Movie Recommendation System

Jin Yao
519030910343

Zhesheng Xu
519030910353

Yanyu Huang
519030910358

I. Introduction

Online video and movie has grown by leaps and bounds in the last few years. New Internet giant like Netflix gradually takes the first place over traditional film and television industry. With video sites competing for user's attention, a feature that comes in handy in attracting users and ensuring repeat business is movie recommendation.

In this project, we combine Content-Based Filtering and Collaborative filtering methods to provide a better recommendation. Besides, we try several fancy algorithm and test the performance of them.

II. Preliminaries

In this section, we introduce the dataset used and the basic notations of the project.

A. Dataset

The dataset used in the article is MovieLens [1], which consists of the following files and contents,

- Ratings data: Each line of the data file after the header row represents one rating of one movie by one user, and has the following format, userId, movieId, rating, timestamp (e.g. 1, 1193, 5, 978300760). Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- Tags data: Each line of the data file after the header row represents one tag applied to one movie by one user, and has the following format: userId, movieId, tag, timestamp (e.g. 2, 60756, Highly quotable, 1445714996).
- Movies data: Each line of the data file after the header row represents one movie, and has the following format, movieId, title, genres (e.g. 1, Toy Story (1995), Animation|Children's|Comedy). Errors and inconsistencies may exist in movies titles. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
- Links data: Each line of the data file after the header row represents one movie, and has the following format, movieId, imdbId, tmdbId (e.g. 1,0114709,862).

B. Notations

According to the dataset, we may extract the following structured information.

Firstly, we define the time, user, movie and movie category space, as is listed below,

1) time space (in seconds): $\mathcal{T} = \{1, 2, 3, \cdots, T\}$
2) user space: $\mathcal{U} = \{1, 2, 3, \cdots, U\}$
3) movie space: $\mathcal{M} = \{1, 2, 3, \cdots, M\}$
4) movie category space: $\mathcal{C} = \{1, 2, 3, \cdots, C\}$

Secondly, we restructure the user rating logs, as,

1) users' ratings matrix $\boldsymbol{R} = (r_{u,m})$: user $u \in \mathcal{U}$ rates movie $m \in \mathcal{M}$ with score $r_{u,m}$. If the user u never rates the movie m, it is set as nan. Otherwie, each element is of a value ranging in $[0, 5]$.
2) time matrix of users' ratings $\boldsymbol{W} = (w_{u,m})$: user $u \in \mathcal{U}$ rates movie $m \in \mathcal{M}$ at time $t \in \mathcal{T}$. If the user u never rates the movie m, it is set as nan.
3) movies rating $\overline{r}_m$:users' weighted average rating of movie $m \in \mathcal{M}$, calculated as,

$$\overline{r}_m = \frac{\sum_{u\in\mathcal{U}} [r_{u,m} \cdot (\frac{\sum_{m\in\mathcal{M}} r_{u,m}\mathbf{1}_{r_{u,m}\neq nan}}{|\{r_{u,m}|\forall m\in\mathcal{M}, r_{u,m}\neq nan\}|})^{-1}]}{\sum_{u\in\mathcal{U}} (\frac{\sum_{m\in\mathcal{M}} r_{u,m}\mathbf{1}_{r_{u,m}\neq nan}}{|\{r_{u,m}|\forall m\in\mathcal{M}, r_{u,m}\neq nan\}|})^{-1}}$$

4) movie's categories matrix $\boldsymbol{L} = (l_{m,c})$: 1 if movie $m \in \mathcal{M}$ has label $c \in \mathcal{C}$ and 0 otherwise.

III. Movie-Based Recommendation

A. Min-Hash Approach

As is introduced during lectures, the min-hash algorithm mainly focuses on the similar items detection. Taking advantage of such an idea, we characterize the features of each movie based on the user ratings of it, which is defined as,

$$\boldsymbol{vec}_{MH} = (vec_{MH,m,u})$$

where, the value of each element $vec_{MH,m,u}$ indicates the user $u \in \mathcal{U}$-'s rating of movie $m \in \mathcal{M}$ and is given by,

$$vec_{MH,m,u} = \begin{cases} 0 & \text{if} \quad r_{u,m} = nan \\ 0 & \text{if} \quad 0 \leq r_{u,m} < 3 \\ 1 & \text{if} \quad 3 \leq r_{u,m} \leq 5 \end{cases}$$

Therefore, each movie $m \in \mathcal{M}$ is represented by the row vector $vec_{MH,m}$.

Based on the movie vector representation, in practice, the recommendation is provided from the following steps,

- Look back and get a user's last movie index, say $m \in \mathcal{M}$.
- Extract the movie's vector $vec_{MH,m}$.
- Find the most similar movie vectors, say, $vec_{MH,m_1}, vec_{MH,m_2}, \cdots, vec_{MH,m_i}$, where i is the number of recommendation results and $m_1, m_2, \cdots, m_i \in \mathcal{M}$.

- Take movies $m_1, m_2, \cdots, m_i$ as the Recommended movies result.

todo

B. Clustering Intuition

Borrowing the idea of clustering algorithms, we may cluster all the movies into several clusters and regard the cluster neighbours of a given movie as its similar ones.

To characterize all the movies, in contrast to the $\boldsymbol{vec}_{mh}$ used above in Section III-A, which highlights user history, here, we leverage the internal movie information, i.e., their categories. The movie vector is defined as,

$$\boldsymbol{vec}_{CLT} = (vec_{CLT,m,c})$$

where, the value of each element $vec_{CLT,m,c}$ indicates the movie $m \in \mathcal{M}$ is of category label $c \in \mathcal{C}$ and is given by,

$$vec_{CLT,m,c} = \begin{cases} 1 & \text{if} \quad l_{m,c} = 1 \\ 0 & \text{if} \quad l_{m,c} = 0 \end{cases}$$

Therefore, each movie $m \in \mathcal{M}$ is represented by the row vector $vec_{CLT,m}$.

Based on the movie vector representation, firstly, we conduct the EM algorithm to extract movie clusters. Therefore, it is assumed that, the distribution of all movie vectors obey Gaussian distribution. Meanwhile, we set the centroid of each cluster by the its mean value.

Then, the recommendation is extracted from the clusters through the following steps,

- Look back and get a user's last movie index, say $m \in \mathcal{M}$.
- Extract the movie's row vector representation $vec_{CLT,m}$.
- Find the cluster that the given row vector falls in with the highest probability.
- Take all the movie vectors $vec_{CLT,m_1}, vec_{CLT,m_2}, \cdots, vec_{CLT,m_n}$ in the cluster found above as candidates, where $m_1, m_2, \cdots, m_i \in \mathcal{M}$.
- For each candidate movie vector vec_{CLT,m_j}, where $1 \leq j \leq n$ assign its weight by its average rating r_{m_j} and its l_2 distance to the cluster centroid.
- According to the weight of each candidate movie vector, sort in a descending order and recommend the first i movies as the results, where i is the number of required movie recommendations.

C. Time Sequence Analysis

As is introduced in Section III-A, III-B, in recommendation results generation, both the similarity-based and clustering-based ideas focus on user's most recent movie. However, in reality, user's movies history might follow a certain pattern. For instance, the one who lately have already finished Star War I, II and III, has a great probability to turn to Star War IV. Therefore, we ought to make use of the time sequence information of all movie logs.

An intuitive approach is that, similar to item prediction in the e-shopping recommendation scenario, train and predict using the movie indices sequence.

As for the implementation, we use the Long-Short-Term-Memory (LSTM for short).

LSTM is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications. [2]

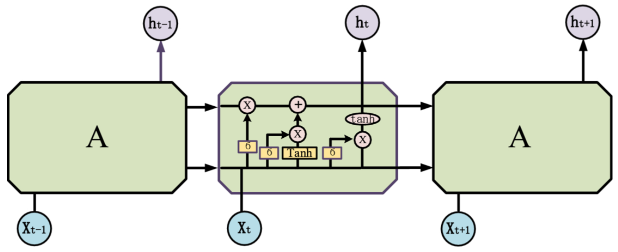

Fig. 1. LSTM Architecture [3]

As is illustrated in Figure 1, the compact forms of the equations for the forward pass of an LSTM unit with a forget gate are,

- $f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$
- $i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$
- $o_t = \sigma_g(W_o x_t + U_f h_{t-1} + b_f)$
- $\widetilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$
- $c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c}_t$
- $h_t = o_t \circ \sigma_h(c_t)$

where,

- $x_t \in R^d$: input vector to the LSTM unit
- $f_t \in R^h$: forget gate's activation vector
- $i_t \in R^h$: input/update gate's activation vector
- $o_t \in R^h$: output gate's activation vector
- $h_t \in R^h$: hidden state vector also known as output vector of the LSTM unit
- $\widetilde{c}_t \in R^h$: cell input activation vector
- $c_t \in R^h$: cell state vector
- $W \in R^{h \times d}$, $U \in R^{h \times h}$ and $b \in R^h$: weight matrices and bias vector parameters which need to be learned during training

As for the training of the above LSTM architecture, we split each user's rating history $m_1, m_2, \cdots, m_n$ into $n - N_{seq} - 1$ sequences $s_1, s_2, \cdots, s_{n-N_{seq}-1}$ of length $N_{seq} \in$

Z, where $m_1, m_2, \cdots, m_n \in \mathcal{M}$, $n \in Z^+$ is the number movies in the user's rating history and $\forall 1 \leq i \leq n - N_{seq} - 1$, $s_i = (m_i, m_{i+1}, \cdots, m_{i+N_{seq}})$. For each sequence s_i, assign its ground truth prediction as $m_{i+N_{seq}+1}$. With feeding the input training data into the network, the model is trained.

In the prediction, after parsing the target user's recent N_{seq} movie history into a sequence of a similar format as described above, we are able to acquire the most possible movie index that the user will turn to next.

D. Time Sequence Analysis and Clustering

However, notice that, the time sequence analysis introduced above in Section III-C uses the sequence of movie indices only. However, such a field is far from informative, comparing with movie categories, ratings etc. Meanwhile, altough we might extract as many predictions as possible, by using the model several times, the outputs are more and more biased, since they highly depends on the previous predictions. As a result, we combine the clustering and the RNN LSTM intuition, as is described below.

Firstly, modify the movie vector $\boldsymbol{vec}_{CLT}$ by adding users' average ratings, as $vec_{CR} = (vec_{CR,m,c})$, where $vec_{CR,m,c} = l_{m,c} \cdot \overline{r}_m$ and $m \in \mathcal{M}, c \in \mathcal{C}$. Therefore, each movie is now represented by a row vector $vec_{CR,m}$ of length $|\mathcal{C}|$.

Secondly, replace the elements (movie indices) m_i $(1 \leq i \leq n)$ in movie sequence described above with its modified movie vector $vec_{CR,m}$. Based on the movie vector representation, similar to Section III-B, with assuming a normal distribution, we form a cluster of all the movies. Meanwhile, we may train a LSTM network, except that, all the elements in the sequence is a row vector of length $|\mathcal{C}|$ instead of a $1-d$ movie index.

In the actual recommendation, we are informed of the user's the most recent N_{seq} movies' indices $m_1, m_2, \cdots, m_{N_{seq}} \in \mathcal{M}$ and then extract the corresponding row movie vectors $vec_{CR,m_1}, vec_{CR,m_2}, \cdots, vec_{CR,m_{N_{seq}}}$ as the input of the trained neural network. The LSTM network replies with a row vector $vec_{CR,m'}$ of length $|\mathcal{C}|$, characterizing the properties of the movie that the user is the most likely to turn to next. Notice that, possibly, $\forall m \in \mathcal{M}, vec_{CR,m} \neq vec_{CR,m'}$, since the "movie vector" is simply an characteristic indication rather than an actual implication of a particular movie, which is exactly the reason why we ought to leverage both the LSTM network and the clustering. With the help of the trained clusters, we are able to discover the cluster that the prediction $vec_{CR,m'}$ falls into. Furthermore, the group of real movies, i.e., its cluster neighbours, are treated as the recommendation results.

IV. User-Based Recommendation

Here we use collaborative filtering algorithm to recommend films based on users. The main idea of this algorithm is to recommend favorite movies of a group users who have similar interest with the target user. Therefore, the algorithm can be divided into two steps, finding similar users and choosing suitable movies.

A. Finding Similar User

We use the users' ratings matrix $\boldsymbol{R}$ to define the attributes of each user. In other words, we measure the interest of each user by his movie record history.

Then we use cosine distance to measure the similarity of two users. However, different users may have different scoring criteria. The users' similarity may be affected by this personal standard. So we normalize rating bias of each users. We first calculate the average movie score of each user as baseline. And we normalize the bias by subtracting baseline for all scored movies in one user.

Suppose we have two users a and b, then these two users are represented by two rows in $\boldsymbol{R}$. The similarity can be written as

$$Sim(a,b) = \frac{|(\boldsymbol{R_a} - Avg_a) \cdot (\boldsymbol{R_b} - Avg_b)|}{\sqrt{|(\boldsymbol{R_a} - Avg_a)| \times |(\boldsymbol{R_b} - Avg_b)|}}$$

Based on the formula above, we can get user-similarity matrix representing the similarity of each pair of users with value between zero and one.

After we calculate the user-similarity matrix, we can sort the row of target user and pick top K most similar users.

B. Choosing Suitable Movies

After we find top K most similar users, we can count the frequency of each movie seen by similar users. Besides, we weight the frequency with user's similarity value to balance the differences in similar users group for a better recommendation.

C. Pros and Cons

User-based collaborative filtering can capture the social relationship between users. However, this method suffers "cold start" problem. It means when user hasn't seen too many movies, we can't provide fitting recommendation based on similar users.

Therefore, we combine the Min-Hash approach to prevent this problem. For a user who has only seen less than five movies, we will just find similar movies in dataset for recommendation rather than finding similar users.

V. Movie-User-Based Recommendation

We use Neural Collaborative Filtering (NCF)[4] to combine user and movie data. It combines generalized matrix factorization and multi-layer perception to predict the probability of the interaction between users and movies.

A. Pre-process and Embedding

We use the same pre-processed data in Min-Hash approach. The input to NCF system is in (user, movie, label) tuple form. If one user scored one movie greater than two, then the label in one. Else the label will be zero.

For the ID of each user or movie, we first embed it into two different latent vectors separately for matrix factorization and multi-layer perception later.

B. Generalized Matrix Factorization

Suppose we have user u and movie i. The embedding latent vectors are p_u and q_i. We perform element-wise product on these two vectors followed by an identity function f, then we can get the result by matrix factorization model.

$$y_{out} = f(\mathbf{1}^{\boldsymbol{T}}(p_u \odot q_i))$$

In neural network, we can use non-linear activation function instead of original identity function to reinforce the expression ability. Besides, we can assign weights on different latent vector dimension learned from input data. This is the improved generalized matrix factorization.

C. Multi-Layer Perception

This is classical architecture in deep learning. The full-connected layer and activation layer are built in sequence. The final result can be written as

$$y_{out} = f(\boldsymbol{W}(p_u, q_i) + \boldsymbol{b})$$

D. Combination of GMF and MLP

In NCF, two operations above are executed in parallel. The result of these two methods are then concated together to learn the weight of different latent vector dimensions. This can be written as

$$y_{out} = \sigma((\alpha h^{GMF}, (1-\alpha)h^{MPL})^T(p_u \odot q_i + f(\boldsymbol{W}(p_u, q_i) + \boldsymbol{b})))$$

Here we use two different weight vectors h^{GMF} and h^{MPL} to model varying importance of latent dimensions. And and α is a hyper-parameter determining the trade-off between the two weights. The whole architecture of NCF is shown below

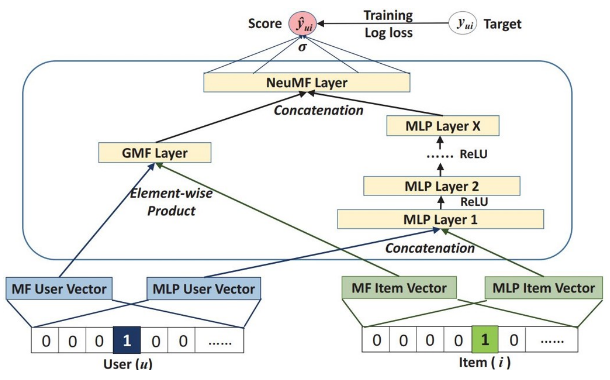

Fig. 2. NCF Architecture

E. Implementation Details

According to origin paper, we use ReLU as the activation function of MLP layers. And the last layer of NCF is sigmoid activation function σ. When training, we adopt the Adaptive Moment Estimation (Adam) with binary cross entropy loss.

F. Prediction and Recommandation

After we train NCF, we can use the model to predict probabilities for recommendation. When we input one user ID and one movie ID, NCF will predict the probability of the interaction between user and movie. The higher the probability is, the more likely the user will watch this movie in future.

Therefore, when given one user ID, we predict the probabilities for watching all movies in dataset and sort them. Then we choose top K movies that the target user hasn't watched and recommend these movies to him.

VI. Experiments

We split MovieLens dataset into train and test set by 8:2 ratio. For one specific scoring record, it will be divided into train set with 0.8 probability. We use test set to measure the performance of different algorithm.

A. Evaluation Metrics

In this project, we use accuracy and recall rate to evaluate the performance.

1) Accuracy: Accuracy represents the ratio for all recommended movies in user's test set. Suppose we have an user u, the test movie set for u is T_u, the recommended movie list is R_u. Then the accuracy for one user is

$$Acc(u) = \frac{|T_u \cap R_u|}{|T_u|}$$

Here $|\cdot|$ means the number of elements in set. The total accuracy for all test users is the average accuracy.

$$Accuracy = \frac{1}{|U|}\sum_{u \in U} Acc(u)$$

2) Recall Rate: Recall rate represents the ratio for all user's test set in the recommended movies. Same as accuracy, the calculation is

$$Recall\ Rate = \frac{1}{|U|}\sum_{u \in U}\frac{|T_u \cap R_u|}{|R_u|}$$

B. Evaluation Result

The accuracy and recall rate for different algorithms is listed in the table I below.

C. Result Analysis

We can find in the table that the performance of LSTM+Clustering and NCF is bad. So we take a deeper analysis on these two methods.

Method	Accurate@10 (%)	Recall Rate@10 (%)
Random	0.2649	0.8775
Min-Hash	4.1225	8.0298
LSTM+Clustering	0.5629	1.5728
Collaborative	6.4901	16.6887
NCF	2.0695	0.5629

TABLE I
Accuracy and Recall Rate

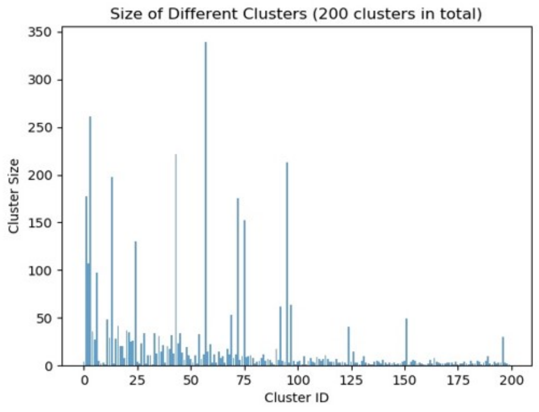

Fig. 3. Sizes of 200 Clusters

1) LSTM+Clustering: We first check the result of clustering, we draw the number of different clusters in the figure 3 below (200 clusters in total). The distribution of different cluster elements is unbalanced. Most clusters have only a few movies while several clusters include a great number of movies. This may lead to the bad performance. When we use the cluster result to recommend movies, most movies may belong to the same cluster or some movies belong to one-element cluster.

Since our clustering algotirhm is mainly based on the label of movies in dataset, we check the distribution of different labels and find that the root of the problem exists in the label numbers. Almost half movies in dataset has the Drama label. This leads to a poor clustering performance.

2) NCF: The feedback from users in recommendation system can be divided into two kinds, explicit one and implicit one. The explicit feedback includes the score or rank given by users for movies. The implicit feedback is mainly about the browsing records or favorite lists from user's behavior.

The MovieLens dataset is the explicit feedback with score of each movie ranked by user. However, Neural Collaborative Filtering mainly targets for implicit feedback (browsing records). Therefore, when using NCF to recommend movies, explicit feedback degrades to implicit feedback. The lost information leads to a poor performance.

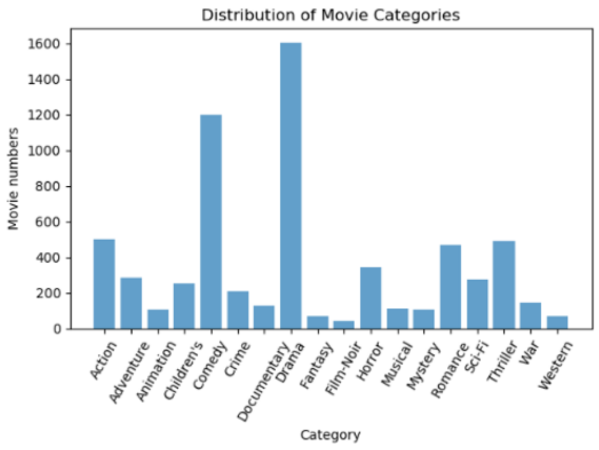

Fig. 4. Label Distribution

VII. Conclusion

In the project, focusing on the topic of movie recommendation, we start from what is introduced during lectures and reach out for some advanced knowledge, such as RNN, LSTM, Collaborative Filtering and Neural Collaborative Filtering. Even though, considering the astronomically large workload in the late semester and prohibitively overwhelming difficulty encountered while handling the data, the final results are not satisfactory enough, we acquire hands-on experience on these knowledge and corresponding deeper insight, which is quite worthwhile, as for the overall struggles.

References

[1] "Movielens dataset," GroupLens. [Online]. Available: https://grouplens.org/datasets/movielens/
[2] "Long short-term memory," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory
[3] J. Hu, X. Wang, Y. Zhang, D. Zhang, M. Zhang, and J. Xue, "Time series prediction method based on variant lstm recurrent neural network," Neural Processing Letters, vol. 52, no. 2, pp. 1485–1500, 2020.
[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in Proceedings of the 26th International Conference on World Wide Web, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 173–182. [Online]. Available: https://doi.org/10.1145/3038912.3052569