BotSpot:A Hybrid Learning Framework to Uncover Bot Install Fraud in Mobile Advertising

Tianjun Yao
bertrand.yao@mobvista.com
Mobvista Inc.
Guangzhou, China

Qing Li
qing.li@mobvista.com
Mobvista Inc.
Guangzhou, China

Shangsong Liang
liangshangsong@gmail.com
Sun Yat-sen University
Guangzhou, China

Yadong Zhu*
yadong.zhu@mobvista.com
Mobvista Inc.
Beijing, China

ABSTRACT

Mobile advertising has become inarguably one of the fastest growing industries all over the world. The influx of capital attracts increasing fraudsters to defraud money from advertisers. There are many tricks a fraudster can leverage, among which bot install fraud is undoubtedly the most insidious one due to its ability to implement sophisticated behavioral patterns and emulate normal users, so as to evade from detection rules defined by human experts. In this work, we propose an anti-fraud method based on heterogeneous graph that incorporates both local context and global context via graph neural networks (GNN) and gradient boosting classifier to detect bot fraud installs at Mobvista, a leading global mobile advertising company. Offline evaluations in two datasets show the proposed method outperforms all the competitive baseline methods by at least 2.2% in the first dataset and 5.75% in the second dataset given the evaluation metric *Recall@90% Precision*. Furthermore, we deploy our method to tackle million-scale data daily at Mobvista. The online performance also shows that the proposed methods consistently detect more bots than other baseline methods.[1]

KEYWORDS

Mobile advertising, Bot install fraud, Fraud detection, Graph neural networks

ACM Reference Format:
Tianjun Yao, Qing Li, Shangsong Liang, and Yadong Zhu. 2020. BotSpot:A Hybrid Learning Framework to Uncover Bot Install Fraud in Mobile Advertising. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3340531.3412690

*corresponding author.

[1]The source code of our proposed method is publicly available from https://github.com/akakeigo2020/CIKM-Applied_Research-2150.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412690

1 INTRODUCTION

With the prevalence of smartphones, mobile ads has become one of the fastest growing industries all over the world. According to [6], the total spend on mobile advertising worldwide has reached to $240.95 billion which increases by $51.82 billion from 2018, and will continue to rise to $286.47 billion in 2020. With ad spend continuously growing for mobile marketing budgets, ad fraud attempts continue to rise as well, following the influx of capital. Among all the fraudulent activities, mobile app install fraud detection is a hot topic in the mobile industry.

As illustrated in Figure 1, a general mobile ad workflow consists of several steps as following: i) an ad request is sent from user device to ad server. ii) the selected ad is displayed in the user device. iii) the user clicks the ad and is redirected to the app store. iv) in the meantime, relevant information such as ad impression and ad clicks is sent to Mobile Measurement Partner(MMP) for attribution and further analysis. v) the app is installed and the related metadata is sent to MMP. vi) MMP performs attribution to determine which ad channel should claim credit for the install. vii) final result is sent to advertiser and the associated ad channel.

In the context of mobile app install fraud which our work aims to tackle with, the advertised product is an app, and the associated ad campaigns for this product are distributed to one or several ad channels. An ad channel can claim credit from the advertiser if some user clicks the ads from this ad channel and installs the app, given that MMP attributes this user install to this ad channel correctly. Furthermore, an ad channel could be a website or a mobile app, or it could be a proxy of several apps. A fraudster, oftentimes cooperating with malicious ad channels, may leverage certain tricks to confuse the attribution system such that the install is falsely attributed and the malicious ad channel collect money from advertisers without subsequent user actions.

Among all the commonly seen fraudulent tricks such as *click injection*, *click spamming*, *device farm* and *bots*. Bots are malicious codes that run a set program or action. While bots can be based on real devices, most bots are server-based. Bots aim to send clicks, installs and in-app events for installs that never truly occurred. Bot install fraud is considered one of the most difficult types of fraud to detect. According to the latest report of Appsflyer[2], they have blocked over 1.5 billion fraud installs over the last three years, the

[2]https://www.appsflyer.com/

fraud installs resulted by bots accounts for more than 50% for the entire mobile ad install fraud activities[1]. At Mobvista there are over one million app installs per day. With too many bots undetected, our customers' advertising budget is wasted without any conversion or effect, which is eventually reaped by the malicious app developer or intermediate malicious ad channels. Furthermore the reputation of Mobvista as an online ad platform is also severely corrupted. Hence in this work, our goal is to detect bot install fraud at Mobvista such that we can prevent advertisers' budget from being wasted without any real conversion. However bots are hard to detect due to several reasons: (1) Most of bots are compromised computers that are essentially normal users without any fraudulent historical events. (2) Bots are programmable and are able to implement sophisticated logic to emulate a normal user's behaviors. (3) In an adversarial setting, bots can get evolved by upgrading its software component and evade updated detection rules. As a result, expert-defined rules oftentimes failed to detect bot install fraud.

Many machine-learning based methods in mobile advertising aim to tackle the problem of ad click fraud or impression fraud, yet very few works solve install fraud, especially those incurred by bots. However, as we mentioned before, bot install fraud is a critical issue that severely affects the mobile advertising industry, causing billions of dollar losses for advertisers globally every year. Furthermore, many of the proposed methods resort to techniques such as ensemble methods, e.g., Random Forest [2] and XGBoost [3]. Although these methods are capable of mining rich fraud patterns with sophisticated feature engineering, they are unable to model various types of relations among entities with structural information, which may be suboptimal as bots oftentimes behave in a clustered manner. As a result, a hybrid learning method that can well capture its local context as well as global context should be superior to the traditional methods, where local context refers to the data instances associated with the same channel, and global context refers to all the data instances in the given dataset.

In this work, we address this challenge by proposing a hybrid learning method that incorporates graph neural networks (GNN) and prior knowledge from gradient boosting machine to detect bot install fraud in mobile advertising. Directly applying either traditional or state-of-the-art GNN [10, 15, 20, 30] is not the best option to tackle our task. Existing GNN models are built upon the assumption that node embeddings in the same cluster tend to be similar. However, in mobile advertising bot installs and normal installs coexist in the same ad channel or publisher, similar embeddings in the same cluster would create sparse feedbacks and are unlikely to help with the classification problem. Hence, in this work we design a novel message passing mechanism based on our specific problem and graph structure, enabling each ad channel learn a representation that is able to reflect its preference towards bot installs and normal installs associated with it.

Our main contributions can be summarized as follows: (1) To the best of our knowledge, we are the first to study the problem of bot install fraud detection in the context of mobile advertising, which is a critical issue in industry yet not covered in the literature so far. (2) We propose a novel bot install fraud detection model which works on heterogeneous bipartite graph at Mobvista. The proposed learning method is able to incorporate both local and global context by leveraging graph neural networks and gradient boosting classifier. (3) We design a novel message passing mechanism that takes into account the graph structure, enabling each channel to learn a representation that is able to reflect its preference towards bot installs and normal installs associated with it. (4) We build a deep ensemble model that incorporates prior knowledge from gradient boosting classifier by leveraging leaf embeddings in our model, which can also be viewed as a model initialization methods using prior knowledge of gradient boosting classifier. (5) We conduct experiments in real-world data at Mobvista, and the results demonstrate the superior of detecting bots by our model.

2 RELATED WORK

Various methods have been proposed for mobile ad fraud detection problem. [31] studied the problem of detecting duplicate clicks over decaying window models [32] in which two streaming algorithm– Group Blooms Filter and Timing Blooms Filter– are proposed which achieves zero false negative and low false positive in detecting duplicate click events. Jiarui and Chen [11] leverage cluster analysis to detect crowdsourcing click fraud, given the observation that there are some distinct characteristics in this form of click fraud, therefore, by constructing IP-Advertiser bipartite graph and perform DP-Means clustering method, the malicious groups can be detected without determining the cluster number.

More recently machine learning based approaches have been leveraged for ad fraud detection [5, 9, 26, 28], and most of them rely on extracting robust features and feed the dataset into an ensemble classifier or boosting machine to detect fraudulent event. Dou et al. [5] studied the problem of download fraud in App Market by setting up a honeypot to capture the ground truth fraudulent activities, and identify the download fraud leveraging feature engineering and XGBoost [3]. Taneja et al. [26] approach the mobile ad click fraud problem by utilizing Recursive Feature Elimination and decision tree classifier with Hellinger-Distance [24] due to its robustness to skewness. Recent years there is a growing interest in deep learning based algorithm on graph. Motivated by the work in [21], various forms of graph embedding methods have been proposed to extract node embeddings [8, 20, 23, 27] without requiring any labels. However, node features are not utilized,and embeddings for unseen nodes in the graph cannot be generated in this scheme. Graph neural network is further presented, in which the limitations of graph embedding are compensated. One of the most prominent progress is known as Graph Convolutional Networks (GCN) [15], which exploits multiple propagation layers to aggregate neighborhood information and enlarge the receptive field for each node in the graph. Hamilton et al. [10] proposed GraphSage and introduces computational sub-graph for each vertex in the graph, which enables scalable graph representational learning. Graph Attention Network [30] further introduces attentional mechanism into graph neural networks which is currently leveraged extensively in Natural Language Processing [13, 18, 29].

3 PROPOSED METHOD

3.1 Preliminary

Graph neural networks are introduced in [7] as a generalization of recursive neural networks that can directly deal with a more general class of graphs, e.g., cyclic, directed and undirected graphs.

Defferrard et al. [4] introduce a graph neural network with polynomial parametrization for the localized filter to reduce the number of hyperparameters to alleviate overfitting and reduce computational cost. Kipf and Welling [15] further refine the model discussed above by proposing a layer-wise linear model where the degree of Chebyshev Polynomial limited to 1. However, by stacking multiple linear layers with non-linearity, a rich class of convolutional filters can still be recovered. Enlightened by Kipf and Welling [15], a more generalized version of graph neural networks is proposed by Hamilton et al. [10] which builds a computational sub-graph for each vertex in the graph and information is propagated from the neighborhood layer-wise consisting of two operators: Aggregation and Concatenation which can be written as follows:

$$h_v^k = W_k \quad AGG_k \quad \eta \; h_u^{k\ 1}, 8u\ 2\ N^1v^o \; o$$
$$h_v^k = \gamma \quad \text{CONCAT} \quad h_v^k, B_k h_v^{k\ 1} \qquad (1)$$
$$8k\ 2\ f1, \ldots Kg$$

where N^1v^o is a function that returns the set of neighbors of vertex v and k denotes the k^{th} layer in the computational graph for vertex v, and there are K layers in total. W_k and B_k are the trainable parameters in the model for affine transformation at layer k. AGG denotes the aggregation function that aggregates features or latent embeddings from the neighborhood of vertex v, and $CONCAT$ is the operator that concatenates self embedding with the embedding from its neighborhood after aggregation function. Finally, γ is the non-linear activation function, e.g., Relu.

3.2 Problem Setup

Our method aims to identify bot install fraud at Mobvista. This problem can be defined as a binary edge classification problem in a bipartite heterogeneous graph, where the graph consists of two types of nodes, namely *device* node and *channel-campaign* node. A *device* node refers to a user device that have installed one or more advertised app, and an *channel-campaign* node refers to an ad campaign on an associated ad channel. An edge connecting a device node and an channel-campaign node refers to an app install occurred at a user device which is attributed to an ad campaign from a specific ad channel. As an edge represents an install in the graph, our method aims to classify every edge into normal install or bot install. Mathematically, the problem can be defined as follows: We have bipartite graph $G = {}^1U, V, E^o$ where U denotes the set of channel-campaign nodes, V denotes the set of devices. An edge e_{ij} is presented in the graph if and only if an app is installed by a certain device $j\ 2\ V$ that can be attributed to the ad campaign from its associated channel, denoted by $i\ 2\ U$.

To this end, we have formulated our problem as an edge classification problem formally.

3.3 Label Acquisition

Bot installs are hard to identify. In this work, we rely on a subset of installs that belongs to some specific ad campaigns for which advertisers has purchased anti-fraud service from a third party such as Appsflyer, therefore labels for these installs are known, although the acquired labels are somewhat noisy. The resulted graph is built using only the subset of installs and can be trained in a fully supervised manner. At inference time, the remaining installs without purchasing anti-fraud service is able to be classified into normal install or bot install.

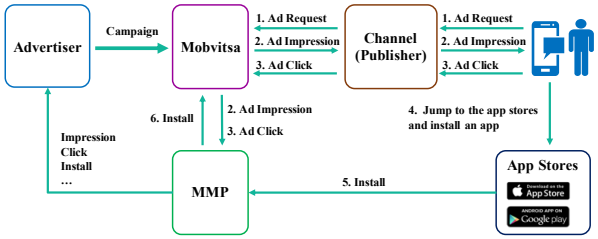


Figure 1: General workflow for serving an mobile ad.

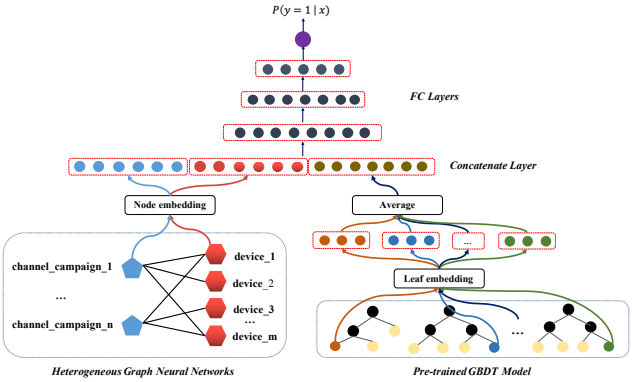


Figure 2: The Model Architecture of BotSpot.

3.4 Model Design

In this section, we detail how we tailor the general GraphSage framework in Eq.(1) that works on a bipartite heterogeneous graph for bot install fraud detection, then we show how to incorporate prior knowledge from gradient boosting classifier as an model initialization method to extract global context to further enrich representations for each install. The overall model architecture of our proposed method is shown in Figure 2.

3.4.1 Asymmetric Message Passing. Due to the service trait at Mobvista, the graph we construct is highly asymmetric, that is, $|N^1j^o| = 1$ and $|N^1i^o| >> |N^1j^o|$. Here, N^1i^o are the set of vertex i's neighbors. Furthermore, due to the fact that $|N^1j^o| = 1$ for most of device vertex $j\ 2\ V$, there will be many isolated sub-graph in the graph. As consequences, the typical multi-layer propagation scheme won't help in this setting as the information couldn't propagate to further vertex. As discussed in [17], GCN is a special form of Laplacian smoothing which makes embedding of the nodes in the same cluster look similar. This characteristic makes GCN a key success in many classification problems in social networks. However, this may be suboptimal for our problem as in the same channel, both normal installs and bot installs present, this kind of smoothing effect will actually make the learning problem more difficult.

In this work, to generate a more discriminative embedding for every edge, we propose a modification of the original message passing mechanism proposed in [10] to force each channel-campaign node to explicitly learn an representation that incorporates its preference

to bot installs and normal installs which serves better purpose for our classification problem.

Channel-campaign node. As mentioned previously, we want every channel-campaign node to generate a discriminative representation by capturing its preference towards bot install and normal install. To achieve this, we force channel-campaign node explicitly aggregate information from bot installs and normal installs separately and obtain a convex combination from both sides based on attentional mechanism. Mathematically, it can be stated as follows:

To classify an edge $e_{ij}, 8i \; 2 \; U, 8j \; 2 \; V$, the embedding aggregated from bot installs for node i is given by:

$$h_i^k = W_k^{bots} \quad AGG_k \quad \begin{matrix} n \\ h_b^{k \; 1}, 8b \; 2 \; N^1 i^o \end{matrix} \\ \text{s.t. class } {}^1 e_{ib}{}^o = 1 \; \& \; e_{ib} < e_{ij} \tag{2}$$

Where $k \; 2 \; f1, 2, \ldots, Kg$ denotes the k^{th} layer for vertex i and we have K layers in our model, W_k^{bots} is the trainable parameter matrix for the aggregation function in the k^{th} layer for bot install and $class^1.^o$ denotes the function that returns the class label of an input edge. It is imperative to satisfy the constraint $e_{ib} < e_{ij}$ for every edge e_{ij} in the training stage, otherwise the ground truth information will leak out and result in overfitting. Similarly in Eq.(2), the embedding obtained from normal installs can be written as:

$$g_i^k = W_k^{normal} \quad AGG_k \quad \begin{matrix} n \\ g_b^{k \; 1}, 8b \; 2 \; N^1 i^o \end{matrix} \\ \text{s.t. class } {}^1 e_{ib}{}^o = 0 \; \& \; e_{ib} < e_{ij} \tag{3}$$

where W_k^{normal} is another set of parameters for linear transformation for neighborhood aggregation from normal installs. The embedding z_i^k that attends over bot installs and normal installs for node i at the k^{th} layer can be obtained by performing a convex combination as following:

$$z_i^k = \lambda_{bots}^{ik} h_i^k \; , \; \lambda_{normal}^{ik} g_i^k, 8i \; 2 \; U \tag{4}$$

where $\lambda_{bots}^{ik}, \lambda_{normal}^{ik}$ denotes the attention coefficient of bot installs and normal installs respectively for node i at the k^{th} layer and is obtained as follows:

$$s_1^{ik} = FFCN_\theta \quad q_i^{k \; 1} \quad h_i^k \\ s_2^{ik} = FFCN_\theta \quad q_i^{k \; 1} \quad g_i^k \tag{5}$$

where $q_i^{k \; 1}$ denotes the final output embedding for vetex i at the $^1k \quad 1^{oth}$ layer and denotes the concatenation operator to concatenate two vectors. The unnormalized score s_1^{ik} and s_2^{ik} are learned from a feed-forward neural network, parametrized by model parameter θ, denoted by $FFCN_\theta{}^1.^o$. The attention coefficient is further obtained with softmax function:

$$\lambda_t^{ik} = \frac{\exp \; s_t^{ik}}{\sum_{T=1}^{2} \exp \; s_T^{ik}}, 8t \; 2 \; f bots, normal g \tag{6}$$

The final representation for node i at the k^{th} layer after concatenation operator in BotSpot is defined as following:

$$q_i^k = \gamma \quad z_i^k \quad B_k q_i^{k \; 1} \tag{7}$$

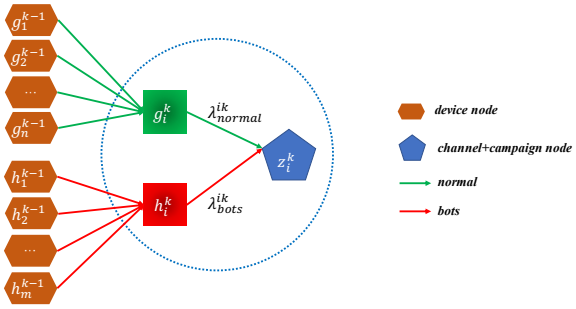


Figure 3: The tailored message passing mechanism for channel-campaign node to summarize information from bot installs and normal installs.

The base case for the multi-layer propagation is as follows:

$$h_i^0 = g_i^0 = q_i^0 = x_i \tag{8}$$

where x_i denotes the original feature vector for node i. The tailored message passing mechanism designed for channel-campaign node is illustrated in Figure 3.

Device node. For the device node, with only one install event occurred for most of the devices, we have much less structural information we could resort to. The message passing mechanism won't help to extract better features in this case, We therefore utilize raw device features without any message passing method.

3.4.2 Incorporating Global Context. Our proposed method above can well utilize structural information as well as expert-defined features to detect bot install fraud. However, this model may still be sub-optimal due to its inability of acquiring global information. As discussed earlier the heterogeneous graph consists of many isolated connect components with almost all the devices connected to only one single neighbor, which makes the multi-layer propagation inadequate for the nodes to generate embeddings with global context. Furthermore, the structural information may sometimes complicate our learning problem. While it is the case that some publishers and ad channels could be inherently malicious that are filled with abundant fraudulent installs, yet in the adversarial game of fraud and anti-fraud, bots are becoming more and more intelligent, they will disguise themselves by seeking for the benign ad channels with plenty of normal users, for which case our model may be hard to detect. However, there may still be traces recognizable for fraudulent activities with global context provided. By incorporating global context we may be able to detect more bot fraud installs. Hence, we leverage LightGBM[12] to extract global context. Specifically, We first train a LightGBM with each data instance to be $h_{ij} = x_i k x_j, 8i \; 2 \; U \& 8j \; 2 \; V$, where x_i and x_j denotes the original feature vector for channel-campaign node i and device node j and h_{ij} denotes the feature vector for edge e_{ij}. With the trained classifier, we then leverage leaf nodes of each decision tree in the boosting classifier for each data instance.

Motivated by [21] where each word is represented as a dense vector rather than a one-hot sparse vector by which word semantics and relative relations are well captured, the index of each leaf node for every week learner in LightGBM also incorporates certain semantic meanings, therefore each leaf node in a decision tree is also embedded in a embedding matrix where the leaf index map to

the corresponding row in the matrix. Formally, let $leaf\_emb^i_j$ be the embedding of the j^{th} leaf node of the i^{th} decision tree T, and $Leaf\_Emb_i$ denotes the embedding matrix for each decision tree T_i. For N total decision trees, the embedding matrix $Leaf\_Emb_i, 8i\ 2\ f0, 1, \ldots, N\ \ 1g$ is our model parameters that can be jointly trained via optimization algorithms such as SGD or Adam[14].

3.4.3 Leaf node aggregation. We discuss the aggregation methods for leaf embeddings in this section. In this work we experiment with mean pooling, concatenation and BiLSTM and adopt mean pooling for leaf embedding aggregation as it requires fewer model parameters than other methods and doesn't get overfitted while the number of decision trees grow. Moreover, it is more efficient in terms of computational resources.

3.4.4 Leveraging high-cardinality features. Several features serves as strong indicators for fraud detection based on previous expert knowledge. However, the nature of high-cardinality makes it prohibitive to be leveraged by gradient-based algorithm: one-hot encoding might bring too many parameters into our model which leads to overfitting. Furthermore, the orthogonality represented by one-hot encoding scheme doesn't reflect the interdependency among various feature values. To deal with this problem, we utilize a similar approach as described in Section 3.3.2: each high-cardinality feature is embedded in a trainable embedding matrix, where each row represents a feature vector for a high-cardinality feature.

3.4.5 Sampling Strategy. According to our observation, bots controlled by bot master tend to act in burst manner, as a result, bot installs occurred in the same duration tend to reveal similar behavioral patterns. Based on this fact, we employ a time-biased approach to sample each data instance: data that are closer to the date of online experiments are sampled with more weights. Intuitively, this helps the model better capture the fraud patterns in the current time duration. Additionally, the positive samples are sampled with more weights to offset the gap between the number of positive samples and negative ones. Mathematically, the weight is calculated as: $w_i = q\frac{1}{\frac{1}{T_i}\ \ C_i}$, where w_i denotes the sampling weight for data instance i, and T_i denotes the time difference to the starting date of the online experiments in terms of days, e.g., the closest day to the starting time is 1. C_i is the coefficient to reweight the data instance i, for positive samples, C_i is the log ratio of negative samples and positive samples,and for negative samples, C_i is equal to 1.

3.4.6 Model Training. To estimate model parameters of BotSpot, we employ cross-entropy loss as the final loss function over each mini-batch: $L = \frac{1}{|D|}\ \ \sum_{i2D} y_i \ln p_i$, where $|D|$ is the size of each mini-batch D and p_i denotes the estimated probability of data instance i. Additionally, the dataset is highly imbalanced in terms of positive and negative samples which poses difficulty for model training as the gradient information from negative samples may overwhelm those from positive samples. To deal with this issue, we leverage hard negative mining to stabilize the training procedure[19] without overwhelming gradient flow from negative samples. Overfitting is a perpetual problem in optimizing deep neural network models, to alleviate this problem, dropout[25] has been applied to our model. Furthermore, L2 regularization is also used.

Table 1: Statistics of the two datasets, each of which consists of 7-day install events. Note that Dev. Node refer to device node and Chan. Node refer to channel-campaign node.

	#Dev. Nodes	#Chan. Nodes	#Normal Edges	#Bot Edges
Dataset-1	889,969	1,968	757,536	140,235
Dataset-2	1,181,278	1,545	1004,623	197,895

4 EXPERIMENTS

In this section, we detail our experimental setup, report and analyze the results.

4.1 Offline Evaluation

4.1.1 Datasets. To evaluate our proposed model, two datasets are built, each of which consists of 7-day install events in Mobvista which exhibits different bot fraud patterns. The statistics of the two datasets are shown in Table 1. Training/Test dataset is split randomly with a ratio of 75% and 25%.

4.1.2 Evaluation Metrics. In practice, our first priority is to detect fraudulent activities with high precision as we don't want to remove normal installs which would harm the interest of downstream ad channels. Second to that, the recall should be as high as possible as more bot installs can be detected such that the advertisers' budget won't be wasted. Thus, as in [16] we also adopt the evaluation metric $Recall@TPrecision$ which is the recall value while the model achieves a precision of T, where T varies depending on the service class. We set T to be 80%, 85% and 90%, respectively. Precision-Recall curve is also illustrated in the experiments to visualize the capacity of various algorithms to identify bot installs.

4.1.3 Baseline Models. To evaluate the performance, we compare our BotSpot with several competitive methods detailed below:

Neural Network(MLP). A 4-layer MLP is used for the same edge classification problem where the feature vector for each edge is obtained by concatenating device node features and channel-campaign node features and the edges features is then fed into the MLP classifier.

MLP+Leaf Embedding. In addition to a 4-layer MLP, the leaf embedding component is included to extract global context. This baseline model also exploits 200 decision trees to obtain leaf embeddings which is the same as in BotSpot. This baseline is designed to facilitate the analysis of leveraging leaf embeddings to extract global information.

LightGBM. A well-known boosting algorithm widely used in industry, serves as a strong baseline model in our experiment. In the experiments, a LightGBM with 300 decision trees are leveraged as our baseline model. We use the same hyperparameters for the baseline method and BotSpot except for the number of decision trees- in BotSpot only 200 trees are leveraged.

BotSpot-Local. The graph neural network component with tailored message passing mechanism in BotSpot serves as another baseline classifier to demonstrate the effectiveness to extract local structural information.

GraphSage. GraphSage[10] is a SOTA graph neural network to extract node features. In this work, it serves as a baseline method to extract node features and oabtain features for each edge which

Table 2: Performance of BotSpot and the baseline methods. Note that in the table we let the notation Rec@x% P be shorten for Recall@x% Precision.

	Dataset 1			Dataset 2		
Model	**Rec@90% P**	**Rec@85% P**	**Rec@80% P**	**Rec@90% P**	**Rec@85% P**	**Rec@80% P**
MLP (Baseline model)	0.6678	0.6981	0.7412	0.1391	0.1962	0.3305
LightGBM	0.7094 (+ 6.23%)	0.7509 (+ 7.56%)	0.7912 (+ 6.75%)	0.1560 (+12.15%)	0.2856 (+45.57%)	0.3723 (+12.65%)
MLP+Leaf Embedding	0.7449 (+11.55%)	0.7859 (+12.58%)	0.8205 (+10.70%)	0.1965 (+41.27%)	0.2998 (+52.80%)	0.4115 (+24.51%)
GraghSage	0.7323 (+ 9.66%)	0.7781 (+11.46%)	0.8159 (+10.08%)	0.1736 (+25.68%)	0.3102 (+58.10%)	0.3943(+19.43%)
GAT	0.7398 (+10.78%)	0.7848 (+12.42%)	0.8194 (+10.55%)	0.1742 (+25.23%)	0.2754 (+40.37%)	0.3859 (+16.76%)
BotSpot-Local	0.7474 (+11.92%)	0.7937 (+13.69%)	0.8281 (+11.72%)	0.1843 (+33.65%)	0.3064 (+56.17%)	0.4045 (+22.39%)
BotSpot	**0.7613 (+14.00%)**	**0.8029 (+15.01%)**	**0.8334 (+12.44%)**	**0.2078 (+49.39%)**	**0.3178 (+61.98%)**	**0.4243 (+28.38%)**

Table 3: Effect of varying number of decision trees in BotSpot. Note that in the table we let the notation Rec@x% P be shorten for Recall@x% Precision.

	Dataset 1			Dataset 2		
Model	**Rec@90% P**	**Rec@85% P**	**Rec@80% P**	**Rec@90% P**	**Rec@85% P**	**Rec@80% P**
BotSpot-Local(Baseline)	0.7474	0.7937	0.8281	0.1813	0.3064	0.4045
BotSpot-50-trees	0.7384(-1.20%)	0.7857(-1.01%)	0.8223(-0.70%)	0.1945(+7.28%)	0.3022(-1.37%)	0.4167(+3.01%)
BotSpot-200-trees	0.7613 (+1.89%)	0.8029 (+1.16%)	0.8334 (+0.64%)	0.2078(+14.61%)	0.3178(+3.72%)	0.4243(+4.89%)
BotSpot-400-trees	**0.7694 (+2.94%)**	**0.8108 (+2.15%)**	**0.8421 (+1.69%)**	**0.2134(+17.70%)**	**0.3245(+5.91%)**	**0.4296(+6.21%)**
BotSpot-800-trees	0.7592(+1.58%)	0.8037(+1.26%)	0.8361(+0.97%)	0.2113(+16.55%)	0.3204(+4.57%)	0.4283(+5.88%)

is then fed into a 2-layer MLP for the final edge classification. This baseline method would facilitate the comparison with Botspot-Local which modified the aggregation methods based on GraphSage.

GAT. Another SOTA graph neural network to extract node features. Similar to GraphSage, this baseline method would serves as a strong baseline method for BotSpot-Local.

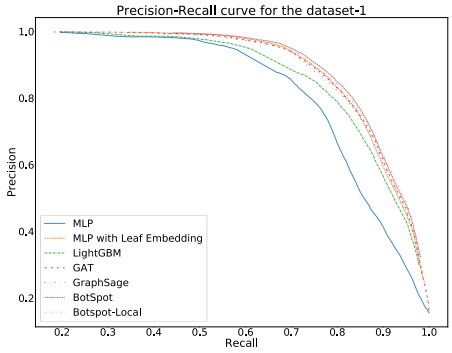
4.1.4 Experiment Results. We report the results of various methods as shown in Table 2. First, an observation is that model performance varies significantly in these two datasets. The underlying reason may be that the two datasets exhibit different bot fraud patterns-bot installs clustered in several malicious ad channels in the first dataset; while for the second dataset bots are distributed more evenly in many ad channels, which poses a harder learning problem. Furthermore, bot fraud patterns tend to shift dynamically and frequently as fraudsters might alter their fraud strategy, trying to evade from being detected. Therefore, it is probable that in the test set, some bot fraud patterns are not covered in the training set for the second dataset, which results in the gap of predictive performance between the two datasets.

Other than that, the two datasets exhibit similar patterns on model performance. As shown in Table 2, without relational information and global information, MLP performs worst among all the methods. However, MLP+Leaf Embedding performs much better, demonstrating the importance of global context. BotSpot, with relational information acquired from our tailored message passing mechanism, performs slightly better than MLP+Leaf Embedding, which proves the effectiveness of our tailored message passing scheme.

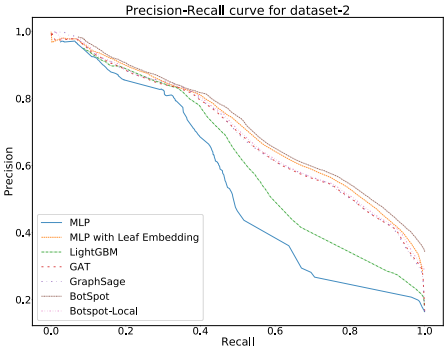
Furthermore, as shown in Table 2 BotSpot-Local outperforms Graphsage and GAT in the first dataset, while for the second one BotSpot-Local and GraphSage performs similarly. This phenomenon may demonstrate that while bots are clustered in several ad channels, by feeding explicit information from neighboring device nodes to channel-campaign node using the proposed aggregation method, feature representation can be extracted to capture the ad channel's preference towards bot installs and normal installs, thus benefit the classification problem. While this characteristic does not exhibit in the graph for the second dataset, BotSpot-Local behaves similarly as GraphSage and GAT. We may also observe that in the first dataset, BotSpot-Local outperforms MLP+Leaf Embedding while for the second dataset, the latter performs better. This may be due to the same reason as above that for the first dataset, local context obtained from graph neural networks matters more as bots are behaved in a clustered manner which is not the case in the second one. As a result, global context acquired from leaf embeddings may carry more importance than local context in the second dataset. However BotSpot, with local information incorporated, still outperforms MLP+Leaf Embedding.

The PR curve for the two datasets is illustrated in Figure **??** to show the performance for various methods. As can be seen, BotSpot consistently outperforms other competitive baseline methods in terms of the area beneath the curve.

4.1.5 Varying number of decision trees in BotSpot. Intuitively, without sufficient number of decision trees, BotSpot is unable to capture a holistic view of global context, while excessive amount of decision trees may result in model overfitting. In this subsection, we study the effect brought by varying the number of tree estimators in BotSpot. We set the number of decision trees to be 50, 200, 400 and 800 for both datasets with the same hyperparameter settings, respectively. The experiment result is shown in Table 3. As shown in the table, BotSpot achieves the best predictive performance while the number of decision trees is 400 for both datasets. However, this is probably not the optimal tree structure as we choose the number of decision trees arbitrarily, and the hyper-parameter setting is fixed which may lead to suboptimal tree structure. Admittedly, one may perform hyper-parameter search over various combination of



(a) dataset-1



(b) dataset-2

Figure 4: PR-curve for Offline Evaluation.

hyperparameters; however it may increase time cost significantly. Thus, how to learn the tree structure efficiently and effectively without extensive model selection is still an open research question, which will be addressed in our future work.

4.2 Online Experiments

In this section, we deploy our proposed method as well as MLP, MLP+Leaf Embedding, GraphSage and BotSpot-Local in our production environment. We evaluate the model performance by counting the total number of detected bot installs. We first obtain the threshold at which every model achieves 90% precision in validation set, and use that threshold in online experiments to detect bot installs. As shown in Figure 5, our proposed method consistently outperforms other baseline methods in terms of number of detected bot installs. As a strong baseline method, MLP+Leaf Embedding also works well without any graph convolutions. Furthermore, BotSpot-Local is able to detect more bots than GraphSage in the given time period consistently. Finally, all methods outperform MLP significantly as is the case in our offline evaluations.

4.3 Implementation

Currently we adopt mini-batch implementation as in GraphSage[10] which is easier to transfer to a distributed learning framework for higher efficiency. We make use of PyTorch[22] for implementation of BotSpot and all the experiments are performed in a server with one NVIDIA RTX 2080Ti GPU. Although our current approach is

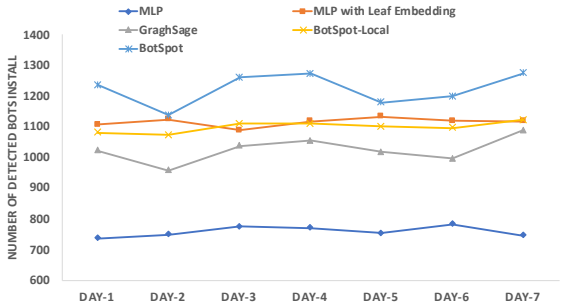


Figure 5: Number of detected bot installs in online experiments with 90% precision.

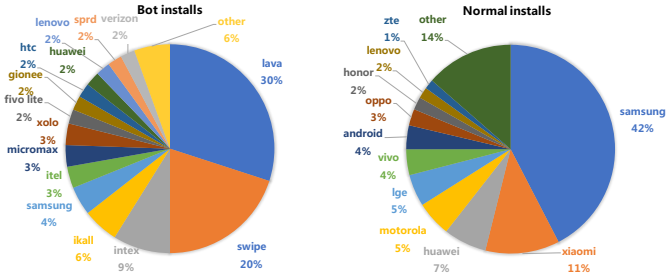


Figure 6: Distribution of device brands for bot install and normal install.

not highly-efficient as it doesn't leverage full batch matrix multiplication or distributed system for model training, thus takes around 3.1 hours per epoch and typically 4 epochs are required for model convergence. However, as we currently build a 5-day graph and detect bot installs for the next day and timeliness is not required, training time isn't an issue. A relative improvement of at least 2.2% in the first dataset and 5.75% in the second dataset may still compensate for the time cost, especially when there are over one million app installs per day. The economical loss would be saved significantly, furthermore, the reputation as an online ad platform can be improved. In the future we are planning to deploy BotSpot for distributed training, and the training time is expected to reduce significantly.

4.4 Case Study

Finally, we present some cases in which bot installs are recalled by BotSpot yet neglected by both BotSpot-Local and MLP with leaf embedding. Unlike image and text data, each app install is represented by various extracted features, which can be hard to interpret. As a result, we choose one particular ad channel where the prediction results vary significantly for all three models. As the features associated with channels are identical for all the samples, interpretability is enhanced. For the ad channel mentioned above, there are 91 bot installs recalled by BotSpot yet not detected by the other two methods. We examine these samples and conclude that these bots share some common patterns- i) the device brands for all the bot installs are rarely used in the local community, while for the normal installs most of the device brands are mainstream, and therefore the distribution significantly varies as illustrated in

Table 4: Presentation of several bot installs recalled by BotSpot yet not detected by other two methods.

device_id	carrier	device_brand	os_version
9c08cf8e4a30	jio 4g	lava	android-6
e36439c3848b	vodafone in	lava	android-6
261f6bade3a3	airtel	intex	android-6
0b86584f574e	idea	swipe	android-6

Figure 6. ii) all the OS versions for the bot devices are Android 6 while for the normal installs over 90% of OS versions are between Android 7 and Android 9 as listed in Table 4. We conjecture that BotSpot is able to recall these samples as it leverages local structure and is able to recognize the local fraud patterns the other two methods fails to detect. Although BotSpot-Local utilizes local structure, it is unable to capture a full picture e.g., the channel-related statistics and ip-related statistics, which leads to the ignorance of these bots. MLP with leaf embedding detect 11 bots in the same ad channel which is also recalled by BotSpot. However, ignoring the local device brand pattern prevents it from detecting another 91 bot installs recalled by BotSpot.

5 CONCLUSION

Bot install fraud is a notoriously hard problem in mobile adverting industry for its dynamically changing behavioral pattern which causes great economical losses for advertisers and undermines the ecosystem of mobile advertising industry. In this work, we address this challenge by proposing BotSpot. We first show how to tailor the naïve graph neural networks to adapt to our problem to detect bot install fraud at Mobvista, and then we analyze its weakness and propose a method to integrate global context via gradient boosting classifier to improve the model performance. Our offline experiments using real-world dataset at Mobvista proves the effectiveness of the proposed method for bot install fraud detection which outperforms all the competitive baseline methods. We then present the results of a 7-day online experiments, in which our proposed method also outperforms all the candidate methods. Finally some canonical cases are studied to facilitate interpretation among various methods. In future, we intend to extract more fine-grained features to enhance model performance and experiment with heuristic-based clustering for the device to make the graph more balanced to take further advantages of the graph structure.

ACKNOWLEDGMENTS

This research was partially supported by the National Natural Science Foundation of China (Grant No. 61906219).

REFERENCES

[1] AppsFlyer. 2019. 2019 Fraud Trends Uncover Fascinating Results. https://scalarr.io/research/articles/latest-trends-in-mobile-ad-fraud-2019. Accessed on 2019-12-19.
[2] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.
[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *KDD*.
[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.
[5] Yingtong Dou, Weijian Li, Zhirong Liu, Zhenhua Dong, Jiebo Luo, and Philip S Yu. 2019. Uncovering download fraud activities in mobile app markets. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 671–678.
[6] eMarketer. 2019. Worldwide mobile ad spending increases to 240.95 billion dollars in 2019. https://www.emarketer.com/forecasts/5a4e4662d8690c0c28d1f233.
[7] Marco Gori, Dipartimento Ingegneria, and Gabriele Monfardini. 2005. A New Model for Learning in Graph Domains. In *IJCNN*.
[8] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable feature learning for networks. In *KDD*.
[9] Ch Md Rakin Haider, Anindya Iqbal, Atif Hasan Rahman, and M. Sohel Rahman. 2018. An ensemble learning based approach for impression fraud detection in mobile advertising. *Journal of Network and Computer Applications* (2018).
[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.
[11] Xu Jiarui and Li Chen. 2016. Detecting crowdsourcing click fraud in search advertising based on clustering analysis. In *IEEE 12th International Conference on Ubiquitous Intelligence and Computing*.
[12] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. In *NIPS*.
[13] Ming-wei Chang Kenton, Lee Kristina, and Jacob Devlin. 2017. BERT paper. *arXiv:1810.04805 [cs]* (2017).
[14] Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A method for stochastic optimization. In *ICLR 2015*.
[15] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[16] Ao Li, Zhou Qin, Runshi Liu, Yiqun Yang, and Dong Li. 2019. Spam review detection with graph convolutional networks. In *CIKM*.
[17] Qimai Li, Zhichao Han, and Xiao Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[18] Shangsong Liang. 2019. Unsupervised Semantic Generative Adversarial Networks for Expert Retrieval. In *The World Wide Web Conference*. 1039–1050.
[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *ECCV*. 21–37.
[20] Zaiqiao Meng, Shangsong Liang, Jinyuan Fang, and Teng Xiao. 2019. Semi-supervisedly co-embedding attributed networks. In *Advances in Neural Information Processing Systems*. 6507–6516.
[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations ofwords and phrases and their compositionality. In *NIPS*.
[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.
[23] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *KDD*.
[24] Archana Purwar and Sandeep Kumar Singh. 2015. Issues in data mining: A comprehensive survey. In *2014 IEEE International Conference on Computational Intelligence and Computing Research*.
[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* (2014).
[26] Mayank Taneja, Kavyanshi Garg, Archana Purwar, and Samarth Sharma. 2015. Prediction of click frauds in mobile advertising. In *2015 8th International Conference on Contemporary Computing, IC3 2015*.
[27] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale information network embedding. In *WWW*.
[28] G. S. Thejas, Kianoosh G. Boroojeni, Kshitij Chandna, Isha Bhatia, S. S. Iyengar, and N. R. Sunitha. 2019. Deep learning-based model to fight against Ad click fraud. In *Proceedings of the 2019 ACM Southeast Conference*.
[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*. 5998–6008.
[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[31] Linfeng Zhang and Yong Guan. 2008. Detecting click fraud in pay-per-click streams of online advertising networks. In *Proceedings - The 28th International Conference on Distributed Computing Systems, ICDCS 2008.* https://doi.org/10.1109/ICDCS.2008.98
[32] Yunyue Zhu and Dennis Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. *VLDB* (2002).