Project progress report: Boosting collaborative filtering with sentiment analysis

Xinyu He
University of Illinois, Urbana-Champaign
Illinois, USA
xhe34@illinois.edu

ABSTRACT

Sentiment analysis has been used to do rating prediction and achieved remarkable success. However, it requires user's review as input, which makes it impossible to predict ratings for items that user has never interacted with. Therefore, only using sentiment analysis for recommendation is impractical.

Given the success of sentiment analysis, researchers start to investigate how to integrate sentiment analysis with collaborative filtering. However, previous work ([1]), [3]) only focus on how to use the benefits of sentiment analysis to boost collaborative filtering, but ignore the feedback that collaborative filtering can contributes to sentiment analysis. Sentiment analysis can provide continuous approximation of ratings instead the discrete values; while collaborative filtering can provide extra user and item information for sentiment analysis. Therefore, it's important to maximize the mutual interest of sentiment analysis and collaborative filtering.

1 PROBLEM DEFINITION

Input: User-item interaction history with reviews: $\{(u, i, s, r)\}$, where the four entities in one history record are user, item, rating, review, respectively.

Output: Let $\mathcal{U}$ and $\mathcal{I}$ be the collection of users and item, for any user-item pair $(u, i) \in \mathcal{U} \times \mathbf{I}$, output probability of u interact with i in the future p_{ui}

2 PROPOSED FRAMEWORK

Maximizing Mutual Information is a popular technique in recent years that tries to maximize the agreement between two models. Although directly maximizing mutual information is intractable, InfoNCE Loss[4] is often adopted to maximize the lower bound of mutual information. Therefore, we can leverage both benefits of semantic analysis and collaborative filtering by maximizing the mutual information (adding InfoNCE Loss term) between two backbone model: semantic analysis backbone model $\mathcal{M}_s$ that output review embedding $\{e_r\}$; collaborative filtering backbone model $\mathcal{M}_c$ that output user embedding $\{e_u\}$ and item embedding $\{e_i\}$.

Specifically, for a record (u, i, s, r), we want to extract the user and item information from e_r, i.e., $e_u^r = f_u(e_r), e_i^r = f_i(e_r)$, where f_u, f_i are user feature and item feature extractor that extracts users' and items' information from reviews' features. Then we want to maximize the mutual information between $\{e_u^r\}$, e_u, and mutual information between $\{e_i^r\}$, e_i. Finally, p_{ui} will be predicted by dot product like in most existing works, e.g., LightGCN [2]. The backbone models can be implemented by Transformer [5] and LightGCN [2]. Note that these backbone models are just selected for this project, it can be replaced by any other models that give desired embeddings. The framework is shown in Figure 1. More details will be introduced in Section 4.

3 DATASET PREPROCESSING

Amazon Digital Music Dataset[1] is used for this project.

During data preprocessing, I only selected those ratings over 3.0 and they are all normalized into $(0, 1]$, i.e, given a rating in range $[3, 5]$, the normaized rating will be

$$\hat{s} = \frac{s-2}{3} \tag{1}$$

The whole dataset are split into test set and training set with proportion 3:7.

After data preprocessing, the dataset contains 16518 users, 11794n digital musics, 116020 ratings and reviews for training and 49599 ratings and reviews for testing.

4 MODEL IMPLEMENTATION

4.1 LightGCN

LightGCN is a GNN-based method that considers users' and items' embeddings as model parameters. It first construct all user-item historical interactions as a undirected and unweighted bipartite graph. To apply message passing on this graph, LightGCN simply considers aggregation of neighbors' embeddings as the messages.

$$m_{j\to i}^{(l)} = \frac{1}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}} e_j^{(l-1)} \tag{2}$$

where $|\mathcal{N}(i)|$ is the number of neighbors of i. Then layer-wise representation ends up in a simple formula:

$$e_i^{(l)} = \sum_{j\in\mathcal{N}_i} m_{j\to i}^{(l)} \tag{3}$$

and the final representation aggregate those layer-wise representations

$$e_i = \frac{1}{L}\sum_{l=1}^{L} e_i^{(l)} \tag{4}$$

Note that this is actually different from the original paper because I adapted some improvement from SimGCL [6]. After updating these embeddings by message passing, embeddings are again updated by gradient descent to minimize BPR loss and L2 regularizartion loss.

$$\mathcal{L} = \mathcal{L}_{BPR} + \mathcal{L}_2 = -\sum_{(u,i,j)\in O} \log\sigma(\hat{p}_{ui} - \hat{p}_{uj}) + \lambda||\Theta||_2^2 \tag{5}$$

where $\Theta = \{E = \{e_i\}\}$ is the collection of model parameters, $\hat{p}_{ui}$ is the predicted probability defined as $\hat{p}_{ui} = e_u \cdot e_i$, $O = \{(u, i, j)|(u, i) \in R^+, (u, j) \in R^-\}$ denotes the pairwise training

[1] https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/

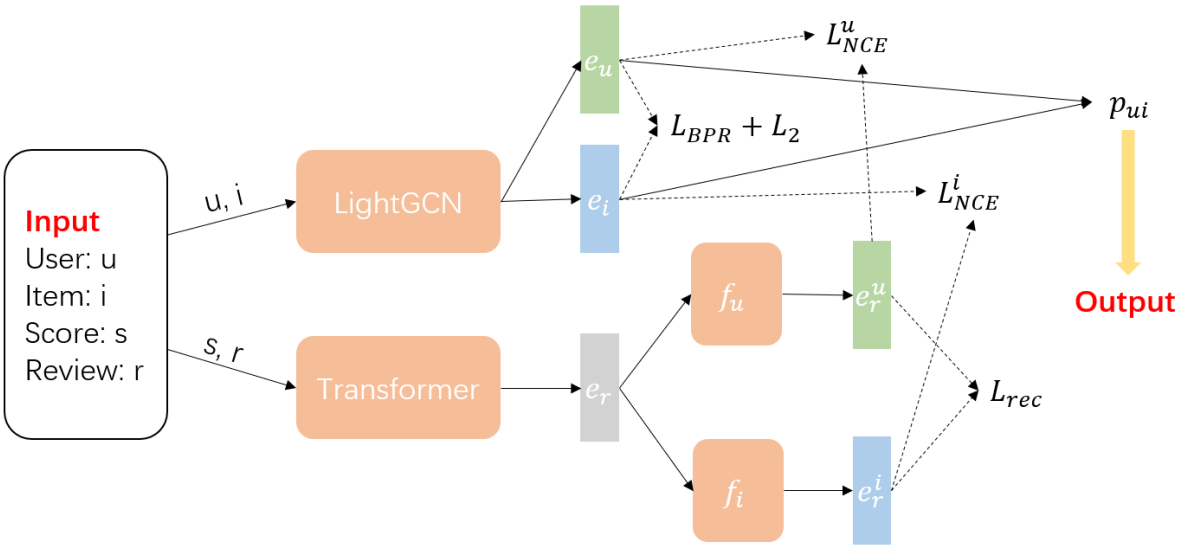

Figure 1: Proposed framework

data, R^+ indicates the observed interactions (positive samples), and R_- is the unobserved interactions (negative samples). In our case, items in negative samples are randomly sampled from item set.

4.2 Transformer

I leverage nn.TransformerEncoderLayer and nn.TransformerEncoder in pytorch to implement transformer. Before feeding our data into the interface of pytorch, I first use tools in torchtext to parse the words in reviews into tokens and build the vocabulary. Then, I also added positional encoding in the tokens, which align with the implemendation of original paper [5]. The d-dimensional output embedding of Transformer will be further fed into f_u and f_i (which are implemented as two fully connected layers) to extract e^r_u, e^r_i. Finally, transformer will use a MAE (mean absolute error) loss to update.

$$\mathcal{L} = \sum_{(u,i,s)} |\hat{p}^r_{ui} - \hat{s}_{ui}| \tag{6}$$

where $\hat{s}_{ui}$ is the ground truth normalized rating, $\hat{p}^r_{ui}$ is the predicted rating defined as $\hat{p}^r_{ui} = e^r_u \cdot e^r_i$.

5 REMAINING TASKS

Thus far, I have finished data preprocessing and implementation of LightGCN and Transformer module, while InfoNCE loss and some other training/evalution code has not been implemented yet. The codes are uploaded to github. The remaining tasks are as below:

- Connect the two implemented module, add evaluation and training codes.
- Compare proposed framework with vanilla LightGCN to show its effectiveness.

REFERENCES

[1] Miguel Á García-Cumbreras, Arturo Montejo-Ráez, and Manuel C Díaz-Galiano. 2013. Pessimists and optimists: Improving collaborative filtering through sentiment analysis. *Expert Systems with Applications* 40, 17 (2013), 6758–6765.

[2] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. 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*. 639–648.

[3] Cane WK Leung, Stephen CF Chan, and Fu-lai Chung. 2006. Integrating collaborative filtering and sentiment analysis: A rating inference approach. In *Proceedings of the ECAI 2006 workshop on recommender systems*. 62–66.

[4] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[6] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*. 1294–1303.