vec2rec: Network Embedding for Item-to-Item Recommendation

Xiaowen Lin
Stanford University
veralin@stanford.edu

Zijian Wang
Stanford University
zijwang@stanford.edu

Bosen Ding
Stanford University
bosend@stanford.edu

Abstract

Recommendation systems are ubiquitous in our life. Many of these recommenders use item-based approaches to predict the "rating" or "preference" a user would give to an item [10]. However, such systems normally use only a part of the available data and/or features. In this report, we study a network embedding approach based on node2vec [8] to incorporate the richer available information and further improve recommendations. Our investigations show that semantic information could be captured from such embedding, for example, `vec(''McDonald's'') - vec(''KFC'') + vec(''Starbucks'')` is close to `vec(''Tim Hortons'')`. These embedded information will enable downstreaming tasks that utilize network embeddings for recommendation systems.

1 Introduction

Recommendation systems are software tools and techniques providing suggestions for items to be of use to a user [10]. It has been proved to be useful for addressing a portion of the information overload phenomenon [2]. One of the most widely-used method is item-based suggestions. One advantage is that it often yields higher click-through rates than those hybrid approaches (e.g., user-item based) [1]. For example, in e-commerce sites, item-based suggestions like frequently bought together have achieved great success. Another advantage is that it does not have to consider a large number of users. The item network makes the computation more efficient and avoids the cold-start problem for new users.

Even though these traditional algorithms performed well in application, they only use part of the available information. With access to significantly increased amount of computational resources, we propose to study a network embedding approach based on node2vec [8] to incorporate the richer information and further improve recommendations. Inspired by word2vec [5], which successfully encodes semantic information in word embedding, we experiment with item embedding from node2vec and show that they can capture the rich contextual information in the network. For example, based on Yelp Business dataset in Toronto, we found that `vec(''McDonald's'') - vec(''KFC'') + vec(''Starbucks'')` is close to `vec(''Tim Hortons'')` where "Tim Hortons" is the largest coffee chain in Canada. The results show that learning good vector representations for network nodes are possible and they could have good application in context-aware recommendations.

2 Related Work

The key method that will be applied in our project is node2vec [3]. Node2vec learns low-dimensional representations for nodes in a graph by optimizing a neighborhood preserving objective. This method of network representation, together with other methods like DeepWalk, was inspired by word2vec [5]. In this section, we will 1) introduce the background of word2vec 2) discuss popular methods for feature representations in the network, e.g., DeepWalk [8] and node2vec 3) explore recent work on link prediction utilizing node2vec [1] 4) outline their applications in our project.

2.1 Word2vec [5]

Word2vec, firstly proposed by Mikolov et al. [5], is one of the most popular word embedding models in the world. It provides an efficient way to learn word embeddings from a large corpus, which utilizes a shallow two-layer neural networks that are trained fast to reconstruct linguistic contexts of words. In detail, it maps distinct words to a vector space, specifically, from one-hot encoding to a low dimension vector space. One of the major benefits of this is the semantic similarity between words could be learned and preserved, e.g., king - man + woman = queen. Inspired by this semantic similarity between words, we are curious about if it is possible for our item/ node embedding to achieve similar effect. For example, if the product embedding can preserve information like brands or categories.

There are a lot of in-depth discussions of Word2Vec. Here, we only highlight a few important technical designs. There are two architectures proposed by Mikolov et al. [5] in the paper - continuous bag-of-words (CBOW), which uses surrounding words to predict the target word, and skip-gram, which uses the centre word to predict the surrounding words [4]. Among these two, CBOW leads to a faster training time with a worse performance on less-frequent words. Besides architecture designs, there are also two optimization tricks, hierarchical softmax and negative sampling, that contributes the efficiency of training significantly. Hierarchical softmax replaces the sparse giant softmax layer to a huffman tree, which improves the time complexity from $O(n)$ to $O(logn)$. However, this method was not used to train the final word2vec model because evaluating less frequent words in Huffman tree still takes a long time. Instead, the authors proposed negative sampling, which updates only the weights of the positive example plus a few sampled negative examples that simulates the process of doing softmax on all the words to reduce computations. It has been tested that this method enables even faster training without losing performance.

One of the biggest tasks people are facing when using word2vec is the out-of-vocabulary (OOV) problem, namely, it does not generate well for unknown words. However, this will not be a huge issue for our specific task on network-based recommendations.

2.2 DeepWalk [8] and Node2vec [3]

Similar to the quintessence of word2vec, Perozzi et al. [8] proposed DeepWalk, a novel approach for learning latent representations of vertices in a network. It uses random walk to generate sequence data and treats nodes as “words” and generates network embeddings with skip-gram and hierarchical softmax. Empirically, DeepWalk is scalable and significantly outperforms other previous methods designed to operate for sparsity (e.g., spectral clustering [7], edge cluster [9], modularity [6]).

However, DeepWalk suffers from a few problems. On the one hand, its random walk strategy is not efficient. On the other hand, DeepWalk uses hierarchical softmax, which has been experimented to be inefficient when compared with some other training trick, i.e., negative sampling, introduced in word2vec.

In order to solve the problems, Grover and Leskovec [3] proposed node2vec. Node2vec uses negative sampling, where an equal number of node pairs from the network which have no edge connecting them were sampled. Further, it uses a well-defined random walk strategy. Specifically, it defines two bias parameters (return the parameter p controls the likelihood of immediately revisiting a node in a walk; in-out parameter q allows the search to differentiate between "inward" and "outward" nodes) [3]. Those two parameters work together to allow us to combine BFS and DFS, which improves the space and time complexity significantly.

Practically, as shown by Grover and Leskovec [3], node2vec outperforms most of previous work in this area for the task of link prediction. Testing out on three different datasets (Facebook, Protein-Protein Interactions/PPI, and arXiv ASTRO-PH), node2vec achieves the best performance in all of the datasets.

Though node2vec has achieved great performances, there are a few things that remain unsolved. On the one hand, in the paper the authors use grid search to tune the hyperparameters p and q. It would be helpful to explore how to adjust these parameters using grid search with domain-specific knowledge giving a new dataset. On the other hand, the datasets being tested all use unweighted graphs [3]. This is fine for some tasks like friendship prediction, however, it loses important information for tasks that have a strong relationship to weights. In our task of recommendation system, we need to take into account the user's ratings toward the items. Two items may have significantly different degree of similarity if a user rate them both very high or if the user rate one of them five stars and the other one star. An unweighted graph will lose such information so finding an effective weighting function is one of the key tasks.

2.3 Item2vec [1]

Traditional recommendation systems usually use collaborative filtering to perform user-user, user-item, and item-item recommendations [11; 12; 14]. Not until recently, network based recommendation has been explored. Although there are some literature that uses node2vec to perform user-item recommendation tasks (e.g. entity2vec [13], there are few studies on the item-based recommendation system with node2vec embedding.

Unlike many other recommendation systems that recommend items based on users' previous experience, Barkan and Koenigstein [1] present an item-based recommendation algorithm that is inspired by word2vec [5]. Item2vec treats a basket of items bought by a single user as a sequence of words in the sense of word2vec. This method then takes each pair of items in the same set as a positive example and applies skip-gram on such a set of items. The method is evaluated on a music dataset for genre classification with SVD as a single baseline model.

In this paper, the author uses the length of each user history as the window size, thus technically building the "context window" around each user. Another approach of generating the context window could be to build the context window based on the shopping history of the whole user group. Equivalently, two items are in the context of each other if each user has bought two items. In all, the paper presents a first step of adopting network embedding techniques on item-based recommendations.

3 Methods

3.1 Overview

Node2vec outperforms most of the previous work in tasks like link predictions. Word2vec [5] has shown to have state-of-the-art performances on various natural language processing tasks. Naturally, it leads us to think about their applications in the recommendation system.

The user-item based recommendation method suffers from the problem of encoding two different types of data into the same embedding domain. It makes comparing the similarity not well defined, as it is hard to compare similarity between user and item, which makes these solutions lose one of the biggest advantages of embeddings. To solve this problem, inspired by word2vec's application in finding words in similar context, we propose a novel approach to apply node2vec in the item-based recommendation task by finding similar items using item embeddings.

We propose to project the user-item network to an weighted item-item network. To reduce computational complexity, we perform graph pruning in the projected graph to reduce large cliques. Then, we apply node2vec on the pruned projected weighted network to retrieve item embeddings and use the embeddings for item recommendations. We further experiment with the linear transformations of the item embeddings to see if the embeddings are able to capture the semantic context of the items.

3.2 Data Collection

We propose to use two public datasets: Amazon product data [1] and Yelp open dataset [2]. Since the Yelp dataset has much richer information, we will use the Amazon dataset for the initial experiments and qualitative evaluation, and use the Yelp dataset for quantitative evaluation and further exploratory analysis.

The two datasets contain data from very different domains: product reviews from the largest Internet retailer in the world and online crowd-sourced restaurant reviews. The Yelp dataset has the richest information including reviews, check-ins, and even user friendship information. We are interested in exploring the performance of the proposed method on the three datasets to see what type of data it works best with.

Due to the large size of the datasets, we choose to experiment with a representative subset of the data. For the Amazon product data, we experiment with a subcategory of the products: Home and Kitchen. For the Yelp dataset, we choose to limit the city to Toronto, a city with the third most number of business in this dataset. Table 1 summarizes some basic information about these datasets.

Dataset	#Users	#Items	#Reviews
Amazon	20,980,000	9,350,000	82,830,000
Amazon: Home and Kitchen	644,510	79,007	24,318,430
Yelp	1,518,169	188,593	5,996,996
Yelp: Toronto	103,262	18,233	474,803

Table 1: Dataset Statistics

[1] http://jmcauley.ucsd.edu/data/amazon
[2] https://www.yelp.com/dataset

3.3 Graph Construction, Projection and Weighting Algorithms

We construct a bipartite graph G_b^A [3] using the Amazon home and kitchen dataset. A node represents a user or a product (item). Each of the user ids and product ids are mapped to a unique node id. An edge only exists between a user and a product, representing the user reviewed the product. Figure 1 summarizes the degree distribution of G_b^A. We can see that the degrees of user and product node both follow the power law.

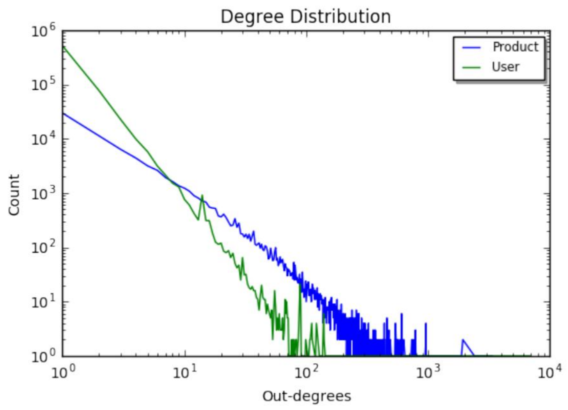


Figure 1: Log-log Degree Distribution of G_b

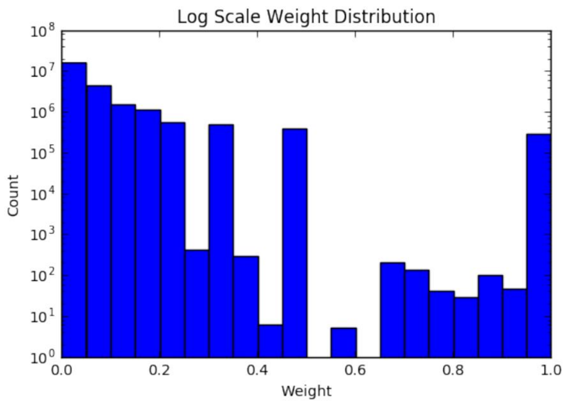


Figure 2: Log Edge Weight Distribution of G_p

We project the user-item bipartite graph G_b^A to a weighted one-mode item-item graph G_p^A [4]. In the projected graph, every node presents a unique product. We define the weight of the edge as

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

where w_{ij} is the weight of an edge between node i and j in G_p^A, $N(i)$ and $N(j)$ are the neighbours of node i and j in G_b^A respectively. The formula is based on the Jaccard Index.

One advantage of our proposed method is that by simply changing the weighting function, the learned embedding can preserve different contextual information. From Figure 2 we can see that there are a large number of edges with high weights. This is likely because if there is one and only one user who bought two unpopular items, the edge between the two products will have a weight of 1. This may be an issue for node2vec as the weight is skewed, we will discuss possible ways to improve it in Section 5.

The graph G_b^Y [5] for Toronto Yelp dataset is contructed in a similar manner, where the node represents a business or a user and the edge represents the review for a business from the user. The projected graph G_p^Y is projected in the same way.

3.4 Graph Pruning

For graph pruning, we use the Amazon dataset as an example, and we apply same techniques for the Yelp dataset. The projected graph G_p^A contains only 58,769 nodes but 24,318,430 edges. Projected networks tend to contain many cliques. However, computational complexity

[3] b denotes bipartite graphs, and A denotes the Amazon dataset.
[4] p denotes projected graphs.
[5] Y denotes the Yelp dataset.

of node2vec depends on the size and number of large cliques. Hence, we propose two graph pruning algorithms to reduce the computational complexity.

3.4.1 Simple Pruning

A simple way to prune the graph is to remove all the edges that have weight less than a threshold value. If a node results in having no edge, we remove the node. From Figure 2 we can see that the majority of the edges have a relatively small weight. Setting the threshold to 0.15, the pruned graph G^A_{pr1} contains 29,682 nodes and 2,863,956 edges.

3.4.2 Node-based Pruning

Although the simple pruning method is efficient and easy to implement, it has several issues. First, since some of the nodes have no edge and are hence removed, we cannot learn the embedding for all nodes. Second, if a popular product has a lot of reviewers, it is unlikely that a large percentage of those reviewers all review another particular product, so the above method will naturally filter out some popular items with a large amount of reviews.

To address the issues, we design a node-based pruning method. In this method, for each node, we sort all the edges by weight, and only retain top N edges with the largest weight. The intuition behind it is that a recommendation engine usually only cares about top N recommendations. Setting the threshold to 200, the pruned graph G^A_{pr2} contains 58,005 nodes and 2,089,334 edges.

3.4.3 Smooth Pruning

A drawback for the node-based pruning is that any node with more than the threshold value of edges will have the same number of edges in the pruned graph. Intuitively, we want the node with larger degrees in the original graph to retain more edges in the pruned graph. We design a smooth pruning algorithm where we keep the top $f(D,T)$ edges with the largest weights for each node, where D is the degree of the node and T is the threshold. We define

$$f(D,T) = \begin{cases} D, & \text{if } D < T, \\ \sqrt{TD}, & \text{if } D \geq T \end{cases}$$

The function $f(D,T)$ is monotonically increasing and sub-modular. Hence, this pruning algorithm results in nice smooth pruned graph. Setting the threshold to 30, the pruned graph G^A_{pr3} contains 58,096 nodes 2,601,904 edges.

3.4.4 Comparison

Figure 3 and Figure 4 compare the degree distribution and weight distribution of the resulting graph pruned by the above methods. As discussed above, the smooth pruning method results in a much smoother graph and retains the relative degree ordering.

3.5 Node2vec

We run node2vec on the pruned graphs G^A_{pr2} and G^A_{pr3} to obtain their item embeddings. Since we are more interested in the macroscopic view of the network neighbourhoods, we use the parameters $p = 1$ and $q = 0.5$ in node2vec. We leave the dimension size to its default parameter $d = 128$. The results will be discussed in the following sections.

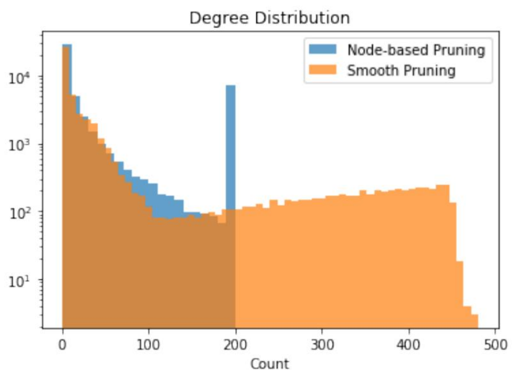


Figure 3: Degree Distribution of Two Pruning

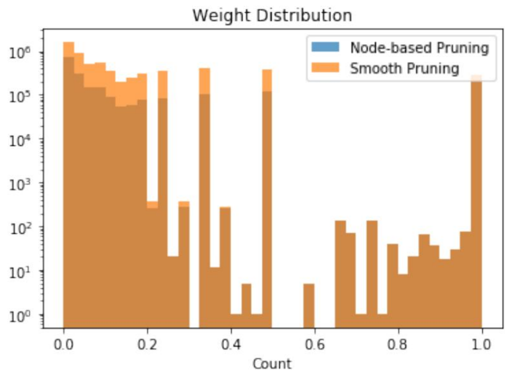


Figure 4: Edge Weight Distribution of of Two Pruning

4 Results and Findings

4.1 Qualitative Evaluation

We conduct qualitative evaluation on the Amazon dataset by analyzing the results on a few random samples. We use the embeddings to find top N *similar* items for a given item. Here, *similar* does not necessarily mean the two items are similar in terms of item type such as movie, book and electronics; it can mean that they are often bought together or that a user who likes one item may also like another. We use cosine similarity to evaluate how similar two items are.

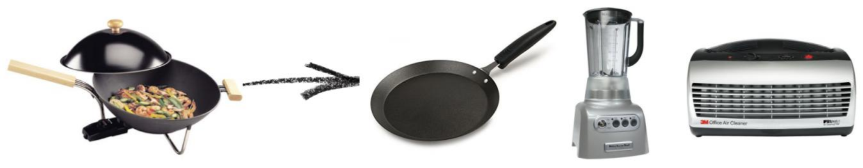


Figure 5: Top 3 Similar Embedding to an Electric Wok

We sampled around 10 items and looked up their top N similar items to get a sense of how well the embedding capture the context information. They are generally reasonable but there are a few exceptions. For example, Figure 5 presents three items that have most similar embeddings to an item "Maxim EW70 Professional 6-1/2-Quart Electric Wok". We can see that the top two items are quite relevant while the third one is not. Given that the node2vec iteration parameter was set to only 10 for this experiment, the result is fairly good.

4.2 Quantitative Evaluation

Quantitative evaluation is tough because there is no well-defined downstream task yet. Therefore, we investigate if the embeddings are semantically meaningful by defining five heuristic metrics for the Yelp dataset. Based on the available data, we choose five important factors of choosing a business: price range, star rating, category, location, and popularity.

Due to the limitation of the computational resources, we sampled 1,000 business out of the 18,233 business and use them to test the six different models. For each business b in the sample, we find top 5 closest embeddings to b and calculate the following metrics:

1. Price Range: the Mean Square Error of the price range.
2. Star: the Mean Square Error of the star rating.
3. #Common Cat: percentage of predictions that have at least one common category
4. Distance: the average distance between the two business in km.
5. #Review: the average difference of the logarithmic review number.

We evaluate the following models:

1. Rnd. Node: a null model that takes 5 random nodes as predictions.
2. Rnd. Neighbour: a smarter null model that takes 5 random neighbours as predictions.
3. Jaccard Index: a model that takes 5 neighbours with the highest Jaccard similarity.
4. n2v-r: our node2vec embedding prediction trained with iteration $r = 10, 100, 1000$.

Notice that model 2 and 3 have one severe drawback - if the degree of the node is less than 5, it will not provide 5 distinct predictions - while the node2vec models do not suffer from this problem. Table 2 presents the results of the evaluation. All of the node2vec models were trained with $p = 1, q = 0.5, w = 80$ and using the smooth pruning with a threshold of 500 on the Toronto Yelp dataset. We can see that our node2vec embeddings are on par with the best results, which means the embeddings are able to capture some semantic information.

Model	**Metrics**				
	Price Range	Star	#Common Cat	Distance	#Review
Rnd. Node	0.939	1.729	0.263	7.854	1.292
Rnd. Neighbor	**0.825**	**1.235**	0.380	5.846	1.876
Jaccard Index	0.860	**1.415**	**0.399**	**4.870**	0.933
n2v-10	0.900	1.823	0.336	7.024	**0.782**
n2v-100	**0.818**	1.545	**0.392**	**5.435**	**0.824**
n2v-1000	0.829	1.500	0.387	5.640	0.931

Table 2: Model Evaluation: Bold denotes the best performance and grey bold denotes the second best.

To investigate the effect of graph pruning. We fix the node2vec parameters to $p = 1, q = 0.5, w = 80, r = 100$ and compare the performance for the two different smooth pruning threshold 500 and 1000. While the graph with threshold 1000 triples the memory requirement for that of 500, Table 3 shows that they have similar performance for the above metrics. It shows that pruning is an effective way to reduce memory requirements while maintaining evaluation performance.

Pruning	**Metrics**				
	Price Range	Star	#Common Cat	Distance	#Review
500	0.818	1.545	0.392	5.435	0.824
1000	0.822	1.481	0.387	5.980	0.997

Table 3: Graph Pruning Evaluation

4.3 Runtime and Memory Performance

In the current node2vec implementation, it is a computationally intensive task if the input graph contains a lot of nodes with high degrees. In such case, the memory usage does not scale linearly with the number of edges but rather polynomially. However, from Table 4, we could see that the runtime does scale linearly with the number of iterations and the computation is trivially parallelable, which makes the task feasible.

#Iterations	Real Time (min)	User Time (min)	Memory (GB)
10	4.7	70.0	97
100	13.1	339.7	102
1000	99.0	3076.6	104

Table 4: Runtime and Memory Usage Statistics

4.4 Semantic Evaluation

Inspired by word2vec [5], which successfully encodes semantic information in word embedding, we experiment with the item embedding from node2vec to see if they can capture the rich contextual information in the network. In word2vec, the result of a vector calculation `vec(‘‘Madrid’’) - vec(‘‘Spain’’) + vec(‘‘France’’)` is closer to `vec(‘‘Paris’’)` than to any other word vector [5].

We tested several combinations of popular chains and most of them yield meaningful result. For example, `vec(‘‘McDonald’’) - vec(‘‘KFC’’) + vec(‘‘Hudson’s Bay’’)` is close to `vec(‘‘Yorkdale Shopping Centre’’)` and `vec(‘‘CF Toronto Eaton Centre’’)` where “Hudson’s Bay”, “Yorkdale Shopping Centre” and “CF Toronto Eaton Centre” are all popular shopping centers in Toronto. `vec(‘‘McDonald’s’’) - vec(‘‘KFC’’) + vec(‘‘Starbucks’’)` is close to `vec(‘‘Tim Hortons’’)` and Tim Hortons is arguably the most popular coffee chain in Canada.

Another example is `vec(‘‘Chipotle Mexican Grill’’) - vec(‘‘Taco Bell’’) + vec(‘‘Sushi Rock’’)` is close to `vec(‘‘Sushi Supreme’’)`. The first two restaurants are Mexican and the latter two are both Japanese restaurants, which shows that the embedding was able to capture some categories information.

A limitation of this experiment is that the Yelp dataset only contains business, which has much less hierarchical relationship between the nodes than that of the words. Additionally, one of the differences between word2vec’ linear transformation and node2vec’s is that in word embedding, each word only has one embedding while in the item embedding, one chain can have multiple business at different locations and their embeddings may be different, which adds to the difficulty. We try to overcome this problem by finding more relevant stores for the first two chain items.

4.5 Challenges

One of the biggest difficulties we face is the intense memory requirement of node2vec. Based on our past experiments, we found that to run node2vec [6] on a projected graph with around 20 million edges, the amount of memory required is unfeasible. To solve the problem, we designed several graph pruning algorithms as presented in previous sections. The simple pruning algorithm reduces the number of edges to around 2 million, which required around 200 GB of memory for node2vec. We were able to compute the encoding using an Amazon Web Services (AWS) machine with a large amount of memory. To further reduce the memory requirement and improve the performance, we identified the bottleneck and designed a new node-based pruning algorithm and a smooth pruning algorithm which further reduces the memory requirement to around 70 GB. However, it’s still not feasible on a laptop computer so we run it on AWS too.

[6]Using the c++ implementation at `https://github.com/snap-stanford/snap/tree/master/examples/node2vec`

Other challenges include efficient hyperparameter tuning and evaluation methods design. We controlled variables to try to find the optimal values for the pruning threshold and node2vec hyperparameters. Further, due to the nature of our task, designing qualitative and quantitative evaluations is a hard task.

5 Future Work

We outline a few future work that may be helpful to improve the performance of our proposed network embeddings.

1. As shown in Section 4, different node2vec parameters can have significant impact on the embedding results. A solution is to perform grid search to find the optimal values for the pruning threshold and node2vec parameters. A more advanced method may be to use the evaluation metrics as label and the parameters as input the train a parameter tuner. With enough computational resources, we may perform one of the above two methods to improve the performance of our embedding.
2. Currently, we use qualitative evaluations and some heuristic metrics as quantitative evaluations. There may be better way to conduct quantitative evaluations, for example, applying the embedding to some downstream task like neural link predictions.
3. In the latest implemention of node2vec, the polynomial memory requirement of node2vec for clique-like graphs is a major bottleneck. The C++ implementation suffers from the problem that the function `preprocess_transition_probs` eats a lot of memory. When running on the projected/ folded graphs which have a lot of cliques, the precomputation of transition probability may make the performance close to $O(E^2)$. This issue could be potentially addressed by adding an LRU cache or adding an option to compute transitional probability on the fly. It is a time-memory tradeoff but it would be great to not rely on AWS machines to run node2vec.
4. One observation is that the walk length of node2vec is the same for any node, but some nodes are dead ends, and it may be useful to represent the dead ends in the random walk to help encode more structural information. In NLP task, this is usually addressed by adding paddings. It may be worth trying to add similar padding method.
5. We only tested the method on two business-related dataset. To see whether this model generalizes in other dataset, we crawl a small twitter dataset. Here, we focus on those users that has a "professor" mention in their biography. The edge was defined as a direct mention between two "professors". However, due to time and twitter data limitation, we only collected around 71,000 users with 53,000 edges, which is relatively small. In the future, if we were able to get access to all twitter data, we may perform a test on Twitter dataset.

6 Conclusion

Recommendation systems play a vital role in our daily life. However, the current widely-used recommenders mostly use traditional collaborative filtering techniques. In this report, we address this by adopting network embeddings using node2vec [3]. We show that network embeddings could learn useful information that could benefit downstreaming recommendation tasks, and advance the possibility of applying deep learning techniques in the field of recommendation systems. The code and dataset produced in this work will be released for public use at `https://github.com/VVCepheiA/cs224w-project`.

Acknowledgement

We thank Michele Catasta, Alexander Haigh, Prof. Jure Leskovec, Srijan Kumar, and all other CS224W staff for their valuable instructions on this project.

Contribution

B.D. worked on idea brainstorming, literature review, team discussions, result analysis, and paper writing. X.L. worked on coding, experiments running, idea brainstorming, literature review, team discussions, result analysis, and paper writing. Z.W. worked on coding, dataset collection, idea brainstorming, literature review, team discussions, result analysis, and paper writing. We believe the work was roughly divided as B.D. 20%; X.L. 40%; Z.W. 40%.

References

[1] Oren Barkan and Noam Koenigstein. Item2vec: Neural item embedding for collaborative filtering. *CoRR*, abs/1603.04259, 2016. URL `http://arxiv.org/abs/1603.04259`.

[2] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[3] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016. URL `http://arxiv.org/abs/1607.00653`.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL `http://arxiv.org/abs/1301.3781`.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013. URL `http://arxiv.org/abs/1310.4546`.

[6] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[7] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[8] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014. URL `http://arxiv.org/abs/1403.6652`.

[9] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9):2658–2663, 2004.

[10] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[11] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0. doi: 10.1145/371920.372071. URL `http://doi.acm.org/10.1145/371920.372071`.

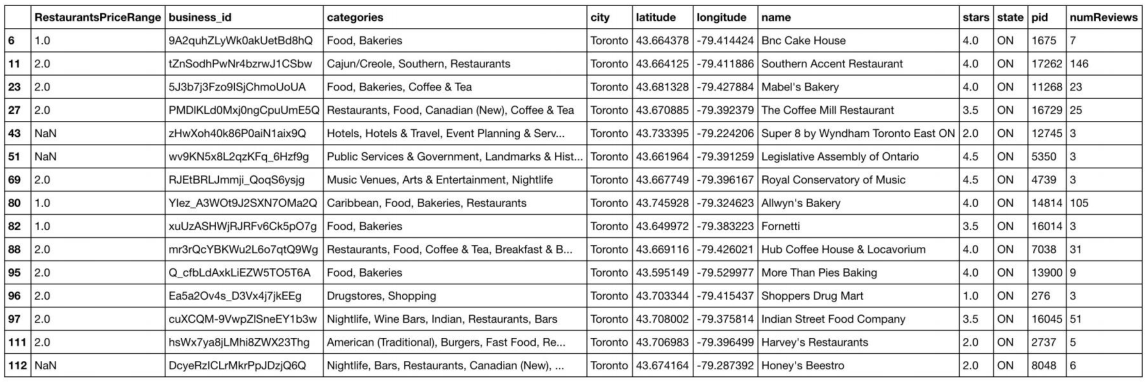
[12] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009:421425:1–421425:19, 2009. doi: 10.1155/2009/421425. URL `https://doi.org/10.1155/2009/421425`.

[13] X. Zhang, C. Zhang, C. Guo, and Y. Ji. A framework for enhancing word embeddings with task-specific information. In *2018 4th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 46–53, Aug 2018. doi: 10.1109/BIGCOM.2018.00014.

[14] Z. Zhao and M. Shang. User-based collaborative-filtering recommendation algorithms on hadoop. In *2010 Third International Conference on Knowledge Discovery and Data Mining*, pages 478–481, Jan 2010. doi: 10.1109/WKDD.2010.54.

Appendix

Figure 6 contains example preprocessed data used in this project.



	RestaurantsPriceRange	business_id	categories	city	latitude	longitude	name	stars	state	pid	numReviews
6	1.0	9A2quhZLyWk0akUetBd8hQ	Food, Bakeries	Toronto	43.664378	-79.414424	Bnc Cake House	4.0	ON	1675	7
11	2.0	tZnSodhPwNr4bzrwJ1CSbw	Cajun/Creole, Southern, Restaurants	Toronto	43.664125	-79.411886	Southern Accent Restaurant	4.0	ON	17262	146
23	2.0	5J3b7j3Fzo9lSjChmoUoUA	Food, Bakeries, Coffee & Tea	Toronto	43.681328	-79.427884	Mabel's Bakery	4.0	ON	11268	23
27	2.0	PMDlKLd0Mxj0ngCpuUmE5Q	Restaurants, Food, Canadian (New), Coffee & Tea	Toronto	43.670885	-79.392379	The Coffee Mill Restaurant	3.5	ON	16729	25
43	NaN	zHwXoh40k86P0aiN1aix9Q	Hotels, Hotels & Travel, Event Planning & Serv...	Toronto	43.733395	-79.224206	Super 8 by Wyndham Toronto East ON	2.0	ON	12745	3
51	NaN	wv9KN5x8L2qzKFq_6Hzf9g	Public Services & Government, Landmarks & Hist...	Toronto	43.661964	-79.391259	Legislative Assembly of Ontario	4.5	ON	5350	3
69	2.0	RJEtBRLJmmji_QoqS6ysjg	Music Venues, Arts & Entertainment, Nightlife	Toronto	43.667749	-79.396167	Royal Conservatory of Music	4.5	ON	4739	3
80	1.0	Ylez_A3WOt9J2SXN7OMa2Q	Caribbean, Food, Bakeries, Restaurants	Toronto	43.745928	-79.324623	Allwyn's Bakery	4.0	ON	14814	105
82	1.0	xuUzASHWjRJRFv6Ck5pO7g	Food, Bakeries	Toronto	43.649972	-79.383223	Fornetti	3.5	ON	16014	3
88	2.0	mr3rQcYBKWu2L6o7qtQ9Wg	Restaurants, Food, Coffee & Tea, Breakfast & B...	Toronto	43.669116	-79.426021	Hub Coffee House & Locavorium	4.0	ON	7038	31
95	2.0	Q_cfbLdAxkLiEZW5TO5T6A	Food, Bakeries	Toronto	43.595149	-79.529977	More Than Pies Baking	4.0	ON	13900	9
96	2.0	Ea5a2Ov4s_D3Vx4j7jkEEg	Drugstores, Shopping	Toronto	43.703344	-79.415437	Shoppers Drug Mart	1.0	ON	276	3
97	2.0	cuXCQM-9VwpZlSneEY1b3w	Nightlife, Wine Bars, Indian, Restaurants, Bars	Toronto	43.708002	-79.375814	Indian Street Food Company	3.5	ON	16045	51
111	2.0	hsWx7ya8jLMhi8ZWX23Thg	American (Traditional), Burgers, Fast Food, Re...	Toronto	43.706983	-79.396499	Harvey's Restaurants	2.0	ON	2737	5
112	NaN	DcyeRzICLrMkrPpJDzjQ6Q	Nightlife, Bars, Restaurants, Canadian (New), ...	Toronto	43.674164	-79.287392	Honey's Beestro	2.0	ON	8048	6

Figure 6: Example of Yelp Data