Nick Dulchin

Foursquare Recommendations from Random Network Walks

Dataset:

I used a dataset containing Foursquare check-ins in New York City between April 3rd, 2012 and February 16th, 2013 Dingqi Yang et al., "Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs.". It contains 227, 428 check-ins at various types of establishments. This includes 1,083 unique users and 38,333 unique venues.

Introduction:

My initial inspiration for this project was to learn about the "DeepWalk" algorithm by using it for some type of machine learning task with some type of network data. I experimented with multiple different datasets but ultimately decided on using the Foursquare dataset because it was large (but not too large for the computing power that I had available) and because I liked that the domain was intuitively interpretable (especially because I am a New Yorker and the data is from NYC check-ins).

I also experimented with multiple types of machine learning tasks including node classification and link prediction before deciding on building a recommendation system. Recommendation systems appeal to me because they are used in many different industries and because most people have a lot of experience interacting with them.

For my project, I used DeepWalk to build a system that recommends venues to users based on their check-ins. In order to test my model against a baseline, I also built three other models. The first is simply a model that recommends randomly and the other two use matrix factorization (with two different loss functions). Matrix factorization is one of the most standard approaches in the field.

Many of the algorithms used are complex and rely on the use of other algorithms that are also complex; so much of this paper will be devoted to the explanation of the intuition and inner workings of those algorithms. I will also discuss recommendation systems generally, the evaluation metrics that I used for my project, and the results that I found. Last, I will talk about future improvements.

Recommendation Systems:

The goal of recommendation systems is to predict the rating or preference that a user would give to an item before they do so. They are used for playlist generation by services like Netflix, YouTube, and Spotify, for ecommerce recommendations by services like Amazon, and for content curation by services like Facebook and Twitter.

A lot of research on recommendation systems was developed in response to the "Netflix Prize" competition held between 2006 and 2009 where Netflix offered $1,000,000 to the team that could best predict users' ratings of specific movies given a training set of about 100,000,000 ratings. In the years since, a few main approaches have become the most popular, but there are many variations depending on the data and the context.

One major trend since 2009 has been the move towards using implicit data rather than explicit data. Explicit data is created when users explicitly rate a product or item (like the 1-5 ratings on Netflix). Implicit ratings are inferred by user activity (like how many times a user played a song or how many times they watched a movie.).

The Foursquare data is implicit data because the amount that a user likes a venue is implied by how many times they check in there. One of the biggest challenges of implicit data is that, unlike explicit data, a zero value doesn't necessarily mean that a user doesn't like a venue, because they might just not have heard of it yet. This makes evaluation a challenge. Fortunately, a lot of work has been done to determine evaluation metrics that are well suited to implicit data as I discuss in the evaluation section.

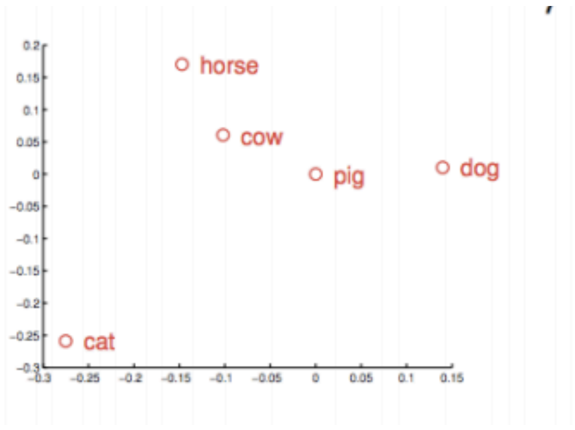
Algorithms:

Word2vec:

The DeepWalk algorithm is basically an abstraction built on top of the "Word2vec" algorithm, so I will start by describing that algorithm. Word2vec was created by Google researchers in 2013 as a natural language modeling tool (Mikolov et al., 2013). The goal of Word2vec is to represent each word from a text in N latent dimensions, where N is less than the total number of words. The latent dimensions are not directly interpretable but they are chosen because they represent a large amount of the information in far fewer dimensions than if all the directly interpretable dimensions were kept (in this case one dimension for each word).

Word2vec achieves this by using a neural network. A text corpus is broken down into sentences and these are fed into the neural network. The weights of the network are updated for each word so that they more accurately predict the words that frequently appear around that word. This part of the Word2vec algorithm is called “SkipGram”. The problem of updating the weights in the neural network is very computationally complex, and so Word2vec uses “negative sampling”, “Hierarchical Softmax”, and a couple other techniques to drastically reduce the computational complexity of the problem without sacrificing much performance. I will not get into how those work but the basic idea is to only adjust a few of the millions of weights on each iteration rather than all of them.

Word2vec then uses the weights for each word as a representation of its semantic meaning. Two words that have similar values across the weights and therefore are “close together” in the vector space (using cosine similarity as a distance measure) are interpreted to be close in semantic meaning. Although the vector space used is often very large, dimensionality reduction techniques can be used to reduce the high dimensional space down to two dimensions where it can be visualized. For example:



In this example, you can see that the semantic meaning of pig is more similar to cow than it is to cat. This is because pig and cow appear close together more often than pig and cat do. Word2vec is useful because the feature vectors that it produces for each word can be used to compare the similarity of words.

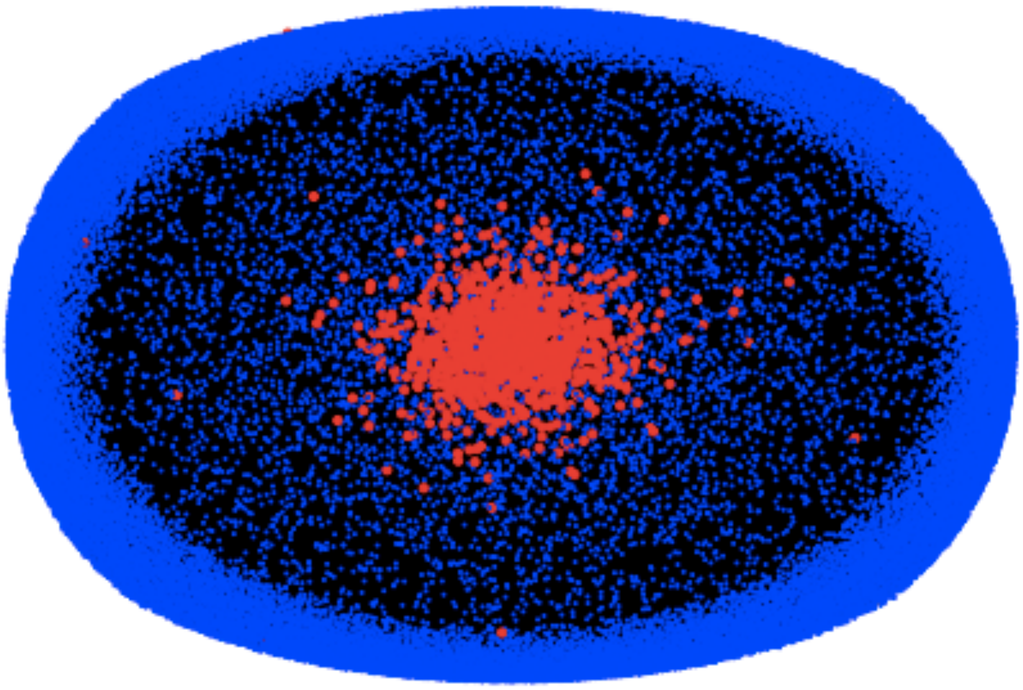
DeepWalk:

Although Word2vec was initially designed for use with text data, there is no reason why it wouldn't work with other types of data as well. The key insight behind DeepWalk is that artificial "sentences" can be created by performing random walks in networks (Perozzi et al., 2014). In the same way that a word is similar to another word if they appear together often, a node in a network is similar to another node if they often appear close together in random walks.

To create the random walks, the algorithm starts at a random node and then randomly selects one of the edges that that node is connected to. It does the same at the next node and continues for a certain number of steps (this value can be adjusted). This process is repeated a certain number of times on each node, and by the end you have a corpus of (# of nodes)*(walks per node) different random walks. That corpus is then treated the same as a text corpus would be. The algorithm updates the weights of the embeddings of each node to better predict the nodes that are often near it in random walks, and the process continues until all of the random walks have been fed into the neural network.

Although the original DeepWalk code only works with uniformly weighted network edges, I found a GitHub repository where a user implemented new functions that allow DeepWalk to be generalized to weighted networks by modifying random walk probabilities based on edge weights (shun1024, *Weighted Random Walk Implementation* *for Deepwalk (in Python)*). In these new walks, an edge with twice the weight of another edge is twice as likely to be the next node in the random walk.

To build my recommendation model for the foursquare data, I started with a heterogeneous weighted network where the nodes were venues and users and the edges represented check-ins for a user at a specific venue (so there were no venue-venue edges or user-user edges, just user-venue edges). The weights of the edges represented the number of times that a user checked in at a venue. Here is a visualization of the network:

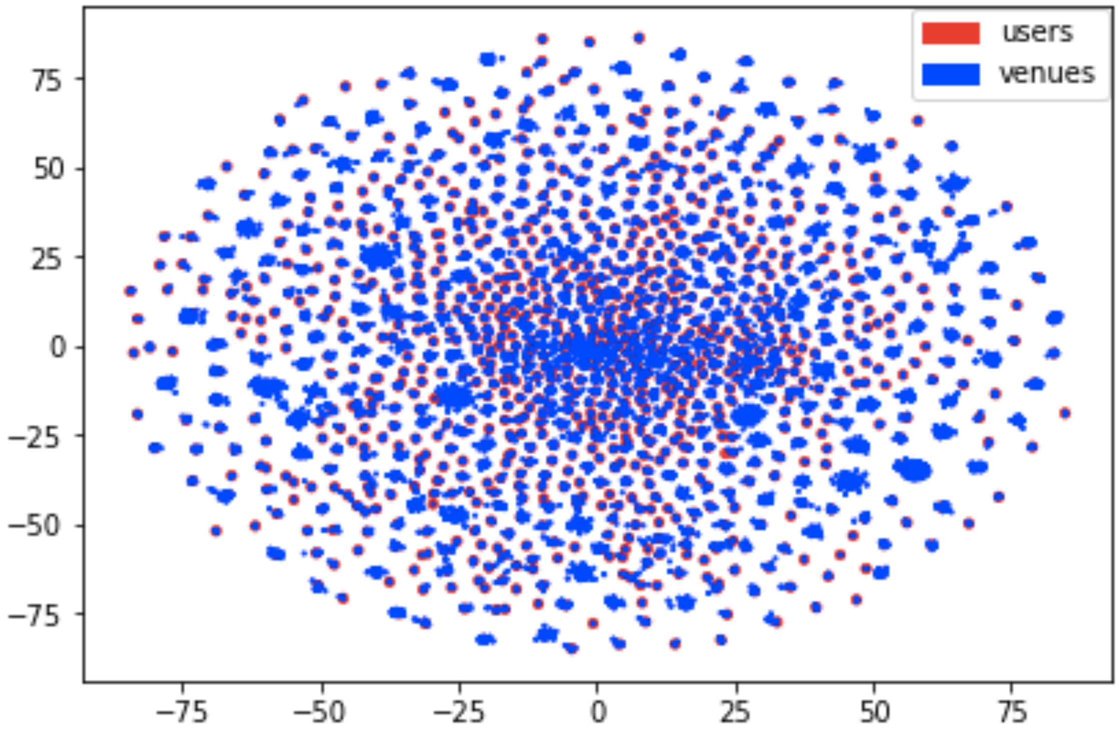


The red nodes represent users and the blue nodes represent venues. The visualization algorithm places nodes with higher degree closer to the middle. Note that because the data was collected by keeping track of all the venues that pre-specified users visited (and not all the users that visited pre-specified venues), the users have much higher degrees and so are usually closer to the middle. The thick ring of venues on the edge of the network represents venues that have a degree of one (meaning that only one of the users in the dataset checked-in at them).

To embed the nodes in this network into latent dimensions, I ran the DeepWalk algorithm with 128 dimensions, 80 random walks per node, walks of length 40, and a window size of 10 (meaning the 10 nodes before and after the target node in a random walk sequence get their probabilities maximized when changing the hidden layer weights in the neural network).

I then had values in all 128 dimensions for every user and venue node, and to determine which venues to recommend I simply found the venues that were most similar to a user across all 128 dimensions. To calculate similarity across dimensions I used cosine similarity. Because I used a heterogeneous network, both venues and users are plotted in the same vector space, so it easy to compare within a category or across categories.

I used a dimensionality reduction technique called "t-Distributed Stochastic Neighbor Embedding" (Maatan & Hinton, 2008) to visualize the network embeddings in 2 dimensions:



The venue recommendations for a given user are based on how close each venue node is to that user node.

Matrix Factorization:

In order to test my DeepWalk model against a baseline, I built two different models using a matrix factorization approach. Matrix factorization is a technique used within the “collaborative filtering” family of recommendation algorithms. Recommendation algorithms are broadly split into two types: content-based models and collaborative filtering models. Content-based models use features that have to do with the direct makeup of an item. For example, in a music recommendation system that would mean using things like genre, song-length, and instruments used. Collaborative filtering, on the other hand, draws its predictive powers from the past behavior of users and from the behavior of similar users.

Collaborative filtering models represent all of the users and all of the items as a massive matrix, and attempt to fill in missing values by determining which users are the most similar (based on the items that they have both rated) and then using the ratings of similar users to fill in missing values. The matrix factorization technique is a way of making these calculations much less complex by representing each user and item in k latent dimensions. If the original matrix is M*N, it is instead represented by two matrices of shapes M*k and k*N respectively, which, when multiplied together equal the original M*N matrix. There are various ways to calculate these latent dimension values and different techniques make different assumptions and are used for different things. In this project I made two different models that represent two different ways of calculating the latent dimension values. Those are the Alternating-Least-Squares approach and the Bayesian Personalized Rankings approach. For all of the matrix factorization models, I used the implementations from a Python library called "Implicit", which are based on papers by Hu et al. (2008) and Rendle et al. (2012).

Alternating-Least-Squares:

The alternating-least-squares algorithm estimates the latent dimension values in the matrix factorization by iteratively holding the user matrix constant while optimizing the item matrix, and then holding the item matrix constant while optimizing the user matrix. The optimization taking place is a minimization of the RMSE between the values predicted by multiplying the latent dimension matrices together and the actual values observed in the original matrix. Once the RMSE drops below a certain threshold, the iterations stop.

Bayesian Personalized Ranking:

The Bayesian Personalized Ranking algorithm generates the matrix factorization matrices in a way that is directly optimized for ranking. While ALS optimizes on an instance level, BPR optimizes on a pair level. It seeks to maximize the posterior probability of the correctly item in item pairs being ranked higher. This is a very computationally complex procedure so instead of going through every pair, the algorithm uses stochastic gradient descent with bootstrap sampling, meaning that it optimizes only a small fraction (which turns out to be good enough for convergence).

Evaluation:

Because these recommendation systems work with implicit data, they are much more complicated to evaluate. Unlike with explicit data where we could determine that a user didn’t like something by seeing that they gave it a low rating, the implicitness of the data means that we have to infer that a user didn’t like a venue by seeing that they never went there. Unfortunately, this is not reliable feedback because there are multiple reasons why a user might have not went to a certain venue (like not knowing about it rather than actively disliking it). For this reason, precision-based metrics are not relevant. We cannot know which venues a user liked, so we must focus on recall-based metrics, which only look at how a model performs on predicting venues that users do visit. For this project I used two recall-based metrics: Expected Percentile Ranking (EPR), and Mean Reciprocal Ranking (MRR).

In training the models I only used 80% of the dataset so that I would have 20% left to use for testing and evaluation. I made the 80-20 split chronological meaning that the test set is made up of the last 20% of the check-ins that were observed.

EPR was used by Hu et. al (2008) and it is formulated as follows:

$$\overline{rank} = \frac{\sum_{u,i} rank_{ui}}{total\ \#\ of\ venues}$$

For every user, all the venues in the dataset are ranked from most recommended to least recommended. All the venues that the user visited in the test set are then checked against the list. If a user visited a venue that was ranked 1st for them by the model, the percentile rank is 0. If they visit the venue that was ranked last for them, the percentile rank is 1. The EPR is calculated by taking an average of all the percentile rankings of all the check-ins in the test set for each user. If a user checks in at a venue that was not in the test set, the observation is ignored.

MRR is a similar metric, but it gives more weight to really good rankings. MRR formulated is as follows:

$$\overline{reciprocal\ rank} = \frac{1}{total\ \#\ of\ venues} \sum{}_{u,i} \frac{1}{rank_{u,i}}$$

I am using two metrics for variety and because MRR adds a slightly different interpretation. While the value of a 1st ranked and a 2nd ranked item in EPR are very close, they are not close with the MRR metrics. The 1st ranked value would be worth 1 and the 2nd would be worth ½. This weighs the highest scores very highly and gives less importance to how low a ranking is once it is out of the very top. This is useful and relevant because in practice you would only really show a user their top rankings so we only care about how good those are.

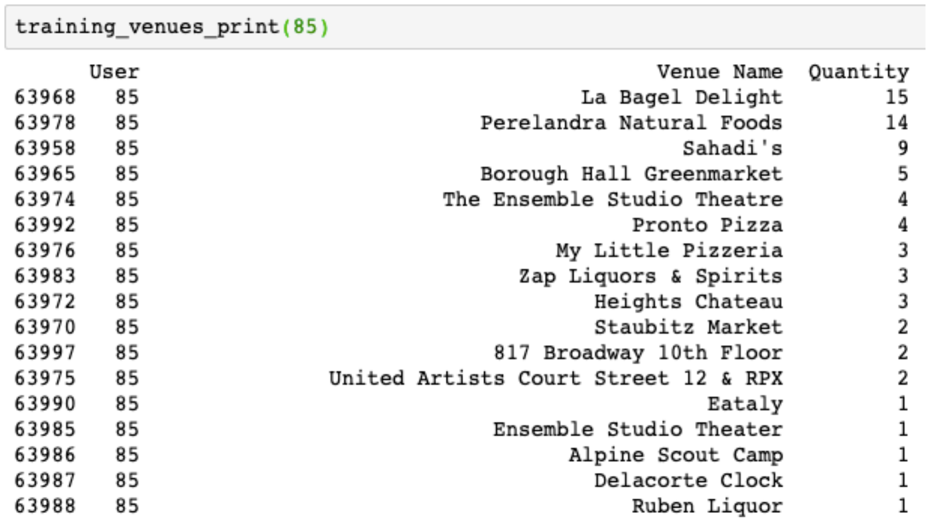
I should note that all offline rankings of recommendation systems are measuring something slightly different than what the real goal is of recommendation systems. What we are measuring with these metrics is essentially how well we can predict which venues people are going to visit, but the real goal of recommendations is actually to predict how much users will enjoy a venue if they did visit it (or at least how much they will appreciate the recommendation). To get closer to answering that question, it would be optimal to do online A/B testing where a subset of the users were given recommendations from one systems and another subset were given recommendations from another system. The recommendation qualities could be measured by keeping track of how often users checked in at recommended venues.

Unfortunately, I obviously don't have the capabilities to run online A/B tests for this project, so will have to stick to offline recall-based metrics.

Results:

Example recommendations (for user 85):

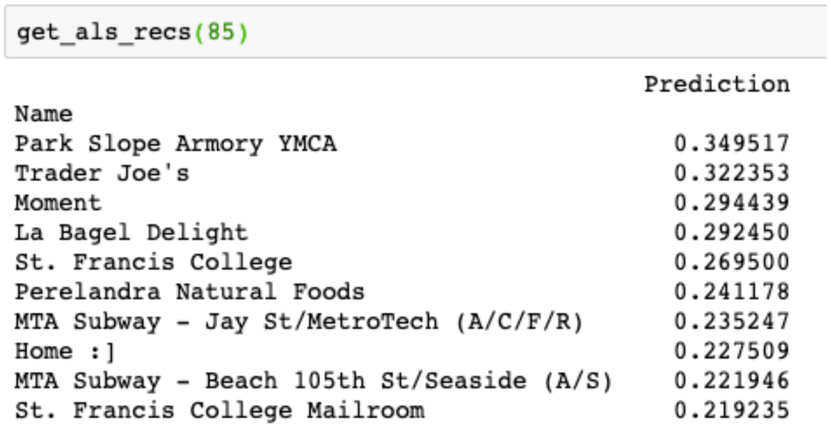
Visited venues:



```
training_venues_print(85)
```

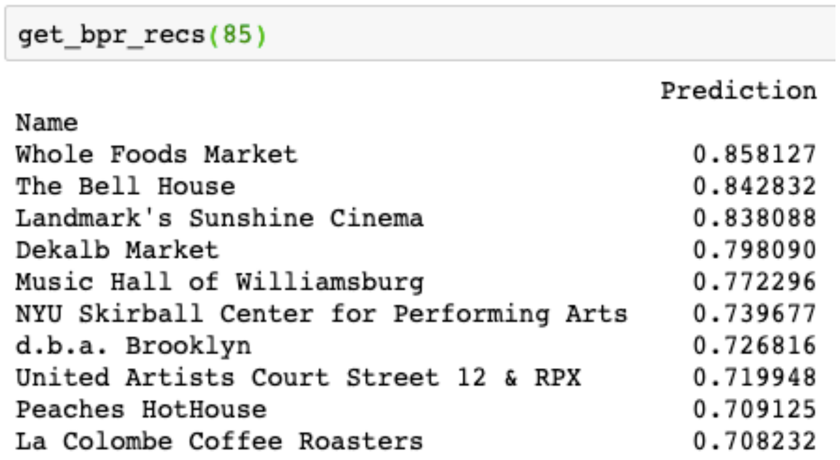
```
      User                                     Venue Name  Quantity
63968   85                                La Bagel Delight        15
63978   85                         Perelandra Natural Foods        14
63958   85                                        Sahadi's         9
63965   85                         Borough Hall Greenmarket         5
63974   85                      The Ensemble Studio Theatre         4
63992   85                                     Pronto Pizza         4
63976   85                               My Little Pizzeria         3
63983   85                            Zap Liquors & Spirits         3
63972   85                                  Heights Chateau         3
63970   85                                  Staubitz Market         2
63997   85                           817 Broadway 10th Floor         2
63975   85             United Artists Court Street 12 & RPX         2
63990   85                                           Eataly         1
63985   85                          Ensemble Studio Theater         1
63986   85                                Alpine Scout Camp         1
63987   85                                   Delacorte Clock         1
63988   85                                     Ruben Liquor         1
```

Recommendations:



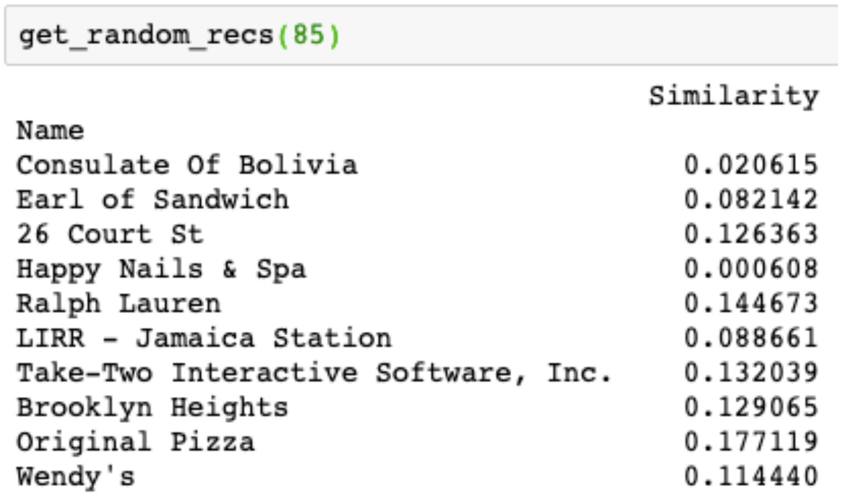
```
get_als_recs(85)
```

```
                                           Prediction
Name
Park Slope Armory YMCA                       0.349517
Trader Joe's                                 0.322353
Moment                                       0.294439
La Bagel Delight                             0.292450
St. Francis College                          0.269500
Perelandra Natural Foods                     0.241178
MTA Subway - Jay St/MetroTech (A/C/F/R)      0.235247
Home :]                                      0.227509
MTA Subway - Beach 105th St/Seaside (A/S)    0.221946
St. Francis College Mailroom                 0.219235
```



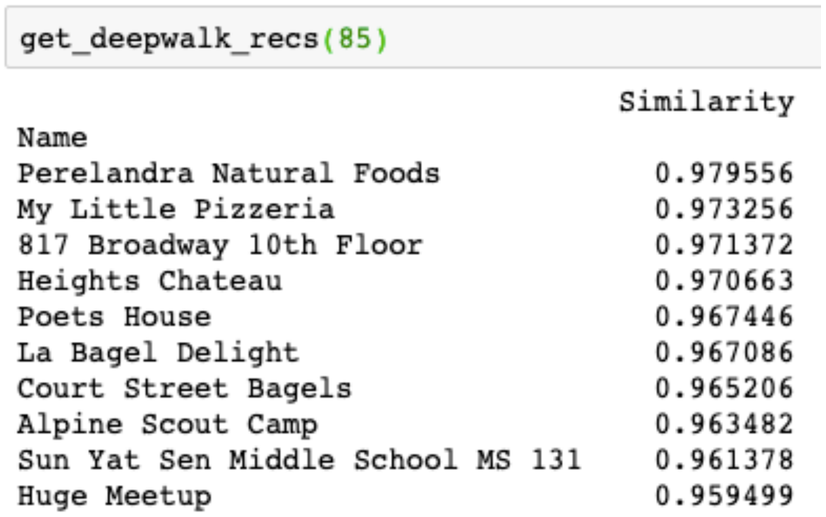
```
get_bpr_recs(85)
```

```
                                           Prediction
Name
Whole Foods Market                           0.858127
The Bell House                               0.842832
Landmark's Sunshine Cinema                   0.838088
Dekalb Market                                0.798090
Music Hall of Williamsburg                   0.772296
NYU Skirball Center for Performing Arts      0.739677
d.b.a. Brooklyn                              0.726816
United Artists Court Street 12 & RPX         0.719948
Peaches HotHouse                             0.709125
La Colombe Coffee Roasters                   0.708232
```



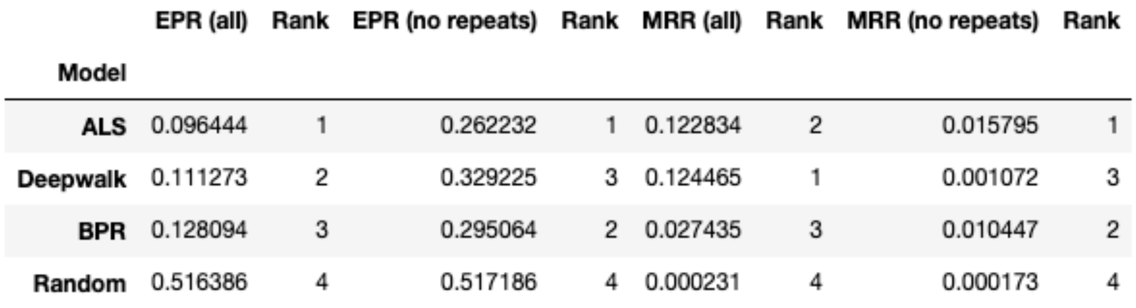
```
get_random_recs(85)
```

```
                                        Similarity
Name
Consulate Of Bolivia                      0.020615
Earl of Sandwich                          0.082142
26 Court St                               0.126363
Happy Nails & Spa                         0.000608
Ralph Lauren                              0.144673
LIRR - Jamaica Station                    0.088661
Take-Two Interactive Software, Inc.       0.132039
Brooklyn Heights                          0.129065
Original Pizza                            0.177119
Wendy's                                   0.114440
```



```
get_deepwalk_recs(85)
```

```
                                    Similarity
Name
Perelandra Natural Foods              0.979556
My Little Pizzeria                    0.973256
817 Broadway 10th Floor               0.971372
Heights Chateau                       0.970663
Poets House                           0.967446
La Bagel Delight                      0.967086
Court Street Bagels                   0.965206
Alpine Scout Camp                     0.963482
Sun Yat Sen Middle School MS 131      0.961378
Huge Meetup                           0.959499
```



Model	EPR (all)	Rank	EPR (no repeats)	Rank	MRR (all)	Rank	MRR (no repeats)	Rank
ALS	0.096444	1	0.262232	1	0.122834	2	0.015795	1
Deepwalk	0.111273	2	0.329225	3	0.124465	1	0.001072	3
BPR	0.128094	3	0.295064	2	0.027435	3	0.010447	2
Random	0.516386	4	0.517186	4	0.000231	4	0.000173	4

I first ran the evaluations by looking at the predictions of all venues, but then I realized that the models may have different competencies in predicting venues that were already visited and that because these are easy predictions it might make sense not to include them in the evaluation.

For both metrics and for all venues and only new venues, it is clear that random is much worse than all three models. For EPR, the random model scores close to the .500 that would be expected theoretically, and for MRR it is orders of magnitude lower than the other scores. To clarify, for EPR low scores are better and for MRR high scores are better.

The DeepWalk model performs relatively comparably to the matrix factorization models (better than both for MRR with all venues), but it clearly suffered more than the others when I took away the recommendations of venues that a user already visited. Overall, the ALS model seems to perform the best according to the metrics I used

Next Steps:

. One thing that I did not look into was how well a recommendation system that simply recommends the most popular overall venues would do. It is possible that this system would score high on these metrics even though it probably wouldn't be that useful to individuals who are interested in discovering new venues via the recommendation system. It is possible that the matrix factorization models are less specialized in this way and are more likely to recommend generically popular venues than DeepWalk. Unfortunately, that is just speculation as I did not include a metric to measure that. If I were to continue working on this project I would attempt to measure this "specificity" or "novelty" characteristic in some way.

Another big limitation in this project was computing power. The DeepWalk algorithm took about 8 hours to run on my computer, so it was not practical to experiment with hyperparameters. If I had had access to more computing power, I would have varied the dimensionality of the output embeddings, the number of walks per node, the length of the walks, and the window size used. There are also a number of hyperparemeter settings involved in the matrix factorization models, and I did not have the time or power to experiment with those either.

Conclusion:

Overall, my analysis shows that the DeepWalk model is significantly better than random and competitive with the matrix factorization models. Judging by the MRR metric when already visited venues were allowed to be recommended, the DeepWalk model beat the matrix factorization models. The results make it clear, however, that DeepWalk fares worse than the other models when already visited venues cannot be recommended.

As noted, my evaluation metric is imperfect and to get a more accurate and complete analysis of it, it would be best to do online A/B testing. There is more research to be done in using DeepWalk for recommendation systems and it is possible that results would be different in different domains.

Bibliography:

Dingqi Yang, Daqing Zhang, Vincent W. Zheng, and Zhiyong Yu. “Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs.” *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45, no. 1 (January 2015): 129–42. https://doi.org/10.1109/TSMC.2014.2327053.

Hu, Yifan, Yehuda Koren, and Chris Volinsky. “Collaborative Filtering for Implicit Feedback Datasets.” In *2008 Eighth IEEE International Conference on Data Mining*, 263–72. Pisa, Italy: IEEE, 2008. https://doi.org/10.1109/ICDM.2008.22.

Karam, Ramzi. “Using Word2vec for Music Recommendations.” Towards Data Science, December 7, 2017. https://towardsdatascience.com/using-word2vec-for-music-recommendations-bb9649ac2484.

Maaten, Laurens van der, and Geoffrey Hinton. “Visualizing Data Using T-SNE.” *Journal of Machine Learning Research* 9, no. Nov (2008): 2579–2605.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed Representations of Words and Phrases and Their Compositionality.” In *Advances in Neural Information Processing Systems 26*, edited by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, 3111–3119. Curran Associates, Inc., 2013. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Ozsoy, Makbule Gulcin. “From Word Embeddings to Item Recommendation,” January 7, 2016. https://arxiv.org/abs/1601.01356v3.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. “DeepWalk: Online Learning of Social Representations.” *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’14*, 2014, 701–10. https://doi.org/10.1145/2623330.2623732.

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. “BPR: Bayesian Personalized Ranking from Implicit Feedback.” *ArXiv:1205.2618 [Cs, Stat]*, May 9, 2012. http://arxiv.org/abs/1205.2618.

shun1024. *Weighted Random Walk Implementation for Deepwalk (in Python): Shun1024/ Weighted-Deepwalk*. Python, 2019. https://github.com/shun1024/weighted-

deepwalk.

Note: I did all the coding in Python. If you would like to see the code let me know.