack detection using correlation hunting. In PVLDB, 2008.
[17] B.Miller, P. Pearce, C. Grier, C. Kreibich, and V. Paxson. What's clicking what? techniques and

[18] innovations of today's clickbots. In DIMVA, 2011. [19] L. Ravindranath, J. Padhye, S. Agarwal, R. Mahajan,I. Obermiller, and S. Shayandeh. Appinsight:
[19] 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. http://bits.blogs.nytimes.com/2012/02/12/disruptions-so-many-apologies-so-much-data-mining, 2012.
[23] Peter Gilbert, Byung-Gon Chun, Landon P Cox, and Jaeyeon Jung. Vision: automated security validation of mobile apps at app markets. In Proceedings of the second international workshop on Mobile cloud computing and services - MCS '11, page 21, New York, New York, USA, 2011. ACM Press.
[24] Google admob: What's the difference between estimated and finalized earnings? http://support. google.com/adsense/answer/168408/.
[25] Microsoft advertising: Build your business. http: //advertising.microsoft.com/en-us/splitter.
[26] iad app network. http://developer.apple.com/support/appstore/iad-app-network/.
[27] Admob publisher guidelines and policies. http://support.google.com/admob/answer/1307237?hl=en&ref topic=1307235.
[28] Microsoft pubcenter publisher terms and conditions.http://pubcenter.microsoft.com/StaticHTML/TC/TC en.html.
[29] L.Breiman. Bagging predictors. Machine Learning,24(2):123{140, 1996.
[30] 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.