A Sub-linear, Massive-scale Look-alike Audience Extension System

Qiang Ma QIANG@YAHOO-INC.COM
Musen Wen MWEN@YAHOO-INC.COM
Zhen Xia ZHENXIA@YAHOO-INC.COM
Datong Chen DATONG@YAHOO-INC.COM
Yahoo! Inc., 701 First Avenue, Sunnyvale, CA 94089

Abstract

Look-alike audience extension is a practically effective way to customize high-performance audience in on-line advertising. With look-alike audience extension system, any advertiser can easily generate a set of customized audience by just providing a list of existing customers without knowing the detailed targetable attributes in a sophisticated advertising system. In this paper, we present our newly developed graph-based look-alike system in Yahoo! advertising platform which provides look-alike audiences for thousands of campaigns. Extensive experiments have been conducted to compare our look-alike model with three other existing look-alike systems using billions of users and millions of user features. The experiment results show that our developed graph-based method with nearest-neighbor filtering outperforms other methods by more than 50% regarding conversion rate in app-install ad campaigns.

1. Introduction

Using look-alike audience in on-line advertising campaigns helps an advertiser reach users similar to its existing customers. It can be used to support many business objectives like targeting users who are similar to a list of past purchasers, web service subscribers, installers of particular apps, brand lovers, customers from customer relationship management (CRM) systems, ad clickers, supporters for a politician, fans of a sports team, etc.

The advantage of look-alike audience extension technology is that it can dramatically simplify the way to reach highly relevant people comparing to other targeting technologies in on-line advertising. An advertiser can just upload a list of users to generate customized audience without knowing any details about the user features used in an advertising system.

The input to a look-alike audience system is a list of user IDs (e.g., browser cookies, addresses, phone numbers or any other identifiers[1]) called "seeds". The seed users can be converters of advertiser's past ad campaigns, or the existing customers who have stronger purchasing power, etc. The output is a list of users who are believed to be look-alikes to the seeds according to certain look-alike model. Then advertisers can target to show ads to those identified look-alike audiences in ad campaigns. The efficacy of a look-alike audience system is measured by multiple business concerns:

1. The original values of Personally identifiable information (PII), such as phone number, are not used. Instead, the hashed and anonymized user IDs are used in the system.

- **Scalability** - What is the maximum size of the look-alike audience output? What are the upper and lower bounds of the seed amount that can lead to a reasonable size of look-alike audience?
- **Performance** - How fast can the system generate a look-alike audience after advertiser uploading the seed list? How much return on investment (ROI) lift can look-alike audience achieve in a real ad campaign?
- **Transparency** - Is the system working as a black box or can it generate feedback and insights during an ad campaign setup? How fast can these feedbacks and insights be generated, in a few seconds or days?

In other words, there is no unique "best" design of a look-alike audience extension system given different requirements on scalability, performance and model transparency. The main contributions of our work are summarized as follows:

- We present a large-scale look-alike audience extension system from Yahoo!, where the similar user query time is sub-linear to the number of query users. The core model of this look-alike system is based on graph mining and machine learning technique on pairwise user-2-user similarities. The system can score 3000+ campaigns, on more than 3 billion users in less than 4 hours.
- Through extensive experiments using real-world app installation campaigns data, we show that the recommended look-alike audience by our method can achieve more than 50% lift in app installation rate over other existing audience extension models.
- Furthermore, we also discuss the challenges and share our experience in developing large-scale look-alike audience extension system. We demonstrate that by using weighted user-2-user similarity graph, the app installation rate can be further improved by 11%.

In the rest of paper, we start with discussions on three existing approaches in Section 2, which are used in experiments for performance comparisons; then in Section 3 we introduce our nearest neighbor filtering based look-alike audience extension method. We present the experiment results in Section 4, which is followed by discussions in Section 5. At last, the related work are discussed in Section 6, and Section 7 has concluding remarks.

2. Existing Look-alike Methods & Systems

The look-alikeness between 2 users is measured by their features. A user u_i can be characterized by a feature vector, $\mathbf{f}_i = (f_{i,1}, \cdots, f_{i,K})$. In an on-line advertising system, each feature $f_{i,k}$ could be continuous (e.g. time spent on a certain site), or in most situations, categorical or binary (e.g. use of a certain mobile app or not). Features can come from a user's attributes, behaviors, social interactions, pre-build audience segments and so on. Different companies may use different features to represent a user based on their businesses and understanding of users. The dimension of a feature vector varies from thousands to millions. In this paper, we consider the case where $f_{ij} \in \{0, 1\}$, and continuous features are bucketized into multiple binary features if needed.

Although the business motivation is simple and straightforward, we share the same view of Shen et al. (2015) that there is a significant lack of prior work of look-alike audience modeling in literature. To our best knowledge, look-alike systems being used in the on-line

advertising industry can be categorized into three categories: simple similarity-based model, regression-based model, and segment approximation-based model.

2.1. Simple Similarity-based Look-alike System

A straightforward method to find look-alike users is to compare all pairs of seed users and available users in the system, then determine look-alikeness based on distance measurements. A simple similarity-based look-alike system can use direct user-2-user similarity to search for users that look like (or in other words, be similar to) seeds. The similarity between two users, u_i and u_j is defined upon their feature vectors $sim(\mathbf{f}_i, \mathbf{f}_j)$. Cosine similarity (for continuous features, Equation 1) and Jaccard similarity (for binary features, Equation 2) are two possible measures.

$$sim_{cosine}(\mathbf{f}_i, \mathbf{f}_j) = \frac{\mathbf{f}_i \cdot \mathbf{f}_j}{||\mathbf{f}_i||||\mathbf{f}_j||} \tag{1}$$

$$sim_{Jaccard}(\mathbf{f}_i, \mathbf{f}_j) = \frac{\sum_{g=1}^{K} min(\mathbf{f}_{ig}, \mathbf{f}_{jg})}{\sum_{g=1}^{K} max(\mathbf{f}_{ig}, \mathbf{f}_{jg})} \tag{2}$$

The similarity between a given user u_i (with feature vector $\mathbf{f}_i$) and seed user set $S = \{u_j : \mathbf{f}_j\}$, could be calculated as the similarity of this user to the most similar seed user, i.e.

$$sim(\mathbf{f}_i, S) = \max_{\mathbf{f}_j \in S} sim(\mathbf{f}_i, \mathbf{f}_j) \tag{3}$$

A more complicated method is probabilistic aggregation of user-2-user similarity measures. This method only works with those user-2-user similarities that are strictly limited between 0 and 1. A pairwise similarity is treated as the probability of triggering a "look-alike" decision. Therefore, the similarity from a user to seed set can be measured as:

$$sim(\mathbf{f}_i, S) = 1 - \prod_{\mathbf{f}_j \in S} (1 - sim(\mathbf{f}_i, \mathbf{f}_j)) \tag{4}$$

The simple similarity-based method is easy to implement on a small scale. It also has the advantage to leverage the information carried by all seeds in the user feature space. However, this method has a couple of the main concerns. The first is scalability because calculating pairwise similarities among a large number of users is not a trivial task. The brute force solution has computational complexity $O(kMN)$, for N candidate users and M seeds with k features on average per user. In a typical online advertising market, there are billions of candidate users and more than tens of thousands of seeds with hundreds of features per user. The second concern is different features of seed users are treated equally, and the model does not distinguish their power of identifying the most relevant candidate users. For example, the predicting power for ad clicks or conversions is not considered. Therefore, some less relevant candidate users may be regarded as look-alike users due to similarity from less important features.

2.2. Regression-based Look-alike System

Another type of look-alike audience systems for online advertising is built with Logistic Regression (LR) Qu et al. (2014). For a given user u_i who has a feature vector $\mathbf{f}_i$, a logistic regression model can be trained and model the probability of being lookalike to seeds as:

$$p(u_i \textit{ is a lookalike to seeds}|\mathbf{f}_i) = \frac{1}{1+e^{-\mathbf{f}_i'\boldsymbol{\beta}}}$$

"Seeds" are usually treated as positive examples. Whereas, there are several options to select negative examples. The simplest negative example set can be a sample of the non-seed users. This is not the best choice since the potential ad converters may also be included in the non-seed users. Another choice is to process ad server logs and find out the users who have seen advertiser's ads in the past but did not convert. However, it suffers from the cold-start problem. If the seeds are from a new advertiser, or will be used for new ad campaigns, there are no users who have seen the ads. When the number of features is large, feature selection is conducted by imposing an L_1 regularization to the loss function to reduce the number of non-zero values in the final model. In practice, machine learning packages, such as Vowpal Wabbit, provide a flexible and scalable solution for training logistic regression on a large scale (e.g., with millions of features) on Hadoop clusters and predicting such probabilities for a massive amount of users.

The advantage of this modeling methods is that information carried by seeds are compressed into a model (e.g. a $\boldsymbol{\beta}$) and there is no need to remember the feature vectors of seeds for scoring users to be a look-alike or not. Apart from the huge benefit, there are some potential caveats for applying regression modeling methods to the look-alike problem. LR is a linear model, although a lot of conjunctions, cross or other dependencies can be embedded into the features, it may not be perfect and efficient for clustering users. We implement an LR based look-alike system for comparison with our proposed system.

2.3. Segment Approximation-based Look-alike System

Another type of look-alike system is based on user segment approximation, where user segments can be user characteristics such as user interest categories. For example, if a user likes NBA games, this user is considered to belong to *Sports* segment. In comparison to other methods, a segment approximation-based system uses pre-built user segments as user features. The general idea of segment approximates based method is to find out top segments that are "shared" by as many seed users as possible.

Here we discuss one the most recent published segment approximation based method from Turn Inc, Shen et al. (2015). In their approach, each user is represented as a bag of segments, $u_i = \{c_{i1}, \cdots, c_{iK}\}$. The optimization goal for look-alike audience extension is, given an advertiser's provided segment set C (i.e., the segments that appear in seed users), to recommend new users with a segment set C', satisfying following three properties:

$$sim(aud(C), aud(C^{'})) > \alpha \tag{5}$$

$$perf(aud(C^{'})) - perf(aud(C)) > \beta \tag{6}$$

$$|aud(C \cup C^{'})| \gg |aud(C)| \tag{7}$$

where $aud(C)$ is the set of users who have any of the segments in C, $perf(aud(C))$ is the performance of users $aud(C)$ regarding ad click-through-rate or conversion-rate. Intuitively, the above criteria mean that: (1). the extended audience set $aud(C^{'})$ is "similar" enough (in terms of user overlap) to the seed audience $aud(C)$ (Eq. 5); (2). The performance of the

extended audience set $aud(C^{'})$ should be better compared to the seed audience set (Eq. 6); (3). And the size of the extended audience set $aud(C^{'})$ should be much larger than the seed audience (Eq. 7).

There can be many segments that satisfy the above properties so a scoring function is needed to distinguish the importance of segments. Shen et al. (2015) discussed a greedy approach, which is an application of the set cover algorithm, and a weighted criteria-based algorithm. For any newly recommended category (or segment) c_{new}, they calculate a score by combining the three functions (Eq. 5, 6, 7) above:

$$score \propto sim(aud(c_{new}), aud(C)) \times perf(aud(c_{new})) \times nov(aud(c_{new})|aud(C)) \quad (8)$$

where $nov(aud(c_{new})|aud(C))$ (corresponds to Eq.7) measures the proportion of users from this new category that are not in seed users. Based on this score, the authors sort all existing user segments and recommend top-k segments to the campaign for user targeting (for more details, refer to Shen et al. (2015)).

Segment approximation works well when the pre-built segments quality is high and have good coverage. It is particularly the case for big branding advertisers, but maybe less useful for small advertisers. The pre-build process also introduces another computation pipeline which may delay the look-alike audience generation. We implemented the algorithm described above, as another method for large-scale comparisons.

3. Graph-Constraint Look-Alike System

In this section, we present the details of our developed graph-constraint look-alike system, which takes advantages of both simple similarity and regression-based methods. First, a global user-2-user similarity graph is built, which enables us to limit candidate look-alike users to the nearest neighbors of seeds. Then the candidate users are ranked by their feature importance specific to ad campaigns.

Training and scoring look-alike audience per ad campaign against all users is a low-efficiency process. A typical look-alike audience size is 1 to 10 million, which is about 0.1% to 1% in a 1 billion user pool. Scoring a lot of irrelevant users wastes a significant amount of computation resources. Also, training to get feature weights against the 99% irrelevant users faces a challenging user sampling problem. We propose to generate look-alike audience in two phases, namely global graph construction and campaign specific modeling.

3.1. Phase I: Global Graph Construction

In this phase, a user-2-user similarity graph is built for all available users in the system, where an edge between two users indicates their Jaccard similarity (Eq. 2). The goal is to find look-alike candidates for a particular set of seeds using this global graph. The core formulation of our approach is a pairwise weighted user-2-user similarity, defined as:

$$sim(\mathbf{f}_i, \mathbf{f}_j) = \frac{\mathbf{f_i}' \mathbf{A} \mathbf{f_j}}{||\mathbf{f_i}|| \times ||\mathbf{f_j}||} \quad (9)$$

where $\mathbf{f_i}$, $\mathbf{f_j}$ are feature vectors of users u_i and u_j. The weight matrix $\mathbf{A}$ can incorporate the importance of linear correlations for both individual features and pairwise feature combinations. For example, a pairwise weighted user-2-user similarity is a weighted cosine similarity if the weight matrix $\mathbf{A}$ is diagonal. A non-diagonal matrix $\mathbf{A}$ can model pairwise

feature combinations. Higher order feature combinations are rarely observed to be useful in a large and sparse feature space. Therefore, our feature modeling is only limited to individual and pairwise features to avoid over-engineering a look-alike system. The scale of the weight matrix is determined by the total number of features which is relatively smaller in comparison to the total number of users. This property allows us to build a specific weight matrix for each ad campaign to leverage more tuning power on features.

The challenges of building the global user-2-user similarity graph live in both the high complexity of user pairwise similarity computation ($O(N^2)$) and similar user query ($O(N)$). To approach these challenges, we use the well-known Locality Sensitive Hashing (LSH) to put similar users into the same clusters by reading user data in one pass and enable similar user query in sub-linear time. Here we briefly introduce the background of LSH.

3.1.1. MinHash Locality Sensitive Hashing (LSH)

Using LSH Rajaraman et al. (2012), each user is processed once in two steps. The first step is using a set of hash functions to transform a user feature vector into a similarity-preserving signature, which consists of a much smaller amount of values than original user feature vector. Then the second step is to assign the user to clusters based on the signature. Signature values are grouped to create clusters of users who are similar to each other (above a predefined similarity threshold) with a bounded probability. At query time when given an input user, we can hash this user to find out the clusters it falls into and the users in those clusters are the candidate similar users. Both cluster construction and the retrieval of similar users are per user based process without pairwise user operations. Therefore, LSH dramatically simplifies the simple similarity-based look-alike system. Below we use MinHash LSH as an example to discuss the details of those two steps:

- **MinHash.** Hashing technique is used in LSH to reduce user feature space while preserving their original similarity with high probability. Different choices of hashing schemes can be applied, such as using random projection to approximate cosine similarity and using MinHash to approximate Jaccard similarity etc. In our case users' Jaccard similarities are computed on their binary features, so MinHash is used:

$$h_{min}(\mathbf{f}_i) = \underset{x \in (1,\ldots,K),\ f_{i,x}=1}{\operatorname{argmin}} h(x)$$

  where $\mathbf{f}_i$ is a K-dimensional feature vector of user u_i, x is the index of features in $\mathbf{f}_i$. Hash function $h(x)$, $[h : (1, \ldots, K) \to R]$ maps the index of a feature into a random number. A function $h_{min}(\mathbf{f}_i)$ is then defined as the feature index that has the minimal hash value, and $h_{min}(\mathbf{f}_i)$ is called the hash signature. It can be proven that MinHash preserves Jaccard similarity (refer to Chapter 3 of Rajaraman et al. (2012)), which means that the probability of two users having the same MinHash value equals to the their Jaccard similarity:

$$sim_{Jaccard}(\mathbf{f}_i, \mathbf{f}_j) = P\left(h_{min}(\mathbf{f}_i) = h_{min}(\mathbf{f}_j)\right)$$

  In other words, if two users' Jaccard similarity is r, and a total of H MinHash functions are used to hash the two users independently. Then the two users are expected to have $H \cdot r$ signatures identical.

- **LSH.** Signatures generated by H Minhash functions form an H dimensional vector. Signatures for all N users form an N-by-H matrix, known as signature matrix Rajaraman et al. (2012). As $H \ll K$, the signature matrix can be considered as a dimension reduction result from the raw user-feature matrix, which is a N-by-K matrix.

  Using the H-dimensional signatures, LSH method provides various flexible ways to cluster similar users into buckets for retrieval purposes. The most popular approach is the "AND-OR" schema, where H signature dimensions are partitioned into b bands. Each band consists of r signature dimensions, where $b \times r = H$. Users are clustered into the same bucket only if they have the same r-dimensional signature within a band. A user can only fall into one bucket in a band. Therefore each user has b buckets in total. If a seed user is found to fall into bucket b_i, all users in b_i can be retrieved as its look-alike candidates, which is a small amount of users compared to the whole user population. The retrieved candidate users can be then scored for exact ranking if needed. The values of b and r are usually determined empirically based on application data similarity distribution, and the desired the item-2-item similarity threshold.

The LSH technique introduced above can be used to reduce the complexity of the system implementation. After conducting MinHash LSH, each user u_i has a list of bucket ids generated by the hash signatures. Candidate look-alike users for a campaign can then be generated by looking up the hash signatures of all the seeds and merging their similar users fetched from the global graph. A reasonable good size of candidates varies from $5\times$ to $10\times$ of the final look-alike audience. A too small candidate set may have risks of missing good look-alike users that may not be selected to the candidate list. A too large candidate set increase the cost of computations in Phase II. When the number of candidate users fetched from the graph is small (e.g., due to a small amount of seeds), one can use regression or segment approximation-based methods to ensure the desired size of look-alike audience. In the rest of this paper, we only consider the cases that sufficient number (e.g., more than 10 million) of candidate users can be fetched.

3.2. Phase II: Campaign Specific Modeling

In this phase, the goal is to train a simple campaign-specific model and refine the audience candidates from Phase I. The candidates are scored by the campaign-specific feature weights, and top ranked users are selected as the final look-alike audience.

Different ad campaigns may care about different user features. A retail store's ad campaign may care more about users' location rather than their education. An insurance ad campaign cares more about users' financial status than their social status. An optimized weight matrix (Eq. 9) for an ad campaign can ignore the irrelevant features and help generate look-alike audience that can lead to a better return-on-investment for ad dollars.

Intuitively a relevant feature should have more power to distinguish seeds from other users. There are many off-the-shelf methods and algorithms for feature selection, feature ranking, and modeling labeled data. The popular ones are simple feature selection and linear model, e.g. logistic regression (with regularization), or decision trees. Here, we discuss a simple feature selection method as an example because the simple method is a non-iterative feature selection method which can be run very efficiently. Also, it gives a

reasonably good performance in a sparse user feature space in comparison to linear models and other complicated methods. However, other methods are still highly recommended if computational cost and response time are not the bottlenecks.

A simple feature selection method calculates a diagonal weight matrix A, where $A_{i,j} = 0,\ for\ i \neq j, 1 \leq i, j \leq L$ and $A_{j,j}$ is the importance of the feature j calculated from a set of seeds S and a set of comparing users U. Let us denote the proportion of users in seeds that has feature j as $p_j = \frac{\sum_{f_i \in S} f_{i,j}}{|S|}$, and the proportion of users in the comparing user set U that has feature j as $q_j = \frac{\sum_{f_i \in U} f_{i,j}}{|U|}$.

The seeds set S is provided by advertisers. But there are multiple options for composing user set U, such as the users who have seen ad impressions from the advertisers but did not click or convert, or the whole pool of candidate users. The benefit of the latter method is that the computed q_j can be shared across all campaigns to improve system efficiency.

There are several options for evaluating feature importance through univariate analysis, such as mutual information Battiti (1994), information gain Lewis and Ringuette (1994) and information value (IV) Siddiqi (2012). The advantage of these methods is that they assume independence of features such that features can be evaluated independently of each other in distributed manner. Based on our experience, these different methods yield similar results. Therefore, in this paper, we take IV as the example implementation, which measures the predictive power of a feature. The importance of the binary feature[2] j is calculated as:

$$A_{j,j} = \begin{cases} (p_j - q_j) \log\left(\frac{p_j(1-q_j)}{(1-p_j)q_j}\right), & \text{if } p_j > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The directional filter $p_j > 0.5$ is optional that enforces only positive features will be selected, which means those features are more significant in seed users than in other users. Negative features can be used to discount a user's rank, and it is one of our future exploration directions. The positive features for a particular campaign can be ranked by their IV values where features with higher values have more predictive power. Furthermore, those top-ranked features can be displayed to advertisers to provide transparency of the model.

To further motivate our proposed IV based method, we can look at the predicted hyperplane for classification in logistic regression (with standardized features) Hilbe (2015):

$$\sum_{i=0}^{K} \hat{\beta}_i x_i = C$$

Where C is a constant. Further, the probability of being positive or negative-labeled can be obtained. If the features, x_i's are binary, those estimated $\hat{\beta}_i$'s are (roughly) a measure of the impact of the log-odd ratio given a presence/absence of the feature $f_{i,j}$. In such situation (i.e. features are binary), the estimated $\hat{\beta}_i$'s are of the same scale and is a measure of the importance of the corresponding feature. In other words, a ranking of the absolute value of these $\hat{\beta}_i$'s corresponds to a ranking of the feature importance.

Following almost exactly the same logic mentioned above for logistic regression, the user scoring method falls into the following framework:

2. In our system, we only consider binary features, where continuous features are bucketized to create binary features as well.

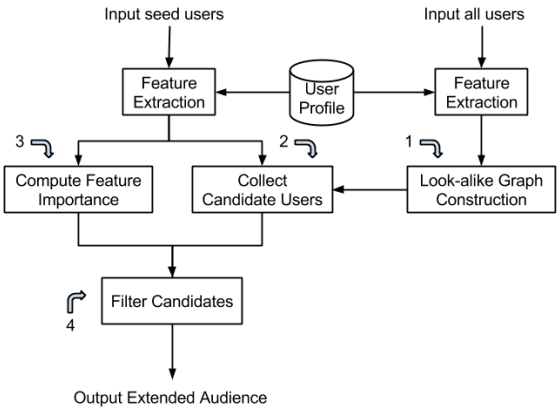

Figure 1: System Architecture and Pipeline

$$s(f_i) = \sum_{j=0}^{K} \beta_j f_{i,j} + \sum_{g,j=0}^{K} \beta_{jg} f_{i,j} f_{i,g}$$

where β_j's are a measure of the variable importance; β_{jg} are a second order measure, a measure of the interactions of the variables. One realization of the above campaign-specific scoring framework is based on IV values, the score of a user for a specific campaign c is,

$$s(f_i, c) = \sum_{1 \le j \le K} A_{j,j} f_{i,j}. \tag{10}$$

Note that the above scoring (Eq. 10) is unbounded. Although in our final audience recommendation, we only need to use this score to rank candidate audience and recommend a specific amount of audience, it will be useful to rank the audience and let advertisers control and leverage the size of the recommended audience and the "quality" of the audience. If one wants to derive a "normalized" score, i.e. to create a mapping from the raw score to a score with range in $[0, 1]$.

$$s() : s(f_i) \mapsto [0, 1]$$

A candidate for this is to use a sigmoid function $s(t) = \frac{1}{1+e^{-\beta t}}$. It is possible to use this normalized score to determine a threshold that top-N users can be selected without sorting.

3.3. System Design and Pipeline

Our graph-constraint look-alike system is running in production at Yahoo! and generating thousands of look-alike audience sets. The sketch of the system is depicted in Figure 1, which contains four major components:

Construct global user graph. The system periodically builds global user graph based on user-2-user similarity. When global graph builder (component 1 in Figure 1) is triggered to run, it loads configured user features from user profile service for each user, and calls our in-house LSH package to hash users into different buckets. If two users' similarity is above our configured threshold, they may be found hashed into the same bucket in multiple bands of the min-hash table. After processing all users, a map-reduce job is run to group users by bucket ids, such that users fall into the same bucket can be stored together to enable efficient user query.

Collect candidate users. Look-alike audience extension system takes a list of seed user ids as input. Component 2 hashes all seed users features and uses the hash signatures to find out the buckets those seed users fall into. Users in those buckets are all merged into the candidate user set. At the end of seed extension process, user profile service is queried for candidate users, and the output user id with feature data are saved on HDFS.

Compute feature importance. To compute feature importance, component 3 reads global user count and seed user count for each feature. Then it calculates feature importance and saves the results on HDFS. The global feature distribution can be shared by all campaigns to make the feature importance learning very efficient.

Score and recommend audience. As the last procedure of look-alike audience extension system, component 4 reads two data sets from HDFS: top K feature list and their importance score, and candidate users and their features. It iterates through all candidate users and scores users by the campaign specific feature importance. As a final step, all candidate users are ranked, and top N users ids are sent to user profile service to update user profiles for campaign targeting.

4. Experiment & Evaluation

In this section, we evaluate look-alike models using back-play tests on real mobile app install campaigns data, where the goal of the campaigns is to get new app installers while meeting the advertiser's target cost-per-install.

4.1. Evaluation Methodology

In app install campaigns, advertisers (usually the app developers) want to get more users to install their apps while meeting their target cost per install. In our evaluations, we focus on the user targeting side and measure how accurate the targeted audience is for specific ad apps. The problem of which app ad should be served to the targeted user at impression time is out of the scope of this paper, and it should be handled at downstream by ad selection algorithms. As running online campaigns for experiments is costly, the performance of a look-alike model can be evaluated by using off-line back-play tests.

Depending on their objectives, advertisers have many choices to put together a seed user list to use a look-alike system. For example, an advertiser may select users who made in-app purchases in the recent past as seeds and use look-alike to find similar users who may potentially also make in-app purchases. Another example is that an advertiser can put together some existing installers as seeds and aim to find similar users who may install the app. In our experiments, we try to simulate the second scenario using the following steps:

1. Fix a date t_0, collect the installers from t_0 to t_1 (e.g., $t_1 - t_0 = \ 1\ month$) as seed users for an app.
2. Use a look-alike audience extension system to extend seed users into a look-alike audience with size n.
3. Calculate the performance (app installation rate) of recommended audience in the test period $(t_1, t_2]$ (e.g., $t_2 - t_1 = \ 2\ weeks$).
4. Compare different look-alike systems' performance by varying audience size n.

4.2. Data and Evaluation Metrics

User features. If app developers use the SDK[3] provided by Yahoo!, app installation and use are tracked and recorded. We also derive user app usage features (e.g., a user is likely to be interested in which categories of apps) from this data to construct user feature vectors. The global graph is built on a user profile snapshot right before the testing period, which includes more than a billion mobile users. The user features include apps installed, app categories, app usage activities, and app meta information. These same features are also used for the simple similarity-based method and logistic regression method in comparison. When we compare with segment-approximation based method, user app interest categories (e.g., Sports, Adventure Games, etc.) are used as the segments.

Seeds. From Yahoo! advertising platform, we sampled 100 apps which have more than 10,000 converters per month. For each app, the seeds were app installers in May 2015. The testing examples are active users in the first two weeks of June, where new installers of the test apps are positive examples and non-installers as negative examples.

Performance metric. We use installation rate (IR) to measure the percent of recommended audience (A) that install the target app (P) in the testing period. It is computed as: $IR(P, A) = \frac{|I|}{|A|}$, where I is the set of new installers from A. Note that in conventional conversion rate computation the denominator is the number of impressions, whereas here the denominator is the number of unique recommended users. The motivation is that we want to capture whether the recommended audience is more likely to install the ad app, without considering ad serving time decisions.

4.3. Experiment Results

To compare the installation rate (IR) of different methods, we compute IR at various recommended audience sizes. For graph-constraint model and the logistic regression model, each candidate audience has a score from the model so that we can rank all the users and take the top-n scored users for evaluation. The candidate audience output from Phase I of our graph-constraint method already have similarity to seeds larger than the pre-configured threshold. Therefore, for the simple similarity-based method, we randomly sample n users from the candidate set to avoid pair-wise user similarity computation between seeds and candidate users. For segment-approximate based model, there is no rank among the candidate users, and we randomly sample n users from the candidate set.

Figure 2 shows the weighted average of installation rates for all four methods. The axises are normalized by constants to comply with company policy. The evaluations were conducted on audience size in the scale of millions. GCL is our proposed graph-constraint look-alike model (section 3.2). LR is logistic regression model (section 2.2), where seed users are positive examples and non-seed users are used as negative examples (the same way to compose training data as the GCL model). Since the majority of users are non-seed users, so downsampling was done on negative samples. $JaccardSim$ is the simple user similarity based method (section 2.1). $Segment$ is the segment approximation method (section 2.3). We can see from the plot that GCL model performs best among the models in comparison, as audience size increases the installation rate decreases. $JaccardSim$ model's

3. https://developer.yahoo.com/flurry/

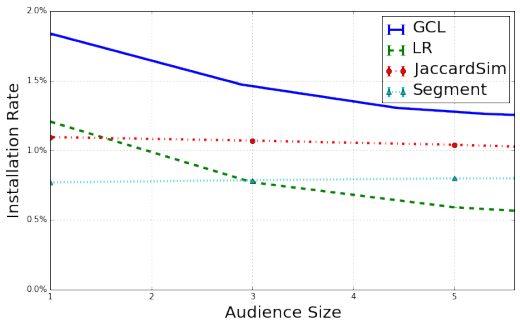

Figure 2: Weighted Average Installation Rate Comparison on 100 Apps. Axises are scaled by constants.

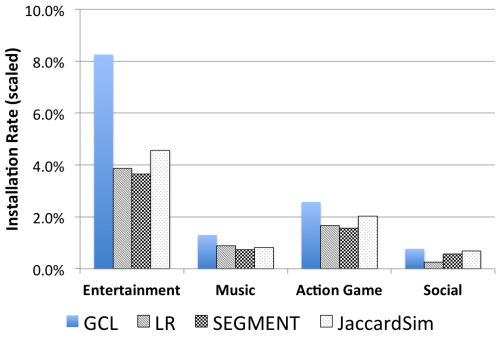

Figure 3: Installation Rate Comparison of 4 Example Apps. Axises are scaled by constants.

performance remains relatively stable across different sizes of audiences, which reflects the average installation rate of users similar to seeds. When audience size is small, top ranking users identified by *GCL* performs more than 50% better than the average similar users.

At smaller recommended audience sizes, the *LR* model performs better than *JaccardSim* model and *Segment* based model. It seems the top features play a key role. When the audience size increases the LR model performance goes down dramatically. This may be due to the sparsity of user features (e.g., apps installed), when none of the apps plays a significant role in the LR model, its effect is down to random user picking. As the audience size gets larger and there is more noise in the larger candidate set.

Figure 3 shows 4 example apps installation rate comparisons from different categories. Since segment based method does not rank users, so we take all the recommended audience by segment based method and select the same amount of top audiences from the logistic regression model and our graph-constraint look-alike model. For all methods in comparison, they work better for entertainment and action game apps than social apps. Our hypothesis is that users usually do not frequently install new social apps, and they stably use the apps where their friends are, so it is hard to get relevant features to predict what additional social apps they will install. On the opposite, users are more likely to explore new entrainment apps (new sources of entertainment), and play new action games when the already installed games are done or got stuck. Therefore, there are more user signals related to those types of apps, and look-alike models can pick up those signals to make better predictions.

5. Discussions

Our look-alike system is a running product at Yahoo!, where the production pipeline can score 3000+ campaigns (using about 3000 mappers on 100 nodes), on more than 3 billion users (with millions of features in total) within about 4 hours. In empirical on-line campaigns, these generated look-alike audiences are driving up to 40% improvements in conversion rates and up to 50% reduction in cost-per-conversion compared to other targeting options like demographic or interests[4]. It is challenging to develop an efficient and effective large-scale look-alike system, in this section we share some experiences in tuning a look-alike system and running on-line look-alike audience targeting campaigns.

4. https://advertising.yahoo.com/Blog/GEMINI-CUSTOM-AUDIENCE-2.html

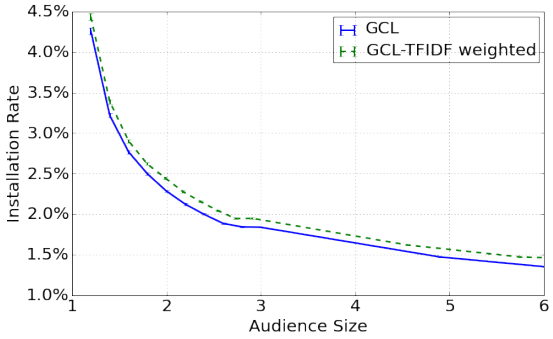

Figure 4: Installation rate improvement achieved by weighted global-graph. Audience sizes are in millions. Axises are scaled by constants.

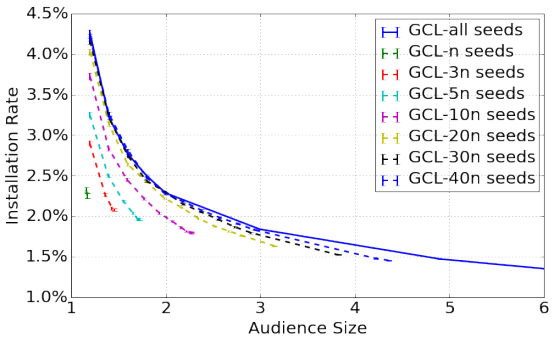

Figure 5: Different seeds set size for 100 apps. $n = 1000$, and audience sizes are in millions. Axises are scaled by constants.

Weighted LSH. Intuitively, users with similar app usage behaviors should be more similar than users who use apps quite differently even if they have the same apps installed. For example, given three users u_1, u_2 and u_3, and they all have a *sports* app installed. u_1 and u_2 use this app more than ten times a day, while user u_3 uses this app only once a week. For these three users, intuitively u_2 should be more similar to u_1 than u_3. To encode this key observation, we compute *tf-idf* weight for each feature of a user, where each user is treated as a document, and the intensity of a feature is treated as the frequency of a word.

MinHash LSH discussed in Section 3.1.1 approximates Jaccard similarity between two sets of items. To extend LSH to encode weighted Jaccard similarity, several prior works proposed different methods Chum et al. (2008); Ioffe (2010); Manasse et al. (2010). One straight-forward method is: suppose w_i is weight of feature i, and w_i is integer. One can generate w_i replicas of feature i to replace the original feature, then proceed with LSH on the transformed feature sets. It can be shown the MinHash LSH on the transformed item sets approximates weighted Jaccard similarity.

In our scenario, the feature weights are real values. So the approaches discussed above is not directly applicable. We adopted the approach proposed in Chum et al. (2008) to do a transformation on the original MinHash value to encode feature weight: $f(X_i) = \frac{-\log X_i}{w_i}$, where w_i is the weight of feature f_i, X_i is the original MinHash value normalized to $(0, 1)$. Intuitively, when feature weight w_i is larger, the hashed value $f(X_i)$ is smaller, so feature f_i is more likely to be selected as the signature, and two users are more likely to have the same hash function signature if both of them have this feature with higher weights.

We use this weighted LSH scheme to build a weighted global graph, and the campaign specific modeling is the same as Section 3.2. Figure 4 compares the performance of weighted graph-constraint look-alike model with the non-weighted model on the same 100 ad apps (see Section 4.2). We can see that when considering user feature weights, audience generated from weighted graph consistently has about 11% lift in installation rate.

Seed size effect. The number of seed users input to look-alike system will affect the number of direct neighbors (or similar direct users) that can be found. By varying the size of the input seed set, we can empirically observe the trade-offs between recommended

audience size and expected installation rate performance. Figure 5 show the 100 campaigns weighted average performance at different sizes of seed sets. A small number $n = 1000$ is used as the base value, and we sample $k \times n$ seeds for each of the 100 ad apps. We can see that as the number of seeds increases: (1). For a fixed amount of recommended audience, larger seed size generates better performance. This is because when seed user size is small, noise is stronger than signal. (2). The maximum size of recommended audience increases, but performance decreases. This is expected in that the more seed users, the more neighbor users can be found in the graph. But meantime more noise are introduced into the model, hence the performance at the maximum number of users decreases. (3). The improved performance and the maximum number of recommended audience do not change much when seed user size reaches $40n$. This value could be used to make suggestions to advertisers as how large the seed size should be to have more freedom in performance and audience reach trade-off. But a rule of thumb is that more seeds are better regarding both installation rate and audience size. However, an optimal choice of the seed size depends on many factors, such as campaign, users features, etc.

6. Related Work

The most related prior work is Turn's audience extension system Shen et al. (2015), the authors proposed a systematic method to generate extended audience using weighted criteria algorithm, which is briefly described in Section 2.3 and employed in experiments for comparison. In that work, users are extended on predefined segments, so the quality of existing segments in the targeting system is critical. Aly et al. (2012) describes a large-scale system to classify users for campaigns, where the goal is to predict the likelihood of a user to convert to an existing campaign. As pointed out by Shen et al. (2015), there is a lack of prior work of audience extension system in the literature. Our proposed method approaches audience extension problem at the individual user level.

Behavioral targeting is an area that focuses on inferring users interests from past behaviors, with the assumption that users' past interests can help improve prediction and recommendation tasks. Barford et al. (2014) conducted a large-scale study to analyze the relationship between users' online profiles and the ads shown to them. They showed that user profile targeting is widely used. Ahmed et al. (2011) introduced a large-scale distributed system to infer users' time varying interests. In recommender systems, Kanagal et al. (2012) combined taxonomies and latent factors to predict users purchase behavior; Ahmed et al. (2013) proposed a content based latent factor model to infer user preferences by combining global and individual users preferences. Using the trending deep learning technique, Djuric et al. (2014) proposed to use hidden conditional random field model to model users online actions, such as click and purchase. These prior work are orthogonal to our proposed system, where the learned user interests and preferences can be converted to user feature vector as input the look-alike system for further improvements.

To deal with a large number of objects, especially when objects pairwise distances is important, hashing techniques are heavily used to improve efficiency. The locality sensitive hashing Slaney and Casey (2008) used in our proposed look-alike system was also applied to many areas, such as detecting duplicate images Chum et al. (2008); Kulis and Grauman (2009); extracting topics from a collection of documents Gollapudi and Panigrahy (2006);

web page clustering Haveliwala et al. (2000), genomic sequence comparison Buhler (2001); 3D object indexing Matei et al. (2006). In our look-alike system, LSH is used as the initial step to cluster users at a coarse level. After user filtering at this step, campaign specific model ranks the candidate audience which has a larger impact on the final performance.

7. Conclusion

In this paper, we present a large-scale look-alike audience extension system from Yahoo!, where the similar user query time is sub-linear. The core model of this look-alike system is based on graph mining and machine learning technique on pairwise user-2-user similarities. Through extensive evaluations on app installation campaigns, we show that the recommended look-alike audience by our method can achieve more than 50% lift in app installation rate over other audience extension models. Furthermore, we also discuss the challenges and share our experience in developing look-alike audience extension system. One of our future work is to leverage the real-time campaign feedback signals into the continuous optimization process to improve look-alike campaigns.

References

Amr Ahmed, Yucheng Low, Mohamed Aly, Vanja Josifovski, and Alexander J Smola. Scalable distributed inference of dynamic user interests for behavioral targeting. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 114–122. ACM, 2011.

Amr Ahmed, Bhargav Kanagal, Sandeep Pandey, Vanja Josifovski, Lluis Garcia Pueyo, and Jeff Yuan. Latent factor models with additive and hierarchically-smoothed user preferences. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 385–394. ACM, 2013.

Mohamed Aly, Andrew Hatch, Vanja Josifovski, and Vijay K Narayanan. Web-scale user modeling for targeting. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 3–12. ACM, 2012.

Paul Barford, Igor Canadi, Darja Krushevskaja, Qiang Ma, and S Muthukrishnan. Adscape: Harvesting and analyzing online display ads. In *Proceedings of the 23rd international conference on World wide web*, pages 597–608. ACM, 2014.

Roberto Battiti. Using mutual information for selecting features in supervised neural net learning. *Neural Networks, IEEE Transactions on*, 5(4):537–550, 1994.

Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.

Ondrej Chum, James Philbin, Andrew Zisserman, et al. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC*, volume 810, pages 812–815, 2008.

Nemanja Djuric, Vladan Radosavljevic, Mihajlo Grbovic, and Narayan Bhamidipati. Hidden conditional random fields with distributed user embeddings for ad targeting. In *IEEE International Conference on Data Mining*, 2014.

Sreenivas Gollapudi and Rina Panigrahy. Exploiting asymmetry in hierarchical topic extraction. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 475–482. ACM, 2006.

Taher Haveliwala, Aristides Gionis, and Piotr Indyk. Scalable techniques for clustering the web. 2000.

Joseph M Hilbe. *Practical Guide to Logistic Regression*. CRC Press, 2015.

Sergey Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010.

Bhargav Kanagal, Amr Ahmed, Sandeep Pandey, Vanja Josifovski, Jeff Yuan, and Lluis Garcia-Pueyo. Supercharging recommender systems using taxonomies for learning user purchase behavior. *Proceedings of the VLDB Endowment*, 5(10):956–967, 2012.

Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009.

David D Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *Third annual symposium on document analysis and information retrieval*, volume 33, pages 81–93, 1994.

Mark Manasse, Frank McSherry, and Kunal Talwar. Consistent weighted sampling. *Unpublished technical report) http://research. microsoft. com/en-us/people/manasse*, 2010.

Bogdan Matei, Ying Shan, Harpreet S Sawhney, Yi Tan, Rakesh Kumar, Daniel Huber, and Martial Hebert. Rapid object indexing using locality sensitive hashing and joint 3d-signature space estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1111–1126, 2006.

Y. Qu, J. Wang, Y. Sun, and H.M. Holtan. Systems and methods for generating expanded user segments, February 18 2014. URL http://www.google.com/patents/US8655695. US Patent 8,655,695.

Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 1. Cambridge University Press Cambridge, 2012. Chapter 3.

Jianqiang Shen, Sahin Cem Geyik, and Ali Dasdan. Effective audience extension in online advertising. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2099–2108. ACM, 2015.

Naeem Siddiqi. *Credit risk scorecards: developing and implementing intelligent credit scoring*, volume 3. John Wiley & Sons, 2012.

Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors (lecture notes). *Signal Processing Magazine, IEEE*, 25(2):128–131, 2008.