GCN-Based CUHKSZ Food Recommendation System and its Improvements

Group 20

Xu Kai 121090650
Xiao Jingli 121090636
Wang Haoxuan 121090538
Wang Yuhui 121090581

The Chinese University of Hong Kong, Shenzhen

Course: DDA4210

1 Significance and Novelty of the Study

The canteen of the Lower Campus of CUHKSZ has a rich menu of dishes. So we want to design a food recommendation system for CUHKSZ students based on our collected student-dish interaction data.

To achieve this, we mainly face two problems.

1. How to efficiently capture the collaborative signals in the data?
2. Since our collected data is small, how to train a strong model while avoiding over-fitting?

For the 1st problem, we use the Neural Graph Collaborative Filtering (NGCF) model to capture the collaborative signals in high-order connectivity of the interactive data (Wang et al., 2019) .

For the 2nd problem, we modify the negative sampler in training process to strengthen the model and simplify the NGCF to avoid overfitting.

2 Data Collection

We collect our data via questionnaires. We first used the Fanfou app to find 50 dishes that had the most sales at the canteens on Lower Campus. Then we collected 142 questionnaires, each asking a student to select five meals they like most from the list of fifty. We treat each selection as an interaction.

What's more, to validate our models, we also introduce another dataset: MovieLens(small) (Harper & Konstan, 2015).

The details of the two datasets are as follows:

Data Set	Users	Items	Interactions	Density
CUHKSZ Food	142	50	710	10%
MovieLens(small)	160	8965	80668	1.4751%

Table 1: Overview of Data Sets

3 Methodology of Natural Graph Filtering (NGCF)

3.1 Recommendation as Link Prediction on Bipartite Graph

Commonly, the input of the recommendation system includes a set of users $U = \{u_i\}_{i=1}^m$, items $V = \{v_i\}_{i=1}^n$, and user-item interactions $O^+ = \{(u, v^+) \mid u \in U, v^+ \in V\}$, where each pair indicates an interaction (buy, click, etc.) between user u and item v^+. We can use a bipartite graph to model them:

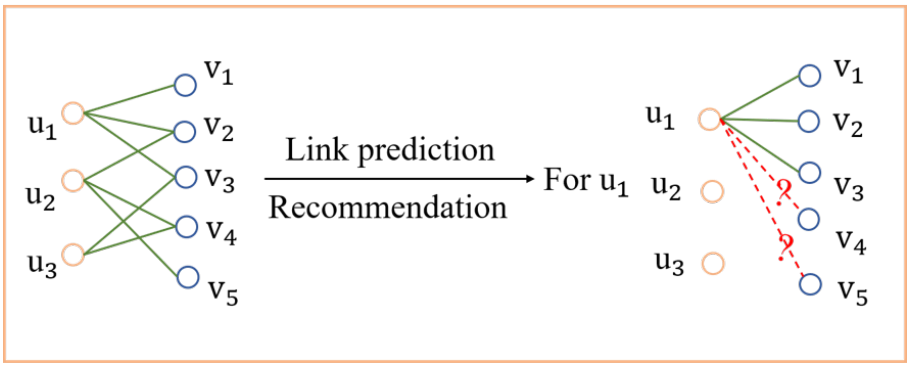

Figure 1: Bipartite Graph

Notice, in a bipartite graph one user node can only have edges with other item nodes. And an edge between (u, v^+) shows that u has an interaction with v^+.

Now, the goal of the recommendation system is to learn embeddings $\{{e_{u_i}}^*\}_{i=1}^m$, $\{{e_{v_i}}^*\}_{i=1}^n$ for each user/item node such that $\hat{y}(u, v) = ({e_u}^*)^T \cdot e_v^*$ can quantify the preference of user u to item v.

Then, for a given user u, we can recommend $\{v_{[1]}, \ldots, v_{[k]}\}$ which haven't had interactions with u but have the top k $\hat{y}(u, v_{[i]})$ among all items that haven't interacted with u.

3.2 NGCF Framework

3.2.1 Core Idea

NGCF uses a methodology similar to GNN to propagate embeddings of nodes in the graph. By doing so, it can exploit the high-order connectivity from user-item interactions. Specifically, we want to map the information of nodes into embeddings . In the process of embedding propagation of GNN, we can refine a user's (or an item's) embedding by aggregating the embedding of the interacted items (or users). By stacking multiple embedding layers, we can enforce the embeddings to capture the collaborative signals in high-order connectivity.

3.2.2 Notation and Initialization

Let $R \in R^{m \times n}$ be the user-item interaction matrix, where $R_{ij} = 1$ if user u_i has interaction with item v_j, else $R_{ij} = 0$. Define adjacency matrix as $A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix} \in R^{(m+n)\times(m+n)}$. Let $D \in R^{(m+n)\times(m+n)}$ be the degree matrix of A, and $L = D^{-\frac{1}{2}} A D^{\frac{1}{2}}$ be the Laplacian matrix. Notice here we do not have $A + I$ operation, since in a bipartite graph self-loop of a node is not allowed. What's more, for user u_i we denote N_{u_i} as the set of items that are interacted by user u_i. And N_{v_i} is the same for item v_i.

Then before training, for each $\{u_i\}_{i=1}^m$, $\{v_i\}_{i=1}^n$ we generate a random vector (uniform) $\{{e_{u_i}}^{(0)}\}_{i=1}^m$, $\{{e_{v_i}}^{(0)}\}_{i=1}^n$ where each $e \in R^d$, and denote $E^{(0)} = \begin{bmatrix} {e_{u_1}}^{(0)} & \cdots & {e_{u_m}}^{(0)} & {e_{v_1}}^{(0)} & \cdots & {e_{v_n}}^{(0)} \end{bmatrix} \in R^{d\times(m+n)}$.

3.2.3 Embedding Propagation

Suppose we have L embedding propagation layers. At layer ℓ ($\ell \le L$), we update embeddings as:

$$ {e_{u_i}}^{(\ell+1)} = \sigma\left(W_1 {e_{u_i}}^{(\ell)} + \sum_{v_j \in N_{u_i}} \frac{1}{\sqrt{|N_{u_i}|\,|N_{v_j}|}} \left(W_1 {e_{v_j}}^{(\ell)} + W_2({e_{v_j}}^{(\ell)} \odot {e_{u_i}}^{(\ell)}) \right) \right), \quad i = 1, \cdots, m \quad (1) $$

$$ {e_{v_i}}^{(\ell+1)} = \sigma\left(W_1 {e_{v_i}}^{(\ell)} + \sum_{u_j \in N_{v_i}} \frac{1}{\sqrt{|N_{v_i}|\,|N_{u_j}|}} \left(W_1 {e_{u_j}}^{(\ell)} + W_2({e_{u_j}}^{(\ell)} \odot {e_{v_i}}^{(\ell)}) \right) \right), \quad i = 1, \cdots, n \quad (2) $$

where ${W_1}^{(\ell)}, {W_2}^{(\ell)} \in R^{d\ell \times d}$ are the trainable weight matrices in layer ℓ. And $\odot$ denotes the element-wise product.

In equation (1), $W_1 {e_{u_i}}^{(\ell)}$ lets ${e_{u_i}}^{(\ell)}$ inherit messages from ${e_{u_i}}^{(\ell)}$, $W_1 {e_{u_j}}^{(\ell)} + W_2({e_{u_j}}^{(\ell)} \odot {e_{v_i}}^{(\ell)})$ lets ${e_{u_i}}^{(\ell+1)}$ take information from ${e_{v_j}}$'s, and such information depends on the affinity between e_{u_i} and e_{v_j}.

Equation (2) holds the same logic as (1).

Equation (1) and (2) can be combined and expressed as a matrix form which we are more familiar with:

$$ E^{(\ell)} = \sigma\left((\mathcal{L} + I) E^{(\ell-1)} {W_1}^{(\ell)} + \mathcal{L} E^{(\ell-1)} (E^{(\ell-1)} {W_2}^{(\ell)}) \right) \quad (3) $$

Where $E^{(\ell)} \in R^{(N+M)\times d\ell}$ are the representations for users and items obtained after ℓ steps of embedding propagation.

3.2.4 Pooling

After embedding propagation, we get $\left\{{e_{u_i}}^{(\ell)}\right\}_{i=1,\cdots,m,\ \ell=0,\cdots,L}$ and $\left\{{e_{v_i}}^{(\ell)}\right\}_{i=1,\cdots,n,\ \ell=0,\cdots,L}$ for users and items. To avoid over-smoothing, concat-based pooling is used, and the final output is:

$$ {e_u}^* = {e_u}^{(0)} \,\|\, \cdots \,\|\, {e_u}^{(L)}, \quad {e_v}^* = {e_v}^{(0)} \,\|\, \cdots \,\|\, {e_v}^{(L)} \quad (4) $$

3.2.5 Evaluation Metric and Optimization

For the evaluation metric, we use Recall@k and Precision@k which have been discussed in the lecture, so we do not show details here. To optimize the model, we use Bayesian Personalized Ranking (BPR) loss here. Recall, $O^+ = \{(u, v^+) \mid u \in U, v^+ \in V\}$, where each pair indicates an interaction between user u and item v^+. Further, we define $O^- = \{(u, v^-) \mid u \in U, v^- \in V\}$, where each pair (u, v^-) indicates u and v^- do not have interaction. Then BPR loss is:

$$L_{\text{BPR}} = -\sum_{u_i \in U} \sum_{(u_i, v_j) \in O^+} \sum_{(u_i, v_k) \in O^-} \ln \sigma \left({e^*_{(u_i)}}^T e^*_{(v_j)} - {e^*_{(u_i)}}^T e^*_{(v_k)} \right) + \lambda \|\Theta\|_2^2 \tag{5}$$

where $\Theta = \{E^{(0)}, \{{W_1}^{(l)}, {W_2}^{(l)}\}_{l=1}^L\}$ denote all the trainable model parameters. For more details of BPR loss, please refer to the Appendix 2.

- Mini-batch training for the BPR loss:

  In our practice, we sample a subset of users $U_{\text{mini}} \subset U$, for each $u_i \in U_{\text{mini}}$, we sample one item v_{ij}, such that $(u_i, v_{ij}) \in O^+$, and one item v_{ik}, such that $(u_i, v_{ik}) \in O^-$, then the mini-batch loss is:

$$L_{\text{BPR}}^{\text{mini}} = -\frac{1}{|U_{\text{mini}}|} \sum_{u_i \in U_{\text{mini}}} \left(-\ln \sigma \left({e^*_{u_i}}^T e^*_{v_{ij}} - {e^*_{u_i}}^T e^*_{v_{ik}} \right) \right). \tag{6}$$

  Then we can use SGD or Adam to update Θ.

4 Simplification of NGCF

Recall in part 3, our trainable model parameters are $\Theta = \left\{ E^{(0)}, \left\{ {W_1}^{(\ell)}, {W_2}^{(\ell)} \right\}_{\ell=1}^L \right\}$, where $E^{(0)} \in R^{d \times (m+n)}$, ${W_1}^{(\ell)}, {W_2}^{(\ell)} \in R^{d\ell \times d}$. Usually, L is small such as 2, 3, 4, or 5. The scale of $\left\{ {W_1}^{(\ell)}, {W_2}^{(\ell)} \right\}_{\ell=1}^L$ is about $O(\tilde{d}_\ell d L) = O(\tilde{d}_\ell d)$ where $\tilde{d}_\ell = \max_\ell d_\ell$. When our input data is large, i.e. $m + n \gg d_\ell$, the NGCF mainly learns to update $E^{(0)}$. So the impact of $\left\{ {W_1}^{(\ell)}, {W_2}^{(\ell)} \right\}_{\ell=1}^L$ may be small. Holding this intuition and referencing He et al.(2019), we decide to drop $\left\{ {W_1}^{(\ell)}, {W_2}^{(\ell)} \right\}_{\ell=1}^L$ in the NGCF model.

Furthermore, Wu et al. (2019), and He et al. (2020), show that for GCN used for recommendation tasks, the nonlinear activation function can be useless(for more details , please refer to Appendix 2). So we also drop the nonlinear activation σ in NGCF.

Now, the simplified NGCF model is the same as NGCF except when doing embedding propagation at the $(\ell + 1)$-th layer:

$${e_{u_i}}^{(k+1)} = \sum_{v_j \in N_{u_i}} \frac{1}{\sqrt{|N_{u_i}|\,|N_{v_j}|}} {e_{v_j}}^{(\ell)} \tag{7}$$

$${e_{v_j}}^{(k+1)} = \sum_{u_j \in N_{v_i}} \frac{1}{\sqrt{|N_{v_i}|\,|N_{u_j}|}} {e_{u_j}}^{(\ell)} \tag{8}$$

$$E^{(\ell+1)} = \mathcal{L} E^{(\ell)} \tag{9}$$

We hope that this simplification of NGCF can lead to faster convergence speed and better generalization ability.

5 Improve the Efficiency of NGCF by Changing the Negative Sampler

5.1 Why Negative Sampler Important and the Problems of Random Negative Sampler

Recall in loss function (6), the process of sampling one item v_{ik}, such that $(u_i, v_{ik}) \in O^-$, for user u_i is called negative sampling. Here we call item v_{ij} a positive sample of user u_i if $(u_i, v_{ij}) \in O^+$, and v_{ik} is a negative sample of u_i if $(u_i, v_{ik}) \in O^-$.

According to the study Huang et al. (2021), the negative sampling strategy plays a critical role in the model training of recommendation. Intuitively, just like in GAN, if the negative samples are close to the positive ones, the model can better learn the boundary between positive and negative samples.

Usually, in NGCF, we apply the strategy of random negative sampling. We scan all the items, put items that do not have interactions with u_i in a list, and then randomly pick one item from the list as a negative sample. However, such a strategy cannot guarantee the number of negative samples, since all items which do not have interactions with u_i have an equal probability of being picked.

5.2 Two-step Negative Sampler

To improve the efficiency of NGCF, we use a strategy called two-step negative sampling instead of random negative sampling. The main idea of this method is: in the first step, we inject information from positive samples to negative ones, making negative candidates which are 'similar' to positive samples. In the second step, we use the pooling operation to combine the negative candidates in Step 1 together to create a fake but informative negative sample. For details and the algorithm of this two-step sampler, please refer to Appendix 4.

6 Numerical Results and Conclusion

- Model 1: NGCF.
- Model 2: Simplified NGCF.
- Model 3: Simplified NGCF using a Two-step Negative sampler.

For hyper-parameters, please refer to Appendix 3.

1. For MovieLens(Small) Dataset we have the following result.

Model	Layers	Average Recall@10	Average Precision@10
1	3	0.1295	0.1962
1	4	0.1301	0.1997
1	5	0.1259	0.1959
2	3	0.1607	0.2473
2	4	0.1620	0.2351
2	5	0.1587	0.2512
3	3	0.1850	0.2817
3	4	0.1897	0.2749
3	5	0.1645	0.2616

Table 2: Result of MovieLens(Small) Dataset

2. For our CUHKSZ Food Dataset we have the following result.

Model	Layers	Average Recall@5	Average Precision@5
1	3	0.1162	0.0404
1	4	0.1608	0.0465
1	5	0.1667	0.0545
2	3	0.1372	0.0424
2	4	0.1751	0.0566
2	5	0.1734	0.0561
3	3	0.1495	0.0462
3	4	0.1926	0.0634
3	5	0.1873	0.0612

Table 3: Result of CUHKSZ Food Dataset

On both datasets, we find the performance of model 3 $>$ model 2 $>$ model 1, which shows the improvement in efficiency based on our modification.

References

Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019, July). Neural graph collaborative filtering. *In Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval* (pp. 165-174).
Harper, F. M., & Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4), 1-19.
He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., & Wang, M. (2020, July). Lightgcn: Simplifying and powering graph convolution network for recommendation. *In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval* (pp. 639-648).
Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., & Weinberger, K. (2019, May). Simplifying graph convolutional networks. *In International conference on machine learning* (pp. 6861-6871). PMLR.
Huang, T., Dong, Y., Ding, M., Yang, Z., Feng, W., Wang, X., & Tang, J. (2021, August). Mixgcf: An improved training method for graph neural network-based recommender systems. *In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (pp. 665-674).

A Appendix

A.1 More Details of BPR Loss

In BPR loss, we assume users prefer items they have interacted with to those that haven't, i.e., users like positive items more than negative ones. So ${e^*_{u_i}}^T e^*_{v_j} - {e^*_{u_i}}^T e^*_{v_k}$ measures the difference in preference of user u_i to positive item v_j and negative item v_k. By minimizing the difference, we can refine the embedding to capture such preference.

A.2 Intuitive Explanation of Why We Can Drop Nonlinear Activation Function and Transformation Matrix in a Collaborative Recommendation System.

As argued by HE (2020), in semi-supervised node classification, each node has rich semantic features as input, such as the title and abstract words of a paper. Thus performing multiple layers of nonlinear transformation is beneficial to feature learning. Nevertheless, in collaborative filtering, each node of the user-item interaction graph is randomly set at layer 0 ($E^{(0)}$). The embeddings in $E^{(0)}$ have no concrete semantics. In this case, performing multiple nonlinear transformations will not contribute to learning better features; even worse, it may add difficulties in training well. In the next subsection, we provide empirical evidence for this argument.

A.3 Hyper-parameters of Our Models

	Latent Dimension	Epochs	Batch Size
MovieLens	64	50	1024
CUHKSZ Food	8	10	40

A.4 Algorithm of Two-step Negative Sampling

Algorithm: Training Process with Two-step Negative Sampler

Input: Number of negative candidate M, Number of layers $\mathcal{L}$.

for t = 1, 2, …, T **do**

Sample a mini-bach of positive pairs $\mathrm{U}_{mini} = \{(u, v_p)\}$.

Initialize loss $L_{BPR}^{mini} = 0$

for each $(u, v_p) \in U_{mini}$ **do**

Sample M raw negative items $\{V_m\}_{m=1}^{M}$ for u.

Let $\varepsilon = \left\{\alpha^{(\ell)} e_{v_p}{}^{(\ell)} + (1-\alpha^{(\ell)}) e_{v_m}{}^{(\ell)}\right\}_{m=1,\ell=0}^{M,\mathcal{L}}$ with $\{\alpha^{(\ell)}\}_{\ell=0}^{\mathcal{L}}$ pre-determined.

Let $e_v{}^{*(\ell)} = arg \max_{e_v{}^{(\ell)} \in \varepsilon^{(\ell)}} (e_u{}^{(\ell)})^T e_v{}^{(\ell)}$

Let $e_v{}^{*} = \sum_{\ell=0}^{\mathcal{L}} \lambda_{(\ell)} e_v{}^{*(\ell)}$

$L_{BPR}^{mini} = L_{BPR}^{mini} - ln\sigma(e_{u_i}^{*\,T} e_{v_{ij}}^{*} - e_{u_i}^{*\,T} e_{v_{ik}}^{*})$

end

Update Θ by decreasing the gradient $\nabla_\Theta L_{BPR}^{mini}$

end