Graph Embedding Random Walk methods under DeepFM for Recommendation system

Tsai Ying Liu
tsai.liu@mail.utornto.ca

Zi You
zi.you@mail.utoronto.ca

ABSTRACT
Representation learning on graph has been utilizing as backbone in the domains like social networks or recommender systems. Recommending "personalize" items to users by learning the informatic latent relationships of users or items is more interpretable than training the black box models alone. The concept of 'nearness' is introduced by random walks to capture the neighbourhood structure of vertices in graph. In particular, we propose an innovative way of combining DeepWalk\Node2vec as the encoders to learn item\user embeddings and DeepFM as the decoder to complete the recommendation classification task. The experiments are based on a movie rating scenario. This paper aims at recommending movies based on users' similar movie preferences and the movies that sharing similar characters indirectly. The performance result shows that the graph with popular movies better learns vertex representations. Among different input settings for decoders, input with latent relationships of subjects and subjects in network effectively help predicting users' rating behaviours. However, the models need further improvement to learn how to better link the new users or movies to the existed ones.

1. INTRODUCTION

Graph G = (V, E) is a common data structure, which often acts as the backbone of diverse scenarios that possess network characteristic, such as social networks and recommender systems. It is composed by vertices and edges, where V is denoted as the set of vertices, E represents as the set of edges, and |V| is the number of vertices of the network. Relational information about interacting subjects can be stored and accessed efficiently in the graph-structured data. Graph can capture the position and the local neighbourhood structure of each vertex and direct connection between vertices. Therefore, extracting the structural information of graph is the problem that we need to solve. The tasks are not limited to link prediction, vertex classification, and recommendation. In our paper, we encode graph structure to learn node embeddings by random walk methods. We aim at classifying the popularity of a movie in a movie interaction graph based on users' rating and recommend new movies that are the closest neighbours of the movie that a user has liked before.

Node embedding methods are under the encoder-decoder framework. They aim at learning latent vertex representations that encoding their graph positions and local neighbourhood structures by the mapping function f: V → $\mathbb{R}^d$. This mapping function projects each vertex as a point into a low-dimensional vector space. The embeddings can then be decoded to extract more information of vertices. In order to get informatic representations, we aim at optimizing the loss function of encoder and decoder. Once embeddings are learned, they can be treated as input for further prediction or classification tasks. The performance of task is used to evaluate the effectiveness of the representations. In our paper, DeepFM is the framework that we choose to complete the classification task, since it is widely used for recommendation purpose in the industry. DNN is also implemented but for baseline purpose.

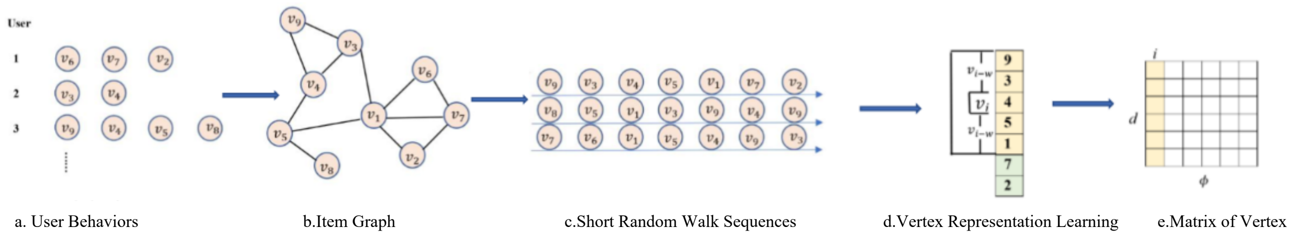
Node embedding methods possess three properties that are suitable to solve the major problems in recommender system, which is a large network of users, items, and other features. First, the low-dimensional vector space provides the scalability for real-world social networks to be processed in short period of time. The time-consuming and scalability problem solved. Secondly, the informatic representations can capture potential connections between items in low-dimensional space. The sparsity problem solved. Thirdly, the undirect-connected items could be recommended to users if they are the neighbours of the users' favour items. We can efficiently measure the distance between the embedding vectors. The neighbourhoods structure problem solved.

In this paper, our vertex representations are learned based on random walks under unsupervised manner. We choose random walk-based methods since they are suitable for large-scale social network scenarios. Firstly, the short random walk sequences provide the concept of 'nearness', which helps to capture the neighbourhood structure of vertices without computing the global structure of the graph. Secondly, each short random walk sequence can efficiently be generated at the same time.

The rest of the paper is organized as follows. In Section2, we discuss different flexibility definition of random walks node embedding methods in more detail, DeepWalk (Section 2.1) and Node2vec (Section 2.2). We propose using the learned representations as input for deep learning frameworks, DNN and DeepFM, for recommendation purpose (Section 2.3). Experimental results are presented in Section3. In Section 4, we conclude our studies and result.

2. GRAPH EMBEDDING METHODS & DEEPFM

2.1. DeepWalk


User
a. User Behaviors
b.Item Graph
c.Short Random Walk Sequences
d.Vertex Representation Learning
e.Matrix of Vertex
Figure 1: Deepwalk Embedding process

DeepWalk is a way to encode the potential relation of vertices to learn the matrix of vertex representations $\phi: v \in V \longmapsto \mathbb{R}^{|v|\times d}$ in low-dimension *d* (Figure1.e). Its input is stream of vertex sequences W_{v_i} that are generated from truncated random walks (Figure 1.c). Each vertex v_i in the sequences W_{v_i} with length *t*, $\{v_1 \dots v_t\}$, is used to predict its contexts, which are the vertices within its window size *w*. Vertex v_i's contexts, also mean neighbor vertices, $v_{i-w} \dots v_{i+w}$. v_{i-w}, may not link directly with v_i in the original graph, but through *w* steps random walks in graph, these contexts indirectly link and co-occur with v_i (Figure 1. d). v_i*'s* local neighborhood structure then formed. If vertices have higher probability co-occur on the random walks sequence, their latent representations ϕ tend to be similar. In the DeepWalk algorithm, the outer loop is based on γ, the number of starting random walks on each vertex. For instance, if γ is 2, each vertex in graph has twice times to be the starting point for random walks. The inner loop will go through all vertices in graph to make sure that each vertex is sampled for random walk at least once.

For each vertex v_i in the vertex sequences W_{v_i}, Skip-Gram predicts v_i*'s* context vertices, $\{v_{i-w}, \dots, v_{i+w}\}$, by optimizing the following equation. Given the latent representation of v_i, $\phi(v_i)$, we maximize the probability that its neighbors will co-occur (Figure 1.d).

$$\min_{\phi} -\log \Pr(\{v_{i-w}, \dots, v_{i+w}\}) \setminus v_i | \phi(v_i))$$

Assuming each vertex is independent, the probability $\Pr(\{v_{i-t}, \dots, v_{i-t}\}) \setminus v_i \mid \phi(v_i))$ is approximated as

$$\Pr(\{v_{i-w}, \dots, v_{i+w}\}) \setminus v_i \mid \phi(v_i)) = \prod_{j=i-w, j\neq i}^{i+w} \Pr(v_j \mid \phi(v_i))$$

In order to speed up the computation, "hierarchical softmax" is used to factorize the conditional probability and binary-tree structure is introduced to change the prediction problem into maximizing the probability of a path to vertex u_k in hierarchy, $(b_0 \dots b_l \dots b_{[\log|v|]})$. b_0 is the root node, b_l is the tree node, and $b_{[\log|v|]}$ is u_k. Each u_k is the context with in window size of v_j. $\psi(b_l)$ is denoted as the representation assigned to the parent of b_l[1].

$$\Pr(u_k \mid \phi(v_j)) = \prod_{l=1}^{[\log|v|]} \Pr(b_l \mid \phi(v_j)) = 1/(1 + e^{-\phi(v_i)\cdot\psi(b_l)}))$$

2.2. Node2vec

The node2vec is an algorithmic framework which is used to learning feature representation for nodes in network. This algorithmic has inherited the main idea of skip-gram model. The skip-gram is a method that has been studied in text analysis scenario. The first step of skip-gram is randomly sample possible contexts of one word and then obtain the vector representation of words by solving an optimization problem using Stochastic Gradient Descent. The distinctive point of node2vec is it adopts a biased truncated random walk to sample neighbors(context) of every node.

Introduction of Feature learning framework. Let *G* = (*V, E*) be a network, $f: V \to \mathbb{R}^d$ be the mapping function from nodes to feature representation. Every source node $u \in V$ and $N_{S(u)} \subset V$ is the network neighborhood of source node *u* which is generated by neighborhood sampling strategy *S*.
The objective function is

$$\max_{f} \sum_{u\in V} \log \Pr(N_s(u) | f(u))$$

where $f(u)$ is the feature representation of source node u. The goal of node2vec is to learn a graph to a low dimensional space of features that maximizes the log-likelihood of preserving network neighborhoods of nodes. This algorithmic assume that the likelihood of observing a neighbor of a source node is independent with observing other neighbors:

$$\Pr(N_S(u) | f(u)) = \prod_{n_i \in N_{S(u)}} \Pr(n_i | f(u)).$$

Also, it assumes that source node and its' each neighbor have same effect on each other. The conditional likelihood of every source-neighborhood node is measured by the dot product of their features:

$$\Pr(n_i | f(u)) = \frac{\exp(f(n_i)\cdot f(u))}{\sum_{u\in V} \exp(f(v)\cdot f(u))}.$$

Because of above two assumption, the objective function can be simplified to

$$\max_{f} \sum_{u\in V} [-\log Z_u + \sum_{n_i \in N_{S(u)}} f(n_i) | f(u))].$$

The objective function is optimized by stochastic gradient descent.

Search strategies. The way node2vec generate context nodes (neighborhood nodes) is different from Deepwalk. Deepwalk generate context nodes by randomly walk on graph. However, this is not reasonable in most real-world network especially commodity network since a majority of commodities in this network don't have high demand, which means relationships between these commodities are weak. If we adopt unweighted random walk to sample neighborhoods of a source node, the sample we get will contain many useless sequences which composed by node with less appearance frequency. The node2vec has resolve this problem by adopted a biased second order random walk.

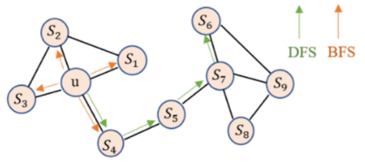


Figure 2: Breadth-first Sampling (BFS) and Depth-fist Sampling (DFS) procedure.

There are two kinds of search strategy. The first one is **Breadth-first Sampling (BFS)**. The $N_{BFS}(u)$ is restricted to nodes that are directed connected with source node u, so that $N_{BFS}(u) = \{S_{1,} S_2, S_3, S_4\}$ with neighborhood size k=4. The second search strategy is **Depth-fist Sampling (DFS)**. The $N_{DFS}(u)$ is a sequence of a nodes that at increasing distance form source node therefore $N_{DFS}(u) = \{S_{4,} S_5, S_7, S_6\}$. The BFS can better capture the homophily between nodes because it only samples the node which are immediate neighbors of the source, however, DFS has better performance on discovering structural equivalence between nodes. For example, u and $S_{1,}$ exhibit structural equivalence since they are in same position on their local structure. Therefore, u and $S_{1,}$ should have similar embedding even though they didn't connect with each other. The node2vec is a flexible algorithm that has ability to learn representation where nodes have similar embedding share similar position in their local structure and representation where nodes have similar embedding from same network close with each other. Node2vec achieve this flexibility through biased random walk sampling.

Biased Random Walk Sampling. Biased random walk is also called weighted walk. It allows us to smoothly control BFS and DFS by adding a bias on transition probability between two nodes. The normalized probability of transitioning from ith node (v) in the walk to $i-1$th node (x) is:

$$P(c_i|c_{i-1}) = \begin{cases} \frac{\pi_{vx}}{Z} & if(v,x) \in E \\ 0 & otherwise \end{cases}$$

where Z is normalizing factor and π_{vx} is the unnormalized transition probability of v and x.

For the original random walk sampling, the weight w_{vx} of edge between v and x is simply equal to π_{vx}, however it cannot allow us to control the search procedure. By adding a biased α_{pq} on π_{vx}, we can guide the walk and explore two type of network neighborhoods by simply adjust two parameters q and p. The biased random walk procedure is showed in figure 2. The figure shows how it evaluate the next step out of node v. The t is the start point of walk and α is calculated by:

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & if\ d_{tx} = 0 \\ 1 & if\ d_{tx} = 1 \\ \frac{1}{q} & if\ d_{tx} = 2 \end{cases}$$

Where d_{tx} denotes the shortest distance between t and x and $d_{tx} \in \{0,1,2\}$.

$$walk\ path: \begin{cases} t \rightarrow v \rightarrow\ t & if\ d_{tx} = 0 \\ t \rightarrow v \rightarrow\ x_1 & if\ d_{tx} = 1 \\ t \rightarrow v \rightarrow x_2 & if\ d_{tx} = 2 \\ t \rightarrow v \rightarrow\ x_3 & if\ d_{tx} = 2 \end{cases}$$

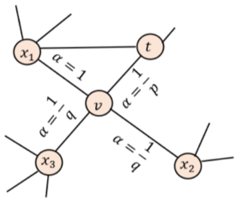


Figure 3: Illustration of the random walk procedure in node2vec

p (return parameter) and q (in-out parameter) play an important role in biased random walk. If $q > 1$, the random walk tends to choose nodes close to node t. This kind of walk can give us a local microscopic view of the graph so as to achieve Breadth-first Sampling behavior. If $q < 1$, it will walk to nodes far away from note t. This behavior is similar to Depth-fist Sampling which provides a global macroscopic view of the graph. The role of p in random walk is to control the range of walk. If $p < \min(q, 1)$, the probability of revisiting a node in the walk is high. If $p > \max(q, 1)$, the walk is more likely to visit those nodes haven't been visit before.

Node2vex algorithm. Setting up: graph $G = (V, E, W)$, walks per node r, walk length l, context size k, return parameter p, in-out parameter q, weight of edges π.
Step 1:sample contexts for all nodes in graph. Execute r biased random walk of a fixed walk length from every node in graph. For each node, we can get r × k neighbors in the end.
Step 2: optimize the mapping function f by using SGD.

2.3. DeepFM

After learning the node embeddings, they will be treated as input for machine learning tasks. In our paper, we focus on classification task for the purpose of recommendation. We choose DNN and DeepFM as the decoder framework since they are widely used deep learning frameworks for recommender system in industry.

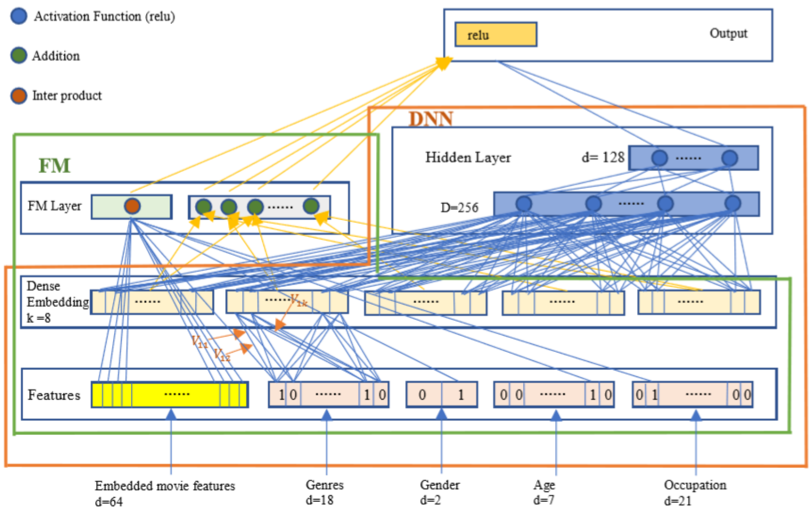


Figure 3: Graph embedding and one-hot embedding as input in DeepFM.

Factorization machine (FM) algorithm is used to solve feature combination problem under a sparsity data. The input of FM is produced by one-hot embedding which makes normal linear models hard to capture the relationship between individual features. However, FM can solve this problem by adding a latent vector on every feature.

Deepwalk is composed by FM component and deep component, that share the same input. FM provide order-2 feature interactions and fed in DNN to model high-order feature interactions. The advantage of DeepFM is that it can capture both low and high order feature interactions.

In our paper, we modify the original DeepFM model by replacing the one-hot embeddings of movie and user features into movie embeddings and user embeddings by DeepWalk and Node2vec respectively. The parameters setting and input setting in Figure3 is used in our experiment. In section 3, we will discuss these settings in detail.

3. EXPERIMENTS

In this experiment, our goal is precisely recommending a right movie to user through predicting user's rating of a movie. If the predicted rating is high (> 3), we assume the user like the movie and then recommend it to the user. Our downstream task is multi-label classification which is done by DNN and DeepFM. For this task, we discuss the quality of obtained representations produced by Deepwalk and node2vec on prediction task.

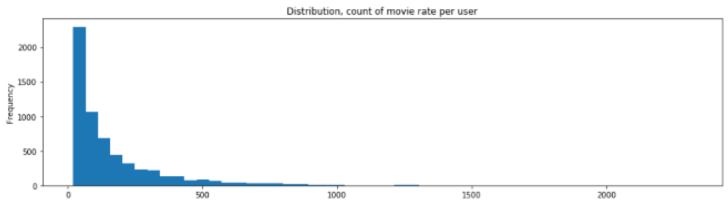
3.1. Experiment Settings

Data Description. We choose the MovieLens-1M dataset in our experiment, since it can be used for the purpose of recommending the popular items to users. These files contain 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000 [3]. We merge three datasets that provided. We take the demographic information of users, such as their genders {"M", "F"}, age groups {1,18, 25, 35, 45, 50, 56}, and occupation assigning value from 0 to 20 (e.g, 6: "doctor/health care", 12: "programmer"). We also pick information of movies, such as genres (e.g, "Animation", "Thriller"). Rating is the feature that we aim to predict, which is made on a 5-star scale. 'Rating', 'Genres', 'Gender', 'Age', and 'Occupation' are all categorical features.

	UserID	MovieID	Rating	Genres	Gender	Age	Occupation
1000203	6040	1091	1	Comedy	M	25	6
1000204	6040	1094	5	Drama|Romance|War	M	25	6
1000205	6040	562	5	Comedy|Drama	M	25	6
1000206	6040	1096	4	Drama	M	25	6
1000207	6040	1097	4	Children's|Drama|Fantasy|Sci-Fi	M	25	6

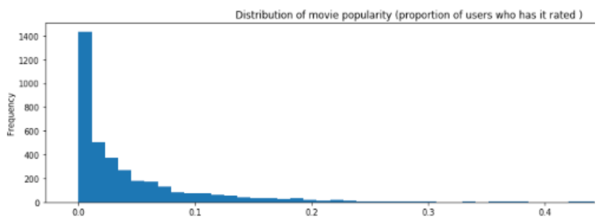
Table1: Dataset with information of users, movies, and ratings

We remove highly active users who have viewed over 500 movies in the embedding stage, since these observations may be pranking cases. We want to avoid adding those low-quality edges into the graph.



Plot1: Distribution of movie rate per user

In our first round of embedding experiment, we exclude movies that are extremely popular to avoid too many edges that are connected to popular movies in the graph since we want to recommend "right" and "personalize" movies to users instead of the popular ones. However, after training the task, the performance is worst. Thus, in our second round of embedding experiment, we include those connections. We discover that top popular movies are important for random walks to capture other potential links between movies. Number of users in our dataset is reduced from 6040 to 5641.



Plot2: Distribution of movie popularity

Graph. For DeepWalk embedding method, our adjacent list is created by collection of unordered movie lists that each UserID has rated. They represent a finite movie-interaction graph. Each list describes the set of neighbors, which are movies that share common features of a movie in the network based on users' behaviors. We have 5641 users that rate movies. 3706 movies have been rated, which means we will have 3706 vertices in the graph. Vertices are connected by edges based on these 5641 users' movie taste.

```
UserID
1    [1, 48, 150, 260, 527, 531, 588, 594, 595, 608...
2    [21, 95, 110, 163, 165, 235, 265, 292, 318, 34...
3    [104, 260, 480, 552, 590, 593, 648, 653, 733, ...
4    [260, 480, 1036, 1097, 1196, 1198, 1201, 1210,...
5    [6, 16, 24, 29, 32, 34, 36, 39, 41, 47, 50, 52...
Name: MovieID, dtype: object
```

Table2: graph for DeepWalk: Adjacency list (movie - movie graph based on users' behaviors)

```
25            (1, 1097)
26            (1, 1721)
27            (1, 1545)
28             (1, 745)
29            (1, 2294)
                 ...
1000178    (6040, 2762)
1000179    (6040, 1036)
1000180     (6040, 508)
1000181    (6040, 1041)
1000182    (6040, 3735)
```

Table2: graph for Node2vec: Edge list (user - movie)

Node Embeddings. After running DeepWalk methods, it maps these 3706 movies into 3706 numbers of vertex representations in low dimension of 64. These movie representations could help capture the neighborhood structure of movies. In other words, these informatic embeddings allow further machine learning models better recommend the movies that share common characters with a movie that a user liked in history.

MovieID	embedded_1	embedded_2	embedded_3	embedded_4	embedded_5	embedded_6	embedded_7	embedded_8	embedded_9	...
593	-0.001839	-0.208218	-0.149449	-0.275163	-0.021372	-0.262893	-0.119091	0.230677	-0.004118	...
110	0.028882	-0.134147	0.074782	0.057043	0.022140	-0.106065	0.060798	0.388916	-0.233957	...
2997	-0.029225	-0.079466	-0.116360	0.006065	0.011736	-0.155851	-0.179037	0.304760	0.065357	...
356	-0.022119	0.052959	0.029533	-0.050222	-0.023208	-0.017691	0.122533	0.333582	0.000737	...
3471	0.033819	-0.051301	0.009521	0.151771	-0.018335	0.066269	0.100171	0.200708	0.198458	...

Table3: Movie Embeddings result from DeepWalk

After running Node2vec methods, it maps 6039 users into 6039 numbers of vertex representations in low dimension of 64. We fix representation sizes from Node2vec same as previous movie representations from DeepWalk. By having user embeddings as input, we are interested in learning the latent user - user relationship. Through these relations, the model could extract their common preference taste on movies.

UserID	embed_1	embed_2	embed_3	embed_4	embed_5	embed_6	embed_7	embed_8	embed_9	...
4169	0.623519	0.710054	0.282839	-0.185800	0.498968	0.031520	0.595000	0.545903	-0.046819	...
1680	1.255707	0.498644	0.454048	-0.173776	-0.046529	-0.843088	-0.144394	0.722826	0.019805	...
1449	0.955785	0.317365	0.418595	-0.092898	0.248555	-0.140392	0.592002	0.459091	-0.046732	...
4277	0.568047	0.401560	0.418902	-0.289528	0.631391	0.003350	0.778597	0.541243	-0.163427	...
1941	0.047503	-0.160254	0.133853	-0.002472	0.532648	0.014277	0.537754	0.971867	-0.889656	...

Table4: User Embeddings result from Node2vec

We then combine movie embeddings and user embeddings respectively with other side information, 'Genres', 'Gender', 'Age', and 'Occupation'. The new datasets are used for the following classification task. 70% of dataset is for training purpose and 30% of dataset is for testing purpose. To be clear, we have 667138 observations in train set and 333070 observations in test set.

UserID	MovieID	Rating	Genres	Gender	Age	Occupation	embedded_1	embedded_2	embedded_3	...
6040	1091	1	Comedy	M	25.0	6.0	-0.092902	0.056900	0.011286	...
6040	1094	5	Drama|Romance|War	M	25.0	6.0	-0.095246	-0.167268	0.120715	...
6040	562	5	Comedy|Drama	M	25.0	6.0	0.031244	-0.122108	-0.208762	...
6040	1096	4	Drama	M	25.0	6.0	-0.075538	0.019253	-0.082550	...
6040	1097	4	Children's|Drama|Fantasy|Sci-Fi	M	25.0	6.0	0.091530	-0.043612	0.164355	...

Table5: Dataset of movie embeddings from DeepWalk with side information

UserID	MovieID	Rating	Genres	Gender	Age	Occupation	embed_1	embed_2	...
6040	1091	1	Comedy	M	25.0	6.0	-0.662957	-0.337576	...
6040	1094	5	Drama|Romance|War	M	25.0	6.0	-0.662957	-0.337576	...
6040	562	5	Comedy|Drama	M	25.0	6.0	-0.662957	-0.337576	...
6040	1096	4	Drama	M	25.0	6.0	-0.662957	-0.337576	...
6040	1097	4	Children's|Drama|Fantasy|Sci-Fi	M	25.0	6.0	-0.662957	-0.337576	...

Table6: Dataset of movie embeddings from Node2vec with side information

3.2. Recommender Classification Task.

Parameter setting. We use DNN(baseline) and DeepFM to train the classification task in recommendation scenario. The dense layer in both models are set as dimension 8. We use Adam as the iterative optimizer to perform several passes on the train dataset to obtain better results. The learning rate is 0.0001. We have batch norm 1 and 0.995 batch norm decay. The two hidden layers are dimension of 256 and 128 respectively. The activation for layers is relu function. Each layer has applied dropout keeping 80% of nodes in layers. Our train dataset has 667138 samples. We have a batch size of 1000, where 1000 training examples present in a single batch. Therefore, in each epoch, we have 670 batches (667138/1000 = 670). Each batch gets passed through the algorithm, so we have 670 iterations per epoch. We set the algorithm to run 30 epochs so there is a total of 20100 iterations (30*670 = 20100) for training.

Evaluation. Normalized gini score is used as our evaluation metric. The advantages of using it instead of AUC is because the minimal value of area under curve is 0.5. Gini score is defined as 2*AUC-1, where the minimal value is 0, and score of 1 indicates perfect classifier. From our experiment performance, we choose the result from one of the epochs that has better balance of score in between train set and test set. However, from the table7, we can see that even the training sets have ideal prediction of a user's rating score for a movie, the models don't really learn the latent information of users or movies. Thus, if we want to predict new users' movie preferences, those models may not be feasible, or the input settings need further investigation.

Classification method / Input	DNN (baseline)		DeepFM	
	train	test	train	test
No graph Embeddings (baseline) with side info	0.5316	0.0013	0.5214	0.0015
DeepWalk Movie Embeddings (without popular movie vertices) with side info	0.0087	0.0022	0.0076	0.0021
DeepWalk Movie Embeddings (with popular movie vertices) with side info	0.7121	0.0048	0.5214	0.0015
Node2vec (p=1, q=2) User Embeddings with side info	0.7802	0.0021	0.8589	0.0008
Node2vec (p=1, q=0.5) User Embeddings with side info	0.8339	0.0019	0.8832	0.0015

Table7: evaluation result (gini norm)

From the performance of each training set, we conclude that input setting including either movie embeddings or user embeddings effectively learn the classification task than the input setting that doesn't have node embeddings. Moreover, we surprisingly discover that including popular movies' connections in a graph effectively learn the potential relations between movies. The vertex representations provide informatic latent interactions between movies, which is helpful for our classification task to predict user rating of a movie through learning the movie-interaction network. From the input settings that have Node2vec as their embedding methods but with different q parameter setting, there is no big difference in the results. Thus, we should try more different parameter settings of p and q to better show their importance.

Node2vec seems have to be outperformed Deepwalk because it can guide search procedure toward the best samples; however, comparing these node embedding methods' performances is not suitable from our experiment setting. Since we want to learn more diverse latent relationships of user or movie but considering the time-consuming manner, we use Node2vec to learn user – movie relationship while implementing DeepWalk to learn movie – movie interactions based on user behaviour. Comparing both methods under same condition settings is what we can explore in our further experiment.

4. CONCLUSIONS

Intuitively, we can say that our experiment network exhibits a mix of homophily and structural equivalence. For instance, two movies will have homophily-based relationship if their share similar feature such as genre types. Some relations will exist between two movies even though they don't belong to similar genre. This kind of relationship is called structural equivalence. In our case, movies that release in same specific time in different year may exhibit this kind of property.

The random walk approach adopted by Deepwalk acts like a depth-first search. Such an approach may quickly learn structural equivalence of nodes by bringing in indirect neighbors, but it may also introduce nodes that are long range away and neglect the homophily between nodes. The node2vec seems have to outperform Deepwalk because it can guide search procedure toward the best samples; however, we can prove this statement by setting same conditions in further experiment.

5. REFERNECES

[1] Perozzi, Bryan and Al-Rfou, Rami and Skiena, Steven, DeepWalk: Online Learning of Social Representations, Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, 2014, 978-1-4503-2956-9, New York, New York, USA, 701—710, 10, http://doi.acm.org/10.1145/2623330.2623732, 10.1145/2623330.2623732, 2623732, ACM, New York, NY, USA

[2] Jizhe Wang, Pipei Huang∗, Huan Zhao, Zhibo Zhang, Binqiang Zhao, Dik Lun Lee, Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba, KDD '18, August 19–23, 2018, London, United Kingdom 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5552-0/18/08, https://doi.org/10.1145/3219819.3219869

[3] William L. Hamilton and Zhitao Ying and Jure Leskovec, Hamilton2017RepresentationLO, Representation Learning on Graphs: Methods and Applications, IEEE Data Eng. Bull., 2017, 40, 52-74

[4] Daokun Zhang and Jie Yin and Xingquan Zhu and Chengqi Zhang, Zhang2017NetworkRL, Network Representation Learning: A Survey, CoRR, 2017, abs/1801.05852

[5] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pages 855–864, 2016.

[7] Guo2017DeepFMAF, DeepFM: A Factorization-Machine based Neural Network for CTR Prediction, Huifeng Guo and Ruiming Tang and Yunming Ye and Zhenguo Li and Xiuqiang He, IJCAI, 2017

[6] R. He, J. McAuley. Modeling the visual evolution of fashion trends with one-class collaborative filtering. WWW, 2016
J. McAuley, C. Targett, J. Shi, A. van den Hengel. Image-based recommendations on styles and substitutes. SIGIR, 2015