Tagging Click-Spamming Suspicious Installs in Mobile Advertising Through Time Delta Distributions

Juan de Monasterio[1]

Jampp

Abstract. Fraud in the mobile advertising world is a topic gaining momentum recently. Different reports agree that invalid traffic is generating losses in the order of billions of dollars[1] and that there is a significant amount of fraud with ongoing efforts against it[5]. Here at Jampp[1] we use an automated fraud detection algorithm for a recent type of mobile advertising fraud, which can be referred to as click-spamming, click-injection or mobile-hijacking. We propose a metric to measure suspicious installs, and use a heuristic to compare the fits of theoretical distributions to this metric. This allows us to derive a threshold for suspicious installs. Our metric is based on the time-delta distributions, which amounts to the time it takes from a click in an ad to be converted into an install. The model is currently in use with satisfactory results. To the best of our knowledge, this is the first algorithm in production used to tackle this specific kind of fraud.

1 Introduction

The mobile advertising industry consists mostly of two big groups of agents: advertisers and publishers. Advertisers pay to place their ads on apps owned by publishers. This process of ad-serving is what is defined as "traffic". Jampp is a demand-side platform (DSP) that allows companies to advertise their mobile applications through publishers' traffic. In short, advertisers pay to drive traffic from publishers' apps to their own apps.

The industry is largely composed of business contracts where advertisers agree to pay a sum of money over a fixed amount of mobile traffic over a given period. Traffic quality is then evaluated according to the user's actions in the advertiser's apps, in response to the ads served. The user actions will certainly depend on the app in question. However, all of them have one common in-app-event which is the install, or the app's first open. In this work we will center our analysis on these events.

Jampp's mission is to help companies grow their mobile business by engaging users and driving new customers. Monthly traffic volume averages around 2.5 billion impressions and 2 million installs on paid traffic. Non-paid traffic, also known as organic traffic[2], refers to the visitors that install or use the advertisers app naturally. This is the opposite of paid traffic, and amounts to 750 million logs per day.

Similar to the desktop ad industry, there are fraudsters intending to abuse profits of traffic advertising. This problem has recently spun attention to companies, government[7], NGOs such as IAB[3], MMA[4] and MRC[5]. In this environment, fraud results not only in investment losses but also in skewed business metrics.

[1] `www.jampp.com`

[2] Organic traffic might in fact be paid for by other DSP platforms and not by Jampp, yet we have no way to distinguish organic traffic from non-Jampp paid traffic.

[3] www.iab.com/

[4] www.mmaglobal.com/

[5] mediaratingcouncil.org/

In this whole ecosystem, revenue models differ across DSPs and advertisers where the quality depends on the volume of events generated by the paid traffic.

Events are both app installs and in-app-events (i.e. purchases, taxi rides, etc.).

Contractual agreements set goals on the expected volume of response events that will arise from an advertiser's spend. Nowadays, the industry's most widely used revenue model is cost-per-install (CPI)[6]. Therefore, there is no pay on the volume of impressions served or on the clicks caused by them.

For any given transaction, starting from an impression all the way to an in-app-event, there are various systems put in place to track the response events generated by them. These user activities are tracked by different agents along the successive chain of actions:

- On the publisher's side, Jampp is receiving click and impression messages, among other things. The user's device ID, publisher, timestamp, context of this click or impression, and other relevant information are logged and messaged to Jampp.
- On the advertiser's side, we receive similar logs from user's installs and events using SDKs integrated into the apps.

The information of every message is received by our platform, which records and forwards it, when necessary, to other actors along the chain. Note that, by this definition, any given ad transaction such as a click or an impression may have multiple response events to that ad. The same user might install an app and then perform successive in-app-events, all in a time period brief enough that the same ad transaction is credited or attributed for these actions.

As a short example, we present here a user's experience in the mobile ad ecosystem for a hypothetical taxi hailing application.

Publisher's Side	Advertiser's Side
Impression of Ad → Click on Ad →	**In-App Events**
	App Install
	Registration
	Search
	Taxi Ride

The general flow of users from an ad in a publisher's site, the click leads them to the App/Playstore and, after installing the app, the advertiser would then log users opening the app as an install. In this hypothetical taxi app, users would then register to use the service and search for available taxi rides. Finally, they would order a taxi ride.

In reality, what we refer to as impressions, clicks, installs, and in-app events are actually HTTP requests made between devices and servers, in response to user actions. User's device IDs, user-agent and other metadata are URL-encoded in the request and these will bounce around different servers, between actors in the ecosystem.

The way an advertiser establishes whether an install is due to organic activity or not[6] is by the attribution method. The most widely used method by advertisers is to attribute the app's install to the DSP that last sent a user's click on an ad for that app before installation. Recall that an "install" is a user's first open of an app.

At the moment, the industry has established this as an attribution model to determine which ads deserve the credit for the user actions. The difficulty lies in verifying that a user flow is genuine, i.e. determining if the publisher's reports of user actions effectively did happen and aren't simulated. In all cases, this means establishing that a real touch on the screen has been made, in representation for the click or a real ad view, in representation of an impression. Faking

[6] Organic events are defined in the industry as events that are not paid for.

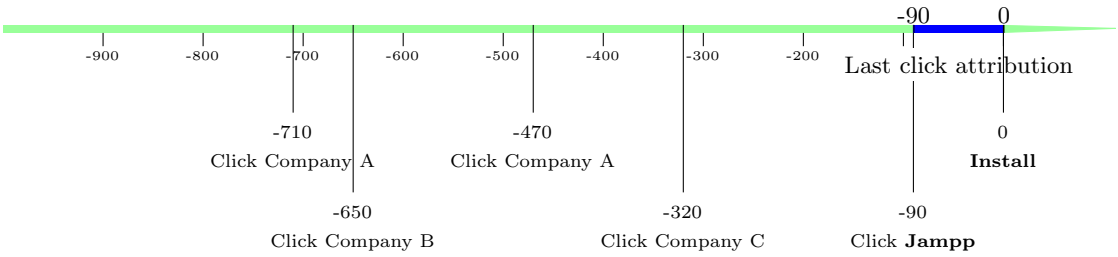

Fig. 1. Timeline of user activity, attributing an install to Jampp.

a request is essentially not difficult since, in technical terms, it only means information flowing between servers and with a specified format. Unlike other industries, fraudulent scenarios are virtually impossible to determine with absolute certainty. Here, legitimate ad transactions can be confused with fraudulent ones.

2 Click-Spamming Fraud

Click-Spamming is a type of sophisticated fraud[3], which is defined by the property that significant human analysis and intervention are needed in order to detect it. It involves a publisher generating fake user click requests to mis-attribute organic installs and simulate good quality traffic.

The fraudster's intent is to steal organic installs by gaming the attribution method. This is usually done by randomly generating thousands of user's click requests with computer programs[4]. The fraudster would then send click requests with different timestamps with a given user device ID. However, he wouldn't be able to produce installs requests (clicks before the install hast taken place), since those can only be generated by the advertiser's servers, and in a special URL parameterized format. This install request information is only exchanged between the DSP and the advertiser. Keeping this in mind, we have that the time difference between the click and the install occurrence is an uncontrolled random event for the fraudster.

Traditional methods of click-spamming detection involve checking CVR (conversion rate, i.e. click-to-install rate) and ER (install-to-events rate). With this type of fraud, advertising campaigns would see huge volumes in clicks and impressions, low CVRs and average in-app-event rates, when compared to other non-fraudulent publishers. However, there is no certainty on which levels of rates would label a fraudulent publisher. We can only tag suspicious publisher activity.

Given this problem setting, we present this as an anomaly detection problem under the umbrella of unsupervised learning, where we would want to classify instances of suspicious fraud[7].

2.1 Time Delta Metric

To counter the fraud effect of click-spamming activity, we derive the Time Delta metric. This is a simple measurement of an install to detect cases of Click-Spamming. Given a transaction, the metric δ^T is $t_1 - t_0$ where t_0 is the time when the click was made, and t_1 is the time when the app was opened by the user. It is important to note that the idea behind this metric is that publishers would be in control of the information sent in the fake clicks, yet they will not

[7] For a list of available unsupervised learning algorithms for fraud detection, please refer to[2].

be able to control the information sent in an install message from the advertiser. In practice, this disorder between click and install timestamps does produce statistical outcomes that are indicative of fraudulent behavior. Due to this, we focus on the empirical distribution defined by this metric, in an advertiser's campaign.

Consider a time window T. Let A be the set of apps (or Advertisers) and, without loss of generality, consider a to be a generic app. From our installs' data, we assume that, for any given instance $a \in A^T$ (during T), the Time Delta measurements are i.i.d random variables $\delta_1, \ldots, \delta_n$[8]. That is, if we consider E to be the set of empirical distributions, we have a mapping $D : A \to E$ such that

$$a \ \to \ \delta_1, \ldots, \delta_{n_a}$$

which maps apps to their Time Delta measurements. The idea behind this metric is that, in general, counting data will statistically appear as a distribution which is exponentially decreasing, and with long thin tails.

As an example, in Figure 2 we show a non-normalized histogram of this distribution for one day worth of installs in a group of apps. All Time Delta measurements shown are aggregated into the same dataset:

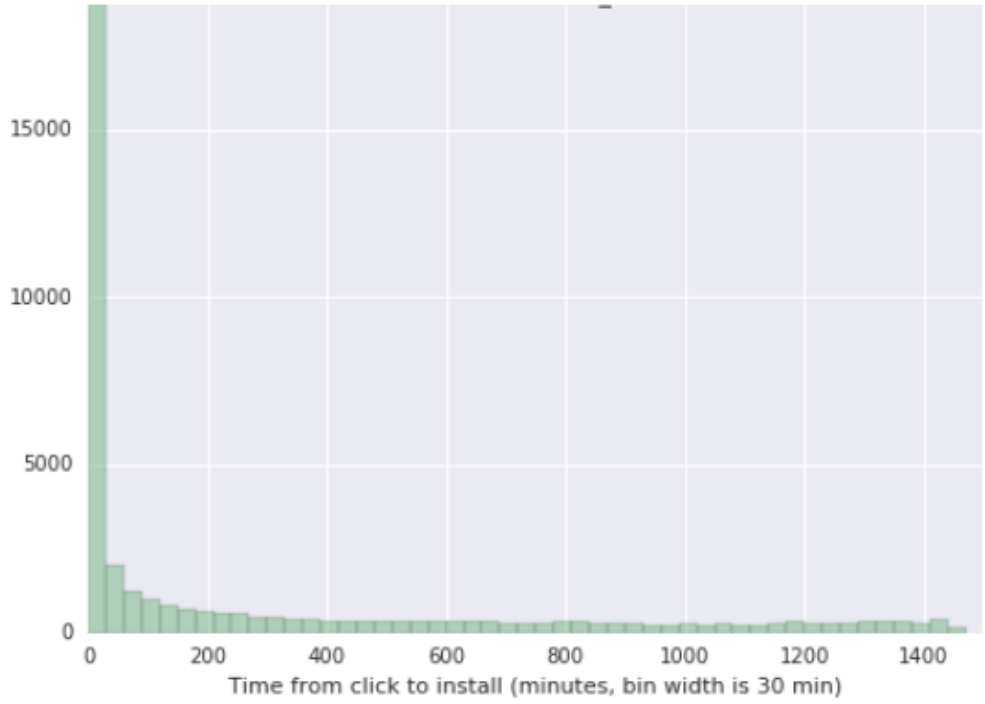

Fig. 2. Histogram of non-fraudulent scenario.

[8] Where the number of random instances depend on each instance a, and we have dropped the T notation as it is implicit.

In our experiments, normal circumstances of the distributions should look as above; with a sharp decay for the lower values of the metric, and with an almost constant decay for higher time delta values. Most of the distribution's mass would be concentrated in smaller Time Delta values, and a small percentage of the distribution would be spread out in higher values.

This can be explained partly due to the nature of the attribution method. We would expect a user to click an ad and be redirected to the app store, in order to download and open the app. As a consequence, most of the users' installs occur in the first hour after the click.

Late installs would also arrive, and this is for two reasons: high-speed Internet unavailability, and people that install an app but don't always open the app after installing it. Added to this, the attribution would last for days, unless the user clicks in another publisher's ad for that same application.

This statistical property in the distributions is ubiquitous across apps of different verticals[9], countries and operating systems. Minor differences arise when looking at different distributions, though. They will differ in maximum values and in the exact rate of decay.

The difficulty with these type of distributions is that there are no accurate ways of fitting them with theoretical ones. In our experiments with goodness of fit, we proposed different data transformations and class of functions as approximators, yet we were not satisfied by the outcomes.

At the same time, if we focus our analysis on what we believe to be fraudulent publishers, we find ourselves with different distributions as in Figure 3[10]:

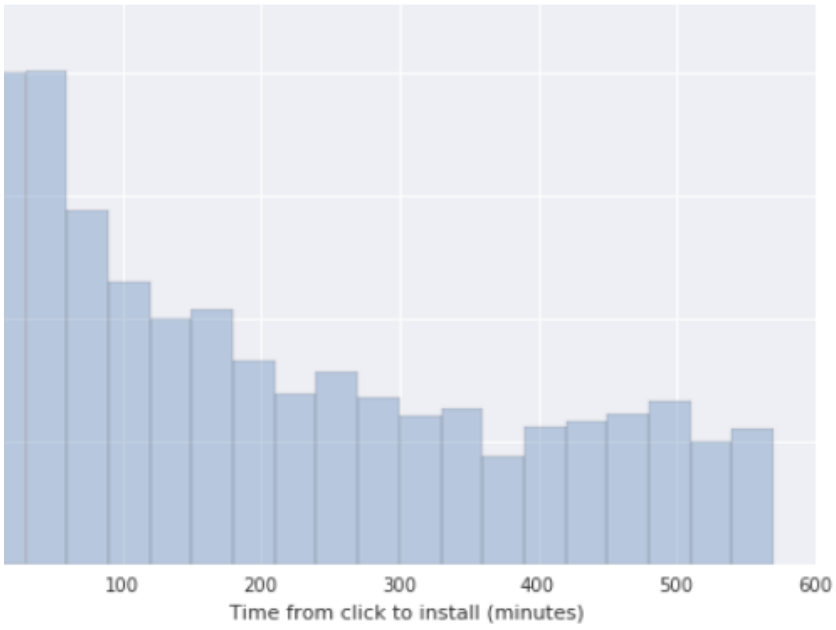

Fig. 3. Example of fraudulent scenario.

We notice here that the empirical distribution has changed substantially. There is no sharp decay in the distribution, and the data has heavier tails. Also, there is a bulk of users "clicking"

[9] Should we explain verticals?
[10] Histogram frequencies have been hidden for confidentiality reasons.

on ads and then opening the app very late in the day. Added to this, there is no reasonable explanation for this high volume of late installs and this behavior often results in suspicious high click and impression volumes, with low CVRs and average ERs. All of this raises the flag of click-spamming.

As a second example of Time Delta distributions, in Figure 4 we look at a scenario where the distribution is multi-modal. At first, the decay and the distribution seems usual, but then the large slump at the end does confirm this is a publisher with Click-Spamming activity.

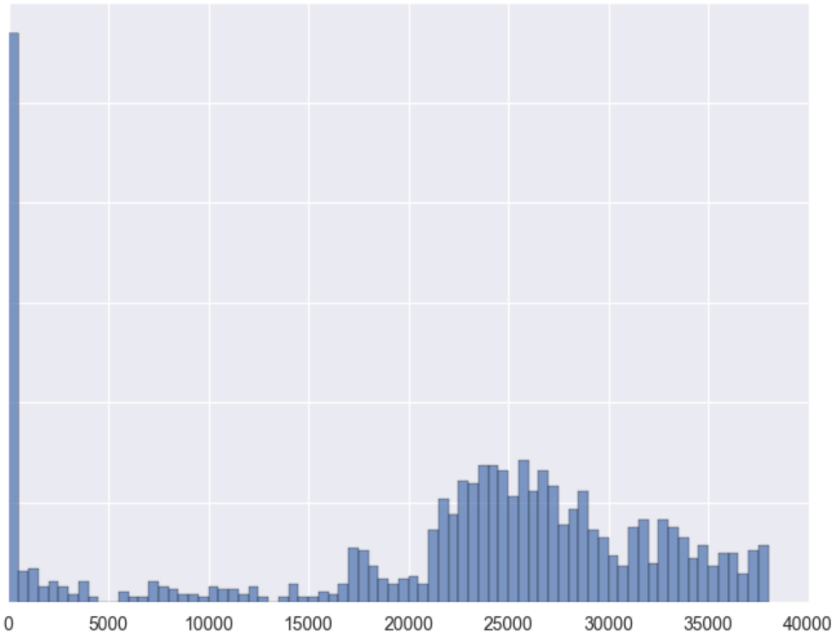

Fig. 4. Second example of fraudulent scenario.

In our data exploration, we found that in fraudulent cases the distributions lack exponential decay of the distribution, and we start to see other different values of central tendency. We consider then that a fraudulent distribution is one which doesn't follow the expected exponential decay, and we would like to find them as anomalies in our dataset.

Using the ideas outlined earlier in the difference among fraudulent and "normal" distributions, we proceed to describe an algorithm, using heuristic techniques, that will provide a time threshold over which our installs are to be considered as suspicious.

3 Click-Spamming Detection

We would like to have a specific threshold $F \in \mathbb{R}$ in the time window T and, for each $a \in A$, F will draw the line for suspicious installs. Each time $\delta > F$, it's a suspicious install for a.

Even though there are arguments in favor of the authenticity of in-app opens coming days after the click, we can safely assume that a significant part of late installs come from fraudulent activity.

Care must be taken, though, not to set a universal value for any application. Install behavior is varied and, for example, a value which fits for regions with fast Internet access and fast smartphones will most probably not fit for a mobile market which is still developing.

To do this, we will compare empirical distributions $D(a^T)$ to a set of theoretical distributions. Following exploratory data analysis, we make the assumption that in this set we have distributions that are either "clean" or "fraudulent", even though we can not prove this.

Later, we will compare their KL divergence with $D(a^T)$ to rank them altogether. With this, we will have a threshold F^T corresponding to the 95th percentile of the best theoretical distribution.[11]

For the first case we used two distributions known for their exponential decay, namely the Exponential and the Generalized extreme value distributions. For the fraudulent case we used two very simple distributions, the Uniform and Chi-squared.

The following list shows the PDF for each distribution.[12]

- Exponential: $\lambda I(x)_{(x\geq 0)}\ e^{-\lambda x}$
- Generalized extreme value: $\frac{1}{\sigma}\left(1+\xi(\frac{x-\mu}{\sigma})\right)^{\frac{-\xi+1}{\xi}} e^{-1+\xi(\frac{x-\mu}{\sigma})}$
- Uniform: $\frac{1}{b-a}I(x)_{x\in(a,b)}$
- Chi-squared: $I(x)_{x>0}\ \frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})}\ x^{\frac{k}{2}-1}e^{-\frac{x}{2}}$

We use the KL divergence

$$KL(Q \mid P) = -\int p(x)\ln(\frac{q(x)}{p(x)})dx$$

as a pseudo-metric to assess which distribution is most similar to $D(a^T)$, since it measures how well Q is used as an approximation of P when p and q are their probability density functions respectively. For our specific case, we will compare each theoretical distribution P with the empirical distribution Q.

For our objective, the fraudulent distributions might not look like they'll fit best. But note that we are not trying to accurately model fraudulent distributions, but rather doing the opposite. In consequence, we need these distributions to be slightly better fitting than the usual non-fraudulent distributions. By doing this, a lower selection barrier is being imposed for the usual distribution to be selected.

Given $a \in A$ and a series of time windows $T_0, \ldots, T_n$, we will find $F_1, .., F_n$ for a as follows:

Data: All installs in $(T_0, \ldots, T_n)$
Result: $(F_1, .., F_n)$
i=1; **while** $i \leq n+1$ **do**
  Fit the four theoretical distributions (D_1, D_2, D_3, D_4) with $D(a^{T_i})$;
  Use the KL divergence to rank the best fit D_{kl};
  if D_{kl} *is non-fraudulent* **then**
    Take $F_i = q \ \ s.t. P_{D_{kl}}(\delta < q) = 0.95$;
  else
    $F_i = NaN$
  end
  i++;
end

[11] Other percentiles such as 97th and 93rd had been tried, but this one has given results that are the most accepted among the system's users at Jampp.

[12] In practice we predefined some of the distribution's parameters which correspond to distributions defined in $\mathbb{R}_{>0}$ only.

The idea is that we would like to take the 95th percentile only for those cases in which the divergence ranks the non-fraudulent distributions to be best approximated by $D(a^{T_i})$. In this way, we omit those situations which we deem more fraudulent.

Finally, to aggregate all $F_1, .., F_n$ into a single threshold for a, we take the median of the previous values (excluding the null ones).

For our experiments we used Python 3.5 with Scipy to do the distribution fitting routine. The setup ran on a standard laptop with a Intel "Core i5" processor (5257U) and 6GB of RAM.

We tried different windows of time for T, using windows between 3 to 7 days of length. We did this to find the minimum length over which we could single out fraudulent outbreaks from the rest of the data, without compromising information. Another caveat is that we need to have enough data in our empirical distribution to use in the KL divergence comparison.

The following images show two examples from different apps where this algorithm was tested.

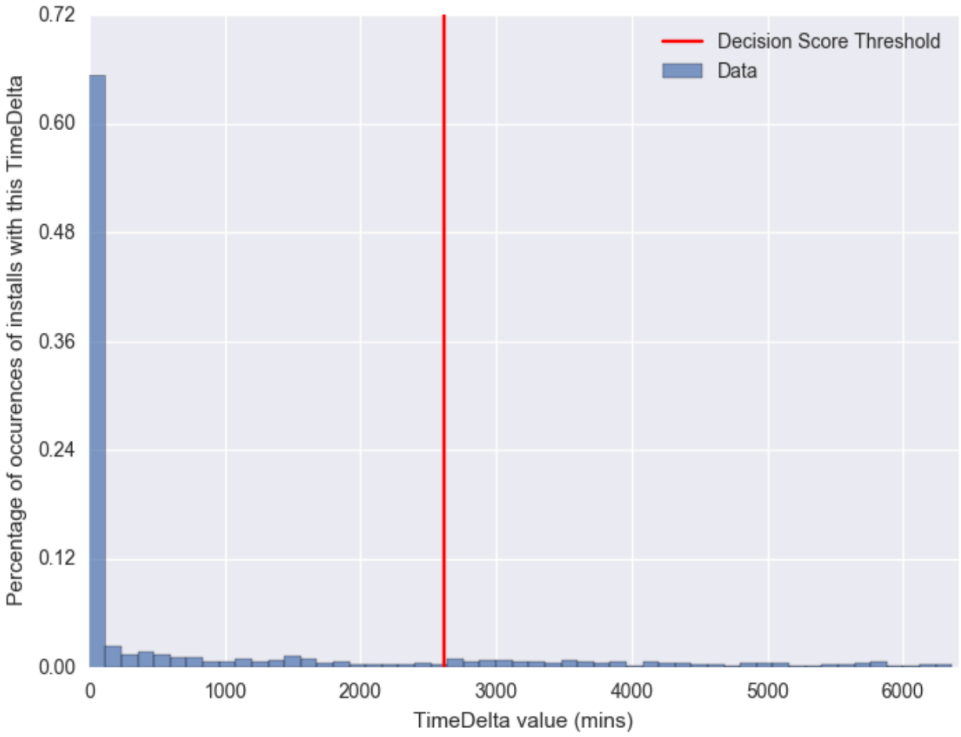

Fig. 5. Long tailed threshold cut.

Figure 5 shows how this application has all Time Delta values extremely wrapped around its most minimum values. Here the 95th percentile value is slightly over a day and a half.

In the case of Figure 6, we observe a similar behavior but we find the decay to be much faster. The tail is much shorter than before and the threshold is set slightly over three hours. Note the significant difference to the previous case.

Currently, we have placed this algorithm in production with satisfactory results. The advertisers use these data to have an elaborate decision on where to put the threshold F in order to

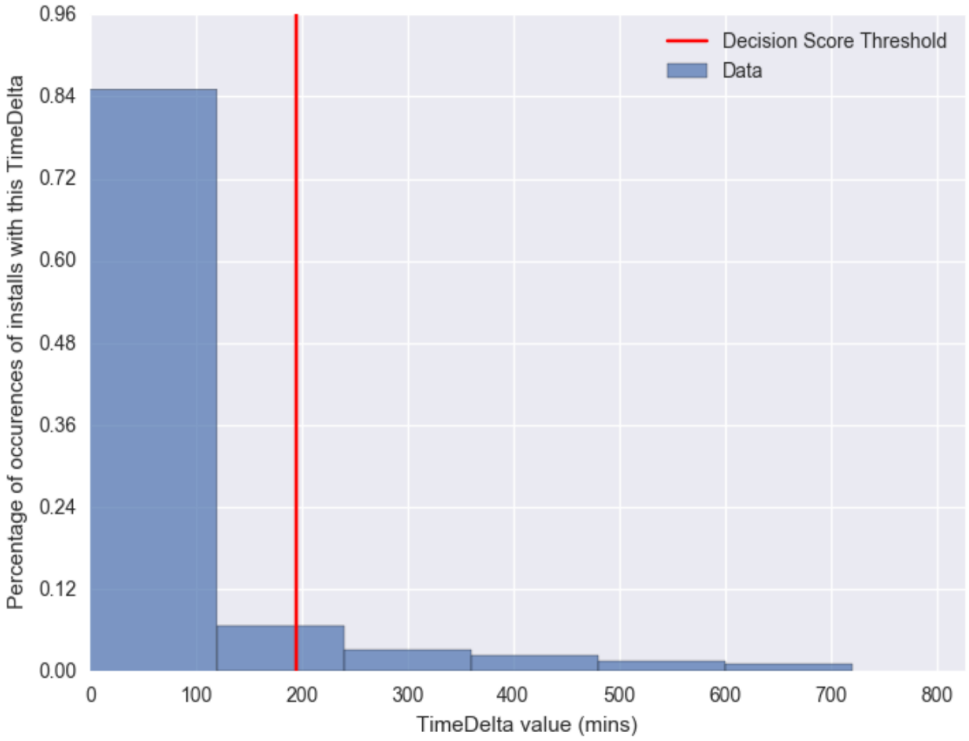

Fig. 6. Short tailed threshold cut.

start rejecting installs. In most cases, the loss of potential installs (those which re now rejected) has been lower than 7% of the whole set.

4 Conclusion

The methods here exposed are a first iteration for fraud detection and classification for Click Spamming. Note here that we rely on the assumption that for all apps there are periods free of fraudulent behavior. These periods are then used to calculate our final threshold, which is robust to data that is contaminated. We are confident on this assumption, since we have seen that fraudulent behavior from publishers will only last, at most, for a few days. Thus, using at least one week of data is enough to identify (most) cases of fraudulent behavior.

We find that this algorithm is robust and flexible in accounting for dissimilarities among applications, where there are significant time differences between Time Deltas. The evaluation measures this difference by automatically fitting the best distributions for different contexts.

However, as we are operating under an unsupervised learning setting, we cannot systematically measure the performance of the algorithm. The lack of certainty over which distributions are fraudulent or not undermines our ability to produce an effective quantification of the algorithm's error. Yet this algorithm is currently in use internally, with positive feedback from its users (those managing advertisers' budgets). Hence, we have seen that the algorithm holds practical value even if the error rate has not been precisely determined.

This is a specific problem solved by an algorithm which could, in principle, be used in other contexts such as analyzing specific publisher installs or applications under different contexts.

References

[1] Ana. *The Bot Baseline: Fraud in Digital Advertising*. [Online; accessed ¡today¿]. 2016. URL: http://www.ana.net/content/show/id/botfraud-2016 (cit. on p. 1).

[2] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (July 2009), 15:1–15:58. ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: http://doi.acm.org/10.1145/1541880.1541882 (cit. on p. 3).

[3] Media Rating Council. *Invalid Traffic Detection and Filtration Guidelines Addendum*. [Online; accessed ¡today¿]. 2015. URL: http://mediaratingcouncil.org/101515_IVT%20Addendum%20FINAL%20%28Version%201.0%29.pdf (cit. on p. 3).

[4] Brad Miller et al. "What's Clicking What? Techniques and Innovations of Today's Clickbots". In: *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA'11. Amsterdam, The Netherlands: Springer-Verlag, 2011, pp. 164–183. ISBN: 978-3-642-22423-2. URL: http://dl.acm.org/citation.cfm?id=2026647.2026661 (cit. on p. 3).

[5] Lenin Ravindranath et al. "Automatic and Scalable Fault Detection for Mobile Applications". In: *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '14. Bretton Woods, New Hampshire, USA: ACM, 2014, pp. 190–203. ISBN: 978-1-4503-2793-0. DOI: 10.1145/2594368.2594377. URL: http://doi.acm.org/10.1145/2594368.2594377 (cit. on p. 1).

[6] John P. Rula, Byungjin Jun, and Fabian Bustamante. "Mobile AD(D): Estimating Mobile App Session Times for Better Ads". In: *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*. HotMobile '15. Santa Fe, New Mexico, USA: ACM, 2015, pp. 123–128. ISBN: 978-1-4503-3391-7. DOI: 10.1145/2699343.2699365. URL: http://doi.acm.org/10.1145/2699343.2699365 (cit. on p. 2).

[7] George Slefo. *Senators Take Aim Against Ad Fraud, Ask FTC for Answers*. [Online; accessed ¡today¿]. 2015. URL: http://adage.com/article/digital/senators-join-fight-ad-fraud-send-letter-ftc/304897/ (cit. on p. 1).