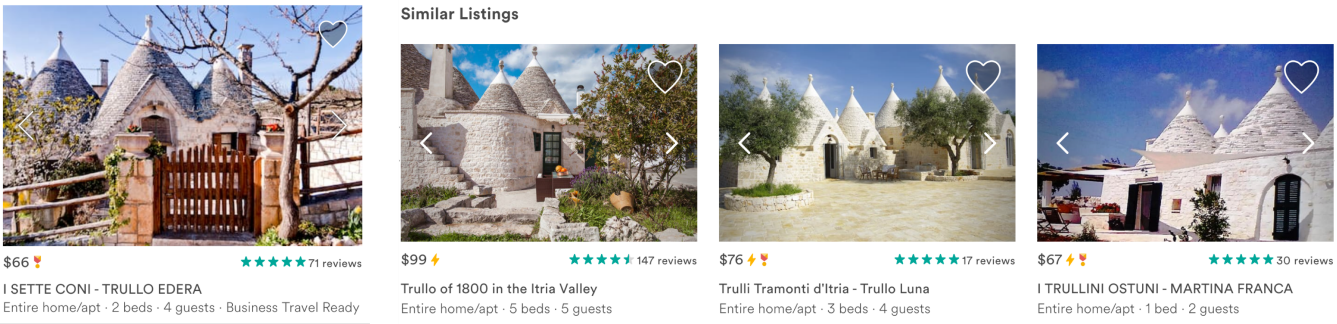
Listing Embeddings in Search Ranking

Listing Embeddings for Similar Listing Recommendations and Real-time Personalization in Search Ranking

Mihajlo Grbovic
Mar 13, 2018 · 11 min read

By Mihajlo Grbovic, Haibin Cheng, Qing Zhang, Lynn Yang, Phillippe Siclait and Matt Jones



Airbnb's marketplace contains millions of diverse listings which potential guests explore through search results generated from a sophisticated Machine Learning model that uses more than hundred signals to decide how to rank a particular listing on the search page. Once a guest views a home they can continue their search by either returning to the results or by browsing the Similar Listing Carousel, where listing recommendations related to the current listing are shown. Together, Search Ranking and Similar Listings drive 99% of our booking conversions.

In this blog post we describe a Listing Embedding technique we developed and deployed at Airbnb for the purpose of improving Similar Listing Recommendations and Real-Time Personalization in Search Ranking. The embeddings are vector representations of Airbnb homes learned from search sessions that allow us to measure similarities between listings. They effectively encode many listing features, such as location, price, listing type, architecture and listing style, all using only 32 float

numbers. We believe that the embedding approach for personalization and recommendation is very powerful and useful for any type of online marketplace on the Web.

Background On Embeddings

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(NLP) applications classic methods for language modeling that represent words as high-dimensional, sparse vectors have been replaced by Neural Language models that learn word embeddings, i.e. low-dimensional representations of words, through the use of neural networks. 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 and Skip-gram language models for word representation learning, the embedding models have been shown to obtain state-of-the-art performance on many traditional language tasks after training on large text data.

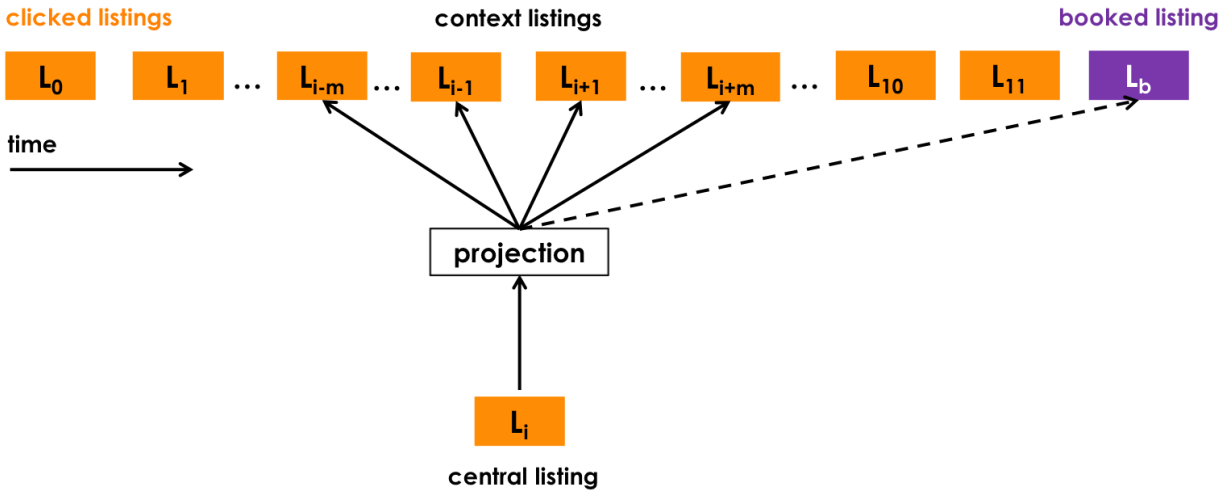
More recently, the concept of embeddings has been extended beyond word representations to other applications outside of NLP domain. Researchers from the Web Search, E-commerce and Marketplace domains have realized that just like one can train word embeddings by treating a sequence of words in a sentence as context, the same can be done for training embeddings of user actions by treating sequence of user actions as context. Examples include learning representations of items that were clicked or purchased or queries and ads that were clicked. These embeddings have subsequently been leveraged for a variety of recommendations on the Web.

Listing Embeddings

Let us assume we are given a data set of click sessions obtained from N users, where each session $s=(L_1,\ldots,L_n)\in S$ is defined as an uninterrupted sequence of n 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 32-dimensional real-valued representation $\mathbf{v}(L_i) \in R^{32}$ of each unique listing Li, such that similar listings lie nearby in the embedding space.

The dimensionality of the listing embeddings was set to $d = 32$, as we found that to be a good trade-off between offline performance (discussed in following section) and the memory needed to store vectors in RAM memory of search machines for the purposes of real-time similarity calculations.

There exist several different ways of training embeddings. We will focus on a technique called Negative Sampling. It starts by initializing the embeddings to random vectors, and proceeds to update them via stochastic gradient descent by reading through search sessions in a sliding window manner. At each step the vector of the ***central listing*** is updated by pushing it closer to vectors of ***positive context listings:*** listings that were clicked by the same user before and after central listing, within a window of length m ($m = 5$), and pushing it away from ***negative context listings:*** randomly sampled listings (as chances are these are not related to the central listing).



For brevity we'll skip over the details of the training procedure and focus on explaining several modifications we made to adapt this approach for our own use-case:

- **Using Booked Listing as Global Context:** We used sessions that end with the user booking the listing (purple listing) to adapt the optimization such that at each step we predict not only the neighboring clicked listings but also the eventually booked listing as well. 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) and is used to update the central listing vector.
- **Adapting to Congregated Search:** Users of online travel booking sites typically search only within a single market, i.e. in the location they want to stay at. As a consequence, for a given *central listing*, the *positive context listings* mostly consist of listings from the same market, while the *negative context listings* mostly consists of listings that are not from the same market as they are sampled randomly from entire listing vocabulary. 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 *Dmn,* sampled from the market of the central listing.

Considering all of the above, the final optimization objective can be formulated as

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

booking global context + market negatives optimization formula

where

- *l* is the *central listing* whose vector **v**(l) is being updated
- *Dp* is a positive set of pairs (*l*,*c*) that represent (*central listing*, *context listing*) tuples whose vectors are being pushed toward one another
- *Dn* is a negative set of pairs (*l*,*c*) that represent (*central listing*, *random listing*) tuples whose vectors are being pushed away from each other
- *lb* is booked listing that is treated as global context and pushed toward central listing vector
- *Dmn* is a market negatives set of pairs (*l*,mn) that represent (*central listing*, *random listing from same market*) tuples whose vectors are being pushed away from each other

Using the described optimization procedure we learned listing embeddings for *4.5 million active listings* on Airbnb using more than *800 million search clicks sessions*, resulting in high quality listing representations.

Cold-start 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 our training data. To create embeddings for a new listing we find 3 geographically closest listings that do have embeddings, and are of same listing type and price range as the new listing, and calculate their mean vector.

What did Embeddings Learn?

To evaluate what characteristics of listings were captured by the embeddings, we examine them in several ways. First, to evaluate if geographical similarity is encoded we performed k-means clustering on learned embeddings. The figure on the left, which shows resulting 100 clusters in California, confirms that listings from similar locations are clustered together. Next, we evaluated average cosine similarities between listings of different types (Entire Home, Private Room, Shared Room) and price ranges and confirmed that cosine similarities between listings of same type and price ranges are much higher compared to similarities between listings of different type 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 these characteristics and to be able to conduct fast and easy explorations in the embedding space we developed an internal Similarity Exploration Tool demonstrated in a video below.

The video provides many examples of embeddings being able to find similar listings of the same unique architecture, including houseboats, treehouses, castles, etc.

Offline Evaluation of Listing Embeddings

Before testing embeddings in recommendation applications on real search traffic, we conducted several offline tests. We also used these tests to compare several different trained embeddings to reach quick decisions regarding embedding dimensionality,

different ideas on algorithm modifications, training data construction, choice of hyperparameters, etc.

One way of evaluating trained embeddings is to test how good they are in recommending listings that the user would book, based on their most recent click.

More specifically, let us assume we are given the most recently clicked listing and the listing candidates that need to be ranked, which contain the listing that user eventually booked. By calculating cosine similarities between embeddings of the clicked listing and the candidate listings we can rank the candidates and observe the rank position of the booked listing.

In the figure below we show results of one such evaluation where listings in search were re-ranked based on similarities in embedding space and rankings of booked listing are averaged for each click leading to booking, going as far back as 17 clicks before the booking to the last click before booking.

We compared several embedding versions **1) d32 regular** trained without any modifications to the original embedding algorithm, **2) d32 booking global** trained with bookings as global context and **3) d32 booking global + market negatives** trained with bookings as global context and explicit negatives from same market (from

our formula). From the consistent lower average ranking of booked listing we can conclude that **d32 booking global + market negatives** outperforms the other two embedding models.

Similar Listings using Embeddings

Every Airbnb home listing page contains a Similar Listings carousel that recommends listings which are similar to it and available for the same set of dates.

At the time of testing embeddings the existing algorithm for Similar Listings consisted of calling our main Search Ranking model for the same location as the given listing followed by filtering on same price range and listing type as the given listing.

We conducted an A/B test where we compared the existing Similar Listings algorithm at the time to our embedding-based solution, in which similar listings were produced by finding k-nearest neighbors in listing embedding space. More precisely, given learned listing embeddings, similar listings for a given listing l are found by calculating cosine similarity between its vector $\mathbf{v}(l)$ and vectors $\mathbf{v}(lj)$ 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=12 listings with the highest similarity were shown as similar listings.

The A/B test showed that embedding-based solution lead to a **21% increase in Similar Listing carousel CTR** and **4.9% more guests discovering the listing they ended up booking** in the Similar Listing carousel.

Real time personalization in Search using Embeddings

So far we've seen that embeddings can be used to efficiently calculate similarities between listings. Our next idea was to leverage this capability in Search Ranking for real-time in-session personalization, where the aim is to show to the guest ***more***

listings that are similar to the ones we think they liked since starting the search session and ***fewer listings similar to the ones we think they did not like.***

To achieve this, for each user we collect and maintain in real-time (using Kafka) two sets of short-term history events:

1. ***Hc*** : set of listing ids that user clicked in last 2 weeks.
2. ***Hs*** : set of listing ids that user skipped in last 2 weeks, where we define skipped listings as ones that were ranked high but skipped by user in favor of a click on a lower positioned listing

Next, each time the user conducts a search we calculate 2 similarity measures for each ***candidate listing*** l_i returned in search.

- ***EmbClickSim*** : similarity between candidate listing embedding and embeddings of listings that user clicked on (from *Hc*).

Specifically, we calculate similarity between market-level centroids from *Hc* and pick the maximum similarity. For example if *Hc* contained listings from New York and Los Angeles, embeddings of listings for each of these two markets would be averaged out to form a market-level centroids.

- ***EmbSkipSim*** : similarity between candidate listing embedding and embeddings of listings that user skipped (from *Hs*)

These two similarity measures were next introduced as additional signal that our Search Ranking Machine Learning model considers when ranking candidate listings.

This was done by first logging the two embedding similarity features along with other search ranking features so we can create a new labeled data set for model training and then proceeding to train a new ranking model that we can be tested against the current production ranking model in a A/B test.

To evaluate if the new model learned to use the embedding similarity features as we intended, we plot their partial dependency plots below. These plots show what would happen to candidate listing ranking score if we fix values of all but a single feature (one we are examining).

In the left subgraph it can be seen that larger values of *EmbClickSim* (listing similar to listings user recently click on) lead to higher model score.

In the right subgraph it can be seen that larger values of *EmbSkipSim* (listing similar to listings user skipped, i.e. did not like) lead to lower model score.

Observations from partial dependency plots confirmed that features behavior matches what we intuitively expected the model will learn. In addition, the new embedding features ranked high in ranking model feature importances and our *offline tests* showed an improvement in performance metrics on a hold-out set when the embedding features were added to the model. This was enough for us to make a decision to proceed to an *online experiment*, which was successful and lead to the launch of embedding features for real-time personalization to production in Summer of 2017.

• • •

Interested in this type of work? We're always looking for talented people to join our Data Science and Analytics team!

Many thanks to the entire Search Ranking team for contributions to the project, *Peng Dai and Luca Luo who helped me launch Similar Listings using Embeddings and Navin Sivanandam who greatly helped with the blog post.*

Thanks to lindsay m pettingill, Robert Chang, and Navin Sivanandam.

Machine Learning Data Science