Real-time Personalization using Embeddings for Search Ranking at Airbnb

Mihajlo Grbovic
Airbnb, Inc.
San Francisco, California, USA
mihajlo.grbovic@airbnb.com

Haibin Cheng
Airbnb, Inc.
San Francisco, California, USA
haibin.cheng@airbnb.com

ABSTRACT

Search Ranking and Recommendations are fundamental problems of crucial interest to major Internet companies, including web search engines, content publishing websites and marketplaces. However, despite sharing some common characteristics a one-size-fits-all solution does not exist in this space. Given a large difference in content that needs to be ranked, personalized and recommended, each marketplace has a somewhat unique challenge. Correspondingly, at Airbnb, a short-term rental marketplace, search and recommendation problems are quite unique, being a two-sided marketplace in which one needs to optimize for host and guest preferences, in a world where a user rarely consumes the same item twice and one listing can accept only one guest for a certain set of dates. In this paper we describe Listing and User Embedding techniques we developed and deployed for purposes of Real-time Personalization in Search Ranking and Similar Listing Recommendations, two channels that drive 99% of conversions. The embedding models were specifically tailored for Airbnb marketplace, and are able to capture guest's short-term and long-term interests, delivering effective home listing recommendations. We conducted rigorous offline testing of the embedding models, followed by successful online tests before fully deploying them into production.

CCS CONCEPTS

• **Information systems** → Content ranking; Web log analysis; Personalization; Query representation; Document representation;

KEYWORDS

Search Ranking; User Modeling; Personalization

ACM Reference format:
Mihajlo Grbovic and Haibin Cheng. 2018. Real-time Personalization using Embeddings for Search Ranking at Airbnb. In *Proceedings of The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, United Kingdom, August 19–23, 2018 (KDD '18),* 10 pages.
https://doi.org/10.1145/3219819.3219885

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3219885

1 INTRODUCTION

During last decade Search architectures, which were typically based on classic Information Retrieval, have seen an increased presence of Machine Learning in its various components [2], especially in Search Ranking which often has challenging objectives depending on the type of content that is being searched over. The main reason behind this trend is the rise in the amount of search data that can be collected and analyzed. The large amounts of collected data open up possibilities for using Machine Learning to personalize search results for a particular user based on previous searches and recommend similar content to recently consumed one.

The objective of any search algorithm can vary depending on the platform at hand. While some platforms aim at increasing website engagement (e.g. clicks and time spent on news articles that are being searched), others aim at maximizing conversions (e.g. purchases of goods or services that are being searched over), and in the case of two sided marketplaces we often need to optimize the search results for both sides of the marketplace, i.e. sellers and buyers. The two sided marketplaces have emerged as a viable business model in many real world applications. In particular, we have moved from the social network paradigm to a network with two distinct types of participants representing supply and demand. Example industries include accommodation (Airbnb), ride sharing (Uber, Lyft), online shops (Etsy), etc. Arguably, content discovery and search ranking for these types of marketplaces need to satisfy both supply and demand sides of the ecosystem in order to grow and prosper.

In the case of Airbnb, there is a clear need to optimize search results for both hosts and guests, meaning that given an input query with location and trip dates we need to rank high listings whose location, price, style, reviews, etc. are appealing to the guest and, at the same time, are a good match in terms of host preferences for trip duration and lead days. Furthermore, we need to detect listings that would likely reject the guest due to bad reviews, pets, length of stay, group size or any other factor, and rank these listings lower. To achieve this we resort to using Learning to Rank. Specifically, we formulate the problem as pairwise regression with positive utilities for bookings and negative utilities for rejections, which we optimize using a modified version of Lambda Rank [4] model that jointly optimizes ranking for both sides of the marketplace.

Since guests typically conduct multiple searches before booking, i.e. click on more than one listing and contact more than one host during their search session, we can use these in-session signals, i.e. clicks, host contacts, etc. for Real-time Personalization where the aim is to show to the guest more of the listings similar to the ones we think they liked since staring the search session. At the same time we can use the negative signal, e.g. skips of high ranked listings, to show to the guest less of the listings similar to the ones we think

they did not like. To be able to calculate similarities between listings that guest interacted with and candidate listings that need to be ranked we propose to use listing embeddings, low-dimensional vector representations learned from search sessions. We leverage these similarities to create personalization features for our Search Ranking Model and to power our Similar Listing Recommendations, the two platforms that drive 99% of bookings at Airbnb.

In addition to Real-time Personalization using immediate user actions, such as clicks, that can be used as proxy signal for short-term user interest, we introduce another type of embeddings trained on bookings to be able to capture user's long-term interest. Due to the nature of travel business, where users travel 1-2 times per year on average, bookings are a sparse signal, with a long tail of users with a single booking. To tackle this we propose to train embeddings at a level of user type, instead of a particular user id, where type is determined using many-to-one rule-based mapping that leverages known user attributes. At the same time we learn listing type embeddings in the same vector space as user type embeddings. This enables us to calculate similarities between user type embedding of the user who is conducting a search and listing type embeddings of candidate listings that need to be ranked.

Compared to previously published work on embeddings for personalization on the Web, novel contributions of this paper are:

- **Real-time Personalization** - Most of the previous work on personalization and item recommendations using embeddings [8, 11] is deployed to production by forming tables of user-item and item-item recommendations offline, and then reading from them at the time of recommendation. We implemented a solution where embeddings of items that user most recently interacted with are combined in an online manner to calculate similarities to items that need to be ranked.
- **Adapting Training for Congregated Search** - Unlike in Web search, the search on travel platforms is often congregated, where users frequently search only within a certain market, e.g. Paris., and rarely across different markets. We adapted the embedding training algorithm to take this into account when doing negative sampling, which lead to capturing better within-market listings similarities.
- **Leveraging Conversions as Global Context** - We recognize the importance of click sessions that end up in conversion, in our case booking. When learning listing embeddings we treat the booked listing as global context that is always being predicted as the window moves over the session.
- **User Type Embeddings** - Previous work on training user embeddings to capture their long-term interest [6, 27] train a separate embedding for each user. When target signal is sparse, there is not enough data to train a good embedding representation for each user. Not to mention that storing embeddings for each user to perform online calculations would require lot of memory. For that reason we propose to train embeddings at a level of user type, where groups of users with same type will have the same embedding.
- **Rejections as Explicit Negatives** - To reduce recommendations that result in rejections we encode host preference signal in user and listing type embeddings by treating host rejections as explicit negatives during training.

For short-term interest personalization we trained listing embeddings using more than 800 million search clicks sessions, resulting in high quality listing representations. We used extensive offline and online evaluation on real search traffic which showed that adding embedding features to the ranking model resulted in significant booking gain. In addition to the search ranking algorithm, listing embeddings were successfully tested and launched for similar listing recommendations where they outperformed the existing algorithm click-through rate (CTR) by 20%.

For long-term interest personalization we trained user type and listing type embeddings using sequences of booked listings by 50 million users. Both user and listing type embeddings were learned in the same vector space, such that we can calculate similarities between user type and listing types of listings that need to be ranked. The similarity was used as an additional feature for search ranking model and was also successfully tested and launched.

2 RELATED WORK

In a number of Natural Language Processing (NLP) applications classic methods for language modeling that represent words as high-dimensional, sparse vectors have been replaced by Neural Language models that learn word embeddings, i.e. low-dimensional representations of words, through the use of neural networks [25, 27]. The networks are trained by directly taking into account the word order and their co-occurrence, based on the assumption that words frequently appearing together in the sentences also share more statistical dependence. With the development of highly scalable continuous bag-of-words (CBOW) and skip-gram (SG) language models for word representation learning [17], the embedding models have been shown to obtain state-of-the-art performance on many traditional language tasks after training on large text data.

More recently, the concept of embeddings has been extended beyond word representations to other applications outside of NLP domain. Researchers from the Web Search, E-commerce and Marketplace domains have quickly realized that just like one can train word embeddings by treating a sequence of words in a sentence as context, same can be done for training embeddings of user actions, e.g. items that were clicked or purchased [11, 18], queries and ads that were clicked [8, 9], by treating sequence of user actions as context. Ever since, we have seen embeddings being leveraged for various types of recommendations on the Web, including music recommendations [26], job search [13], app recommendations [21], movie recommendations [3, 7], etc. Furthermore, it has been shown that items which user interacted with can be leveraged to directly lean user embeddings in the same feature space as item embeddings, such that direct user-item recommendations can be made [6, 10, 11, 24, 27]. Alternative approach, specifically useful for cold-start recommendations, is to still to use text embeddings (e.g. ones publicly available at https://code.google.com/p/word2vec) and leverage item and/or user meta data (e.g. title and description) to compute their embeddings [5, 14, 19, 28]. Finally, similar extensions of embedding approaches have been proposed for Social Network analysis, where random walks on graphs can be used to learn embeddings of nodes in graph structure [12, 20].

Embedding approaches have had a major impact in both academia and industry circles. Recent industry conference publications and

talks show that they have been successfully deployed in various personalization, recommendation and ranking engines of major Web companies, such as Yahoo [8, 11, 29], Etsy [1], Criteo [18], Linkedin [15, 23], Tinder [16], Tumblr [10], Instacart [22], Facebook [28].

3 METHODOLOGY

In the following we introduce the proposed methodology for the task of listing recommendations and listing ranking in search at Airbnb. We describe two distinct approaches, i.e. listing embeddings for short-term real-time personalization and user-type & listing type embeddings for long term personalization, respectively.

3.1 Listing Embeddings

Let us assume we are given a set $\mathcal{S}$ of S click sessions obtained from N users, where each session $s = (l_1, \ldots, l_M) \in \mathcal{S}$ is defined as an uninterrupted sequence of M listing ids that were clicked by the user. A new session is started whenever there is a time gap of more than 30 minutes between two consecutive user clicks. Given this data set, the aim is to learn a d-dimensional real-valued representation $\mathbf{v}_{l_i} \in \mathbb{R}^d$ of each unique listing l_i, such that similar listings lie nearby in the embedding space.

More formally, the objective of the model is to learn listing representations using the skip-gram model [17] by maximizing the objective function $\mathcal{L}$ over the entire set $\mathcal{S}$ of search sessions, defined as follows

$$\mathcal{L} = \sum_{s \in \mathcal{S}} \sum_{l_i \in s} \Big(\sum_{-m \geq j \leq m, i \neq 0} \log \mathbb{P}(l_{i+j} | l_i) \Big), \tag{1}$$

Probability $\mathbb{P}(l_{i+j}|l_i)$ of observing a listing l_{i+j} from the contextual neighborhood of clicked listing l_i is defined using the soft-max

$$\mathbb{P}(l_{i+j}|l_i) = \frac{\exp(\mathbf{v}_{l_i}^\top \mathbf{v}'_{l_{i+j}})}{\sum_{l=1}^{|\mathcal{V}|} \exp(\mathbf{v}_{l_i}^\top \mathbf{v}'_l)}, \tag{2}$$

where $\mathbf{v}_l$ and $\mathbf{v}'_l$ are the input and output vector representations of listing l, hyperparameter m is defined as a length of the relevant forward looking and backward looking context (neighborhood) for a clicked listing, and $\mathcal{V}$ is a vocabulary defined as a set of unique listings ids in the data set. From (1) and (2) we see that the proposed approach models temporal context of listing click sequences, where listings with similar contexts (i.e., with similar neighboring listings in search sessions) will have similar representations.

Time required to compute gradient $\nabla\mathcal{L}$ of the objective function in (1) is proportional to the vocabulary size $|\mathcal{V}|$, which for large vocabularies, e.g. several millions listing ids, is an infeasible task. As an alternative we used negative sampling approach proposed in [17], which significantly reduces computational complexity. Negative sampling can be formulated as follows. We generate a set $\mathcal{D}_p$ of *positive pairs* (l, c) of clicked listings l and their contexts c (i.e., clicks on other listings by the same user that happened before and after click on listing l within a window of length m), and a set $\mathcal{D}_n$ of *negative pairs* (l, c) of clicked listings and n randomly sampled listings from the entire vocabulary $\mathcal{V}$. The optimization objective then becomes

$$\underset{\theta}{\operatorname{argmax}} \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_l}}, \tag{3}$$

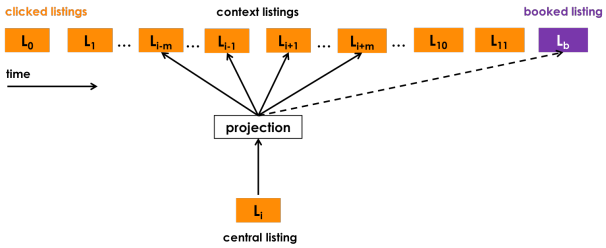

Figure 1: Skip-gram model for Listing Embeddings

where parameters θ to be learned are $\mathbf{v}_l$ and $\mathbf{v}_c$, $l, c \in \mathcal{V}$. The optimization is done via stochastic gradient ascent.

Booked Listing as Global Context. We can break down the click sessions set $\mathcal{S}$ into 1) *booked sessions*, i.e. click sessions that end with user booking a listing to stay at, and 2) *exploratory sessions*, i.e. click sessions that do not end with booking, i.e. users were just browsing. Both are useful from the standpoint of capturing contextual similarity, however *booked sessions* can be used to adapt the optimization such that at each step we predict not only the neighboring clicked listings but the eventually booked listing as well. This adaptation can be achieved by adding booked listing as *global context*, such that it will always be predicted no matter if it is within the context window or not. Consequently, for *booked sessions* the embedding update rule becomes

$$\underset{\theta}{\operatorname{argmax}} \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_l}} + \log \frac{1}{1 + e^{-\mathbf{v}'_{l_b} \mathbf{v}_l}}, \tag{4}$$

where $\mathbf{v}_{l_b}$ is the embedding of the booked listing l_b. For *exploratory sessions* the updates are still conducted by optimizing objective (3).

Figure 1 shows a graphical representation of how listing embeddings are learned from *booked sessions* using a sliding window of size $2n + 1$ that slides from the first clicked listing to the booked listing. At each step the embedding of the central listing $\mathbf{v}_l$ is being updated such that it predicts the embeddings of the context listings $\mathbf{v}_c$ from $\mathcal{D}_p$ and the booked listing $\mathbf{v}_{l_b}$. As the window slides some listings fall in and out of the context set, while the booked listing always remains within it as global context (dotted line).

Adapting Training for Congregated Search. Users of online travel booking sites typically search only within a single market, i.e. location they want to stay at. As a consequence, there is a high probability that $\mathcal{D}_p$ contains listings from the same market. On the other hand, due to random sampling of negatives, it is very likely that $\mathcal{D}_n$ contains mostly listings that are not from the same markets as listings in $\mathcal{D}_p$. At each step, for a given central listing l, the positive context mostly consist of listings from the same market as l, while the negative context mostly consists of listings that are not from the same market as l. We found that this imbalance leads to learning sub-optimal within-market similarities. To address this issue we propose to add a set of random negatives $\mathcal{D}_{m_n}$, sampled from the market of the central listing l,

$$\underset{\theta}{\operatorname{argmax}} \sum_{(l,c) \in \mathcal{D}_p} \log \frac{1}{1 + e^{-\mathbf{v}'_c \mathbf{v}_l}} + \sum_{(l,c) \in \mathcal{D}_n} \log \frac{1}{1 + e^{\mathbf{v}'_c \mathbf{v}_l}} + \log \frac{1}{1 + e^{-\mathbf{v}'_{l_b} \mathbf{v}_l}} + \sum_{(l,m_n) \in \mathcal{D}_{m_n}} \log \frac{1}{1 + e^{\mathbf{v}'_{m_n} \mathbf{v}_l}}. \tag{5}$$

where parameters θ to be learned are $\mathbf{v}_l$ and $\mathbf{v}_c$, $l, c \in \mathcal{V}$.

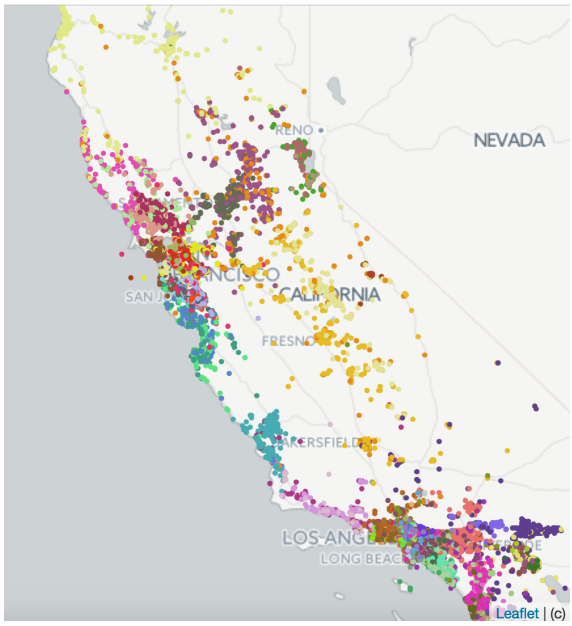

Figure 2: California Listing Embedding Clusters

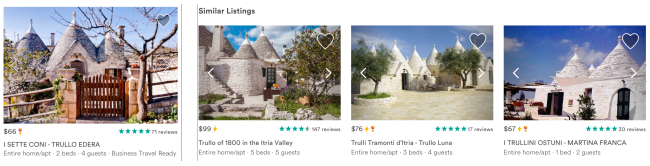

Figure 3: Similar Listings using Embeddings

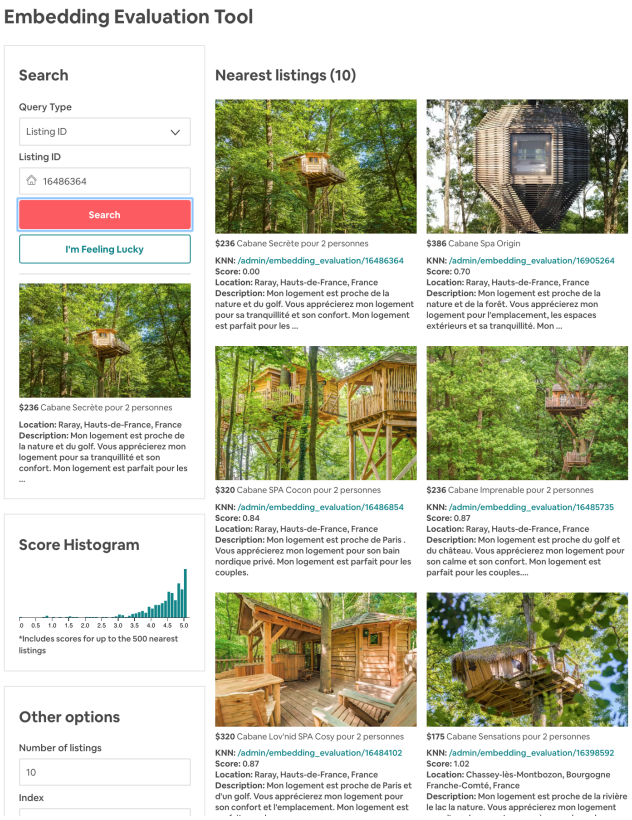

Figure 4: Embeddings Evaluation Tool

Cold start listing embeddings. Every day new listings are created by hosts and made available on Airbnb. At that point these listings do not have an embedding because they were not present in the click sessions $\mathcal{S}$ training data. To create embeddings for new listings we propose to utilize existing embeddings of other listings.

Upon listing creation the host is required to provide information about the listing, such as location, price, listing type, etc. We use the provided meta-data about the listing to find 3 geographically closest listings (within a 10 miles radius) that have embeddings, are of same listing type as the new listing (e.g. Private Room) and belong to the same price bucket as the new listing (e.g. $\$20 - \$25$ per night). Next, we calculate the mean vector using 3 embeddings of identified listings to form the new listing embedding. Using this technique we are able to cover more than 98% of new listings.

Examining Listing Embeddings. To evaluate what characteristics of listings were captured by embeddings we examine the $d = 32$ dimensional embeddings trained using (5) on 800 million click sessions. First, by performing k-means clustering on learned embeddings we evaluate if geographical similarity is encoded. Figure 2, which shows resulting 100 clusters in California, confirms that listings from similar locations are clustered together. We found the clusters very useful for re-evaluating our definitions of travel markets. Next, we evaluate average cosine similarities between listings from Los Angeles of different listing types (Table 1) and between listings of different price ranges (Table 2). From those tables it can observed that cosine similarities between listings of same type and price ranges are much higher compared to similarities between listings of different types and price ranges. Therefore, we can conclude that those two listing characteristics are well encoded in the learned embeddings as well.

While some listing characteristics, such as price, do not need to be learned because they can be extracted from listing meta-data, other types of listing characteristics, such as architecture, style and feel are much harder to extract in form of listing features. To evaluate if these characteristics are captured by embeddings we can examine k-nearest neighbors of unique architecture listings in the listing embedding space. Figure 3 shows one such case where for a listing of unique architecture on the left, the most similar listings are of the same style and architecture. To be able to conduct fast and easy explorations in the listing embedding space we developed an internal **Similarity Exploration Tool** shown in Figure 4.

Demonstration video of this tool, which is available online at **https://youtu.be/1kJSAG91TrI**, shows many more examples of embeddings being able to find similar listings of the same unique architecture, including houseboats, treehouses, castles, chalets, beachfront apartments, etc.

Table 1: Cosine similarities between different Listing Types

Room Type	Entire Home	Private Room	Shared Room
Entire Home	**0.895**	0.875	0.848
Private Room		**0.901**	0.865
Shared Room			**0.896**

Table 2: Cosine similarities between different Price Ranges

Price Range	<$30	$30-$60	$60-$90	$90-$120	$120+
<$30	**0.916**	0.887	0.882	0.871	0.854
$30-$60		**0.906**	0.889	0.876	0.865
$60-$90			**0.902**	0.883	0.880
$90-$120				**0.898**	0.890
$120+					**0.909**

3.2 User-type & Listing-type Embeddings

Listing embeddings described in Section 3.1. that were trained using click sessions are very good at finding similarities between listings of the same market. As such, they are suitable for short-term, in-session, personalization where the aim is to show to the user listings that are similar to the ones they clicked during the immanent search session.

However, in addition to in-session personalization, based on signals that just happened within the same session, it would be useful to personalize search based on signals from user's longer-term history. For example, given a user who is currently searching for a listing in Los Angeles, and has made past bookings in New York and London, it would be useful to recommend listings that are similar to those previously booked ones.

While some cross-market similarities are captured in listing embeddings trained using clicks, a more principal way of learning such cross-market similarities would be to learn from sessions constructed of listings that a particular user booked over time. Specifically, let us assume we are given a set $\mathcal{S}_b$ of booking sessions obtained from N users, where each booking session $s_b = (l_{b1}, \ldots, l_{bM})$ is defined as a sequence of listings booked by user j ordered in time. Attempting to learn embeddings $\mathbf{v}_{l_{id}}$ for each *listing_id* using this type of data would be challenging in many ways:

- First, booking sessions data $\mathcal{S}_b$ is much smaller than click sessions data $\mathcal{S}$ because bookings are less frequent events.
- Second, many users booked only a single listing in the past and we cannot learn from a session of length 1.
- Third, to learn a meaningful embedding for any entity from contextual information at least $5-10$ occurrences of that entity are needed in the data, and there are many *listing_ids* on the platform that were booked less than $5-10$ times.
- Finally, long time intervals may pass between two consecutive bookings by the user, and in that time user preferences, such as price point, may change, e.g. due to career change.

To address these very common marketplace problems in practice, we propose to learn embeddings at a level of *listing_type* instead of *listing_id*. Given meta-data available for a certain *listing_id* such as location, price, listing type, capacity, number of beds, etc., we use a rule-based mapping defined in Table 3 to determine its *listing_type*. For example, an *Entire Home* listing from *US* that has a 2 person capacity, 1 bed, 1 bedroom & 1 bathroom, with Average Price Per Night of $60.8, Average Price Per Night Per Guest of $29.3, 5 reviews, all 5 stars, and 100% *New Guest Accept Rate* would map into $listing\_type = US\_lt_1\_pn_3\_pg_3\_r_3\_5s_4\_c_2\_b_1\_bd_2\_bt_2\_nu_3$. Buckets are determined in a data-driven manner to maximize for coverage in each *listing_type* bucket. The mapping from *listing_id* to a *listing_type* is a many-to-one mapping, meaning that many listings will map into the same *listing_type*.

Table 3: Mappings of listing meta data to listing type buckets

Buckets	1	2	3	4	5	6	7	8
Country	US	CA	GB	FR	MX	AU	ES	...
Listing Type	Ent	Priv	Share					
$ per Night	<40	40-55	56-69	70-83	84-100	101-129	130-189	190+
$ per Guest	<21	21-27	28-34	35-42	43-52	53-75	76+	
Num Reviews	0	1	2-5	6-10	11-35	35+		
Listing 5 Star %	0-40	41-60	61-90	90+				
Capacity	1	2	3	4	5	6+		
Num Beds	1	2	3	4+				
Num Bedrooms	0	1	2	3	4+			
Num Bathroom	0	1	2	3+				
New Guest Acc %	<60	61-90	>91					

Table 4: Mappings of user meta data to user type buckets

Buckets	1	2	3	4	5	6	7	8
Market	SF	NYC	LA	HK	PHL	AUS	LV	...
Language	en	es	fr	jp	ru	ko	de	...
Device Type	Mac	Msft	Andr	Ipad	Tablet	Iphone	...	
Full Profile	Yes	No						
Profile Photo	Yes	No						
Num Bookings	0	1	2-7	8+				
$ per Night	<40	40-55	56-69	70-83	84-100	101-129	130-189	190+
$ per Guest	<21	21-27	28-34	35-42	43-52	53-75	76+	
Capacity	<2	2-2.6	2.7-3	3.1-4	4.1-6	6.1+		
Num Reviews	<1	1-3.5	3.6-10	> 10				
Listing 5 Star %	0-40	41-60	61-90	90+				
Guest 5 Star %	0-40	41-60	61-90	90+				

To account for user ever-changing preferences over time we propose to learn *user_type* embeddings in the same vector space as *listing_type* embeddings. The *user_type* is determined using a similar procedure we applied to listings, i.e. by leveraging meta-data about user and their previous bookings, defined in Table 4. For example, for a user from San Francisco with MacBook laptop, English language settings, full profile with user photo, 83.4% average Guest 5 star rating from hosts, who has made 3 bookings in the past, where the *average* statistics of booked listings were $52.52 Price Per Night, $31.85 Price Per Night Per Guest, 2.33 Capacity, 8.24 Reviews and 76.1% Listing 5 star rating, the resulting *user_type* is $SF\_lg_1\_dt_1\_fp_1\_pp_1\_nb_1\_ppn_2\_ppg_3\_c_2\_nr_3\_l5s_3\_g5s_3$. When generating booking sessions for training embeddings we calculate the *user_type* up to the latest booking. For users who made their first booking *user_type* is calculated based on the first 5 rows from Table 4 because at the time of booking we had no prior information about past bookings. This is convenient, because learned embeddings for *user_types* which are based on first 5 rows can be used for cold-start personalization for logged-out users and new users with no past bookings.

Training Procedure. To learn *user_type* and *listing_type* embeddings in the same vector space we incorporate the *user_type*

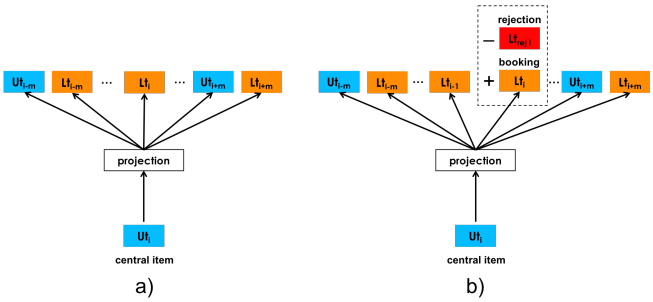

Figure 5: Listing Type and User Type Skip-gram model

into the booking sessions. Specifically, we form a set $\mathcal{S}_b$ consisting of N_b booking sessions from N users, where each session $s_b = (u_{type_1} l_{type_1}, \ldots, u_{type_M} l_{type_M}) \in \mathcal{S}_b$ is defined as a sequence of booking events, i.e. (*user_type*, *listing_type*) tuples ordered in time. Note that each session consists of bookings by same *user_id*, however for a single *user_id* their *user_types* can change over time, similarly to how *listing_types* for the same listing can change over time as they receive more bookings.

The objective that needs to be optimized can be defined similarly to (3), where instead of listing l, the center item that needs to be updated is either *user_type* (u_t) or *listing_type* (l_t) depending on which one is caught in the sliding window. For example, to update the central item which is a *user_type* (u_t) we use

$$\underset{\theta}{\text{argmax}} \sum_{(u_t,c)\in\mathcal{D}_{book}} \log \frac{1}{1+e^{-\mathbf{v}'_c \mathbf{v}_{u_t}}} + \sum_{(u_t,c)\in\mathcal{D}_{neg}} \log \frac{1}{1+e^{\mathbf{v}'_c \mathbf{v}_{u_t}}}, \quad (6)$$

where $\mathcal{D}_{book}$ contains the *user_type* and *listing_type* from recent user history, specifically user bookings from near past and near future with respect to central item's timestamp, while $\mathcal{D}_{neg}$ contains random *user_type* or *listing_type* instances used as negatives. Similarly, if the central item is a *listing_type* (l_t) we optimize the following objective

$$\underset{\theta}{\text{argmax}} \sum_{(l_t,c)\in\mathcal{D}_{book}} \log \frac{1}{1+e^{-\mathbf{v}'_c \mathbf{v}_{l_t}}} + \sum_{(l_t,c)\in\mathcal{D}_{neg}} \log \frac{1}{1+e^{\mathbf{v}'_c \mathbf{v}_{l_t}}}. \quad (7)$$

Figure 5a (on the left) shows a graphical representation of this model, where central item represents *user_type* (u_t) for which the updates are performed as in (6).

Since *booking sessions* by definition mostly contain listings from different markets, there is no need to sample additional negatives from same market as the booked listing, like we did in Session 3.1. to account for the congregated search in *click sessions*.

Explicit Negatives for Rejections. Unlike clicks that only reflect guest-side preferences, bookings reflect host-side preferences as well, as there exists an explicit feedback from the host, in form of accepting guest's request to book or rejecting guest's request to book. Some of the reasons for host rejections are bad guest star ratings, incomplete or empty guest profile, no profile picture, etc. These characteristics are part of *user_type* definition from Table 4.

Host rejections can be utilized during training to encode the host preference signal in the embedding space in addition to the guest preference signal. The whole purpose of incorporating the rejection signal is that some *listing_types* are less sensitive to *user_types* with no bookings, incomplete profiles and less than average guest star ratings than others, and we want the embeddings of those *listing_types* and *user_types* to be closer in the vector space, such that recommendations based on embedding similarities would reduce future rejections in addition to maximizing booking chances.

We formulate the use of the rejections as explicit negatives in the following manner. In addition to sets $\mathcal{D}_{book}$ and $\mathcal{D}_{neg}$, we generate a set $\mathcal{D}_{rej}$ of pairs (u_t, l_t) of *user_type* or *listing_type* that were involved in a rejection event. As depicted in Figure 5b (on the right), we specifically focus on the cases when host rejections (labeled with a minus sign) were followed by a successful booking (labeled with a plus sign) of another listing by the same user. The new optimization objective can then be formulated as

$$\underset{\theta}{\text{argmax}} \sum_{(u_t,c)\in\mathcal{D}_{book}} \log \frac{1}{1+\exp^{-\mathbf{v}'_c \mathbf{v}_{u_t}}} + \sum_{(u_t,c)\in\mathcal{D}_{neg}} \log \frac{1}{1+\exp^{\mathbf{v}'_c \mathbf{v}_{u_t}}} + \sum_{(u_t,l_t)\in\mathcal{D}_{reject}} \log \frac{1}{1+\exp^{\mathbf{v}'_{l_t} \mathbf{v}_{u_t}}}. \quad (8)$$

in case of updating the central item which is a *user_type* (u_t), and

$$\underset{\theta}{\text{argmax}} \sum_{(l_t,c)\in\mathcal{D}_{book}} \log \frac{1}{1+\exp^{-\mathbf{v}'_c \mathbf{v}_{l_t}}} + \sum_{(l_t,c)\in\mathcal{D}_{neg}} \log \frac{1}{1+\exp^{\mathbf{v}'_c \mathbf{v}_{l_t}}} + \sum_{(l_t,u_t)\in\mathcal{D}_{reject}} \log \frac{1}{1+\exp^{\mathbf{v}'_{u_t} \mathbf{v}_{l_t}}}. \quad (9)$$

in case of updating the central item which is a *listing_type* (l_t).

Given learned embeddings for all *user_types* and *listing_types*, we can recommend to the user the most relevant listings based on the cosine similarities between user's current *user_type* embedding and *listing_type* embeddings of candidate listings. For example, in Table 5 we show cosine similarities between *user_type* = $SF\_lg_1\_dt_1\_fp_1\_pp_1\_nb_3\_ppn_5\_ppg_5\_c_4\_nr_3\_l5s_3\_g5s_3$ who typically books high quality, spacious listings with lots of good reviews and several different *listing_types* in US. It can be observed that listing types that best match these user preferences, i.e. entire home, lots of good reviews, large and above average price, have high cosine similarity, while the ones that do not match user preferences, i.e. ones with less space, lower price and small number of reviews have low cosine similarity.

Table 5: Recommendations based on type embeddings

User Type	
$SF\_lg_1\_dt_1\_fp_1\_pp_1\_nb_3\_ppn_5\_ppg_5\_c_4\_nr_3\_l5s_3\_g5s_3$	
Listing Type	**Sim**
$US\_lt_1\_pn_4\_pg_5\_r_5\_5s_4\_c_2\_b_1\_bd_3\_bt_3\_nu_3$ (large, good reviews)	0.629
$US\_lt_1\_pn_3\_pg_3\_r_5\_5s_2\_c_3\_b_1\_bd_2\_bt_2\_nu_3$ (cheaper, bad reviews)	0.350
$US\_lt_2\_pn_3\_pg_3\_r_5\_5s_4\_c_1\_b_1\_bd_2\_bt_2\_nu_3$ (priv room, good reviews)	0.241
$US\_lt_2\_pn_2\_pg_2\_r_5\_5s_2\_c_1\_b_1\_bd_2\_bt_2\_nu_3$ (cheaper, bad reviews)	0.169
$US\_lt_3\_pn_1\_pg_1\_r_5\_5s_3\_c_1\_b_1\_bd_2\_bt_2\_nu_3$ (shared room, bad reviews)	0.121

4 EXPERIMENTS

In this section we first cover the details of training Listing Embeddings and their Offline Evaluation. We then show Online Experiment Results of using Listing Embeddings for Similar Listing Recommendations on the Listing Page. Finally, we give background on our Search Ranking Model and describe how Listing Embeddings and Listing Type & User Type Embeddings were used to implement features for Real-time Personalization in Search. Both applications of embeddings were successfully launched to production.

4.1 Training Listing Embeddings

For training listing embeddings we created 800 million click sessions from search, by taking all searches from *logged-in users*, grouping them by user id and ordering clicks on listing ids in time. This was followed by splitting one large ordered list of listing ids into multiple ones based on 30 minute inactivity rule. Next, we removed accidental and short clicks, i.e. clicks for which user stayed on the listing page for less than 30 seconds, and kept only sessions consisting of 2 or more clicks. Finally, the sessions were anonymized by dropping the user id column. As mentioned before, click sessions consist of *exploratory sessions* &. *booked sessions* (sequence of clicks that end with booking). In light of offline evaluation results we oversampled *booked sessions* by 5x in the training data, which resulted in the best performing listing embeddings.

Setting up Daily Training. We learn listing embeddings for 4.5 million Airbnb listings and our training data practicalities and parameters were tuned using offline evaluation techniques presented below. Our training data is updated daily in a sliding window manner over multiple months, by processing the latest day search sessions and adding them to the dataset and discarding the oldest day search sessions from the dataset. We train embeddings for each *listing_id*, where we initialize vectors randomly before training (same random seed is used every time). We found that we get better offline performance if we re-train listing embeddings from scratch every day, instead of incrementally continuing training on existing vectors. The day-to-day vector differences do not cause discrepancies in our models because in our applications we use the cosine similarity as the primary signal and not the actual vectors themselves. Even with vector changes over time, the connotations of cosine similarity measure and its ranges do not change.

Dimensionality of listing embeddings was set to $d = 32$, as we found that to be a good trade-off between offline performance and memory needed to store vectors in RAM memory of search machines for purposes of real-time similarity calculations. Context window size was set to $m = 5$, and we performed 10 iterations over the training data. To implement the congregated search change to the algorithm we modified the original word2vec c code[1]. Training used MapReduce, where 300 mappers read data and a single reducer trains the model in a multi-threaded manner. End-to-end daily data generation and training pipeline is implemented using Airflow[2], which is Airbnb's open-sourced scheduling platform.

4.2 Offline Evaluation of Listing Embeddings

To be able to make quick decisions regarding different ideas on optimization function, training data construction, hyperparameters, etc, we needed a way to quickly compare different embeddings.

One way to evaluate trained embeddings is to test how good they are in recommending listings that user would book, based on the most recent user click. More specifically, let us assume we are given the most recently clicked listing and listing candidates that need to be ranked, which contain the listing that user eventually booked. By calculating cosine similarities between embeddings of clicked listing and candidate listings we can rank the candidates and observe the rank position of the booked listing.

[1]https://code.google.com/p/word2vec
[2]http://airbnb.io/projects/airflow

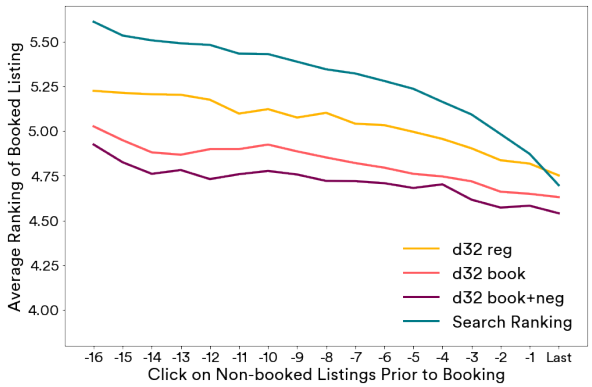

Figure 6: Offline evaluation of Listing Embeddings

For purposes of evaluation we use a large number of such search, click and booking events, where rankings were already assigned by our *Search Ranking* model. In Figure 6 we show results of offline evaluation in which we compared several versions of $d = 32$ embeddings with regards to how they rank the booked listing based on clicks that precede it. Rankings of booked listing are averaged for each click leading to the booking, going as far back as 17 clicks before the booking to the *Last* click before the booking. Lower values mean higher ranking. Embedding versions that we compared were 1) **d32**: trained using (3), 2) **d32 book**: trained with bookings as global context (4) and 3) **d32 book + neg**: trained with bookings as global context and explicit negatives from same market (5).

It can be observed that *Search Ranking* model gets better with more clicks as it uses memorization features. It can also be observed that *re-ranking* listings based on embedding similarity would be useful, especially in early stages of the search funnel. Finally, we can conclude that **d32 book + neg** outperforms the other two embedding versions. The same type of graphs were used to make decisions regarding hyperparameters, data construction, etc.

4.3 Similar Listings using Embeddings

Every Airbnb home listing page[3] contains *Similar Listings* carousel which recommends listings that are similar to it and available for the same set of dates. At the time of our test, the existing algorithm for *Similar Listings* carousel was calling the main *Search Ranking* model for the same location as the given listing followed by filtering on availability, price range and listing type of the given listing.

We conducted an A/B test where we compared the existing similar listings algorithm to an embedding-based solution, in which similar listings were produced by finding the k-nearest neighbors in listing embedding space. Given learned listing embeddings, similar listings for a given listing l were found by calculating cosine similarity between its vector $\mathbf{v}_l$ and vectors $\mathbf{v}_j$ of all listings from the same market that are available for the same set of dates (if check-in and check-out dates are set). The K listings with the highest similarity were retrieved as similar listings. The calculations were performed online and happen in parallel using our sharded architecture, where parts of embeddings are stored on each of the search machines.

[3]https://www.airbnb.com/rooms/433392

The A/B test showed that embedding-based solution lead to a 21% increase in Similar Listing carousel CTR (23% in cases when listing page had entered dates and 20% in cases of dateless pages) and 4.9% increase in guests who find the listing they end up booking in Similar Listing carousel. In light of these results we deployed the embedding-based Similar Listings to production.

4.4 Real time personalization in Search Ranking using Embeddings

Background. To formally describe our Search Ranking Model, let us assume we are given training data about each search $D_s = (\mathbf{x}_i, y_i), i = 1...K$, where K is the number of listings returned by search, $\mathbf{x}_i$ is a vector containing features of the i-th listing result and $y_i \in \{0, 0.01, 0.25, 1, -0.4\}$ is the label assigned to the i-th listing result. To assign the label to a particular listing from the search result we wait for 1 week after search happened to observe the final outcome, which can be $y_i = 1$ if listing was booked, $y_i = 0.25$ if listing host was contacted by the guest but booking did not happen, $y = -0.4$ if listing host rejected the guest, $y_i = 0.01$ is listing was clicked and $y_i = 0$ if listing was just viewed but not clicked. After that 1 week wait the set D_s is also shortened to keep only search results up to the last result user clicked on $K_c \leq K$. Finally, to form data $\mathcal{D} = \bigcup_{s=1}^{N} D_s$ we only keep D_s sets which contain at least one booking label. Every time we train a new ranking model we use the most recent 30 days of data.

Feature vector $\mathbf{x}_i$ for the i-th listing result consists of **listing features**, **user features**, **query features** and **cross-features**. Listing features are features associated with the listing itself, such as *price per night*, *listing type*, *number of rooms*, *rejection rate*, etc. Query features are features associated with the issued query, such as *number of guests*, *length of stay*, *lead days*, etc. User features are features associated with the user who is conducting the search, such as *average booked price*, *guest rating*, etc. Cross-features are features derived from two or more of these feature sources: listing, user, query. Examples of such features are *query listing distance*: distance between query location and listing location, *capacity fit*: difference between query number of guests and listing capacity, *price difference*: difference between listing price and average price of user's historical bookings, *rejection probability*: probability that host will reject these query parameters, *click percentage*: real-time memorization feature that tracks what percentage of user's clicks were on that particular listing, etc. The model uses approximately 100 features. For conciseness we will not list all of them.

Next, we formulate the problem as pairwise regression with search labels as utilities and use data $\mathcal{D}$ to train a Gradient Boosting Decision Trees (GBDT) model, using package [4] that was modified to support Lambda Rank. When evaluating different models offline, we use NDCG, a standard ranking metric, on hold-out set of search sessions, i.e. 80% of $\mathcal{D}$ for training and 20% for testing.

Finally, once the model is trained it is used for online scoring of listings in search. The signals needed to calculate feature vectors $\mathbf{x}_i$ for each listing returned by search query q performed by user u are all calculated in an online manner and scoring happens in parallel using our sharded architecture. Given all the scores, the listings are shown to the user in a descending order of predicted utility.

[4] https://github.com/yarny/gbdt

Table 6: Embedding Features for Search Ranking

Feature Name	Description
EmbClickSim	similarity to clicked listings in H_c
EmbSkipSim	similarity to skipped listings H_s
EmbLongClickSim	similarity to long clicked listings H_{lc}
EmbWishlistSim	similarity to wishlisted listings H_w
EmbInqSim	similarity to contacted listings H_i
EmbBookSim	similarity to booked listing H_b
EmbLastLongClickSim	similarity to last long clicked listing
UserTypeListingTypeSim	user type and listing type similarity

Listing Embedding Features. The first step in adding embedding features to our Search Ranking Model was to load the 4.5 million embeddings into our search backend such that they can be accessed in real-time for feature calculation and model scoring.

Next, we introduced several user short-term history sets, that hold user actions from last 2 weeks, which are updated in real-time as new user actions happen. The logic was implemented using using Kafka [5]. Specifically, for each user_id we collect and maintain (regularly update) the following sets of listing ids:

(1) H_c**: clicked listing_ids** - listings that user clicked on in last 2 weeks.
(2) H_{lc}**: long-clicked listing_ids** - listing that user clicked and stayed on the listing page for longer than 60 sec.
(3) H_s**: skipped listing_ids** - listings that user skipped in favor of a click on a lower positioned listing
(4) H_w**: wishlisted listing_ids** - listings that user added to a wishlist in last 2 weeks.
(5) H_i**: inquired listing_ids** - listings that user contacted in last 2 weeks but did not book.
(6) H_b**: booked listing_ids** - listings that user booked in last 2 weeks.

We further split each of the short-term history sets H_* into subsets that contain listings from the same market. For example, if user had clicked on listings from New York and Los Angeles, their set H_c would be further split into $H_c(NY)$ and $H_c(LA)$.

Finally, we define the embedding features which utilize the defined sets and the listing embeddings to produce a score for each candidate listing. The features are summarized in Table 6.

In the following we describe how *EmbClickSim* feature is computed using H_c. The rest of the features from top rows of Table 6 are computed in the same manner using their corresponding user short-term history set H_*.

To compute *EmbClickSim* for candidate listing l_i we need to compute cosine similarity between its listing embedding $\mathbf{v}_{l_i}$ and embeddings of listings in H_c. We do so by first computing H_c market-level centroid embeddings. To illustrate, let us assume H_c contains 5 listings from NY and 3 listings from LA. This would entail computing two market-level centroid embeddings, one for NY and one for LA, by averaging embeddings of listing ids from each of the markets. Finally, *EmbClickSim* is calculated as maximum out of

[5] https://kafka.apache.org

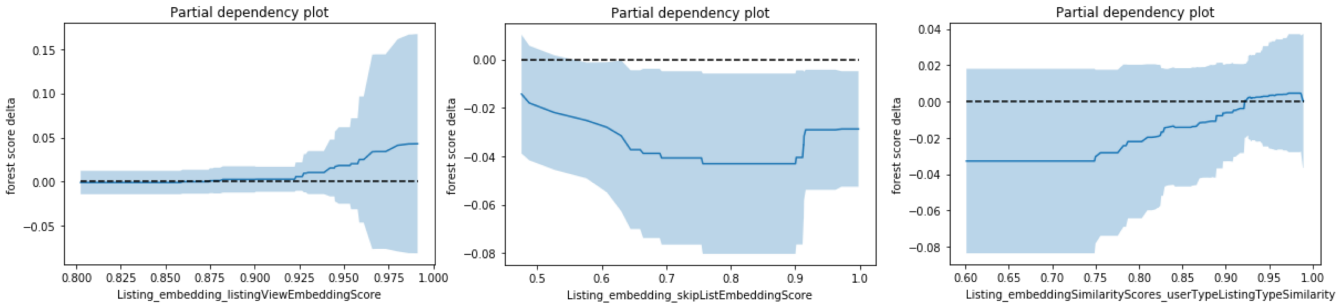

Figure 7: Partial Dependency Plots for EmbClickSim, EmbSkipSim and UserTypeListTypeSim

two similarities between listing embedding $\mathbf{v}_{l_i}$ and H_c market-level centroid embeddings.

More generally *EmbClickSim* can be expressed as

$$EmbClickSim(l_i, H_c) = \max_{m \in M} cos(\mathbf{v}_{l_i}, \sum_{l_h \in m, l_h \in H_c} \mathbf{v}_{l_h}), \quad (10)$$

where M is the set of markets user had clicks in.

In addition to similarity to all user clicks, we added a feature that measures similarity to the latest long click, *EmbLastLongClickSim*. For a candidate listing l_i it is calculated by finding the cosine similarity between its embedding $\mathbf{v}_{l_i}$ and the embedding of the latest long clicked listing l_{last} from H_{lc},

$$EmbLastLongClickSim(l_i, H_{lc}) = cos(\mathbf{v}_{l_i}, \mathbf{v}_{l_{last}}). \quad (11)$$

User-type & Listing-type Embedding Features. We follow similar procedure to introduce features based on user type and listing type embeddings. We trained embeddings for 500K user types and 500K listing types using 50 million user booking sessions. Embeddings were $d = 32$ dimensional and were trained using a sliding window of $m = 5$ over booking sessions. The user type and listing type embeddings were loaded to search machines memory, such that we can compute the type similarities online.

To compute the *UserTypeListingTypeSim* feature for candidate listing l_i we simply look-up its current listing type l_t as well as current user type u_t of the user who is conducting the search and calculate cosine similarity between their embeddings,

$$UserTypeListingTypeSim(u_t, l_t) = cos(\mathbf{v}_{u_t}, \mathbf{v}_{l_t}). \quad (12)$$

All features from Table 6 were logged for 30 days so they could be added to search ranking training set $\mathcal{D}$. The *coverage* of features, meaning the proportion of $\mathcal{D}$ which had particular feature populated, are reported in Table 7. As expected, it can be observed that features based on user clicks and skips have the highest coverage.

Finally, we trained a new GBDT Search Ranking model with embedding features added. Feature importances for embedding features (ranking among 104 features) are shown in Table 7. Top ranking features are similarity to listings user clicked on (*EmbClickSim*: ranked 5th overall) and similarity to listings user skipped (*EmbSkipSim*: ranked 8th overall). Five embedding features ranked among the top 20 features. As expected, long-term feature *UserTypeListingTypeSim* which used all past user bookings ranked better than short-term feature *EmbBookSim* which takes into account only bookings from last 2 weeks. This also shows that recommendations based on past bookings are better with embeddings that are trained using historical booking sessions instead of click sessions.

Table 7: Embedding Features Coverage and Importances

Feature Name	Coverage	Feature Importance
EmbClickSim	76.16%	**5/104**
EmbSkipSim	78.64%	**8/104**
EmbLongClickSim	51.05%	20/104
EmbWishlistSim	36.50%	47/104
EmbInqSim	20.61%	12/104
EmbBookSim	8.06%	46/104
EmbLastLongClickSim	48.28%	11/104
UserTypeListingTypeSim	86.11%	22/104

To evaluate if the model learned to use the features as we intended, we plot the partial dependency plots for 3 embedding features: *EmbClickSim*, *EmbSkipSim* and *UserTypeListTypeSim*. These plots show what would happen to listing's ranking score if we fix values of all but a single feature (the one we are examining). On the left subgraph it can be seen that large values of *EmbClickSim*, which convey that listing is similar to the listings user recently click on, lead to a higher model score. The middle subgraph shows that large values of *EmbSkipSim*, which indicate that listing is similar to the listings user skipped, lead to a lower model score. Finally, the right subgraph shows that large values of *UserTypeListingTypeSim*, which indicate that user type is similar to listing type, lead to a higher model score as expected.

Online Experiment Results Summary. We conducted both offline and online experiments (A/B test). First, we compared two search ranking models trained on the same data with and without embedding features. In Table 8 we summarize the results in terms of DCU (Discounted Cumulative Utility) per each utility (impression, click, rejection and booking) and overall NDCU (Normalized Discounted Cumulative Utility). It can be observed that adding embedding features resulted in 2.27% lift in NDCU, where booking DCU increased by 2.58%, meaning that booked listings were ranked higher in the hold-out set, without any hit on rejections (DCU -0.4 was flat), meaning that rejected listings did not rank any higher than in the model without embedding features.

Table 8: Offline Experiment Results

Metrics	Percentage Lift
DCU -0.4 (rejections)	+0.31%
DCU 0.01 (clicks)	+1.48%
DCU 0.25 (contacts)	+1.95%
DCU 1 (bookings)	+2.58%
NDCU	+2.27%

Observations from Table 8, plus the fact that embedding features ranked high in GBDT feature importances (Table 7) and the finding that features behavior matches what we intuitively expected (Figure 7) was enough to make a decision to proceed to an online experiment. In the online experiment we saw a statistically significant booking gain and embedding features were launched to production. Several months later we conducted a back test in which we attempted to remove the embedding features, and it resulted in negative bookings, which was another indicator that the real-time embedding features are effective.

5 CONCLUSION

We proposed a novel method for real-time personalization in Search Ranking at Airbnb. The method learns low-dimensional representations of home listings and users based on contextual co-occurrence in user click and booking sessions. To better leverage available search contexts, we incorporate concepts such as global context and explicit negative signals into the training procedure. We evaluated the proposed method in Similar Listing Recommendations and Search Ranking. After successful test on live search traffic both embedding applications were deployed to production.

ACKNOWLEDGEMENTS

We would like to thank the entire Airbnb Search Ranking Team for their contributions to the project, especially Qing Zhang and Lynn Yang. We would also like to thank Phillippe Siclait and Matt Jones for creating the Embedding Evaluation Tool. The summary of this paper was published in Airbnb's Medium Blog [6].

REFERENCES

[1] Kamelia Aryafar, Devin Guillory, and Liangjie Hong. 2016. An Ensemble-based Approach to Click-Through Rate Prediction for Promoted Listings at Etsy. In *arXiv preprint arXiv:1711.01377*.

[2] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[3] Oren Barkan and Noam Koenigstein. 2016. Item2vec: neural item embedding for collaborative filtering. In *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*. IEEE, 1–6.

[4] Christopher J Burges, Robert Ragno, and Quoc V Le. 2011. Learning to rank with nonsmooth cost functions. In *Advances in NIPS 2007*.

[5] Ting Chen, Liangjie Hong, Yue Shi, and Yizhou Sun. 2017. Joint Text Embedding for Personalized Content-based Recommendation. In *arXiv preprint arXiv:1706.01084*.

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

[7] Nemanja Djuric, Hao Wu, Vladan Radosavljevic, Mihajlo Grbovic, and Narayan Bhamidipati. 2015. Hierarchical neural language models for joint representation of streaming documents and their content. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 248–255.

[8] Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, Ricardo Baeza-Yates, Andrew Feng, Erik Ordentlich, Lee Yang, and Gavin Owens. 2016. Scalable semantic matching of queries to ads in sponsored search advertising. In *SIGIR 2016*. ACM, 375–384.

[9] Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, and Narayan Bhamidipati. 2015. Context-and content-aware embeddings for query rewriting in sponsored search. In *SIGIR 2015*. ACM, 383–392.

[10] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, and Ananth Nagarajan. 2015. Gender and interest targeting for sponsored post advertising at tumblr. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1819–1828.

[11] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[12] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[13] Krishnaram Kenthapadi, Benjamin Le, and Ganesh Venkataraman. 2017. Personalized Job Recommendation System at LinkedIn: Practical Challenges and Lessons Learned. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 346–347.

[14] Maciej Kula. 2015. Metadata embeddings for user and item cold-start recommendations. *arXiv preprint arXiv:1507.08439* (2015).

[15] Benjamin Le. 2017. Deep Learning for Personalized Search and Recommender Systems. In *Slideshare: https://www.slideshare.net/BenjaminLe4/deep-learning-for-personalized-search-and-recommender-systems*.

[16] Steve Liu. 2017. Personalized Recommendations at Tinder: The TinVec Approach. In *Slideshare: https://www.slideshare.net/SessionsEvents/dr-steve-liu-chief-scientist-tinder-at-mlconf-sf-2017*.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[18] Thomas Nedelec, Elena Smirnova, and Flavian Vasile. 2017. Specializing Joint Representations for the task of Product Recommendation. *arXiv preprint arXiv:1706.07625* (2017).

[19] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1933–1942.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 701–710.

[21] Vladan Radosavljevic, Mihajlo Grbovic, Nemanja Djuric, Narayan Bhamidipati, Daneo Zhang, Jack Wang, Jiankai Dang, Haiying Huang, Ananth Nagarajan, and Peiji Chen. 2016. Smartphone app categorization for interest targeting in advertising marketplace. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 93–94.

[22] Sharath Rao. 2017. Learned Embeddings for Search at Instacart. In *Slideshare: https://www.slideshare.net/SharathRao6/learned-embeddings-for-search-and-discovery-at-instacart*.

[23] Thomas Schmitt, François Gonard, Philippe Caillou, and Michèle Sebag. 2017. Language Modelling for Collaborative Filtering: Application to Job Applicant Matching. In *IEEE International Conference on Tools with Artificial Intelligence*.

[24] Yukihiro Tagami, Hayato Kobayashi, Shingo Ono, and Akira Tajima. 2015. Modeling User Activities on the Web using Paragraph Vector. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 125–126.

[25] Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 384–394.

[26] Dongjing Wang, Shuiguang Deng, Xin Zhang, and Guandong Xu. 2016. Learning music embedding with metadata for context aware recommendation. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*.

[27] Jason Weston, Ron J Weiss, and Hector Yee. 2013. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 65–68.

[28] Ledell Wu, Adam Fisch, Sumit Chopra, Keith Adams, Antoine Bordes, and Jason Weston. 2017. StarSpace: Embed All The Things! *arXiv preprint arXiv:1709.03856*.

[29] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. 2016. Ranking relevance in yahoo search. In *Proceedings of the 22nd ACM SIGKDD*.

[6] https://medium.com/airbnb-engineering/listing-embeddings-for-similar-listing-recommendations-and-real-time-personalization-in-search-601172f7603e