Recommender Systems for Online Knowledge Sharing Platform

SI 650 Project Final Report

Ximiao Li
ximiaoli@umich.edu

Xiying Gao
xiying@umich.edu

Yuqi Yan
yukei@umich.edu

Abstract

In the context of the rapid development of the Internet and software industry, recommender systems exist in all aspects of our lives. Youtube and Spotify would generate personalized playlists, Amazon would recommend products according to our purchase history, Instagram would discover potential interests based on our social network, etc. Especially, applications, such as Quora, share and grow the world's knowledge by matching the most important questions and best-suited users, and the best answers to the public. Driven by this mission, we explore several recommender systems using the information retrieval techniques learned in class and compare their performance for knowledge-sharing.

1 Introduction

This paper mainly focuses on Collaborative Filtering (CF) based recommender systems for *Zhihu*[1], the most popular online knowledge-sharing platform in China. A successful recommender system for such platforms would enable users to have more related search results and expand their knowledge base by recommending related topics that may interest them. To the best of our knowledge, no one has applied CF-variants on the ZhihuRec dataset, which is a brand new one, so we are working on a non-solved problem and have few references. Normal distribution recommender, popularity recommender, and user-item effects recommender as selected as baseline models. Further, we explored Matrix Factorization, Variational Autoencoders, and Graph Convolution Networks to compare the recommendation performance. All of the high-level models except for Matrix Factorization outperform baseline models and among them, Variational Autoencoder performs best according to selected evaluation metrics. LightGCN, which is the variant of classical GCN, is also proven to have higher performance than the original one. We discuss potential reasons for these facts and prospect some future work in the later part. Through this project, we learned how to utilize cutting-edge collaborative filtering variants to build an ideal recommender system and successfully implemented these models in real-world data.

[1] https://www.zhihu.com/

2 Related Work

Collaborative filtering (CF) is known as one of the most successful approaches in the recommender system area. Researchers have then developed a series of CF-variants to overcome the challenges that the CF algorithm faced. With the fast-growing number of data and the development of computing techniques, researchers are now able to train and learn the model features with deep neural networks. Specifically, for the social media platforms, we can further include users' friendship relationship information and find out the similarities. Graph neural networks have shown good performance in such tasks. In this section, we'll discuss their advantages and drawbacks.

2.1 Collaborative Filtering (CF)

Collaborative filtering uses a database of preferences for items by users to predict additional topics or products a new user may like(Su and Khoshgoftaar, 2009). These data are converted to a user-item (U-I) rating matrix. The similarity matrix of users (user-based) / items (item-based) is calculated and we can make recommendations by aggregating the ratings of similar users/items. This is also known as memory-based CF. The method is easy to implement but suffers from the sparse U-I matrix, which brings in the cold start problem(Said et al., 2012). The user to rate or item needs to be rated sufficient times to enable the system to capture the user's preferences. Therefore, researchers have developed different algorithms to learn the latent factors from the U-I matrix. Among all the CF-variants, ma-

trix factorization (MF) is one of the widest adapted methods used for RS for its efficiency shown in the Netflix Prize Challenge(Funk, 2006)(Koren et al., 2009). We implemented this method in this project to futher explore its performance on the ZhihuRec dataset.

2.2 Neural Network-based CF

Auto Encoder-CF In light of the previous success of the encoder-decoder architecture in SegNet(Badrinarayanan et al., 2017) for computer vision tasks, researchers use this networks architecture to learn the latent factors and build RS based on it. Autorec(Sedhain et al., 2015) and BiVAE(Truong et al., 2021) are two variants using the Autoencoder architecture. In this project, we perform variational Autoencoder (VAE) to obtain recommendation results on our brand-new dataset.
Graph Neural Networks (GNN) In the real world, most objects around us are explicitly or implicitly connected with each other; in other words, our living world follows a graphic structure. This characteristic is obvious in the recommendation system as well. The objects in an RS, including users, items, attributes, and context, are tightly connected with each other and influence each other via various relations(Hu et al., 2014), as shown in Figure 1. Therefore, a part of further RS algorithm improvements is mainly based on its graph structure, such as incorporating Graph Neural Network (GNN) or Graph Convolution Network (GCN) with classic CF tasks. Through an iterative message passing, the neural network can better learn user/item representations by capturing the high-order information in the user-item bipartite graph(Wu et al., 2020). Numerous successful GNN-based models have been proposed and shown promising results for CF tasks, such as PinSage(Ying et al., 2018), NGCF(Wang et al., 2019), LightGCN(He et al., 2020), and MCCF(Wang et al., 2020). In this project, we figure out the difference between LightGCN and classical GCN, and implement these two models to compare their results.

3 Dataset

In this section, we'll introduce the data we use for our exploration of recommender systems for the online knowledge-sharing platform. The data is already annotated but is encoded due to the privacy policy. Therefore, we can't provide further annotation results or meaningful examples. We'll conduct

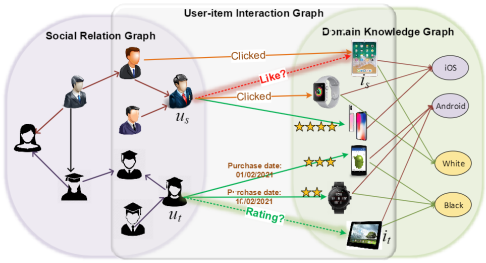

Figure 1: The demonstration of graph learning-based recommender systems

Exploratory Data Analysis to provide some insights about the dataset. (suggested by the instructor)

3.1 Introduction

Our dataset is ZhihuRec (Hao et al., 2021a), which is collected from Zhihu, a socialized knowledge-sharing community in Q&A scenarios. It is the largest public recommendation dataset, with rich user profiles (gender, login frequency, geographical location, etc.), huge user interaction behaviors (impressions, clicks, timestamps, search queries, etc), and abundant item information (textual content and numerical information).

The ZhihuRec dataset contains a full dataset called Zhihu100M, which includes about 100M user-answer interactions, and two smaller datasets randomly sampled from the Zhihu100M dataset called Zhihu20M and Zhihu1M. Some statistical information is displayed in Table 1.

Dataset	Zhihu100M	Zhihu20M	Zhihu1M
#users	798,086	159,878	7,963
#answer	554,976	342,737	81,214
#impressions	99,978,523	19,999,502	1,000,026
#queries	3,900,243	779,104	38,148
#users with queries	501,918	100,611	4,975
#clicks	26,981,583	5,395,962	271,725
avg #impressions	125.27	125.09	125.58
avg #clicks	33.81	33.75	34.12
#clicks: #non-clicks	1 : 2.71	1 : 2.71	1 : 2.68
avg #queries per user	4.89	4.87	4.79

Table 1: ZhihuRec Statistics (Hao et al., 2021b)

The highlight of the ZhihuRec dataset is user interaction logs. User clicks are viewed as positive interactions, and user impressions (without clicks) are viewed as negative interactions:

- Label = 0: impression = 1, click = 0
- Label = 1: impression = 1, click = 1

Table 2 shows the fields of each impression record in ZhihuRec to give a general idea of the user interaction logs.

Field	Description
userID	Id of user
#impressions	#user-answer impression
answerID_i	Id of the i th answer
showtime_i	show time of the i th answer
readtime_i	read time of the i th answer

Table 2: The Impression Fields of ZhihuRec Dataset (Hao et al., 2021b)

3.2 Exploratory Data Analysis

We performed some basic analysis on **ZhihuRec1M** to get an overview of the data.

3.2.1 Answers

First, we analyzed the distribution of the answers' creation time, and plotted the number of answers created against the dates formatted in *'Year-Month'* (Figure 2). It shows that the answers recorded began from April 2011 and ended in May 2018. There's a sharp increase in the number of answers in 2018, with the maximum count reaching 48886 in May.

Next, we took *#likes, #comments, #collections* as basic features and drew histograms and kernel density estimate (KDE) plots between them (Figure 5). The plots show a slightly right skewed pattern in distribution.

Then we analyzed the correlation between attributes (Figure 4). Note that we dropped invalid data with no tokenID (i.e. empty answers) and counted the number of tokens in each answer during pre-processing. It indicates that if an answer is *recommended by the editor*, it is more likely to be *labeled high-value answer*. Moreover, *#thanks*, *#likes*, *#comments*, *#collections*, *#dislikes* and *#helpless* have significantly positive correlations between each other.

We also plotted the distribution of answers with the different number of topics (Figure 3). There are 14407 answers have no topics related, and we only considered the answers having at least one topic in the analysis. The results indicate that most answers have 5 topics related, while few answers relate to 6 or more topics.

3.2.2 Questions

The distribution of questions' creation time (Figure 6) shows that the dataset records questions between December 2010 and May 2018. Similar to the distribution of answers, there's also a sharp increase in the number of questions starting from 2018. The maximum count of questions created is 6662 in May 2018.

Taking *#answers*, *#followers*, *#invitations*, *#comments* as basic features, the distribution of these items was plotted (Figure 9). While density distributions of *#answers*, *#followers* and *#comments* have a right-skewed pattern respectively, *#invitations* presents a bimodal distribution in the plot.

After dropping invalid data with no tokenID (i.e. empty questions) and calculating the number of tokens in each question, the correlations between attributes are plotted (Figure 8). Concluded from the plot, *#followers*, *#comments* and *#followers* are significantly correlated with each other.

Similar to the result of answers (Figure 3), the distribution of questions with different *#topics* (Figure 7) indicates that the majority of questions are related to 5 or fewer topics.

3.2.3 Users

To explore the basic features of users, we draw distribution plots for *#followers, #topics_followed, #questions_followed*, and found that all the three features show slightly right skewed patterns in distribution (Figure 11).

Then we introduced more attributes to our analysis, through the plot below (Figure 10), we discovered that *#thanks_received, #comments_received, #dislike_received, #answers*, and *#comments* are strongly positively correlated to each other.

As the user-based CF model utilizes users' similarity based on users' features, we simply explore it by constructing a network with the lists of topic that each user followed, *#topicID_followed*, before building models. We first found the top 10 topics which have the most followers, then split the top 10 topics into two sets: one is the top 5 topics, and the other is the top 6-10 topics. For each of the two sets, we added users who followed those topics as nodes, with user's *gender*, and *#followers* as node's attributes. We would add an edge between users if they followed the same topic, and the weight of an edge is the number of topics that each pair of users commonly followed. By randomly choosing

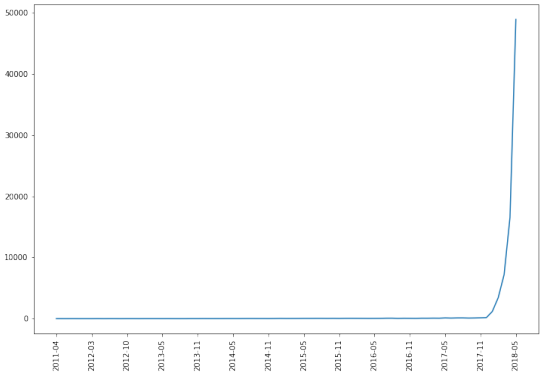

Figure 2: Distribution of Answers Created

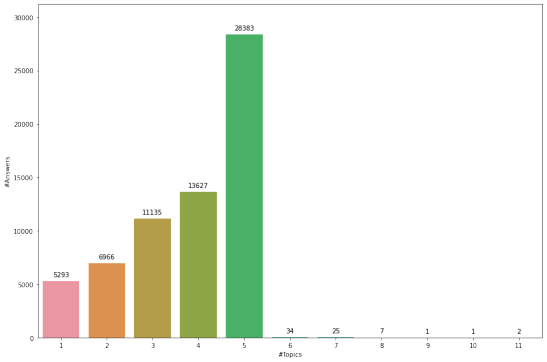

Figure 3: Distribution of answers with different #topics, each answer has at least one topic

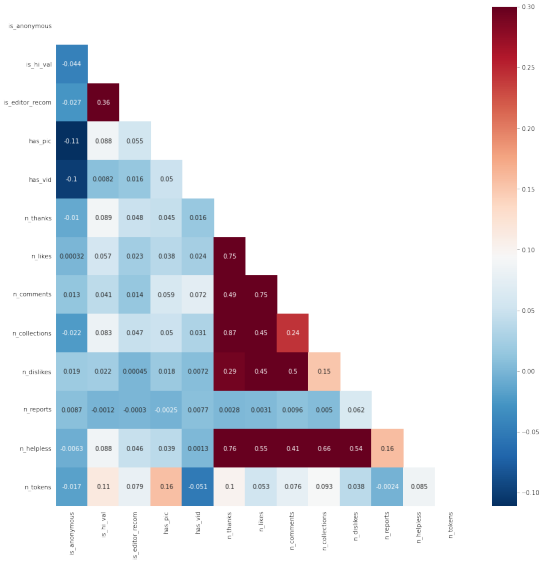

Figure 4: Correlation Between Answers' Features

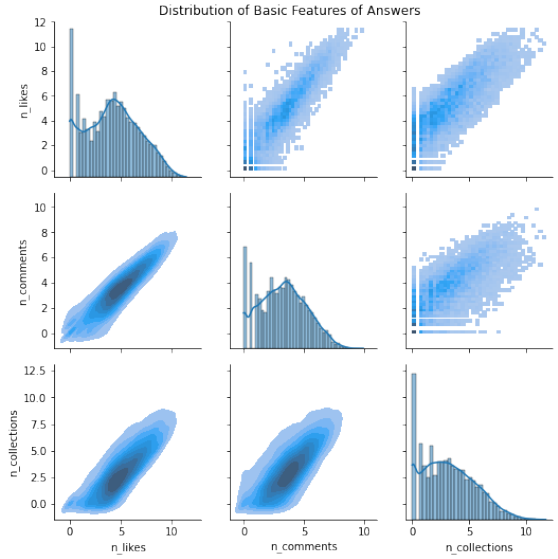

Figure 5: Distribution of Answers Basic Features

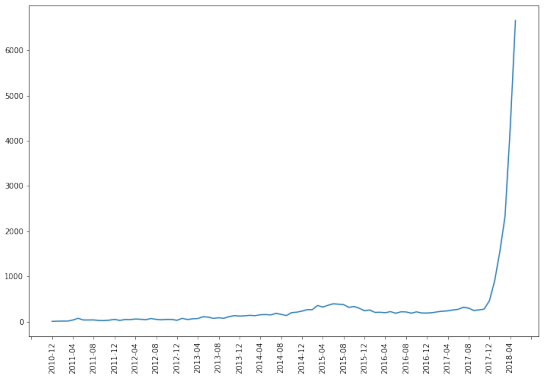

Figure 6: Distribution of Questions Created

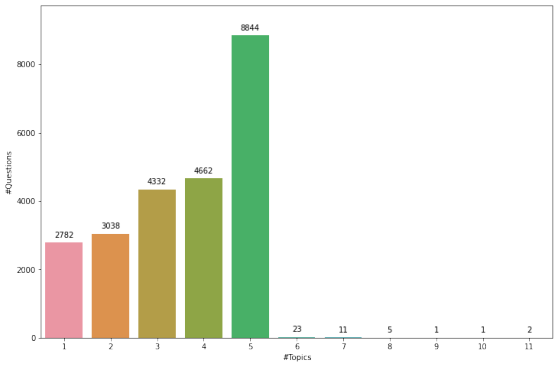

Figure 7: Distribution of questions with different #topics, each question has at least one topic

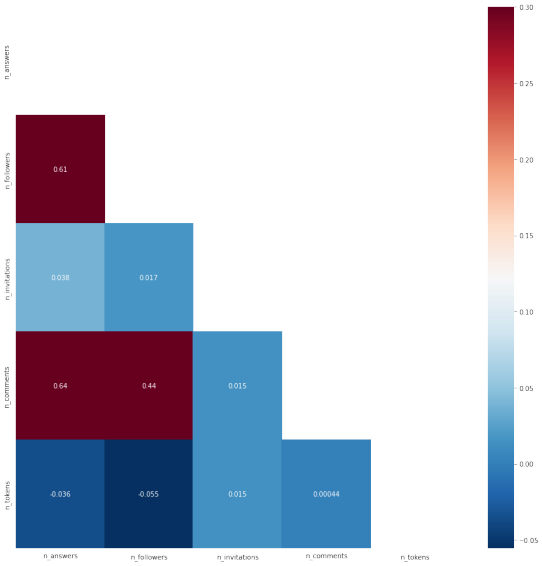

Figure 8: Correlation Between Questions' Features

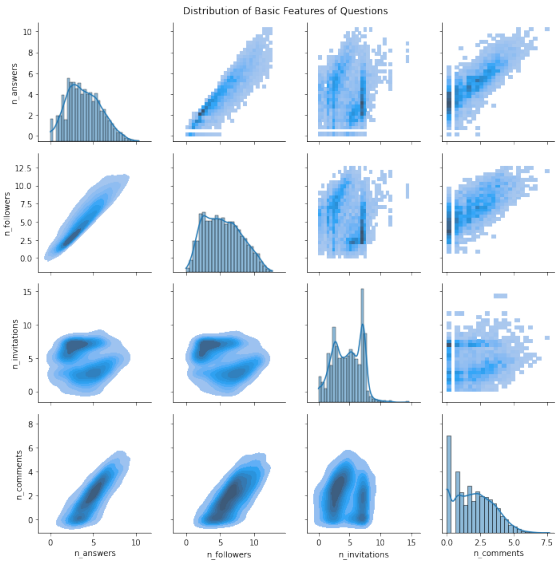

Figure 9: Distribution of Questions Basic Features

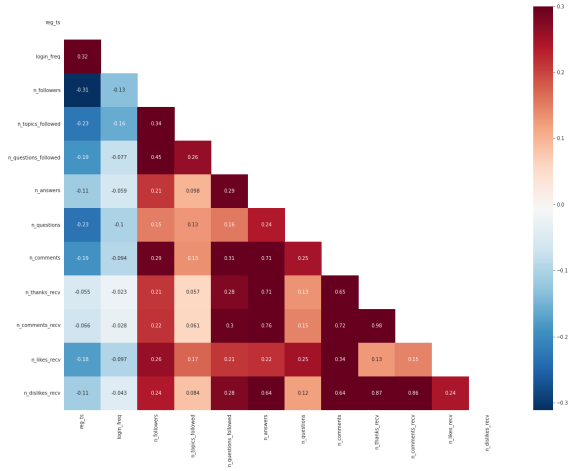

Figure 10: Correlation Between Users' Features

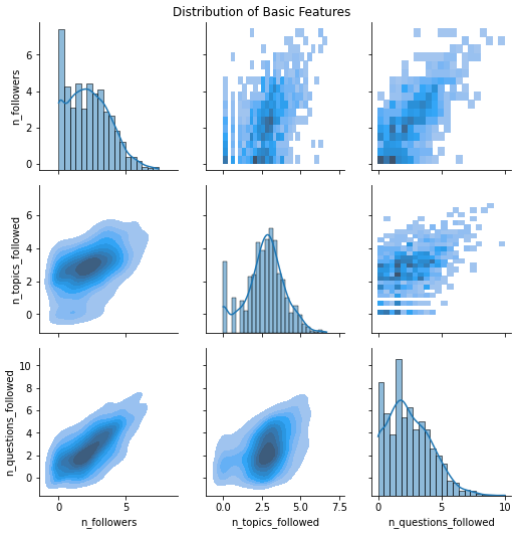

Figure 11: Distribution of Users' Basic Features

20 users, we got the graphs (Figure 12 and Figure 13) representing the network.

Note that the node size shows *#followers*, the node color represents *gender* (orange for women, blue for men, grey for unknown type), and the node label is *#userID*. Unfortunately, we didn't find significant relationships between the commonly followed topics with either *gender* or *#followers*. However, we did conclude that the top 5 topics are followed by most users and should not be considered useful features for our models.

4 Methodology

In this section, we'll briefly introduce how we preprocess the data and what models we have implemented. We utilized two packages Recbole[2] and Surprise[3] to perform models. In addtion, we manually implement and add MF method into Recbole. The estimated rating of user u for item i is notated as $\hat{r}_{ui}$ in this part.

4.1 Data Preprocessing

Due to the limitation of computing resources and time, our project focuses on the Zhuhu1M dataset only. After retrieving the dataset, we split it into train, validation, and test sets. For each user, the last click and impressions after the last click are treated as the test set, the click right before the last click, and the impressions that happened between the click right before the last click and the last click are treated as the validation set, and others are treated as the training set.

We also create a graph to include the connection between the users. Since we don't have detailed follower IDs for each user, we depict the relationship between users by the number of common topics they follow. The more common topics two users follow, the more likely that these users share the same interests. Hence, for every two users, we add an edge between them with the number of common topics as the weight.

As for the feature selection task, we take both the results of our "Exploratory Data Analysis" part and the common sense into consideration, and finally choose features as listed in Table 3.

[2]https://recbole.io/

[3]https://surpriselib.com/

Data	Features
user	userID, gender, #followers, #topics_followed, #questions_followed, #answers, #questions, #n_comments, #thanks_recv, #comments_recv, #likes_recv, #dislikes_recv, province
answer	answerID, is_anonymous, is_hi_val, is_editor_recom, has_pic, has_vid, #thanks, #n_likes, #comments, #collections, #dislikes, #reports, #helpless
iteraction	userID, answerID, imp_ts, is_clicked

Table 3: The Selected Features of Zhihu1M

4.2 Baseline Models

4.2.1 Normal Distribution Recommender

This baseline algorithm predicts a random rating based on the distribution of the training set, which is assumed to be normal. The prediction $\hat{r}_{ui}$ is generated from a normal distribution $N(\hat{\mu}, \hat{\sigma}^2)$ where $\hat{\mu}$ and $\hat{\sigma}$ are estimated from the training data using Maximum Likelihood Estimation:

$$\hat{\mu} = \frac{1}{|R_{\text{train}}|} \sum_{r_{ui} \in R_{\text{train}}} r_{ui}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{\text{train}}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{\text{train}}|}}$$

4.2.2 Popularity Recommender

This algorithm always recommends the most popular answers (i.e. most clicked by the users) in the training set.

4.2.3 User-Item Effects Recommender

This method is chosen as the baseline model by Yahoo Research (Koren, 2010). Denote μ the overall average click possibility. The estimate for an unknown rating r_{ui} accounts for the user and item effects:

$$\hat{r}_{ui} = \hat{\mu} + \hat{b}_u + \hat{b}_i$$

where $\hat{b}_u, \hat{\mu}, \hat{b}_i$ are calculated by

$$\min \sum_{(u,i)} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

If the user is unknown, then the bias b_u is assumed to be zero. The same applies to b_i.

4.3 General Models

4.3.1 Matrix Factorization

The traditional CF model based on the similarity score between user/item pairs would generate a

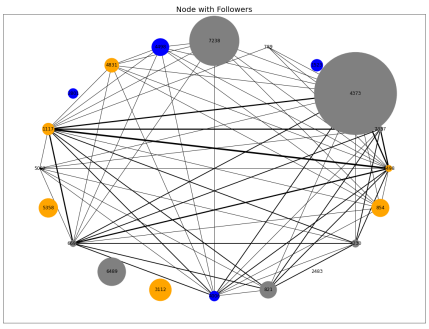

Figure 12: Users and Common Followed Topics: Top 5

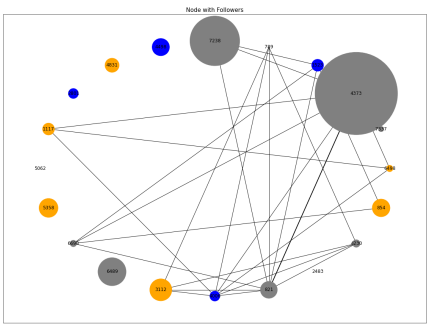

Figure 13: Users and Common Followed Topics: Top 6-10

large and sparse matrix, which induces complex computation. To solve this problem, Matrix Factorization (MF), which is the first generation of the machine learning-based CF approach can find the user and item latent factor representation by decomposing the user-item matrix(the interaction) based on linear algebraic technique.

4.3.2 Variational Autoencoders

Variational AutoEncoders (VAEs) introduce a generative model with multinomial likelihood and use Bayesian inference for parameter estimation since multinomial likelihood is particularly well suited to modeling user-item implicit feedback data(Liang et al., 2018).

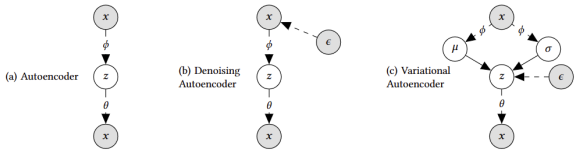

Figure 14: An illustration of how Variational Autoencoders differ from other autoencoders.

In this model, an introduction to a different regularization parameter for the learning objective is crucial for achieving competitive performance. It also utilizes KL annealing to tune the additional parameter introduced in a practical and efficient way. In this scenario, this non-linear probabilistic model would enable us to go beyond the limited modeling capacity of linear factor models.

4.4 Graph Convolution Network

Methods that generate user's (or item's) embeddings by mapping from pre-existing features, such as classic MF, cannot capture latent user-item interactions. Therefore, introducing a bipartite graph structure into the embedding process with Graph Convolution Network (GCN) was proposed to integrate the latent user-item interactions.

4.4.1 Neural Graph Collaborative Filtering

Neural Graph Collaborative Filtering (NGCF) exploits the high-order connectivity relationships in the user-item bipartite graph by propagating embeddings on it. The illustration of this idea is shown as Figure 15.

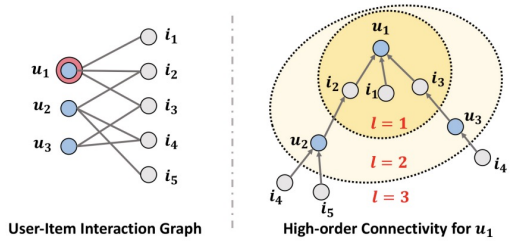

Figure 15: An illustration of the user-item interaction graph and the high-order connectivity. (Wang et al., 2019)

Embeddings. NGCF describes user u_i (item i_j) as vector $e_{u_i} \in \mathbb{R}^d$ ($e_{i_j} \in \mathbb{R}^d$):

$$E = [\underbrace{e_{u_1}, e_{u_2}, ..., e_{u_N}}_{\text{user embeddings}}, \underbrace{e_{i_1}, e_{i_2}, ..., e_{i_M}}_{\text{item embeddings}}].$$

Unlike traditional recommender models such as MF and neural collaborative filtering(He et al.,

2017), the embeddings are initial state instead of directly fed into an interaction layer to achieve the prediction score. The interaction information would be added in the future propagation part to realize higher efficiency.

Embedding Propagation Layers. Following the standard GCN(Kipf and Welling, 2016), NGCF defines the propagate embeddings as:

$$e_u^{(k+1)} = \delta(W_1 e_u^{(k)} + \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}}(W_1 e_i^{(k)} + W_2(e_i^{(k)} \odot e_u^{(k)}))),$$

$$e_i^{(k+1)} = \delta(W_1 e_i^{(k)} + \sum_{u \in N_i} \frac{1}{\sqrt{|N_u||N_i|}}(W_1 e_u^{(k)} + W_2(e_u^{(k)} \odot e_i^{(k)}))),$$

where $e_u^{(k)}$ and $e_i^{(k)}$ respectively denote the refined embedding of user u and item i after k layers propagation; δ is the nonlinear activation function; N_u denotes the set of items that are interacted by user u, N_i denotes the set of items that are interacted by item i, W_1 and W_2 are trainable weight matrix of feature transformations in each layer. As shown in Figure 16, NGCF concatenates L+1 embeddings $\{e_u^{(0)}, e_u^{(1)}, ..., e_u^{(L)}\}$ after propagating with L layers to obtain the final embedding e_u^* (similar for e_i^*). Finally, it generates the prediction score using the inner product:

$$\hat{y}_{\text{NGCF}}(u, i) = e_u^{*T} e_i^*$$

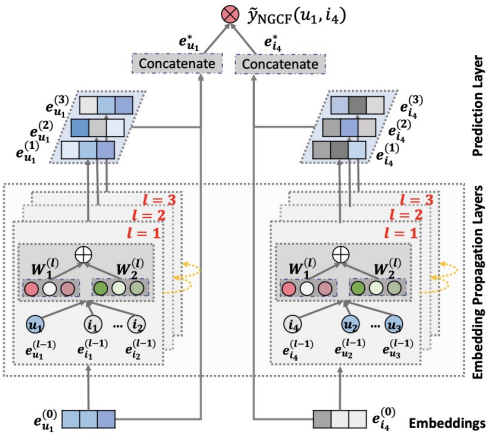

Figure 16: An illustration of NGCF model architecture (the arrowed lines present the flow of information). (Wang et al., 2019)

4.4.2 Light Graph Convolution Network

In order to avoid overfitting and reduce the computation resources, Light Graph Convolution Network (LightGCN) simplifies GCN by removing feature transformation and nonlinear activation parts but only keeping the neighborhood aggregation part for collaborative filtering aims. Therefore, the propagation rule in LightGCN is defined as:

$$e_u^{(k+1)} = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} e_i^{(k)},$$

$$e_i^{(k+1)} = \sum_{u \in N_i} \frac{1}{\sqrt{|N_u||N_i|}} e_u^{(k)}.$$

Moreover, it uses linear weighted sum to combine the embeddings obtained at each layer to generate the final representation:

$$e_u^* = \sum_{k=0}^{K} \alpha_k e_u^{(k)}; \quad e_i^* = \sum_{k=0}^{K} \alpha_k e_i^{(k)};$$

where $\alpha_k \geq 0$ denotes the importance of the k-th layer embedding. The structure of LightGCN is shown as Figure 17. Such a simple, linear, and neat model is much easier to implement and train.

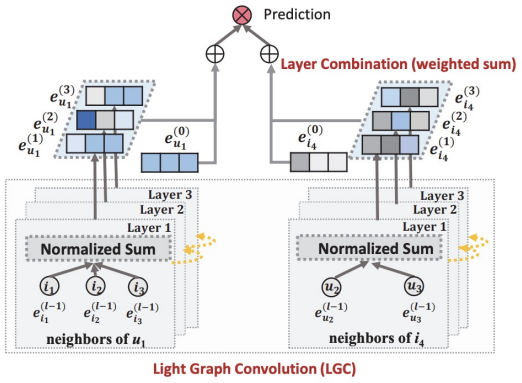

Figure 17: An illustration of LightGCN model architecture (He et al., 2020)

5 Evaluation and Results

5.1 Evaluation Metrics

To compare different recommender systems, it's essential to select proper metrics to evaluate the results. According to(Silveira et al., 2019), we can explore a number of metrics to compare these models' accuracy, fairness, coverage, and more. We mainly focus on accuracy so we pick the following metrics according to our dataset and the survey(Wu et al., 2020).

HR measures the proportion of users who have at least one click on the recommended items:

$$\textbf{HR}@K = \frac{1}{|\mathcal{U}|}\Sigma_{u\in\mathcal{U}} I\left(\left|R^K(u) \cap T(u)\right| > 0\right)$$

where $T(u)$ denotes the ground truth item set, $R^K(u)$ denotes the top-K recommended item set, and $I(\cdot)$ is the indicator function.

Precision, **Recall** are widely adopted to evaluate accuracy of top-K recommendations. Precision@K measures the fraction of the items the user will click among the recommended K items. Recall@K measures the proportion of the number of user clicks in the recommended K items to the entire click set.

$$\text{Precision@K(u)}=\frac{\left|R^K(u) \cap T(u)\right|}{K}$$

$$\text{Recall}\,@K(u) = \frac{\left|R^K(u) \cap T(u)\right|}{|T(u)|}$$

The overall metric is the average over all the users, e.g.,

$$\text{Precision@K}= \frac{1}{|\mathcal{U}|}\Sigma_{u\in\mathcal{U}}\ \text{Precision}\ @K(u)$$

NDCG differentiates the contributions of the accurately recommended items based on their ranking positions:

$$\text{NDCG@K}= \frac{1}{|\mathcal{U}|}\Sigma_{u\in\mathcal{U}} \frac{\sum_{k=1}^{K} \frac{I\left(R_k^K(u)\in T(u)\right)}{\log(k+1)}}{\sum_{k=1}^{K} \frac{1}{\log(k+1)}}$$

where $R_k^K(u)$ denotes the k^{th} item in the recommendation list $R^K(u)$.

5.2 Results

Metrics of results are listed in Table 4 and Table 5.

Model	HR@K	Prec@K	Rec@K	NDCG@K
POP	0.034	0.004	0.013	0.008
MF	0.036	0.004	0.012	0.008
VAE	**0.104**	**0.013**	**0.034**	**0.026**
NGCF	0.07	0.008	0.025	0.017
LGCN	0.084	0.010	0.027	0.019

Table 4: Results (K = 10): With Recbole

Model	Prec@K	Rec@K	NDCG@K
NDR	0.168	0.060	0.081
UIER	0.168	0.060	0.110
MF	0.241	0.071	0.150

Table 5: Baseline Results (K = 10): With Scikit-surprise

6 Discussion

The performance of all the methods is close to our expectations. For the approaches to reducing matrix dimensionality, VAE outperforms MF in all evaluation metrics. That's because VAE introduces non-linear probabilistic methods with denoising functionality, it is much better than MF which only uses linear calculations. In addition, MF does not beat the baseline, POP, which may be caused by the high sparsity in the ZhihuRec dataset. As for the approach to learning user-item interaction with the bipartite graph structure, LightGCN outperforms NGCF in all cases although it has a lighter and more concise structure, which is consistent with the results in the original LightGCN paper(He et al., 2020). One reason might be that LightGCN fits the training data better than NGCF since NGCF is too heavy to reflect the overfitting problem. In this project, VAE is the one with the best results, which implies the efficiency of statistical methods.

We've noticed that the evaluation results for MF differ significantly in the two packages. Therefore we investigated and compared the implementation of the evaluation metrics in the two packages. The definitions of the metrics are different in these two packages so we can't directly compare the performance. Because of the privacy policy, we're unable to get the original text and look into the recommendation results, which makes it even harder to figure out how the model performs on the end-user side. To better understand how well our model performs, we'll need to get more information from the Zhihu platform (e.g. how well their recommender system works).

7 Conclusion

In this project, we performed various methods to build a recommender system for a brand-new dataset ZhihuRec, which is collected from the largest and most popular socialized knowledge-sharing community in China. Three models (Normal Distribution Recommender, Popularity Recommender and User-Item Effects Recommender) were chosen as baseline models. We also performed MF, VAE, NGCF, and LightGCN as high-level models and compared their results. It turned out that all of the high-level models other than MF performed better than baseline models and VAE outperformed others according to the evaluation metrics.

8 Other Things We Tried

We initially planned to use GNN to dig out the relationship between users according to the common topics/users they followed. We've successfully created the graph we need, and we attempted to train the DiffNet using the graph and other information we have. But we failed to train the network using the RecBole-GNN[4] package. We've checked the code and figured out it was caused by the package implementation. Unfortunately, we're unable to create a new model pipeline from scratch due to the limited time we have. But still, it's worth exploring the GNN performance within users' information.

9 Reflection and Prospect

- Our project used two different packages at first. Due to the different definitions and implementations of the evaluation metrics, we found it difficult to compare the model performance across them. To avoid this, we should keep the models implemented within one framework.
- Unfortunately we failed to implement the network using the user graph. But intuitively the commonly followed topics should have insightful information on users' common preferences. There is still a lot of space we can explore in how should we define the user graph and includes it in our network.
- The dataset we used in this project is significantly sparse. We're using different kinds of encoders to generate latent representations in our project. Another interesting research question is, with limited computing resources, how should we handle the sparse dataset?
- VAE outperforms GNN methods in our project. This result reminds us that neural network is not necessarily effective in all situations. We should not rely too much on these "black boxes". On the contrary, it is more practical to analyze the problem from the root and build some mathematical models.

References

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495.

Simon Funk. 2006. Netflix update: Try this at home.

Bin Hao, Min Zhang, Weizhi Ma, Shaoyun Shi, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2021a. A large-scale rich context query and recommendation dataset in online knowledge-sharing.

Bin Hao, Min Zhang, Weizhi Ma, Shaoyun Shi, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2021b. A large-scale rich context query and recommendation dataset in online knowledge-sharing. *arXiv preprint arXiv:2106.06467*.

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*, pages 639–648.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. *CoRR*, abs/1708.05031.

Liang Hu, Jian Cao, Guandong Xu, Longbing Cao, Zhiping Gu, and Wei Cao. 2014. Deep modeling of group preferences for group-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Yehuda Koren. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1–24.

Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference*, pages 689–698.

Alan Said, Brijnesh J Jain, and Sahin Albayrak. 2012. Analyzing weighting schemes in collaborative filtering: cold start, post cold start and power users. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 2035–2040.

Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112.

Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. 2019. How good your recommender system is? a survey on evaluations in recommendation. *International Journal of Machine Learning and Cybernetics*, 10(5):813–831.

[4] https://github.com/RUCAIBox/RecBole-GNN

Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

Quoc-Tuan Truong, Aghiles Salah, and Hady W Lauw. 2021. Bilateral variational autoencoder for collaborative filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pages 292–300.

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174.

Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. 2020. Multi-component graph convolutional collaborative filtering. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 6267–6274.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2020. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys (CSUR)*.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.