Received December 6, 2019, accepted February 1, 2020, date of publication February 27, 2020, date of current version March 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976491

Generating Realistic Users Using Generative Adversarial Network With Recommendation-Based Embedding

PARICHAT CHONWIHARNPHAN[1], PIPOP THIENPRAPASITH[2], AND EKAPOL CHUANGSUWANICH[1]
[1]Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330, Thailand
[2]Home Dot Tech Company, Ltd., Bangkok 10330, Thailand

Corresponding author: Ekapol Chuangsuwanich (ekapol.c@chula.ac.th)

This work was supported in part by the Joint Research Collaboration between Home Dot Tech Company, Ltd. and the Faculty of Engineering, Chulalongkorn University.

ABSTRACT User data has been used by many companies to understand user behaviors and finding new business strategies. However, common techniques cannot be used when it comes to new products that have not yet been released due to the fact that there are no prior data available. In this work, we propose a framework for generating realistic user data on new products which can then be analyzed for insights. Our model uses Conditional Generative Adversarial Network (CGAN) with the Straight-Through Gumbel estimator which can also handle discrete-valued outputs. The CGAN is conditioned on product features learned using a recommendation system which can better capture the relationship between products. Experiments using a dataset consisting of view logs from a real estate listing website shows that our model outperforms other baselines on four performance metrics, and can effectively predict the finer characteristics of new products.

INDEX TERMS Generative adversarial networks, deep learning, generative model, data generation, Gumbel-softmax trick, product embedding.

I. INTRODUCTION

With the rapid growth of online service, websites become one of the main channels where people access their favorite content. The data from these websites, often collected in web logs become a source for mining insights about the users, which can be very valuable for business. Techniques such as association mining [1], sequential pattern mining [2], or clustering [3] can be used on web log data to improve profitability. For e-commerce websites, recommendation engines [4] can be trained on the logs to improve conversion rate and thus increase revenue.

Another business use case for the study of web log is for product development. Based on the web log data, if the business can identify the characteristics of target users who are likely to be interested in the new product before its release, they will have more useful insights for user targeting, marketing channel strategy and the best time to launch new campaigns without conducting a market survey. The goal of this work is to forecast users' responses to new products by learning from web log data.

Generative Adversarial Network (GAN) [9] is a generative model for learning from such distribution and generating new realistic samples. It has been successfully applied to computer vision [15], [16], [20], [21] and natural language processing [26], [27] domains. The model consists of two neural networks: a generator and a discriminator. The generator learns to generate new (fake) samples that look similar to the real sample. The discriminator learns to discriminate between real data and fake generated sample and send the feedback to the generator so that it can generate better fake samples.

Moreover, GAN can be trained to control the mode of generated output by adding some kind of conditional signal, such as class label, to both the generator and the discriminator.

The associate editor coordinating the review of this manuscript and approving it for publication was Kaitai Liang.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

This model is called a Conditional Generative Adversarial Network [11]. A relevant work [22] was proposed to apply CGAN for generating new plausible orders on the e-commerce website in order to forecast the demand, the seasonality, the characteristic of the customer, etc. The input of the model used for conditioning the order is the product name, so the model can be used to estimate the orders of new products on the platform. However, the orders generated by GAN are embeddings that are used as inputs to a separate set of models which will convert the embedding into meaningful information. This is due to the limitation of GAN that it cannot handle discrete value generation well. As a result, classifiers are also needed to be trained to extract the characteristics from generated order representation which makes the training process sophisticated and hard to maintain.

Similarly, our work also tries to generate realistic logs of user for new products. However, the data on the real estate domain comes from Home.co.th, a real estate listing website in Thailand, which has more than one million views per month. Given characteristics of a new real estate, such as the location, the price, the size, the facilities, etc. we can generate web logs which can be then aggregated to predicted the characteristics of target users for the real estate. Our key technical contributions are enumerated below:

- Recommendation based product embedding: prior works used product descriptions for conditioning signal. The product description was converted into word embeddings which mean only explicitly written product characteristic would be captured. It cannot capture other associations such as a person that likes condominiums that are closer to the public transportation system might prefer a condominium with a gym. In order to capture these kinds of relationship, we propose the use of recommender based embedding which is learned via a deep recommender system.
- A generative approach for generating web log data with discrete outputs: prior works used a two-step process in order to summarize user characteristics. On the other hand, we handle the limitation of discrete outputs for GAN by using Straight-Through Gumbel Estimator [23], [24] which help simplifying the pipeline and reducing error propagation of the models.

For evaluation, to subjectively evaluate outputs, we use four metrics to evaluate the results: Relative Similarity Measure (RSM) [22], Correlation Coefficient (CORR), Earth-Mover Distance (EMD) and Root Mean Squared Error (RMSE). We compare our model with two baseline generation approaches: the nearest neighborhood and a Conditional Variational Autoencoder (CVAE) [7]. The results show that our generator generates the realistic users that are indistinguishable and outperforms the baseline approaches in these metrics.

In the rest of this paper, the background and related works are described in section II. Section III explains the detail of our methodology and the experiment results are shown in section IV. The last section is the conclusion of this work.

II. BACKGROUND AND RELATED WORK

In this section, we review the related theories and existing works related to product embedding, Variational Autoencoder (VAE), Generative Adversarial Network (GAN) and Gumbel-Softmax distribution.

A. PRODUCT EMBEDDING

An embedding can be considered as a dense and compact representation of feature vectors. Good embedding should have the property that if the features are similar in some sense, their embedding vectors are closer. Word2Vec, which was proposed in [28], is a popular embedding technique for textual data. Two words are considered similar semantically by their surrounding words and co-occurrence. The authors proposed two methods to learn Word2Vec embedding: the Continuous Bag of Words (CBOW), which predict the current word based on surrounding words, and the Skip-gram, which predict context words given the current word.

For recommendation system, item embedding is a continuous vector which represents item and tries to capture the relationship of items. In [29], the authors proposed Item2Vec by using the Word2Vec framework. They assumed that the items which share the same basket are similar in some sense regardless of the order that user generates. They predicted a item based on other items in the same basket. The authors in [4] applied item embedding to improve the session-based recommendation task in an e-commerce website. They embedded an item description to its embedding by Word2Vec. In [30], the authors want to represent the relation between users and content for news recommendation. They generated user representation by using a recurrent neural network (RNN), whereas the content embedding was learned via denoising autoencoder. They shown that the click-through rate improved by 23% and the total duration improved by 10% over not using embeddings.

B. VARIATIONAL AUTOENCODER

Variational Autoencoder (VAE) [6] is a kind of Autoencoder that can learn the distribution of the data. VAE can be used to generate new data samples by sampling from the learned distribution. The model consists of an encoder and a decoder. The encoder embeds the input into a Gaussian distribution, which will be sampled and decoded into the original input by decoder. Unlike the Autoencoder, decoder of the VAE learns not only from a single point in the latent space but also from the nearby points in the latent space.

In application domains, the authors in [8] applied VAE to generate new items which maximally satisfy the preference of a group of user. They learned the shared latent representation between user and item features from user-item ratings. Embedding for product recommendations can be generated through weighted maximum coverage in a greedy manner.

The item decoder maps these latent representations to item features of new items.

However, VAE usually fails to capture multimodal distributions and usually generate lower quality outputs due to the gap between the lower bound of approximate posterior distribution and true data distribution [10]. Therefore, GAN is used as a variant of VAE since no variational bound is needed, which often leads to better generated results.

C. GENERATIVE ADVERSARIAL NETWORK

Generative Adversarial Network (GAN) [9] is a deep generative model which can generate realistic samples that are similar to the training data. The model is composed of a generator (G) and a discriminator (D). The generator takes a random noise input and learns to generate new realistic samples. The discriminator decides whether a sample is real or generated. GAN works as a minimax game where D tries to maximize the likelihood to recognize real samples as real and generated samples as fake, whereas G would like to minimize it. GAN aims to achieve an equilibrium between the generator and the discriminator so, the objective function of the original GAN for discriminator is defined as:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}\big[\log D(x)\big] - \frac{1}{2}\mathbb{E}_{z\sim p_z}\big[\log(1-D(G(z)))\big] \tag{1}$$

where z is a random normal distribution (0, 1) or uniform distribution (0, 1). On the generator side, the objective function is defined as:

$$J^{(G)} = -J^{(D)} \tag{2}$$

The authors in [11] proposed conditional GAN to control the output of the generator by incorporating some information (c) such as class labels and other modalities into both the generator and the discriminator. InfoGAN [12] is another work which conditioned the generation by adding the mutual information between the latent code and the generator output as a regularization term in the loss function.

GANs often suffer from various problems such as non-convergence, mode collapse, diminished gradients, etc. Wasserstein GAN was proposed in [13] to improve GAN by changing the loss function to Earth-Mover distance or Wasserstein distance. The output of the discriminator is a scalar which gives a score on how real the input sample is, so the authors renamed the discriminator to critic. To satisfy the constraint of Lipschitz continuity which is required to compute the distance, they added the weight clipping bound the weight of the critic. The difficulty in WGAN is tuning the bounds in weight clipping. If the bound is large, it can take a long time for the weights to converge. If the bound is small, it can lead to vanishing gradients when the network has many layers. To alleviate this issue, WGAN-GP [14] was subsequently proposed. Instead of the weight clipping, the authors added a gradient penalty term into the loss function to enforce the Lipschitz constraint during the training phase. The loss function of both the discriminator ($J_W^{(D)}$) and the generator ($J_W^{(G)}$) are defined as:

$$\begin{aligned} J_W^{(D)} &= -\mathbb{E}_{x\sim p_g}\big[D(x)\big] \\ &\quad + \mathbb{E}_{x\sim p_r}\big[D(x)\big] \\ &\quad - \lambda\mathbb{E}_{x\sim p_{x'}}\big[\parallel \nabla_x D(x) \parallel_2 - 1\big]^2 \qquad (3) \\ J_W^{(G)} &= -J_W^{(D)} \qquad (4) \end{aligned}$$

where the last term is gradient penalty. $p_{x'}$ is a uniform distribution along the straight lines between pairs of points which are sampled from the data distribution p_r and the generated data distribution p_g.

GAN has been successfully applied in many application domains such as computer vision and natural language processing. In [15], the authors proposed CycleGAN to transform images from one domain to another domain. The authors in [16], [17] applied VAE and GAN to repair and fill the missing parts of images, a tasked is called image inpainting. To improve the quality of generated samples, there are many works that combine VAE [6] and GAN, especially for images [18]–[21]. VAE is an excellent generative model for learning representations but generates blurry outputs, whereas GAN generates sharp outputs but cannot explicitly learn the embeddings like VAEs. In [21], the authors used two generative models to capture the latent spaces of hand poses and depth images for 3D hand pose estimation. VAE embedded features to the share latent representation and GAN's generator generated the 3D hand pose by latent representation.

For NLP, the limitation of GAN is in generating discrete outputs such as text because the gradient cannot be back propagated through the argmax function used to generate a discrete output. REINFORCE, a technique used in the reinforcement learning literature, can be used to circumvent this issue [26]. However, it can lead to slow convergence and training instability. In [27], the authors applied Gumbel-Softmax trick which was proposed in [23] for text generation to handle discrete outputs. For business applications, [22] was recently proposed to apply GAN with e-commerce data to generate the plausible orders related to a particular product in order to understand the characteristics of future orders.

D. GUMBEL-SOFTMAX DISTRIBUTION

Because of the difficulty of training stochastic networks with discrete variables, the authors in [23], [24] proposed Gumbel-softmax trick which can be used to backpropagate through the softmax. A multinomial distribution can be efficiently sampled by the following function which was proposed in [25]:

$$z = \text{onehot}(\text{argmax}_i[g_i + \log \pi_i]) \tag{5}$$

where π_i is class probability. g_i is a noise which is drawn from Gumbel (0,1). Onehot is a function to encode categorical variable into binary column for each category and returns a sparse matrix.

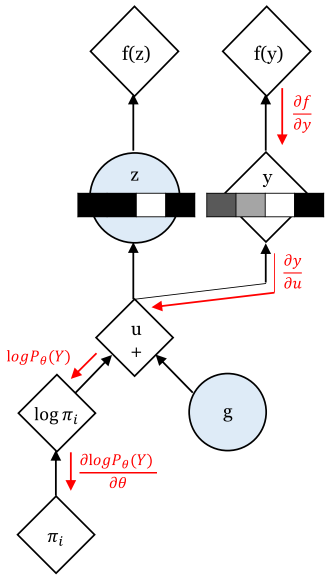
f(z)
f(y)
$\frac{\partial f}{\partial y}$
z
y
$\frac{\partial y}{\partial u}$
u
+
$\log P_\theta(Y)$
$\log \pi_i$
g
$\frac{\partial \log P_\theta(Y)}{\partial \theta}$
π_i

FIGURE 1. Gradient estimation in the Straight-Through Gumbel Estimator. Black arrows indicate the forward pass, and red arrows indicate the backward pass.

Because argmax is non-differentiable function, the Gumbel-Softmax estimator [23], [24] was proposed by using the softmax function to approximate the argmax as:

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} \exp((\log(\pi_i) + g_i)/\tau} \tag{6}$$

where τ is the softmax temperature parameter. When τ approaches zero, the expected value of the Gumbel-Softmax distribution is identical to a discrete distribution. At higher τ, the expected value converses to a uniform distribution. The Gumbel-Softmax function yields a smooth gradient at high τ, but can become unstable at low τ. At high τ can be used at the start of the training and anneal to a small but non-zero value.

For scenarios that requires sampling of discrete values such as ones in GAN, the authors proposed the Straight-Through (ST) Gumbel Estimator. The forward pass is done by normal sampling, but the backward pass is done by backpropagating the Gumbel-softmax to approximate the gradient which is shown in Figure 1.

III. METHODOLOGY

In this section, we describe the details of the dataset and our system for generating user logs. Figure 2 shows an overview of our system. First, we embed the product features with some embedding model. Then, we feed the product embedding to the generation model which will output user logs. The logs can be analyzed to extract insight about the product used as the input.

A. DATASET

Our web log data are from a real-estate search engine website[1] from January 2018 to February 2018. There are around 1.5 million records and 5,400 property projects. An example of the web log data is given in Table 1. It consists of the user ID, the project that the user visited, the device that the user used to access the website, the agent or operation system, the referring page that the user was referred from, and the time of visit.

[1]https://www.home.co.th

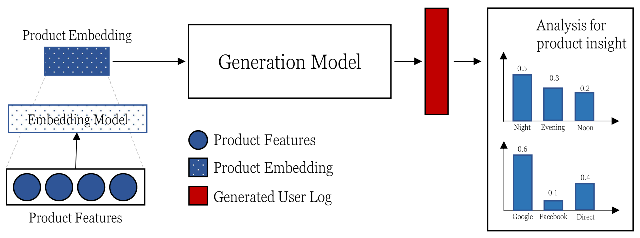

FIGURE 2. The overview of our system.

TABLE 1. Example web logs in our dataset.

user_id	project	device	agent	referring	y	m	d	h
8bx-xx-xxx	5174	Mobile	iPhone	Google	2018	1	15	8
8bx-xx-xxx	5476	Mobile	iPhone	Google	2018	1	15	8
1ax-xx-xxx	7956	Desktop	Windows	Direct	2018	1	15	12
f7x-xx-xxx	3924	Mobile	Android	Others	2018	1	16	21

TABLE 2. Examples of user features extracted.

device	agent	referring	cluster	weekday	period
Mobile	iPhone	Google	2	Mon	Morning
Desktop	Windows	Direct	1	Thu	Night
Mobile	Android	Facebook	35	Sat	Afternoon

The user features (U_i) are created from the web log which contain the follow features:

- Customer characteristics: device (mobile, desktop), agent or operation system (Android, iPhone, iPad, Macintosh, Windows, others), customer segmentation (based on k-mean clustering)
- Channel: referring pages (Google, Facebook, direct, others)
- Visit period: day of the week, period (morning, afternoon, evening, night)

An example of features constructed from each log entry is shown in Table 2.

To give a sense of our data, we show the histogram of each features in Figure 3. The users mostly use mobile devices to access the website (60.6%). Most sessions were referred from Google and the peak time period is during 13:00-18:59.

We also use product features (C_j) which capture the characteristics of the each property products. The product features include: the starting price, the location, the nearest train station, the latitude, the longitude, the area, the district, the product type, and the facilities. The product feature is used to condition the generation algorithm. We grouped features to their category as follow:

- Start period of project
- Location such as latitude, longitude, and district

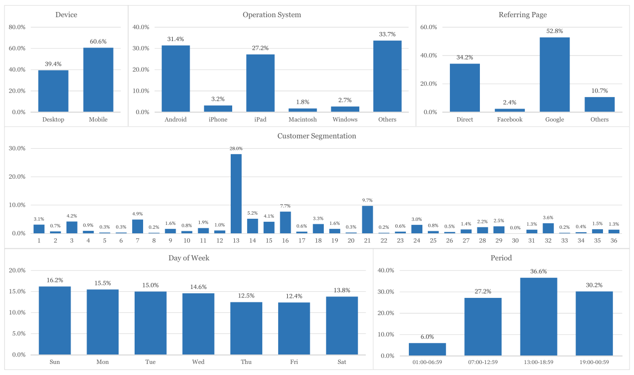

FIGURE 3. The distribution of each feature.

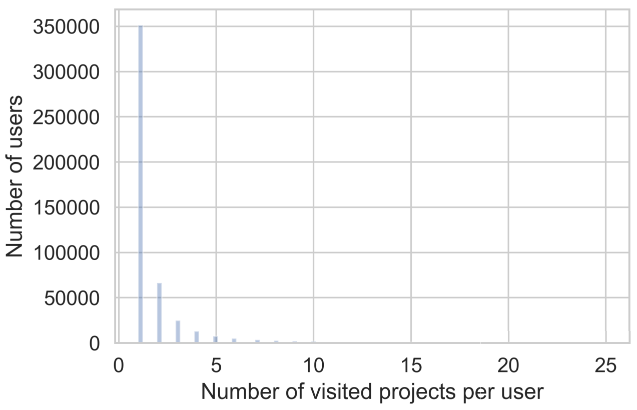

FIGURE 4. The distribution of number of property products per user who visited more than a single property product.

- Transportation and landmark such as train station, express way, and supermarket
- Facility such as swimming pool, parking lots, and gym
- Project type and style such as condominium, detached house, and home office
- Others such as starting price, and area

To reduce spurious information, we filtered out some projects that were visited less than 50 users which result in 4,876 projects remaining.

B. PRODUCT EMBEDDING

One of the key components of our model is the product embedding. It aims to encapsulate each product's peculiarity so that the generator can have an easier time generating new visit logs. We compare two kinds of model for learning our embeddings: Autoencoder and Recommendation system.

1) AUTOENCODER (AE)

the objective of the AE is to compress the original input and learn the best embedding that can be used to reconstruct the original input. The embedding is usually lower dimension than the original input features so that the mapping is not trivial. We use a six-layer autoencoder with (256, 128, 96, 96, 128, 256) neurons at each layer, resulting in an embedding of size 64.

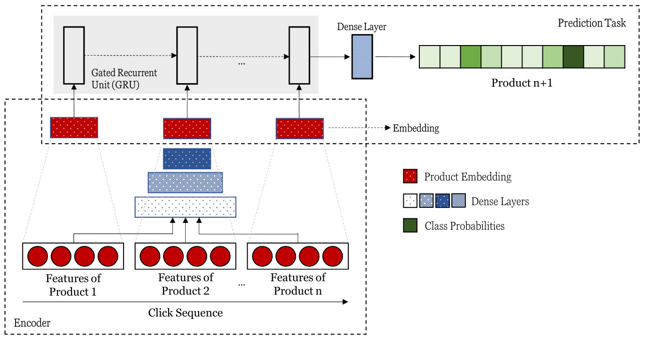

FIGURE 5. The recommendation system used to produce product embeddings.

2) RECOMMENDATION SYSTEM (REC)

We can also extract the embedding via a trained recommendation system. The goal is to capture additional properties of the product that are not captured in the pattern of how users explore the products. Our system is similar to the recurrent neural network-based recommender. Our model consists of two parts: the encoder and the predictor as shown in Figure 5. The recommendation model takes the sequence of products visited by each user as input and tries to predict the next product that the same user would visit. The sequence of product is represented by their product features. The product features are embedded into an embedding via a three fully connected layers. After that, the embeddings are fed to a Gated Recurrent Unit (GRU) which is used to predict the next product.

After training the model, the embedding vector from the encoder is the embedding for each product. This product embedding vector should capture the relationship between products which is not only the similar characteristics but also the similar user preference.

C. GENERATIVE MODEL

The key of our system is the generative model which takes in a product embedding and outputs user logs. Our GAN-based generator is shown in Figure 6. The model is similar to [22] which generates the orders of new product in e-commerce website. However, their generated orders are in a form of embedding due to the limitation of GAN to handle discrete output generation, they need to train classifiers to extract the characteristics from the generated representation. Our work uses the Straight-Through Gumbel Estimator [23], [24] in order to deal with discrete generation.

- The generator (G) generates fake user features conditioned on the product embedding. In other words, the generator will try to generate users that are likely to interact with the particular product. The generator is a fully-connected network with three hidden layers and uses LeakyRelu as an activation function at each layer. For the last layer, we use the Straight-Through Gumbel estimator to generate the discrete outputs and the tanh function to generate the continuous outputs which were normalized to the range of [-1, 1].

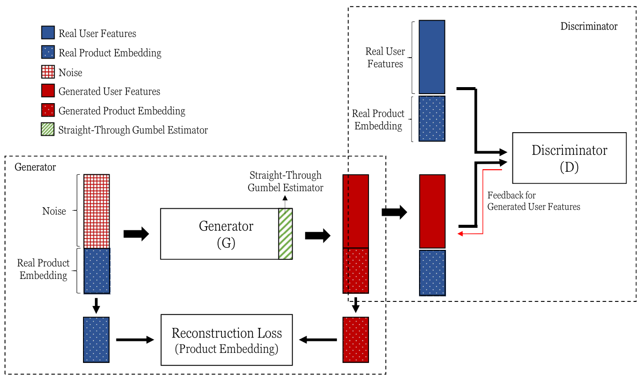

FIGURE 6. The GAN model for web log generation.

- The discriminator (D) takes the concatenation of real or generated user features and the real product embedding and then decides whether the user feature is real or fake. This model is a fully-connected network with two hidden layers and uses LeakyRelu as an activation function of each layer. The last layer use a linear activation function to output the score indicating how real the users are based on the real product embedding. This information can be used to guide the generator to generate better fake users via backpropagation. If this value is small, the generated users are close to the real data.
- To ensure that the model learns to generate users based on the product used for conditioning, we also forced the model to learn about the product by adding a reconstruction loss ($L^{(p)}$). The generator not only generates the users, but also the product embedding. The reconstruction loss is the cosine distance between the input product embedding and the product embedding at the output. For the generated users, the loss function is the WGAN-GP loss. Thus, the overall loss function for the generator is the weighted sum of reconstruction loss and user generator loss that is defined as $\alpha L^{(u)}+(1-\alpha)L^{(p)}$ where α is a hyperparameter that we need to tune.

For hyperparameter tuning, we used grid-search to obtain the optimal configurations or hyperparameters of the models which provide the highest performance score for this dataset.

- For the generator noise, we tried different vector sizes $\{36, 64, 96, 128\}$ and found that 64-dimension noise gives the highest performance for this model.
- The generator configuration uses a structure $64 \longrightarrow 128 \longrightarrow 256$ as the hidden units. The last layer outputs a vector of length $52 + 70 = 122$ which is the number of user and product features.
- The discriminator has two hidden layers $128 \longrightarrow 64$ and the last layer is linear with size 1 to measure how real the input users are.
- τ in ST Gumbel Estimator, we used 0.9 to be the initial value with anneal rate of 0.005. The final value is 0.35.
- The optimizers, we tried $\{$Adam, SGD, RMSprop$\}$ and found that Adam with $\beta = 0.7$ gave the best result.
- The ratio of number of times the generator is trained to the discriminator is 1:5.

IV. EXPERIMENT

We performed experiments to verify the effectiveness of our generation system. In this section we will talk about the experimental setups and the results of each experiment.

A. EXPERIMENTAL SETUP

We construct the training and test set by randomly select 50 products from the total products to be treated as new products. The training set contains all of the products that are not selected for testing, meaning the test are completely unseen by the model. We repeat the selection 10 times to construct model 10 independent training and test sets.

B. EVALUATION METRICS

Our goal is to have the model predict the distribution of web logs given unseen products. Thus, we cannot measure the performance of the web log individually. We have the model generate 10, 000 web log per test products and then measure the statistically properties of the log generated with respect to the ground truth. Prior work used Relative Similarity Measure (RSM) which captures the characteristic of each attribute of the generated product relative to other products [22]. However, this measure does not capture higher order statistics. Moreover, some use cases might require precise knowledge of the distribution rather than relative difference. Thus, we also propose three other metrics that can be used to measure the quality of the generated logs. The four metrics can be summarized as follow:

1) RELATIVE SIMILARITY MEASURE (RSM)

this measure is used to measure the relative similarity between real and generated samples by comparing between two products. The concept of computing this measure is shown in Algorithm IV-B.1.

An example in Figure 7, the first low level of referring page feature is Facebook. The portion of real users that were referred from Facebook is 30% ($s^r_{A(facebook)}$) and 20% ($s^r_{B(facebook)}$) for product A and product B respectively. The $s^r_{A(facebook)}$ is 10% higher than $s^r_{B(facebook)}$. In the same way, the product A's generated users were referred from Facebook 50% ($s^g_{A(facebook)}$) and is also higher than $s^g_{B(facebook)}$. Thus, we count this as an relative similarity between real users and generated users. Similarly, the portion of real users that were referred from Google is 20% for product A ($s^r_{A(google)}$) and 50% for product B ($s^r_{B(google)}$). The $s^r_{A(google)}$ is lower than the $s^r_{B(google)}$. In comparison with generated users, $s^g_{A(google)}$ is also lower than $s^r_{B(google)}$. The last one is directing visitors. The real direct users is 50% ($s^r_{A(direct)}$) and 30% ($s^r_{B(direct)}$) for product A and product B respectively. The $s^r_{A(direct)}$ is higher than $s^r_{B(direct)}$. This is the same direction as the comparison between $s^g_{A(direct)}$ and $s^g_{B(direct)}$. Thus, we count the generated users as high relative similarity with the real users on all three low-level referred features.

Algorithm 1 RSM Metric Calculation

```
Require: the list of testing products ($p$), the number of testing products ($n_p$), the real users statistics ($U^r$), the generated users statistics ($U^g$)
1:  $n = 0$
2:  $n_{count} = 0$
3:  for $f$ in referring feature do
4:      select $u^r$ == f and $u^g$ == f from $U^r$ and $U^g$ respectively
5:      for each product pair $(i, j)$ do
6:          select the $s_i^r$ and $s_j^r$ from $u^r$
7:          select the $s_i^g$ and $s_j^g$ from $u^g$
8:          if $(s_i^r > s_j^r \wedge s_i^g > s_j^g) \vee (s_i^r < s_j^r \wedge s_i^g < s_j^g) \vee (s_i^r = s_j^r \wedge s_i^g = s_j^g)$ then
9:              $n_{count} = n_{count} + 1$
10:         end if
11:         $n = n + 1$
12:     end for
13:     $rsm = n_{count} \div n$
14:     return $rsm$
15: end for
```

Algorithm 2 Correlation Coefficient Calculation

```
Require: the list of testing products ($p$), the number of testing products ($n_p$), the real users statistic ($U^r$), the generated users statistic ($U^g$).
1:  $n = 0$
2:  $n_{count} = 0$
3:  for each product $i$ do
4:      select $s_i^r$ from $U_i^r$
5:      select $s_i^g$ from $U_i^g$
6:      if $\rho_{s_i^r, s_i^g} >= 0.85$ then
7:          $n_{count} = n_{count} + 1$
8:      end if
9:  end for
10: $corr = n_{count} \div n_p$
11: return $corr$
```

2) CORRELATION COEFFICIENT (CORR)

this metric measures the strength of the relationship between real and generated samples as the formula:

$$\rho_{p_r, p_g} = \frac{\text{cov}(p_r, p_g)}{\sigma_{p_r}\sigma_{p_g}} \tag{7}$$

where $\text{cov}(p_r, p_g)$ is the covariance of p_r and p_g, σ_{p_r} is the standard deviation of p_r and σ_{p_g} is the standard deviation of p_g. The value is between -1 and 1 where -1 means negative correlation and 1 means positive correlation. An example in Figure 7, the real users were referred from Google 20%, Facebook 30% and direct 50%. Meanwhile, the generated users were from Google 10%, Facebook 50% and direct 40%. The correlation between [0.2, 0.3, 0.5] and [0.1, 0.5, 0.4] is 0.58. If the correlation coefficient ≥ 0.85, we count that as a high correlation between real and generated users.

3) WASSERSTEIN DISTANCE OR EARTH MOVER'S DISTANCE (EMD)

this metric measures the distance between two probability distributions as the formula:

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\ \| \text{x-y} \|\ \right] \tag{8}$$

where P_r is the probability distribution of real users, P_g is the probability distribution of generated users and $\Pi(P_r, P_g)$ is the set of all distributions. We use this metric to calculate the minimum cost of transforming the generated user distribution into the real user distribution. As Figure 7, the distance between real user distribution and generated user distribution is 0.2. The advantage of EMD is that even when two distributions are not overlaps, this measure can provide the distance value between two distributions, whereas Kullback-Leibler divergence (D_{KL}) provides the infinity when two distributions are disjoint.

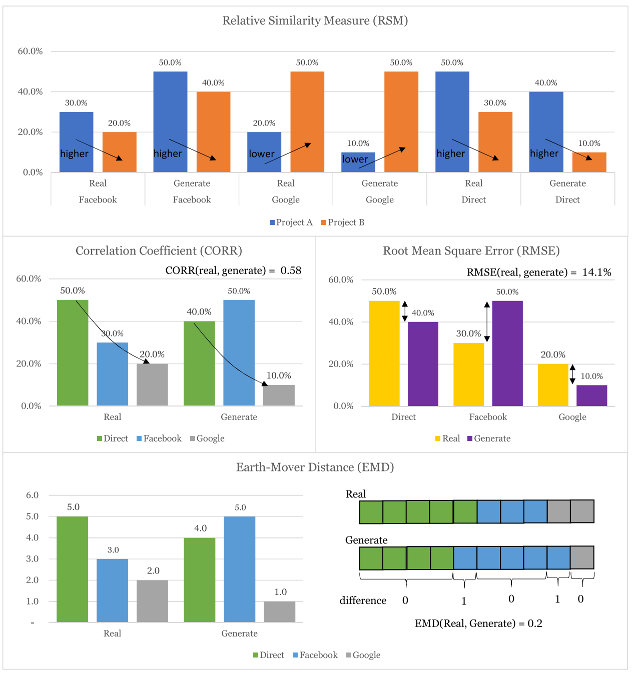

FIGURE 7. Example of our metrics.

4) ROOT MEAN SQUARE ERROR (RMSE)

this metric is the average of difference between proportion of real and generated users of each feature as the formula:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(p_r - p_g)^2}{n}} \tag{9}$$

where p_r is the proportion of real users, p_g is the proportion of generated users and n is the number of testing projects. RMSE measures how accurately the generated users are in the same unit as the data. Thus, this measure is easy for interpretation. The lower score is better. As Figure 7, the RMSE between [0.5, 0.3, 0.2] and [0.4, 0.5, 0.1] is 0.141 or 14.1%. It means that the difference between real and generated users is 14.1%.

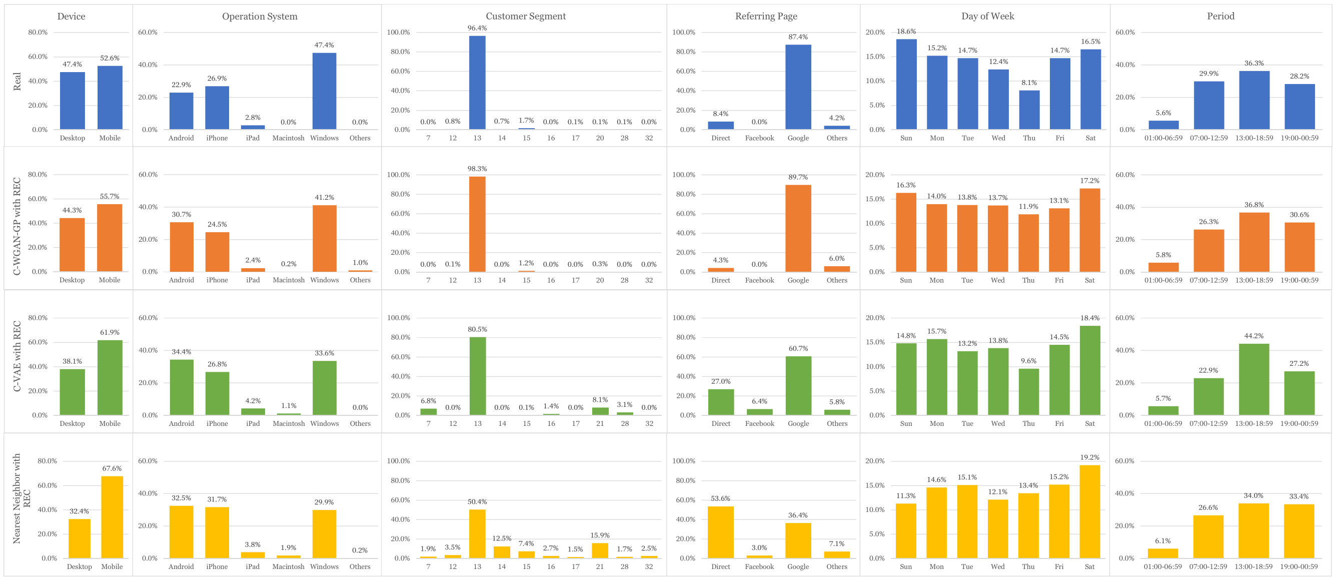

FIGURE 8. An example of the distribution comparison of each user features between the ground-truth users and generated users.

To summarize the evaluation metrics as Figure 7, this is an example based on referring page feature. RSM score measures the relative similarity of real and generated users by comparing between two products. This example shows that the generated users are counted as high relative similarity with the real data. The correlation coefficient (CORR) is used to measure the correlation of real and generated users within the same product. The correlation of this example is 0.58, we do not count as high correlation with the real data. The next one is Earth-Mover distance (EMD) that measure the minimum cost to transform one distribution into the other. This shows that the cost of converting the generated distribution to the real distribution is 0.2. The last one is RMSE that shows the precision of the generator. This metric measures how close the proportion of generated users on each characteristic is to the real users.

The metrics are measured on each feature and average. However, we can also measure the metrics in a multivariate manner (2 features). For example, RSM will be measure the relative similarity between a tuple of two features instead such as (referring page, weekday), (device, operation system), etc.

C. BASELINE GENERATION APPROACH

We compare our approach with two baseline approaches: nearest neighborhood approach and Conditional Variational Autoencoder (CVAE).

- We apply the nearest neighbor (NN) concept to summarize the characteristics of users who are likely to be interested in new product. We select the user log of top 5 existing products that their characteristics are similar to the characteristics of new product. Thus, we know the list of possible values of each user feature for new product and sample based on that distribution.

TABLE 3. The overall performance for 1 feature based on 4 metrics.

Model	1 Feature			
	RSM	CORR	EMD	RMSE
C-WGAN-GP with REC Embedding	**72.5%**	**88.9%**	**0.59**	**16.2%**
C-WGAN-GP with AE Embedding	69.7%	87.8%	0.77	18.1%
C-WGAN-GP with Product Features	67.9%	86.6%	0.83	18.2%
C-VAE with REC Embedding	65.3%	85.6%	1.23	20.3%
NN with REC Embedding	54.7%	71.6%	1.69	28.0%

- The conditional variational autoencoder (CVAE), which was proposed in [7], is an extension of Variational Autoencoder [6]. This model can control on the data generation process to generate some specific output by adding the additional information to both encoder and decoder. We use this model and add the product embedding to generate the user logs of new product.

D. RESULT ANALYSIS

The overall performance of our methodology (C-WGAN-GP) that is compared against several baseline approaches as shown in Table 3 and 4 for 1 feature and 2 features respectively. Our proposed approach which used conditional GAN with embedding learned from recommendation system performed the best on every metrics. This shows the effectiveness of our approach in learning the distribution of new products. The effectiveness of the learned embedding is shown when we compare different embeddings. For embedding from autoencoder, the original product features only improve the performance slightly on several metrics, but using recommendation embedding shows significant gain on all metrics.

Our simplest baseline is a nearest neighbor model (NN). The nearest products in the training set are used as the statistics of the new product. This model uses cosine distance on the recommendation embedding to select the top 5 nearest

TABLE 4. The overall performance for 2 features based on 4 metrics.

Model	2 Feature			
	RSM	CORR	EMD	RMSE
C-WGAN-GP with REC Embedding	**81.9%**	**82.5%**	**1.17**	**19.4%**
C-WGAN-GP with AE Embedding	78.8%	80.1%	1.98	21.4%
C-WGAN-GP with Product Features	78.4%	78.4%	2.56	21.6%
C-VAE with REC Embedding	72.2%	76.5%	3.27	23.8%
NN with REC Embedding	62.9%	39.8%	5.81	34.3%

TABLE 5. The performance of our approach for each feature.

Feature	Model	RSM	CORR	EMD	RMSE
Device	**C-WGAN-GP with REC**	**69.3%**	**84.4%**	**0.059**	**10.0%**
	CVAE with REC	52.4%	83.4%	0.087	10.7%
	NN with REC	51.6%	83.2%	0.102	14.1%
Operation System	**C-WGAN-GP with REC**	**68.9%**	**99.5%**	**0.261**	**12.6%**
	CVAE with REC	57.5%	97.8%	0.357	14.3%
	NN with REC	49.5%	92.7%	0.431	15.9%
Customer Segmentation	**C-WGAN-GP with REC**	**80.4%**	**75.1%**	**2.091**	**24.4%**
	CVAE with REC	71.4%	60.4%	4.492	53.1%
	NN with REC	57.2%	1.5%	8.195	85.8%
Referring	**C-WGAN-GP with REC**	**68.8%**	**80.2%**	**0.148**	**19.4%**
	CVAE with REC	59.6%	78.6%	0.271	26.6%
	NN with REC	50.1%	54.7%	0.341	36.1%
Weekday	**C-WGAN-GP with REC**	**69.6%**	**100%**	**0.038**	**6.5%**
	CVAE with REC	50.7%	98.5%	0.052	7.1%
	NN with REC	46.2%	99.8%	0.068	7.1%
Time Period	**C-WGAN-GP with REC**	**73.9%**	**97.4%**	**0.094**	**8.7%**
	CVAE with REC	51.8%	90.6%	0.125	10.4%
	NN with REC	50.1%	96.3%	0.185	9.0%

products. Unsurprisingly, this method performs the worst, since for real estate there are rarely two products that are similar to each other.

We also trained another baseline based on CVAE with recommendation embedding. The CVAE baseline performed worse than all other GAN models, showing the effectiveness of GANs in learning the distribution of the customers.

We also show the performance of models for each feature as shown in Table 5. The most interest is in the customer segmentation which has 36 possibilities. The proposed model can get 75% CORR meaning that it can be used to give some guidance on what kind of customer would prefer the product. On the other hands, a NN approach which is something a human might have done based on his limited experience, would yield abysmal results.

We show one example of the generated distribution in Figure 8. Note how our model yields an estimate for customer segment number 15 as 1.2%, which is very close to the actual distribution of 1.7%. In a highly imbalance case such as this one, it is very hard for models besides GANs to uncover the long tail of the distribution. This shows the effectiveness of our model.

V. CONCLUSION

We have proposed an approach for generating realistic users that would be interested in a new product on a real-estate search engine website. Our method consists mainly of two parts: a product embedding model and a generation model. Each product is embedded into its low-dimensional representation by using a recommendation system that can help capturing the relationship between similar products. The generation model is a CGAN with WGAN-GP loss. We evaluated the quality of the distributions learned based on four metrics: the relative similarity measure (RSM), the correlation coefficient (CORR), the earth-mover distance (EMD) and the root mean square error (RMSE). The result demonstrates that our approach outperforms the baselines, and can generate users which are similar to the real data even for highly imbalanced cases. In the future we plan to add a demand forecasting component, since the demand could not be inferred from just the generated logs.

ACKNOWLEDGMENT

The authors would like to thank P. Imemkamon for the user clustering used in this work, and E. Leelasornchai for his help on the early prototypes.

REFERENCES

[1] G. Suchacka and G. Chodak, "Using association rules to assess purchase probability in online stores," *Inf. Syst. e-Bus. Manage.*, vol. 15, no. 3, pp. 751–780, Sep. 2016.
[2] P. Weichbroth, "Frequent sequence mining in Web log data," in *Proc. Int. Conf. Man–Mach. Interact.* Springer, 2018, pp. 459–467.
[3] S. Hao, S. Zhaoxiang, and Z. Bingbing, "A user clustering algorithm on Web usage mining," in *Proc. 1st Int. Conf. Electron. Instrum. Inf. Syst. (EIIS)*, Jun. 2017, pp. 1–4.
[4] A. Greenstein-Messica, L. Rokach, and M. Friedman, "Session-based recommendations using item embedding," in *Proc. 22nd Int. Conf. Intell. User Interfaces (IUI)*, 2017, pp. 629–633.
[5] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. ICML Workshop Unsupervised Transf. Learn.*, 2012, pp. 37–49.
[6] P. D. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. 2nd Int. Conf. Learn. Represent.*, 2014, pp. 1–14.
[7] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Proc. Adv. Neural Inf. Process. Syst. 28*, 2015, pp. 3483–3491.
[8] T. V. Vo and H. Soh, "Generation meets recommendation: Proposing novel items for groups of users," in *Proc. 12th ACM Conf. Rec. Syst. (RecSys)*, 2018, pp. 145–153.
[9] I. Goodfellow, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 2672–2680.
[10] I. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," Apr. 2017, *arXiv:1701.00160*. [Online]. Available: https://arxiv.org/abs/1701.00160
[11] M. Mirza and S. Osindero, "Conditional generative adversarial nets," Nov. 2014, *arXiv:1411.1784*. [Online]. Available: http://arxiv.org/abs/1411.1784
[12] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. NIPS*, 2016, pp. 2172–2180.
[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 214–223.
[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. NIPS*, 2017, pp. 5767–5777.
[15] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2223–2232.
[16] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5505–5514.
[17] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: feature learning by inpainting," in *Proc. CVPR*, 2016.
[18] A. B. L. Larsen, S. K. Sønderby, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Proc. 33rd Int. Conf. Int. Conf. Mach. Learn.*, 2016, pp. 1558–1566.
[19] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua, "CVAE-GAN: Fine-grained image generation through asymmetric training," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 2764–2773.
[20] S. H. Khan, M. Hayat, and N. Barnes, "Adversarial training of variational autoencoders for high fidelity image generation," 2018, *arXiv:1804.10323*. [Online]. Available: http://arxiv.org/abs/1804.10323

[21] C. Wan, T. Probst, L. Van Gool, and A. Yao, "Crossing nets: Dual generative models with a shared latent space for hand pose estimation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017.
[22] A. Kumar, A. Biswas, and S. Sanyal, "eCommerce-GAN: A generative adversarial network for e-commerce," 2018, *arXiv:1801.03244*. [Online]. Available: http://arxiv.org/abs/1801.03244
[23] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-softmax," 2016, *arXiv:1611.01144*. [Online]. Available: https://arxiv.org/abs/1611.01144
[24] J. C. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–20.
[25] J. C. Maddison, D. Tarlow, and T. Minka, "A* sampling," in *Proc. Annu. Conf. Neural Inf. Process. Syst.*, 2014, pp. 1–9.
[26] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence generative adversarial nets with policy gradient," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017.
[27] M. J. Kusner and J. M. Hernández-Lobato, "GANS for sequences of discrete elements with the Gumbel-softmax distribution," 2016, *arXiv:1611.04051*. [Online]. Available: https://arxiv.org/abs/1611.04051
[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. ICLR Workshop*, 2013, pp. 1–12.
[29] O. Barkan and N. Koenigstein, "ITEM2VEC: Neural item embedding for collaborative filtering," in *Proc. IEEE 26th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Sep. 2017, pp. 1–6.
[30] S. Okura, Y. Tagami, S. Ono, and A. Tajima, "Embedding-based news recommendation for millions of users," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2017, pp. 1933–1942.

PARICHAT CHONWIHARNPHAN received the B.Sci. degree from the Faculty of Commerce and Accountancy, Cuhlalongkorn University, Bangkok, Thailand, in 2014, where she is currently pursuing the M.Sci. degree with the Faculty of Computer Engineering.

PIPOP THIENPRAPASITH received the Ph.D. (Eng.) degree from the Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand, in 2017. He currently works as a Chief Data Officer at Home Dot Tech Company, Ltd.

EKAPOL CHUANGSUWANICH received the B.S. and S.M. degrees in electrical and computer engineering from Carnegie Mellon University, in 2008 and 2009, respectively, and the Ph.D. degree from Massachusetts Institute of Technology (MIT), in 2016. He joined the Spoken Language Systems Group, MIT Computer Science and Artificial Intelligence Laboratory. He is currently a Faculty Member at the Department of Computer Engineering, Chulalongkorn University. His research interests include speech processing, assistive technology, and health applications.