App2Vec: Vector Modeling of Mobile Apps and Applications

Qiang Ma*
Yahoo! Inc.
qiang@yahoo-inc.com

S. Muthukrishnan
Rutgers University
muthu@cs.rutgers.edu

Wil Simpson§
Yahoo! Inc.
wil@yahoo-inc.com

ABSTRACT

We design a way to model apps as vectors, inspired by the recent deep learning approach to vectorization of words called `word2vec`. Our method relies on how users use apps. In particular, we visualize the time series of how each user uses mobile apps as a "document", and apply the recent `word2vec` modeling on these documents, but the novelty is that the training context is carefully weighted by the time interval between the usage of successive apps. This gives us the `app2vec` vectorization of apps. We apply this to industrial scale data from Yahoo! and (a) show examples that `app2vec` captures semantic relationships between apps, much as `word2vec` does with words, (b) show using Yahoo!'s extensive human evaluation system that 82% of the retrieved top similar apps are semantically relevant, achieving 37% lift over bag-of-word approach and 140% lift over matrix factorization approach to vectorizing apps, and (c) finally, we use `app2vec` to predict app-install conversion and improve ad conversion prediction accuracy by almost 5%. This is the first industry scale design, training and use of app vectorization.

1. INTRODUCTION

Mobile device usage is a significant part of people's daily life. According to Flurry Analytics, in 2014, overall mobile application (app) usage grew by 76% [2]; consistently, Americans average about nine app downloads per month. This drives the mobile app-install ad business, where app developers run ad campaigns to get more people to download and install their apps. According to BI Intelligence [1] the US mobile app-install ad revenue will reach 6.8 billion by the end of 2019. This in turn motivates efforts to understand users' interests and their preference for one app over the other.

Some time ago, website browsing was the primary mode of accessing online content. There was progress in understanding how users consume online content and how to use their historic patterns to predict future behavior through machine learning techniques [18, 7, 5, 13]. Just as website visits reveal a user's online content consumption interests, user activities on mobile devices are represented by the mobile apps they install and use and the app sessions generated over time. According to a Nielsen report [3], in 2013, American users used on average about 27 apps per month. Therefore, it is critical to find out what (new) mobile applications are useful to a user.

*§Work done at Flurry.

In this paper, we present a first-of-its-kind industrial scale modeling of mobile apps based on how users use them, and use that to improve prediction of users' preference for different apps. Our contributions are as follows:

- *(Vectorization of Apps.)* Based on the usage history of apps across Yahoo!'s users, we propose a model `app2vec` to represent apps in a vector space without *a priori* knowledge of their semantics. Our approach is inspired by the recent deep learning based embedding of words into vectors called `word2vec` [4], but our novelty is that, in our training context, the time between the usage of apps is crucially used (without that, we show the modeling is not as effective). We show by examples that vector operations on the `app2vec` representations capture semantic relationships between apps, similar to how `word2vec` vectors capture linguistic relationships among words and phrases [15].
- *(Quantitative Evaluation.)* A common challenge for the `word2vec` model and newly developed models on top of `word2vec` is evaluating the quality in some quantitative manner beyond empirical examples. Because of the industrial context of our work, we are able to use Yahoo!'s human editorial infrastructure. We conduct large-scale editorial evaluations on the quality of learned app vectors and compare several models including bag-of-words, matrix factorization, and `app2vec`. The evaluation results show that 82% of the retrieved top similar apps based on `app2vec` are semantically relevant, achieving 37% lift over bag-of-word approach and 140% lift over matrix factorization approach. We share our insights from this exercise which we believe can help the research community to understand the results from even `word2vec` framework and its many variants better.
- *(`app2vec` Application.)* In mobile advertising, we need to predict the probability of a user converting on an ad for an app. We design a method of extracting features for users based on clustering using `app2vec`. Through extensive experiments using ad impressions from ad server logs, we demonstrate that these features improve ad conversion prediction accuracy (AUC) by 5% which is significant in practice.

The rest of the paper is organized as: Section 2 motivates and defines the problem; Section 3 introduces the background of the `word2vec` model, and presents our proposed `app2vec` model with qualitative and quantitative evaluations. Section 4 discusses the application of `app2vec` in ad

conversion prediction in mobile app-install advertising. Section 5 discusses related work and Section 6 has concluding remarks.

2. THE APP SIMILARITY PROBLEM

As discussed in the previous section, in the mobile world, on one hand users are installing and trying new apps over time, on the other hand, there are thousands of mobile app developers who want to get more users to install their apps and grow their user bases. Therefore, how should app developers target mobile users to install new apps? More specifically, for app developers, the question is how to find users interested in their apps? First, let us look at the example below:

EXAMPLE 2.1. *Assume there are three users X, Y and Z, and each of them uses a set of apps. User X uses {"Angry Bird", "NY Times", "House Finder"}; user Y uses {"Six Flags", "Boston Post", "Zillow"}; user Z uses {"Tumblr", "Yahoo Mail", "Flickr"}. Suppose the developers of "Redfin" app want to promote it and ask the question "who are the users interested in real estate?" It is natural to think that users X and Y are probably better candidates than Z to target for advertising, because they use apps "House Finder" and "Zillow" that are in the real estate category, similar to "Redfin".*

From this example above, the reason that users X and Y seem to be better candidates to target for advertising is that they use apps similar (e.g., of the same app store category) to the advertiser's app. However, how to determine which apps are similar to advertiser's apps? Formally, the fundamental problem we want to solve is: given a set of users U, and the historic app usage sessions S_u of a user $u \in U$, where each app session $s_i \in S_u$ is represented by $\langle u_i, a_j, t_s, t_e \rangle$, meaning user u_i used app a_j starting at time t_s and ending at time t_e. We would like to learn a similarity function $sim(a_i, a_j)$ for two apps a_i and a_j.

Two potential ways of solving this problem are:

- **Bag-of-words.** One approach to computing app similarity is through the bag-of-words method using app meta information. We can analyze the textual information in an app's title, description, market categories, user reviews, etc., or even enriched information from search engine results [19, 20]. There are several drawbacks to this approach: the app textual information are mostly provided by app developers, so the quality of the texts differ, and the data is noisy. For example, a game app description says how one can run a cafe, but based on the texts it is hard to distinguish it from a real cafe shop app. Another drawback of bag-of-words approach is that it is language specific, although people speak different languages and use apps developed in different languages, their app usage behavior may be similar. For example, people like to take photos of the highlights of a day, then use photo editing apps to beautify the photo and share it with friends in social apps. To identify such common user behavior patterns across languages, one has to build a different model for each language.

- **Vectorization.** Another approach is to learn latent vectors of apps [12], and then the distance between any pair of apps can be computed easily based on their latent vectors. In this approach, one can compose a user-app matrix based on user app usage history. In the matrix, a cell value is 1 if the corresponding user uses the specified app, and 0 if the user does not use the app. Or one can use normalized app usage (e.g., based on the number of app sessions, the amount of time spent in apps, etc) as the cell values to represent the app usage intensity. Then low-rank approximation algorithms can be used to get the latent vectors of users and apps. This approach does not suffer from the high dimension problem as the bag-of-words approach. In later sections, we compare the learned app distances from this method to our proposed app vectorization method.

A common drawback of the approaches discussed above is they do not take app usage context into account. Similar apps that have different descriptions, or complementary apps which are used in similar contexts but have different functionalities, are not learned. In our study, we model apps based on their empirical app usage contexts (i.e., the sequence of apps used in a relatively short period) generated by millions of users. Based on usage context, we learn a latent vector representation for each mobile app. Distances between the vectors of apps can then be used to measure similarity.

3. MOBILE APP MODELING – APP2VEC

In this section, we present our app vectorization method and the results from using a large scale real data.

3.1 Word2vec Background

Our work is inspired by the recent work on neural network models [15, 14], where the continuous bag-of-words (CBOW) and skip-gram (SG) models were proposed. In these works, the goal is to learn a function to map words to vectors using a large corpus of documents, where the semantic distance between two words can be measured using their latent vectors.

Continuous bag-of-words. In this model, a window of context is moving through the sentences. A log-linear classifier is used to predict the target word of a context window based on its preceding and succeeding words, whose vector representations are averaged as the context vector. With different applications in real world problems, how to form the input from context words should be adjusted according to domain knowledge. The objective of CBOW is to maximize the log-likelihood:

$$\mathcal{L} = \sum_{t=1}^{T} \log \mathcal{P}(w_t | w_{t-c} : w_{t+c})$$

where T is the size of word sequence, w_t is the target word in the context, c is the context length (or half of the sliding window size), and $w_{t-c} : w_{t+c}$ is the sub-sequence $(w_{t-c}, \ldots, w_{t+c})$ without w_t. $\mathcal{P}(w_t | w_{t-c} : w_{t+c})$ is defined using softmax,

$$\mathcal{P}(w_t | w_{t-c} : w_{t+c}) = \frac{\exp(\bar{v}^T v'_{w_t})}{\sum_{w=1}^{W} \exp \bar{v}^T v'_w}$$

where v'_{w_t} is the output vector representation of word w_t,

Querying App	Based on user-app (binary) matrix factorization	Direct word2vec without preprocessing and weighting	app2vec
A social chatting app	A photo editing app	A screen theme app	A text and video chat app
	A music discovery and sharing app	A toolbox app	A music player app
	An emergency call app	A solitaire game app	A schoolmate chatting app (non-English app)

Table 1: Examples of top-3 similar apps generated by different methods

and $\bar{v}$ is the average of word vectors in the context,

$$\bar{v} = \frac{1}{2c} \sum_{-c \leq j \leq c, j \neq 0} v_j$$

where v_w is the input vector representation of word w.

The other model proposed in [15, 14] is skip-gram, which uses the target word to predict the surrounding context words. Since our work is extending CBOW, here we skip the details of skip-gram.

3.2 App2vec

Taking the analogy to word and document modeling, we can treat each app as a word and each user as a document consisting of app sessions that take place in the order of time. `word2vec` can be directly applied in this formulation, however, the results (discussed in Section 3.3) are not much better than the simple bag-of-words approach. There are special properties to app session sequences that do not exist in text documents. Consider this example:

EXAMPLE 3.1. *app session sequence of a user is – (a, g_1, a, g_2, a, g_3, b, g_4, c, g_5, a). a, b, c are three different apps, g_i is the elapsed time gap between two app sessions. a could be a social app that this user uses quite frequently.*

The example above illustrates two challenges in processing app session "documents": (1) there are cases that an app is used multiple times in a row. In this case, the repeated app sessions of the same app are likely to be usage independent of other apps. Intuitively, we want to capture the similarity between apps within close usage context. Therefore, we pre-process the app session sequences to de-duplicate the repeated app sessions from the same app by merging its consecutive sessions. So the app session sequence in above example becomes $(a, g_3, b, g_4, c, g_5, a)$. (2) In the app usage context, if two app sessions are far apart from each other, they are likely to be less relevant to each other. Therefore, we need a modified `word2vec` model to consider the weight between apps, where weight is measured by the time elapsed between two app sessions.

In the original CBOW model proposed in `word2vec`, each word in the context is treated equally ([11] considered weighted context words). It is a natural and simple assumption that words in the context are equally relevant to the target word. However in the scenario of modeling mobile apps, app sessions have varying intervals in-between, and intuitively the app sessions within short time gaps to the target app should contribute more in predicting the target app. Therefore, we modify CBOW algorithm to include the weight of context words. Let weight of word w_i to the target word w_t be:

$$r(w_i, w_t) = \alpha^l$$

where α is empirically chosen as 0.8, and l is the amount of the gap (e.g., number of minutes) between app w_i within the current context and target app w_t.

Plug in the recency weighting to CBOW model, $\bar{v}$ becomes,

$$\bar{v} = \frac{\sum_{-c \leq j \leq c, j \neq 0} r(w_j, w_t) v_j}{\sum_{-c \leq j \leq c, j \neq 0} r(w_j, w_t)}$$

In the rest of the paper, the app vectors are trained by using this modified CBOW algorithm.

3.3 App2vec Training Results

Dataset: Our data is obtained from app usage information available at Yahoo! (*https://developer.yahoo.com/*). If app developers use the SDK provided by Yahoo!, app installation and use are tracked and recorded. From this data set, we can find out how many times an app was used, used with what other apps in a short period, and how users switched from one app to another.

We sampled 300 million iOS users[1] from one day in February 2015 and used all app sessions generated by those users to build the app vectors, which includes several hundreds of thousands of apps. A user's app sessions are concatenated to form a user document. We pre-process the user documents to de-duplicate the repeated app sessions and compute the elapsed time between app sessions. Then the whole data was used to train our `app2vec` model.

The `app2vec` model generates a real vector for each app, such that the similarity between two apps can be computed by taking their cosine similarity.

Table 1 shows some examples comparing the quality of learned app similarity using the matrix factorization method, direct application of `word2vec`, and our proposed `app2vec` model[2]. We can see that the top similar apps make better sense for the `app2vec` model than the other methods. In this comparison, the dataset used in training `app2vec` model is also used to form a user-app matrix with binary cell value to indicate whether a user uses an app. Then we take low-rank approximation to get latent vectors of apps (see details in quantitative evaluation part), where the vector size is the same as `app2vec` vectors. We can see from the table that the querying app is a social chatting app, by matrix factorization method the top-3 similar apps fetched are not other social apps, but the apps may be used in complementary. By direct application of `word2vec`, the top-3 similar apps are not relevant. However, `app2vec` can fetch two other social apps and one complementary app.

One advantage of the `app2vec` model is that it is language

[1]For anonymity, all user IDs were removed for our data analysis and modeling training.

[2]Due to the privacy policy, app names are not displayed here.

Queried App	Based on user-app matrix factorization	app2vec
A dictionary app	A fitness and exercise app	An app to watch, create and share short videos
	A food take-away order app	A horoscope app
	A trivia app	An app to share fun conversations, photos, videos with friends
	A casino game app	An app to share secrets and send messages anonymously
An app to search for house or apartments for sale or rent	A video streaming app	An app to search local real estate or find homes for sale
	An app to get free credit reports, credit scores, and daily credit monitoring	An app to get free credit reports, credit scores, and daily credit monitoring
	A coffee shop app	A finance app to manage bank or financial accounts
	An app to publish items for sale or to buy	An app for home improvement ideas and designs
An app to make online payment	A wallpaper app	An app to search for deals, coupons
	A navigation app	An app to search for cheap flights and compare prices
	A news app	An app that help mobile device to accept credit and debit cards payment
	A car racing app	An email client app
A role-playing game	All 4 apps are role-playing and RPG games	All 4 apps are role-play and RPG games

Table 2: Examples of top-4 similar apps from matrix factorization (binary cell values) and app2vec.

agnostic. For example, one of the two social apps fetched by the app2vec model is in non-English, where most of the apps in training data are in English.

To get a further understanding of the learned app similarity, Table 2 displays several more examples of relevant apps from the built app2vec model compared to the matrix factorization (MF) method. In the first example, from the app2vec model, the dictionary app's most similar apps are not only limited to other dictionary apps, but also apps that are very likely needed to lookup a (novel) word or phrase, such as in social communication and messaging apps. Whereas the similar apps obtained through the MF method are less meaningful, except the trivia app where one can imagine a dictionary is handy. In the second example, the querying app is used to search for a house for sale or rent. For the app2vec model, the most similar apps include other house hunting app, credit checking, and home interior improvement design apps, which make sense in real life scenarios. But for the MF method, only one of the top four similar apps are related to house/apartment hunting. In the third example, the querying app is used to make online payments. The relevant apps from app2vec include shopping apps and mail client app (the latter one makes sense intuitively because it can be used to check email receipts after making a payment). But the top 4 similar apps from the MF method do not make much sense. In the last example, all queried relevant apps for a role-playing game app are the same type of RPG game apps. For this query, both app2vec and the MF method work well.

From these examples presented above, we can see app2vec method can be used to generate relevant results for querying apps. Next we conduct evaluation on a larger scale.

Quantitative evaluation. In addition to the specific examples discussed above, we also measure the quality of app similarity in large scale through manual review. We sample 1000 apps from the data set used to train app2vec model and use different models to retrieve the top-1 similar app to those 1000 apps. The retrieved app pairs are categorized into three types through editorial efforts, (1). Strongly relevant: the two apps have (almost) the same functionality, e.g. two apps are the same type of game. (2). Relevant: the two apps have complementary functions and can be used together to achieve one task, or the use of one app needs actions from another app. (3). Not relevant: there is no obvious connection between two apps. The manual review process involves reading the apps' app store page and make an assessment based on the apps' description and screenshot images. Table 3 shows examples for those three categories.

Table 4 summarizes the evaluation results for the six different models below:

- BoW: bag-of-words approach, computes two apps' cosine similarity based on the keywords extracted from app's meta information, which includes the app name, category, and description. Standard text pre-processing steps, such as stop word removal and stemming, were taken before extracting keywords. Each keyword is then tf.idf weighted.

- BoWCategory: intuitively, two apps from the same app store category are more similar than apps from different categories. This method is also based on bag-of-words approach to computing two apps similarity, but we retrieve the top similar app from the querying app's category.

- MFBinary[3]: matrix factorization approach, the matrix used to do low rank approximation is user-to-app matrix M, where $m_{i,j} = \{0, 1\}$ to indicate whether user

[3]For matrix factorization approach, one month app usage of 300 million users are used, since a user may not use all installed apps in one day.

App A	App B	Evaluation Category
Car racing game app developed by company X	Car racing game app developed by company Y	Strongly relevant
A restaurants review app	Food ordering and delivery service app	Relevant (complementary functions)
A news app	A flash light app	Not relevant

Table 3: App similarity quantitative evaluation criteria.

u_i uses app a_j. The rank is set to be the same as `app2vec` latent vector size.

- `MFIntensity`: the matrix used to do low-rank approximation is user-to-app matrix M, where $m_{i,j}$ is the percent of app's (a_j) sessions used by user (u_i) among all apps used. The rank is set to be the same as `app2vec` latent vector size.
- `word2vecOnApp`: direct application of `word2vec`: treat each user as a *document*, apps used as *words* and the sequence of app sessions as *sentence*. Directly apply `word2vec` on this set of user documents.
- `app2vec`: proposed in section 3.2, where a user's app sessions are pre-processed to combine adjacent sessions from the same app, to apply a larger weight to apps used within a short time frame.

We can see from Table 4 that across all models, numbers of app pairs belong to the strongly relevant category are much larger than those have complementary functionalities (in the category – Relevant). Especially for `BoW` and `BoWCategory`, almost all the semantically relevant pairs (about 58%) are apps of the same or similar functions (in the category – Strongly relevant). It is expected in that if two apps' keyword sets overlap, they are likely used to describe the same functions of the apps. However, due to the ambiguity of keywords, sometimes, the same set of keywords are used in different types of apps. For example, {"food", "drink", "order", "check-out"} can be used in an app to make food delivery orders and in a restaurant simulation game app. A simple way to improve on this aspect, one can limit to retrieve top similar apps from the same app store category. By applying this method, `BoWCategory` retrieves 7% more strongly relevant than `BoW`. `BoW` method relies on the quality of the app meta information (e.g., app description provided by developers), and it does not take into account user usage information. As to the few cases of relevant functionality app pairs, it is largely by chance. For example, in one such pair, one app is an RPG game, and the other app is a user guide app for this game.

`MFBinary` and `MFIntensity` generate similar results and complementary app pairs (in Relevant category) are about 17%. The amount of strongly relevant app pairs is much less than `BoW` approach, but app pairs that have relevant functionalities get improved greatly. And by using user-app usage intensity information, `MFIntensity` captures 4% more relevant function app pairs than `MFBinary`. It is expected to have more app pairs that have relevant functionalities using MF model since by formulation it captures the app co-use patterns across the users. For example, one such app pair includes an app to help users look for nearby outdoor fitness parks which have exercise equipment and a fitness exercise tracking app to record amount of exercises done. However, overall MF model does not generate better interpretable app pair relationships than simple `BoW` approaches. Our hypothesis is that an individual user installs a diverse set of apps on his/her device, spanning from utility app, wallpaper app, battery life monitoring app, to stock trading app. A well trained MF model probably retrieves pairs of apps that are co-used by a lot of similar users. However, those pairs of apps are not necessarily semantically related.

Comparing direct application of `word2vec` on users app usage data (`word2vecOnApp`) with the simple `BoWCategory` method, the total numbers of semantically relevant app pairs (Strongly relevant and Relevant categories) are almost the same. But we can see 14% of the app pairs are identified as having complementary functions, comparing to 2% by `BoWCategory` model. It is because `word2vecOnApp` utilizes the app usage contexts from users, where the function relevant apps are likely to appear in the same contexts, and the signals can be picked up by `word2vec` model. However, different from word documents, (1). in users' app sessions the same app can have many consecutive sessions (e.g., usage sessions from messaging apps, social apps), these are noise to `word2vec` model which does training on moving window concept and the context windows are not well formed; (2) the semantic contexts can be formed better by utilizing the time intervals between app sessions. Therefore, `word2vecOnApp` can be improved further by approaching these two issues.

`app2vec` has 82% of the retrieved app pairs evaluated as semantically relevant. Comparing to `word2vecOnApp`, `app2vec` identifies 18% more pairs that are strongly relevant (same or similar functions), and 5% more pairs that have complementary functionalities.

App analogy examples. In the original `word2vec` paper [15], the authors illustrated interesting examples of word analogies can be found through word vector operations, such as ("man + king - woman = queen"). When the same operations are conducted on the learned app vectors from `app2vec`, interesting app analogies also can be found. Table 5 shows some examples of app pairs with nice semantic implications. For example (as shown in the first row), an email client app is related to notes taking and reminder app. When we apply the vector operation with a public transportation schedule app, we discovered that a social events app to complete the analogous pair. It makes sense that these two are used in complementary for going to and coming back from social events. Another example in the table (second row), for a pair of apps including a travel guide and trail maps app. The analogous pairs found include a daily nightlife events discovery app and an app for fitness exercise and training tracking. This example makes sense in that people who are social butterflies probably care more about being in shape.

4. APP2VEC IN APP-INSTALL ADVERTISING CONVERSION PREDICTION

We use `app2vec` for a key application at Yahoo! When an ad impression is available, ad selection algorithms need to estimate the probabilities of click and post-click conversion (app installation) for each eligible ad. A typical ad selection method is to rank ads by their expected revenue. For ad a_i, it is calculated as:

$$E(a_i) = b_i \times p(click) \times p(conversion|click)$$

Where b_i is the bid price for a click from the advertiser.

	BoW (Bag-of -words)	BoWCategory (Bag-of-words within same app category)	MFBinary (Binary matrix factorization)	MFIntensity (Usage intensity matrix factorization)	word2vecOnApp (Direct application of word2vec)	app2vec
Strongly relevant	51%	58%	21%	18%	45%	63%
Relevant	1%	2%	13%	17%	14%	19%
Not relevant	48%	40%	66%	65%	41%	18%

Table 4: App similarity evaluations using different models. Each model is used to query top similar app to 1000 sampled apps, and editorial evaluation accesses the quality of similar apps.

v_A - v_B	+ v_C	= v_D
Mail client app for hotmail & outlook - Sticky notes & reminders app	An app of real time bus, subway & metrotracker with off-line schedules.	An app for finding cool clubs, fun bars, and other events.
	Advanced photography app.	App that encrypts and protects documents, e.g., photos, videos etc.
An app for personal trip advisor, local travel guide & on road trip planner - An app of US trail maps	Social network app for television fans and movie buffs.	Community app for writers, authors and readers.
	An app that shows which bars, night clubs and pop-up parties are most popular every night of the week.	An app that helps people do exercises, track training progress.
An app that provides healthy restaurant nutrition guide - An fitness app that focus on walking training plan, and gives how-to-lose-weight tips	An app that lists grocery cash back, coupons, freebies, and rewards on healthy & popular brands.	An app that lets users earn rewards just for taking photos of their shopping receipts.
	An app for small business that enable phone to accept credit card payments.	An app that lets users send money for free. Transfer to any bank overnight.

Table 5: Examples of app analogies.

The estimations of click and post-click conversion probabilities usually use information from impression context (e.g., publisher meta-information, past performance), current user's profile (e.g., demographics, geo-location, interests) and current ad (e.g., type of ad, ad format, past performance, etc.). In practice, since ad conversions are very rare events, click prediction and post-click conversion prediction have separate models.

In this section, we focus on how to build user's app usage features to improve the post-click conversion prediction. One can use all apps as binary features in the model, but there are hundreds of thousands of apps in the markets and used by all users, these features will be very sparse, and rigorous regularization is probably needed to do feature reduction. Suppose a set of apps A are used by user u_i, if apps can be categorized into different types, one can compose user's app usage profile by the types of used apps, $C_{u_i} = \{c | c \; is \; type \; of \; a_j, a_j \in A\}$.

Then the question is how to categorize apps into different types. From our app2vec model, a vector representation of apps is learned from a large set of users app usage. Then apps can be clustered based on the distances computed based on their latent vectors. One can apply k-means clustering algorithm on the app vectors to cluster apps, and use the cluster membership as app type.

4.1 Experiment and Results

In our experiment, we use a logistic regression model to test the impact of models trained with user app usage profile created by different methods.

A logistic regression model is trained for each of the sampled 17 advertisers, where positive examples are users[4] who installed the advertised app after clicking on the ad, and negative examples are users who clicked on the ad but did not install. In our experiments, we sampled ten of thousands of positive examples for each advertiser, and down sampled negative examples to be about four times of positive examples (from our empirical experience this ratio yields better accuracy).

The baseline features used to train the models include features from the user (e.g., age, gender, categories of clicked ads, categories of converted ads), publisher (e.g., publisher id), ad (e.g., the category of the ad).

Here we want to compare the impact of users' app usage features created from different app categorization methods, as described below:

- Store category: apps' store categories are determined by app developers, who should presumably have the best understanding of the app and choose the most relevant app category.
- k-means: use k-means clustering algorithm to cluster apps based on the learned app vectors from app2vec. Here we let $k = 60$, which is about the amount of app categories in the app store.

For each advertiser, we conduct 3-fold cross validation to evaluate the model performance, where 66% of the data are used for model training, and the other 33% are used for accuracy evaluation. Figure 1 shows the conversion prediction models accuracy (AUC) comparison (with 95% confidence interval) by using different user app usage profiles. It is clear that adding user app usage features can improve baseline model's accuracy. Simply using app store categories

[4]Anonymized user IDs from ad servers were used to evaluate method accuracy.

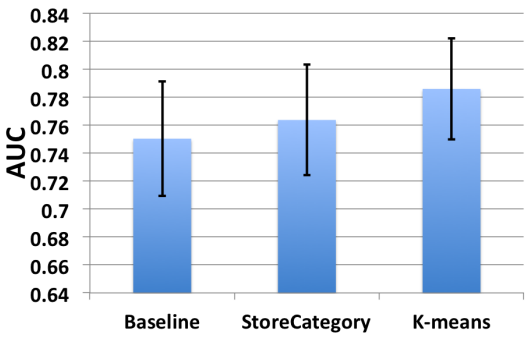

Figure 1: Install prediction model, AUC comparison using different methods to provide user used app category information. Based on 17 install prediction models for 17 different advertisers. AUC for each model was taken from the mean of 3-fold cross-validation, and error bar for 95% confidence interval is provided.

achieves the smallest improvement. Maybe the predefined app store categories have ambiguity and developers of similar apps may assign their apps into different app store categories. However, the app similarity from `app2vec` results are from a huge number of users usage patterns, and we have shown the learned app similarities have high quality (see Table 4 in Section 3.3). Therefore, clustering apps into the same amount of categories as store categories generates better accuracy, with 4.7% AUC lift over the baseline.

5. RELATED WORK

Our work is directly related to mobile application recommendation work. In [9] authors proposed that the context of user state and time can help significantly in recommending apps to users. [16] takes the user to user social network structure and social influence into account to predict app installation, instead of using the user to user graph constructed by commonly installed apps. Besides user interests, [12] considers the privacy preferences of a user to generate app recommendations, and using the relationships between apps and their requested access permissions [20] introduced a bipartite graph based approach for computing similarities among mobile apps. However for the cold-start problem, when there is no prior information about a user's preference, [10] proposed to use Twitter followers app preferences to make recommendations. In our study, we augment the state-of-art collaborative filtering approach with semantic relationships among apps from their usage contexts.

Our `app2vec` is inspired by `word2vec` work [15, 14]. The proposed concept of using shallow learning, but taking advantage of big data opens up research and applications in various areas. There is various follow-up work applying `word2vec` concept to other domains. [6] builds event level model of user's mobile activities, authors demonstrated the learned latent vector of events are effective in predicting what the next app user is going to use. [8] proposed a hierarchical model that by maintaining a document as the global context of words in the document, after the training process word vectors and document vector can be learned together. In our work, we treated each mobile app as a word and the app usage sequence from one user as a document. Through examples, we show that the learned app semantic similarity is meaningful. [11] also proposed that the words in a context should not be treated equally in the CBOW model, which are weighted by a softmax over context words. As an alternative approach to app representation learning, if one can construct an app-to-app graph, the method proposed in [17] can be applied, which optimizes an objective function to preserve both the local and global network structures. In the app world, we applied domain knowledge that app sessions happen with a time gap in between, and the context apps are weighted by the time gap to target app.

6. CONCLUSION

In this paper, we introduce a new way to vectorize mobile app usage. This is inspired by deep learning technique in `word2vec` for word documents (in contrast to the conventional bag-of-words or collaborative filtering approaches), and this models mobile applications based on app usage contexts and captures semantics. We show using Yahoo!'s human evaluation that `app2vec` captures semantic relationships between apps, where 82% of the retrieved top similar apps are semantically relevant, achieving 37% lift over bag-of-word approach and 140% lift over matrix factorization approach. Furthermore, we use `app2vec` for predicting conversions and show it improves prediction accuracy by almost 5% which is significant. We believe that `app2vec` will find many more uses in mobile app ad research.

7. REFERENCES

[1] The mobile app-install ad is driving a boom in mobile ad spend - and not just among game makers. *http://www.businessinsider.com/the-mobile-app-install-ad-is-driving-a-boom-in-mobile-ad-spend-and-not-just-among-game-makers-2015-1*. 2015/06/15.

[2] Shopping, productivity and messaging give mobile another stunning growth year. `http://flurrymobile.tumblr.com/post/115194992530/shopping-productivity-and-messaging-give-mobile`. 2015/01/06.

[3] Smartphones: So many apps, so much time. `http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html`. 2014/07/01.

[4] word2vec. `https://code.google.com/p/word2vec/`. Accessed: 2015/03/30.

[5] A. Ahmed, Y. Low, M. Aly, V. Josifovski, and A. 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.

[6] R. Baeza-Yates, D. Jiang, F. Silvestri, and B. Harrison. Predicting the next app that you are going to use. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 285–294. ACM, 2015.

[7] Y. Chen, D. Pavlov, and J. F. Canny. Large-scale behavioral targeting. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge*

discovery and data mining, pages 209–218. ACM, 2009.

[8] N. Djuric, H. Wu, V. Radosavljevic, M. Grbovic, and N. Bhamidipati. Hierarchical neural language models for joint representation of streaming documents and their content. In *Proceedings of the 24th International Conference on World Wide Web*, pages 248–255. International World Wide Web Conferences Steering Committee, 2015.

[9] A. Karatzoglou, L. Baltrunas, K. Church, and M. Böhmer. Climbing the app wall: enabling mobile app discovery through context-aware recommendations. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2527–2530. ACM, 2012.

[10] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua. Addressing cold-start in app recommendation: Latent user models constructed from twitter followers. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 283–292. ACM, 2013.

[11] W. Ling, L. Chu-Cheng, Y. Tsvetkov, and S. Amir. Not all contexts are created equal: Better word representations with variable attention. 2015.

[12] B. Liu, D. Kong, L. Cen, N. Z. Gong, H. Jin, and H. Xiong. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 315–324. ACM, 2015.

[13] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[16] W. Pan, N. Aharony, and A. Pentland. Composite social network for predicting mobile apps installation. In *AAAI*, 2011.

[17] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. ACM, 2015.

[18] J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen. How much can behavioral targeting help online advertising? In *Proceedings of the 18th international conference on World wide web*, pages 261–270. ACM, 2009.

[19] H. Zhu, H. Cao, E. Chen, H. Xiong, and J. Tian. Exploiting enriched contextual information for mobile app classification. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 1617–1621. ACM, 2012.

[20] H. Zhu, H. Xiong, Y. Ge, and E. Chen. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 951–960. ACM, 2014.