A hybrid filtering movie recommendation system

Zhengqi Dong, Yuntian He
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{dong.760, he.1773}@osu.edu

1 Introduction

With ever-growing volume of information in the Internet, people start to have hard time in finding the content that they actually want to see, including news, articles, musics, movies, and etc. Recommender Systems(RSs) is a software tool that provides suggestions for a item to be use to the user. [5] In everyday life, human will be heavily rely on the recommendation systems to discover the most interesting and valuable information. The need to build a robust movie recommendation system is extremely important given the huge demand for personalized content of modern consumer. An example of recommendation system is such as this:

- User A watches Game of Thrones and Breaking Bad.
- User B does search on Game of Thrones, then the system suggests Breaking Bad from data collected about user A.

In general, there are three type of RSs: 1) Content-based filtering: Content-based filtering relies on the "similarity" between two items being recommended. If a user like an Item-A, then the user tend to in favor of a item similar to Item-A. 2) Collaborative Filtering(CF): The CF use the assumption that the use who have agreed in the past tend to agree in the future. For example, if the User D bought an Item-1 and Item-2. In the meantime, User-A, B, C also bough these two items along with another item Item-3, then it is recommended that Item-3 will be worth purchasing for User-D as well. The CF system can be further categorized to Memory-based CF (e.g., User-based CF, Item-based CF) and Model-based CF (e.g., Matrix factorization, Clustering CF, Neural Network-based CF). 3) Hybrid filtering: A Hybrid filtering refers to the RSs that combines the techniques from former two. [6, 7]

The CF is the most popular and widely adopted techniques for building the recommendation systems, but there are still many challenges and limitations remain unsolved, such as highly sparse data in very large product sets, the scalability for growing users and items, responsiveness for newly uploaded items and the actions taken by user, shilling attacks, and etc. [6]

In this project, we propose a hybrid filtering movie recommendation system that combines the idea of content-based filtering and collaborative filtering and integrates the representation learned from multiple sources. Section 2 will give a formal problem statement for movie recommendation. Section 3 will introduce the framework and each module's functionality. Section 4 presents our experiments on a real-world movie datasets including metadata of more than 45,000 movies and 26 million ratings.

2 Problem Statement

Here is a formal formulation of the **Movie Recommendation Problem**. Given a set of users $\mathcal{U}$, movies $\mathcal{M}$, rating score field $\mathcal{S}$, and a rating record set $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{M} \times \mathcal{S}$, we aim to provide user $u \in \mathcal{U}$ with recommendations on movies in $\mathcal{M}$ that he has never watched. In general, there are two kinds of queries for this problem:

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

- **Rating prediction:** The goal is to learn a function $f_s : \mathcal{U} \times \mathcal{M} \rightarrow \mathcal{S}$ to predict the rating that user u would give to an unseen movie m.
- **Top-k recommendation:** The goal is to learn a function $f_t : \mathcal{U} \rightarrow \mathcal{M}^k$ to predict k unseen movies in which user u might be interested.

Note that the second query can be answered from the first one by simply sorting all movies with their predicted ratings and confidence scores. Since it is hard to find the ground truth data of top-k recommendation, in this project we focus on the first query for the problem.

3 System framework

This section introduces the functionalities of the modules of our hybrid filtering recommendation system. Figure 1 shows the architecture of this workflow and the following subsections describes the individual modules in our framework. Note that our framework has great flexibility since each module is independent, one can try multiple alternatives for the desired performance on each module.

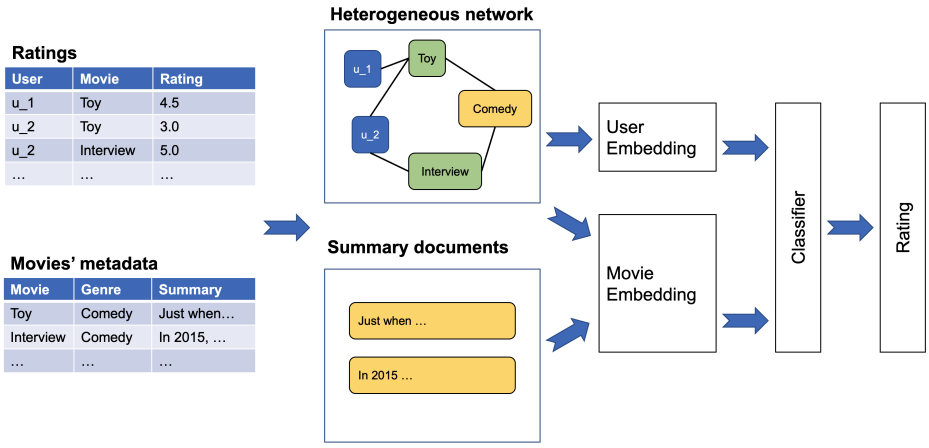

Figure 1: System architecture

3.1 Text embedding

A movie metadata usually contains multiple types of textual information, including genres, tags, and overviews. Here the goal is to learn representations of such textual data to describe the movies. The intuition is to recommend in a *content filtering* way such that two movies may be liked by the same user if their content is similar.

In most cases, genres and tags are words or phrases which is too short and usually used as attributes rather than a text corpus. Longer textual data such as overviews may represent the main content of each movie, therefore it is reasonable to leverage it to learn text embeddings. Doc2vec [4], BERT [1], and CNN [3] are common options for text embedding in data mining tasks. Finally, the text embedding module will learn a e_{mt}-dimensional vector for each movie.

The text embedding algorithm that we use for out implementation is Doc2vec, which was known as Paragraph Vector in [4]. Doc2vec text embedding is a unsupervised algorithm that can learn fixed-length feature representation from variable-length of text, such as sentences, paragraphs, and documents. It consist of several advantages when considering the type of text embedding algorithm in our project. First, the Doc2vec algorithms is capable in learning vector representation from unlabeled data and generalize well for data that do not have enough labels. Second, Doc2vec takes into consideration of word orders while learning the semantic meaning of documents.

3.2 Graph contextual embedding

In the metadata of movies, items are associated with attributes of different types. They may have nominal attributes including genres and production companies, interval attributes such as release date, ratio attributes such as budgets and revenues. Leveraging these attributes is expected to facilitate the representation learning of movies.

Graph is a structure that can describe the complex relationship among objects. Motivated by the success of representation learning of heterogeneous information network (HIN) [2], we created an HIN with users (**U**), movies (**M**), and its selected attributes, i.e., genres (**G**) and casts/crew (**C**). The model first performs random walks in the HIN and feeds the sequence of nodes to a skip-gram model to learn node embeddings. The intuition is if two users appear together in the sampled walks for many times, their representations are expected to be very close in the embedding space. Since the random walk contains different types of nodes, this model recommends in a hybrid way. To capture both the structural and semantic correlations in the HIN, this model use three metapaths: U-M-U, U-M-C-M-U, and U-M-G-M-U.

A challenge for directly using traditional information network embedding method in this setting is that the interaction between users and movies are usually signed (or labeled), which is represented as ratings. Ratings are numerical values having unique semantics, i.e., a score of 5 is always more positive than 1. To tackle this, we design a novel rating-aware sampling policy which ensures that ratings associated with neighboring moves in a random walk are close with high probability. Suppose a random walk under the metapath U-M-U starts from s_0 and traverses through s_1, the policy will sample s_2 under this distribution:

$$\begin{aligned} & P(s_2 = U_k | s_1 = M_j, s_0 = U_i) \\ = \quad & softmax\{-|R(U_i, M_j) - R(U_k, M_j)|\} \end{aligned}$$

Finally, the graph contextual embedding module will learn a e_{ug}-dimensional vector for each user and a e_{mg}-dimensional vector for each movie.

3.3 Classification

The last module is a classifier which takes the learned embeddings of users and movies in the first two modules as input and predict a rating score. A common option for the classification module is MLP.

4 Experiments

This section will introduce our dataset as well as experimental setup, and evaluate the performance of our proposed system.

4.1 Dataset

This project uses the Movies dataset from Kaggle[1] containing 26 million ratings from 270,000 users on all the 45,000 movies in the dataset. Besides, the metadata of movies has more than 20 attributes for each movie. In this project, we only use genre, cast, and crew as movie attributes.

One important step in our data preprocessing is to reduce the number of attributes for movies. The intuition is that one movie may contains tens or hundreds of casts and crews, but users may be aware of only the most important ones. We reduce the number of cast of each movie to up to 8, and only use director as each movie's crew attribute.

4.2 Experimental setup

4.2.1 Baselines and metrics

We evaluate the performance of our proposed system with different configurations. We use three different ways to generate the movie embeddings:

[1] https://www.kaggle.com/rounakbanik/the-movies-dataset. The original dataset was released by MovieLens website, https://grouplens.org/datasets/movielens/latest/. It is a project run by GroupLens, a NLP research lab at the Univeristy of Minnesota.

- Use text embedding (TEXT)
- Use graph embedding (GRAPH)
- Use both text and graph embedding (BOTH)

We evaluate the performance of these methods using following metrics:

- Mean absolute error (MAE)

$$\text{MAE} = \frac{\sum_{i=1}^{n} \left| Y_i - \hat{Y_i} \right|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{1}$$

- Mean squared error (MSE)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y_i} \right)^2 = \frac{\sum_{i=1}^{n} e_i^2}{n} \tag{2}$$

- Accuracy (ACC)

$$\text{ACC} = \frac{\text{Total correct predictions}}{\text{Total predictions}} * 100 \tag{3}$$

4.2.2 Hyperparameters for representation learning

We use doc2vec [4] and metapath2vec [2] as our text and graph contextual embedding modules, respectively. For metapath2vec, we specifically design the metapaths and rating-aware sampling policy introduced in Section 3.2. We set $e_{mt} = e_{mg} = e_{ug} = 128$ for the dimensionalities of learned embeddings. We set the number of random walks for each user to 60 and the length of each work is 50. For other hyperparameters, we follow the choices used in their paper for the best performance.

4.2.3 Miscellaneous

The rating records are randomly divided into training (90%) and test (10%) set. Only the training set is used to learn embeddings. We use a three-layer MLP as our classifier, the output size of each layer is 128, 32, and 10. The classifier is trained for 5 epochs, and the learning rate is 10^{-3}. The experiments are conducted on a single node of OSC Owens cluster.

4.3 Evaluation

This subsection will investigate the performance of different methods and the impact of parameters on its performance.

4.3.1 Overall performance

Table 1 shows the results of all methods where the batch size is set to 1000. It shows that method BOTH outperforms TEXT and GRAPH in terms of all metrics, which is expected since BOTH leverages both textual and graph contextual information from the dataset. It takes longer time for training because the input for BOTH's classifier is larger.

We note that the gap between BOTH and the other two methods is not very large, which can be explained by the fact that the training set is very big - it has 23 million records for training, which alleviates the disadvantage of using only single information source (either text or graph). In addition, all methods use the user embeddings learned from the graph embedding module, which capture each user's preference well.

Table 1: Overall Performance

Name	MAE	MSE	ACC	Time (sec)
TEXT	0.738	1.092	32.014	735.380
GRAPH	0.719	1.061	33.043	740.422
BOTH	**0.713**	**1.034**	**33.052**	781.779

4.3.2 Impact of batch size

In this subsection, we vary the batch size from 500 to 10000 to observe the impact of batch size on the performance of each method. The results are shown in Figure 2.

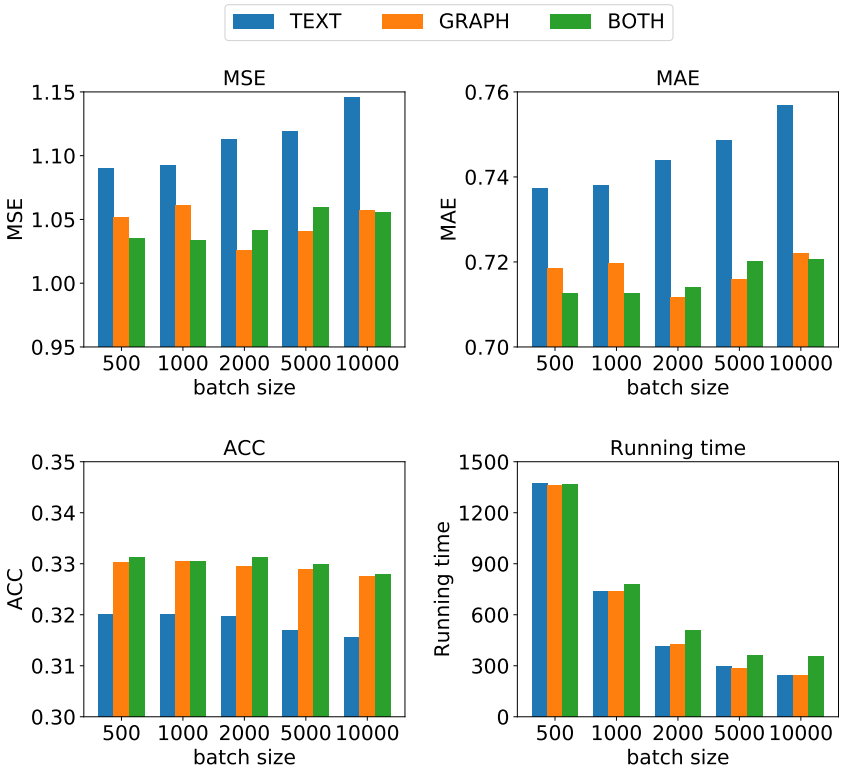

Figure 2: Performance with varying batch sizes

We found that when the batch size increases, the value of ACC (or MSE/MAE) of each method slightly decreases (or increases). While the change is very limited, which demonstrates our method's robustness. In addition, when the batch size is larger, each method runs expectedly faster.

5 Conclusion

In this project, we design a hybrid filtering movie recommendation system which provides flexibility for module selection. We explore the performance of text embedding method by using it as our text embedding module, and design unique metapaths and a novel rating-aware sampling policy for graph embedding. Leveraging both text and graph contextual information, the system further improves the quality of recommendation compared to methods using single information source.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 135–144, 2017.

[3] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[4] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[5] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

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

[7] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.