2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Movie Recommendation using YouTube Movie Trailer Data as the Side Information

Debashish Roy
Department of Computer Science
Ryerson University
Toronto, Canada
debashish.roy@ryerson.ca

Chen Ding
Department of Computer Science
Ryerson University
Toronto, Canada
cding@ryerson.ca

***Abstract*— The user feedback data such as likes, dislikes, comments on movie trailers posted on YouTube can be a useful information source for movie recommender systems. In this paper, we study the effect of adding the feedback data on trailers as a type of the side information to the movie rating data. We propose a recommendation framework that can integrate the trailer and rating data adopting different integration strategies: integrating all the trailer data as movie features, using sentiment scores derived from the trailer comments as a rating matrix to integrate with the movie rating matrix and treating others as the movie features, or only integrating the sentiment score based rating matrix with the movie rating matrix. Our experiment shows that if we include the movie trailer data, recommendation accuracy is improved. We also find that the most accurate result is achieved if all the trailer feedback data is integrated as movie features. To design our system, we use both Matrix Factorization (MF) and Deep Neural Network (DNN) Models. We find that the DNN model performs better than the MF model.**

Keywords*— *recommender system; deep learning; matrix factorization; sentiment score; knowledge transfer; multi-source

I. INTRODUCTION

Nowadays, the immense volume of accessible online data is creating an information overload problem. It can be a big challenge for users to find items of their interest efficiently and effectively. Recommender systems provide a solution to ease such problems. Movie is one of the most commonly used items when testing recommendation algorithms. MovieLens [1], Netflix [2], IMDB [3] are a few popular movie datasets used in many research papers. In a movie recommender system, on top of explicit ratings, there are various types of implicit rating data we can use such as likes, comments, the watch history. Both types of ratings are commonly used in Collaborative filtering (CF) based recommender systems. In content-based systems, content information is usually defined by various movie features such as title, plot, genre, director, actors, posters, movie clips or trailers. In many hybrid systems [4], these features are treated as the side information to be included into the model built upon the rating matrix.

Movie trailer is one of the movie features, playing a similar role as the plot description or the movie poster, it is the highlight of a movie. Most of the movie trailers have YouTube as the main hosting site. People may watch these trailers before the movie release date and leave comments. These comments reflect on how much they like the trailer, and oftentimes also their expectations on the movie, but not on whether they like the movie itself. After they watch the movie, they may come back to watch the trailer again and leave comments. In this case, these comments may reflect on whether they like the movie. Although in both cases, we cannot equate a user's rating on a movie trailer to the rating on the movie itself, the rich feedback information collected on movie trailers can provide some hint on popularity of a movie and on whether the user likes the movie. It is our goal in this work to investigate whether adding the user feedback data retrieved on movie trailers can improve the movie recommendation result. To the best of our knowledge, most of the studies on YouTube focused on recommending the hosted video itself based on its feedback data, whereas we focus on using the feedbacks on trailers to recommend movies. We would like to study the effect of adding the feedback data on trailers as a type of the side information into the movie recommender system.

In this work, we adopt three strategies to integrate movie ratings with the trailer data. We consider four types of feedback data on trailers: the like count (the number of likes), the comment count (the number of comments), the view count (the number of views) and the sentiment score calculated from each comment. The first three are per movie basis and can be treated as movie (or trailer) features. The last one is per (movie, user) pair basis and is considered as the implicit rating. In the first strategy, we treat all of them as the side information and integrate them with the movie ratings. In the second strategy, we use sentiment scores as implicit ratings to integrate with the explicit movie ratings, ignoring other features. In the third strategy, we use sentiment scores as a rating matrix and the other trailer features as the side information to integrate with the movie ratings. We design the recommender system using both the matrix factorization model and the deep neural network. Our experimental result shows that if we integrate all the trailer feedback data as the side information with movie ratings, we can provide the most accurate result and this finding is observed in both the matrix factorization model and the deep neural network model. We also find that the deep neural network model performs better than the factorization model.

The rest of the paper is organized as follows. Section II reviews the related work. Section III explains the overall system architecture and recommendation strategies used in our research. Section IV provides the details of the experiment design, data collection and preparation steps, and result analyses. Finally, Section V concludes the paper with a summary and future research directions.

II. RELATED WORK

We first review some of the research works that use the matrix factorization (MF) model to implement recommender systems using both the rating data and the side information. A two-level hybrid matrix factorization model is proposed in [5]. It computes the semantic relations between items using weighted textual MF, in which a textual corpus is represented by a term document matrix. In [6], users' friendship data is added as a social regularization term to recommend items.

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 © 2020 IEEE

YouTube recommends personalized videos to its users based on their activities such as videos watched, favored and liked [7]. It uses the personal activity as the seeds and expands the set of videos by traversing a co-visitation graph of videos which is created using the association rule mining.

In recent years, deep learning models have become one of the most popular and effective options to implement recommender systems. In [8], a neural collaborative filtering (NCF) approach is proposed, which generalizes the matrix factorization and models the non-linear relationship between users and items using a multi-layer perceptron (MLP) neural network. To recommend apps in Google Play, a deep neural network architecture "wide and deep learning" is proposed in [9]. The wide learning component is a generalized linear model using a single layer perceptron. The deep learning component is a non-linear model that uses multi-layer perceptron. Deep Factorization Machine in [10] integrates the factorization machine with the MLP. The pairwise and linear interactions between different features are captured by the factorization machine and the deep learning component is used to learn higher-order interactions.

Multiple data sources can offer a richer set of user interaction data and provide a deeper insight in user preferences. In [11], a user profile is built using the cross-linking function in four social networks (Foursquare, Twitter, Instagram, Facebook). In [12], cross-network collaborations are used to recommend YouTube videos. To build a user profile, the system extracts the auxiliary information of users from their corresponding Twitter accounts and uses the user profile to recommend YouTube videos. A multi-source based Cross-network Collaborative Matrix Factorization (CCMF) framework is designed in [13]. Information from one network to another network is transferred by aligning the similar items between the two networks.

Compared with the previous work, especially the work on movie recommender systems and the work on integrating multiple data sources, we use the feedback data on movie trailers as the side information (not the feedbacks on the full-length movies), which means that the ratings from two sources are not of the exact same type. To the best of our knowledge, it is novel to use trailer feedback data in this way. Also, in the past, the side information is usually treated as the item features to be included into the recommendation model. In our work, since the comments can be viewed as implicit ratings, we explore different ways of integrating them. Compared with [13], although we follow their broad learning framework, we take a two-step process – knowledge transfer first and then regularization based on item similarities, whereas they include all regularization terms in one model.

III. Our Porposed Movie Recommender System

We collect the movie rating data from MovieLens and the movie trailer data from YouTube, though theoretically we can get data from any source that has movie ratings or movie trailer data. We propose three approaches to integrate the movie trailer feedback data with the rating data. In the first approach, we only use the sentiment scores calculated from the comments as a rating matrix to integrate with the movie ratings, ignoring other feedbacks. We use VADER [14] to calculate the sentiment scores. A higher score (closer to 1) means that a comment is positive, and a lower score (closer to 0) indicates a negative comment. In the second approach, we use sentiment scores as a rating matrix and the other trailer feedbacks as the side information to integrate with the movie ratings. Here, the side information is infused as a movie trailer feature vector. In the third approach, sentiment scores are also considered as one of the trailer features, and then all the trailer features are infused as a feature vector with movie ratings.

A. System Architecture

Fig. 1 shows the overall system architecture of our movie recommender system.

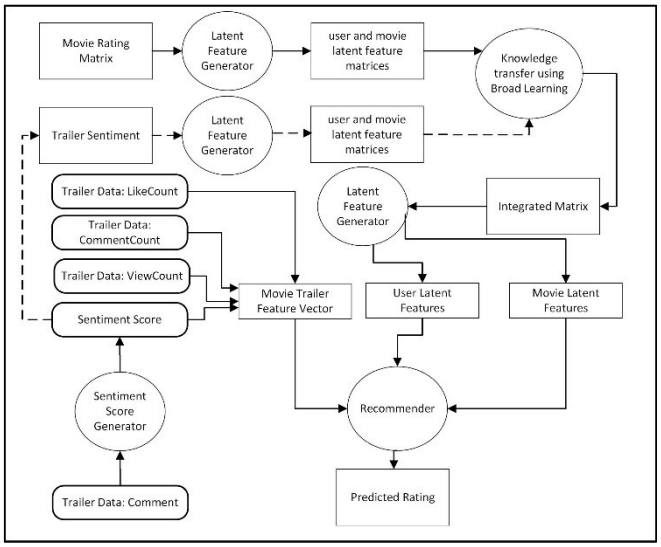

Fig. 1. Overall system architecture of our movie recommender system

We use both matrix factorization and deep neural network to implement the recommendation model. For all the movies, we extract the associated trailer data. The sentiment scores of the comments are used in two ways. In one approach, shown as dotted lines in Fig. 1, it is used as a rating matrix. The two rating matrices are fed into the latent feature generators (implemented using either the MF model or the DNN model, with latent feature vectors as the output instead of the predicted ratings). Then we transfer the knowledge learned from the sentiment-based matrix to the movie rating matrix using a broad learning algorithm [13]. Since users who provide feedbacks on trailers could be different from users who provide movie ratings, it is hard to align users. Therefore, we align movies based on their titles. After the knowledge transfer, we get the integrated matrix (same size as the movie rating matrix). Applying the latent feature generator on this new matrix, we get the final user and movie latent features. In the second approach, sentiment scores are used as movie features. Together with the other three trailer features, they are combined with user and movie latent features to feed into the recommendation component. In this case, the knowledge transfer and matrix integration are not required.

The side information is included in the recommender system through a trailer feature vector. The recommender system takes three vectors as input: a) the movie trailer feature vector, b) the vector for user latent features, and c) the vector for movie latent features. Again, we use either the matrix factorization model or the deep neural network for implementing the recommender. In the matrix factorization implementation, we add a new regularization term, which is based on the similarity between two movie trailers. The assumption is that the similarity between two movies in the latent space is consistent with the similarity based on their trailer features.

The network structure of the deep neural network implementation is shown in Fig. 2. We use one-hot encoding vectors for both users and movies, and numerical values for trailer features. Once we get the dense embedding vectors for users and movies, we concatenate them with the trailer feature vector to create the input vector. In the current implementation, we include three linear transformation layers using the Rectified Linear Unit (ReLU) function.

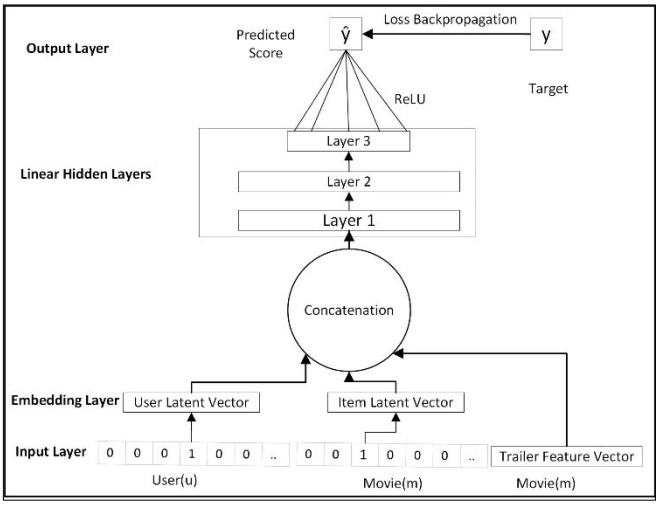

Fig. 2. DNN implementation of the recommender system

B. Matrix Factorization Implementation

The process of matrix factorization starts with a user-item rating matrix R. The size of matrix R is $m \times n$ where m denotes the total number of users and n denotes the total number of items. Matrix factorization decomposes the rating matrix R into two low rank latent feature matrices P for users and Q for items; here, the size of matrix P is $m \times d$ and the size of matrix Q is $n \times d$, d is the rank of the matrices and defines the dimension for the latent features. If $\hat{R}$ represents a matrix of predicted ratings, matrix factorization approximates $\hat{R}$ in such a way that $\hat{R} = PQ^T$. To predict a rating from user u on item i, the inner product between P_u and Q_i is calculated. To decompose a sparse rating matrix, the following objective function is used [9]:

$$L = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in C} \left(R_{u,i} - P_u Q_i^T\right)^2 + \frac{\lambda}{2}\left(\|P\|_F^2 + \|Q\|_F^2\right) \quad (1)$$

In equation (1), C represents the set of (user, item) pairs of known ratings; to avoid overfitting, two regularization terms on the sizes of P and Q are added as constraints and λ is used as a regularization parameter. In this work, for matrix factorization using only the rating data, we use the objective function defined in equation (1). However, this objective function does not include the side information retrieved from the movie trailer data. To include the side information, we derive a regularization term using the trailer similarity score. We define S_{jh} as the similarity co-efficient between trailers of two movies j and h which satisfies: i) $S_{jh} \in [0,1]$; ii) $S_{jh} = S_{hj}$; iii) the larger S_{jh} is, the more similar the movies are. With the similarity co-efficient, the similarity regularization is to minimize the following term:

$$\min \quad \frac{\alpha}{2} \sum_{j=1}^{n} \sum_{h=1}^{n} \left(S_{jh} - Q_j^T Q_h\right)^2 \quad (2)$$

In equation (2), α is used as a regularization parameter. The similarity between trailers of different movies (movie j and movie h) is calculated using the cosine similarity function. Adding the regularization term defined in equation (2) to the previous objective function defined in equation (1), we get the following objective function:

$$L = \min_{P,Q} \frac{1}{2} \sum_{(u,i)\in C} \left(R_{u,i} - P_u Q_i^T\right)^2 + \frac{\lambda}{2}\left(\|P\|_F^2 + \|Q\|_F^2\right) + \frac{\alpha}{2} \sum_{j=1}^{N} \sum_{n=1}^{N} \left(S_{jn} - Q_j^T Q_n\right)^2 \quad (3)$$

We use this updated objective function to design a hybrid recommendation model using rating and movie features.

C. Deep Nerual Network Implementation

In addition to the matrix factorization model, we use deep neural network to model the interaction between users and movies. To learn the interaction patterns, we use a multi-layer perceptron (MLP) with linear transformation layers to form our deep neural network as shown in Fig. 2. The network takes three vectors: i) the movie feature vector, ii) the user latent vector, and iii) the movie latent vector. To format the input for the neural network, we combine the vectors using the concatenation operation. As a simple vector concatenation does not consider the interaction between user and movie latent features, we add hidden layers on the concatenated vector to construct an MLP. In this MLP based neural network, we use both linearity and non-linearity to learn the interaction between users and movies. To add non-linearity, we use the activation function ReLU. We choose ReLU function for the following reasons: it does not suffer from saturation; it is well-suited for sparse data; it helps to avoid overfitting. We also use the deep neural network to generate the latent features in the latent feature generator. The embedding layer is used to map the high-dimensional user and movie one-hot encoding vectors into the low-dimensional dense vectors in the latent space. The number of latent features determines the size of the embedding vector. Here, user and item embeddings have the same size.

D. Knowledge Transfer

In our research, to transfer knowledge between movie ratings and trailers, we align the movies using their titles, and transfer movie latent features between the sources [13]. Assume that S^1 and S^2 are two information sources and M is a movie which is represented as M^1 in information source S^1 and M^2 in S^2. Even if M^1 and M^2 refer to the same movie, as they are from different sources, their latent features could be different and thus we cannot directly set $M^1 = M^2$. L is a matrix that is used to store the information for two matching movies between two information sources. We use this matrix to ensure that only the latent feature vectors of the same item are restricted to be the same and we want to set $L^T M^1 = L^T L M^2$. To make $L^T M^1$ and $L^T L M^2$ to be the same or close to each other, we want to make $\|L^T M^1 - L^T L M^2\|^2$ to be as small as possible. An item latent source adaptation matrix H is used to bridge the differences between S^1 and S^2. Essentially, the transfer of latent features from one source to another is considered as a minimization problem and defined as the following regularization term [13]:

$$\frac{1}{2} \|L^T M^1 H - L^T L M^2\|^2 \quad (4)$$

IV. EXPERIMENT

In this section, we first explain how we design the experiment and how we collect and prepare the data. Then, we show the results that are obtained from the testing phase of the experiment. Lastly we compare different approaches with analyse and discussion on the results.

A. Experiment Design

In this experiment, we investigate our proposed three approaches for data integration and recommendation. We compare the results to identify the most effective solution. To design the recommender system, we use both the deep neural network and matrix factorization models. So, we also compare the performance of the two implementations. Finally, we want to compare our approaches with a few baseline models implemented using deep learning models or matrix factorization models. To run the experiment, we use a computer with Intel core i7 processor 2630QM, 2.5 GHz clock speed, 16GB RAM and Windows-10 as the operating system. The programming language used is Python 3.7 and Python packages used include Torch, Numpy and Scipy. To download the YouTube data, we have used google-api-python-client and the source code for the project can be found on github[1].

B. Dataset Preparation

In this work, we use the MovieLens dataset for the movie rating data. We extracted 7805 ratings for randomly selected 1000 users and 1000 movies which were released between 1998 and 2005. In this dataset, the maximum number of ratings on a movie is 33, the minimum is 2, and the average is 8. For each movie, we extract four types of implicit feedback data from the YouTube movie trailers: i) the comment count, ii) the view count, iii) the like count, and, iv) top 100 comments on the movie trailers given by 1000 users. Note that these 1000 users are different from those from the MovieLens dataset. From YouTube, we extracted a total of 3158 comments from these 1000 users. The maximum number of comments on a movie is 17, the minimum is 1, and the average is 3. In general, there are more ratings from the MovieLens than those from the YouTube.

C. Results and Analyses

a) Performance Optimization on Hyperparameters: In the deep neural network implementation, we need to determine the size of the embedding vector and the number of iterations (step). To determine the embedding size, we run the model on the integrated matrix, testing different values including 10, 20, 30, 40 and 50, and for each run we save the RMSE values. An embedding vector of size 40 gives us the best performance and a similar kind of experiment gives us an optimal step value of 15. So, in the neural network model, we use 40 latent features for both users and movies and iterate it for 15 times. After concatenation, we have altogether 84 neurons: 40 for user latent features, 40 for movie latent features and 4 for movie trailer features. We have three hidden layers in our neural network model: first layer has 84 neurons; the second and third layer have 42 and 21 neurons respectively. In the matrix factorization implementation, we need to decide the optimal values for the parameter d (the number of latent features) and for number of iterations (step). To determine the values, we apply the similar kind of approach that we use to determine the size of the embedding layer and the best performance is achieved when we choose d as 20 and step as 50. So, in this work, we use 20 latent features for the two latent feature vectors and we iterate the model for 50 times. In equation (3), for the regularization parameters α and λ, we choose very small values 0.0002 and 0.02 respectively.

b) Evaluation of Results using RMSE and F1-Score: For evaluation, we calculate the RMSE and F1 scores for top-10 recommended items generated using MF and DNN models. We also have the results for the precision and recall values and all accuracy measures on top 20 items. Since the patterns observed are similar, we only report these two metrics in the paper. Fig. 3 shows the RMSE scores for both models.

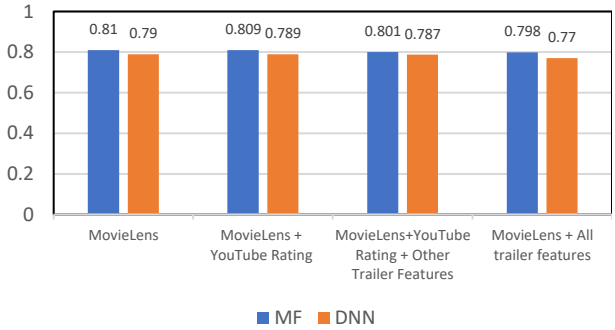

Fig. 3. RMSE for Matrix Factorization and Deep Neural Network models

From the figure, we see that using trailer features can improve the prediction accuracy (smaller RMSE scores). We have the smallest RMSE scores when we use all the trailer feedback data as the movie features. When we use the trailer sentiment scores as ratings and integrate them with movie ratings, the result is slightly worse. Also, comparing between MF and DNN, DNN models produce better RMSE scores. Fig. 4 shows the evaluation using F1@10 for both models.

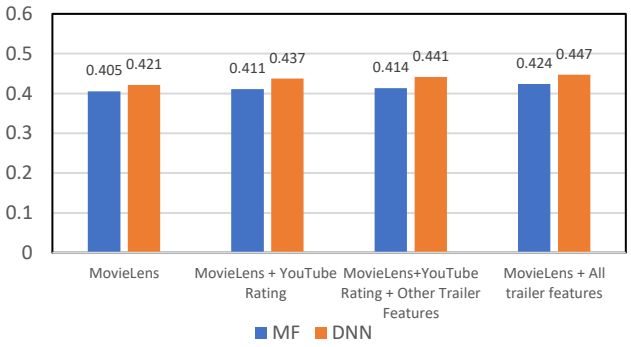

Fig. 4. F1@10 for Matrix Factorization and Deep Neural Network models

From the figure, we see that the highest F1-scores are achieved when using all the trailer feedback data as movie features. The DNN model increases F1@10 by 6.2% when using all trailer features vs. MovieLens rating only. Again, the DNN model produces better results than the MF model.

From these figures, we see that when adding only the sentiment scores as the rating matrix, there is very little improvement compared to the basic model. When adding all the trailer feedback data, adding sentiment scores as another feature shows better result than adding them as a rating matrix. The possible reason could be that when we integrate these features as the side information into the rating-based models (MF or DNN), we use them directly without any data transformation, while the other two approaches require data transformations in the form of generating latent features and transferring the knowledge between two matrices. These figures also show that the DNN model performs better than

[1] https://github.com/movieReco/hybridrecommender

the MF model. For example in Fig. 3, comparing with the MF model, the DNN based model lowers the error by 2.47% for the basic model, 2.53% when sentiment scores are added as a rating matrix, 1.75% when sentiment scores are added as a rating matrix with other features, and 3.5% when we add all the trailer feedback data as movie features.

We compare our best-performing DNN model (using all the trailer feedback data as features) with some of the baseline algorithms, including the basic MF model, SVD, SVD++, NMF [15], NCF [8], RBM [16] and SAR [17]. For all of the baselines, we take their default parameters. Fig. 5 shows the comparison based on the RMSE scores.

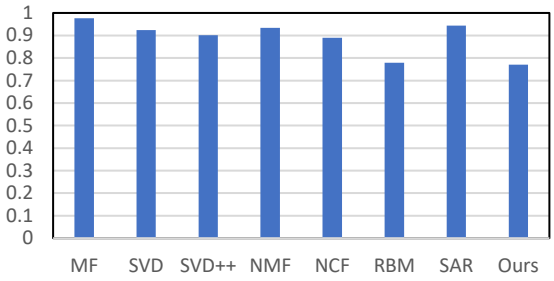

Fig. 5. Comparing with baseline algorithms using RMSE

From the figure, we see that our model has the smallest RMSE score compared to the baselines. Out of all the baseline methods, RBM has the lowest error rate and our model lowers that error by 1.18%. Fig. 6 shows the comparison of our model with these baseline algorithms on F1@10.

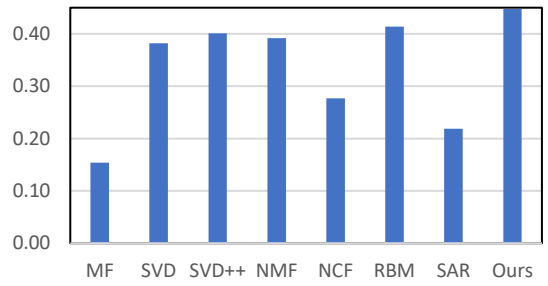

Fig. 6. Comparing with baseline algorithms using F1@10

The figure shows that our model has the most accurate result compared to the other models in terms of the F1-scores. Out of the matrix factorization models, SVD++ generates the most accurate result while our model improves it by 11.5%. Out of the neural network models, RBM generates the most accurate result, while our model improves it by 8.1%.

In terms of the running time, for our dataset, MF-based model takes longer time than the DNN-based model. When adding the trailer feedback data, we record longer running time (~35% longer) from both models compared to the case when the original model is used without the side information. This is expected considering the extra processing required.

V. CONCLUSION AND FUTURE WORK

This work evaluates the effectiveness of adding movie trailer feedbacks as the side information to the movie rating data for movie recommendation. To integrate the trailer data, we have used three approaches: integrating all of them as movie features; treating sentiment scores as a rating matrix to integrate with the movie rating matrix and others as the movie features; only integrating the sentiment rating matrix with the movie rating matrix. Overall, the evaluation results show that if we include movie trailer data, it reduces the prediction error and increases the recommendation accuracy. As for the way of integration, if all the trailer feedback data is integrated as the movie features, our recommender system provides the most accurate result. We also find that deep neural network model performs better than the matrix factorization model.

In future, we want to extend our system to add temporal signals as the side information. As users' criteria to find a movie and user preferences on a movie may change over time, if we can add the temporal signals in our model, we might be able to recommend movies based on users' changing interests. We also want to try different DNN models to implement our recommender system. For example, instead of MLP, we may try the CNN model, or the RNN model if we consider the sequence information (time of ratings).

REFERENCES

[1] F. M. Harper and J. A. Konstan, The movielens datasets: History and context, Acm transactions on Interactive Intelligent Systems, vol. 5, no. 4, pp. 1-19, 2015.

[2] J. Bennett and S. Lanning, The netflix prize, In the Proceedings of KDD Cup and Workshop, pp. 35, 2007.

[3] IMDb, [Online]. Available: https://datasets.imdbws.com/. [Accessed 1 12 2019].

[4] P. Lops, M. de Gemmis and G. Semeraro, Recommender systems handbook, Springer, pp. 1-186, 2011.

[5] F. Li, G. Xu and L. Cao,Two-level matrix factorization for recommender systems, Neural Computing and Applications, vol. 27, no. 8, pp. 2267-2278, 2016.

[6] Z. Sun, L. Han, W. Huang, X. Wang, X. Zeng, M. Wang and H. Yan, Recommender systems based on social networks, Journal of Systems and Software, vol. 99, pp. 109-119, 2015.

[7] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston and D. Sampath,The YouTube video recommendation system, In the Proceedings of the fourth ACM conference on Recommender systems, pp. 293-296, 2010.

[8] X. He, L. Liao, H. Zhang, L. Nie, X. Hu and T. S. Chua, Neural collaborative filtering, In the Proceedings of the 26th international conference on world wide web, pp. 173-182, 2017.

[9] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir and R. Anil, Wide & deep learning for recommender systems, In the Proceedings of the 1st workshop on deep learning for recommender systems, pp. 7-10, 2016.

[10] H. Guo, R. Tang, Y. Ye, Z. Li and X. He, DeepFM: a factorization-machine based neural network for CTR prediction, arXiv preprint arXiv:1703.04247, 2017.

[11] A. Farseev, L. Nie, M. Akbari and T. S. Chua, Harvesting multiple sources for user profile learning: a big data study, In the Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 235-242, 2015.

[12] M. Yan, J. Sang and C. Xu, Unified YouTube video recommendation via cross-network collaboration, In the Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 19-26, 2015.

[13] J. Zhu, J. Zhang, L. He, Q. Wu, B. Zhou, C. Zhang and P. S. Yu, Broad Learning based Multi-Source Collaborative Recommendation, In the Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 1409-1418, 2017.

[14] C. J. Hutto and E. Gilbert, Vader: A parsimonious rule-based model for sentiment analysis of social media text., In Eighth international AAAI conference on weblogs and social media, 2014.

[15] H. Lee, J. Yoo and S. Choi, Semi-supervised nonnegative matrix factorization, IEEE Signal Processing Letters, 17(1), pp. 4-7, 2009.

[16] R. Salakhutdinov and A. a. H. G. Mnih, Restricted Boltzmann machines for collaborative filtering, In the Proceedings of the 24th international conference on Machine learning, pp. 791-798, 2007.

[17] B. Sarwar, G. Karypis and J. a. R. J. Konstan, Item-based collaborative filtering recommendation algorithms, In the Proceedings of the 10th international conference on World Wide Web, pp. 285-295, 2001.