STAT 4850G/9850B FINAL PROJECT

Recommender System

Jia Liang, Jiachen Yan, John Dominic Tomaszewski, Kecheng Xiao, Ruiying Zhang, Xinyi Zeng

Group number: 10

ARTICLE HISTORY
Compiled April 6, 2020

Abstract
A recommender system is a system that predicts the future preferences of items to users. We are in an era of information explosion, and people face multiple choices when it comes to making a decision. This article will start from two typical ways for building a recommender system, Content-based Recommendation, and Collaborative Filtering. The first one would rely on the nature of items; the second one would make the recommendation based on users' past behavior. We would then combine the knowledge of deep learning, applying a state-of-art model Neural Collaborative Filtering to the movie and music datasets. Because of the changing of users' preference and updating of the items, there would not be any explicit and firm methods to evaluate the model, here the RMSE, precision, recall, and the nDCG would be applied to do the evaluation.

KEYWORDS
Recommender system, Content-based Recommendation, Collaborative Filtering, Matrix Factorization, Neural Collaborative Filtering, prediction

1. Introduction

We now live in an era of big data. Compared to the past time, we are exposed to more products, more advertisements, and other entertainment resources. Having a good recommendation system would provide significant chances to businesses, educators, and customers of all kinds. It is a mutually beneficial system, the customers get what they want while the business providers achieve the maximum profit. Based on the motivation to create a win-win situation for both the business side and customer side, we become interested in the recommender system and decide using several advanced methods to build it.

We are using two dataset.One is MovieLens, and another one is LastFM. Detail provided in following table.

Table Statistics of the MovieLens-LastFM datase

Dataset	Num.users	Nums.items	Nums.rating	Nums.genres
MovieLens	600	9000	100836	18
LastFM	1892	17632	92834	11946

There are several datasets available from MovieLens; The MovieLens Latest Dataset (Small) is the one we will use. This dataset includes 100,836 ratings, and was chosen for its accessibility: it is compact enough to work with on a personal computer efficiently. Similar conditions applied to LastFM dataset.

MovieLens is an explicit dataset, as it gives a qualitative score (out of five stars) for each movie a user has reviewed. LastFM is similar to MovieLens dataset, except the rating is not given; instead, it uses weight to show users' preferences to the artists. We would demonstrate how we proceed with this in the following part. They are both suitable for Matrix Factorization, SVD, MLP and NeuMF model, as we can easily create latent factors out of the users and items.

MovieLens provides a clear genre of classification to do a content-based filtering recommendation system. Lastfm has no precise musical style classification for musicians; instead, the dataset provides the tags file based on how the user defines the style of musicians, which has no standard to follow and is less accurate. It causes difficulty in building up a similarity matrix.

MovieLens dataset is widely used for the recommendation system's study and development, from traditional methods such as content-based techniques (MJ Pazzani 2007), Collaborative filtering(JB Schafer 2007), to more advanced model such as NeuralCollaborative filtering(X He, 2017). They explore the data well and provide very instructive learning. Dataset LastFM is not so commonly used, so it leaves us more space to do the data mining. Also, we aware that the choices of parameters would have a significant influence on the performance of the model, so we would also do parameter optimization.

Just as "No Free Lunch Theorems for Optimization" (Wolpert, D. H., Macready, W. G. (1997)), we cannot obtain a perfect model for the recommendation system. Also, there is always a tradeoff between model performance and training time. However, we could gradually improve it by combining state-of-art techniques.

2. Notation and Model

- Models:
  - CBF: *Content-based Filtering*
  - MF: *Matrix Factorization*
  - SVD: *Singular Value Decomposition*
  - MLP: *Muti-Layer Perceptron*
  - NeuMF: *Neural Matrix Factorisation*
- Evaluation Methods
  - MAE: *Mean Absolute Error*
  - RSME: *Root Mean Squared Error*
  - DCG: *Discounted Cumulative Gain*
  - IDCG: *Ideal Discounted Cumulative Gain*
  - NDCG: *Normalized Discounted Cumulative Gain*
- Avoid overfitting techniques
  - SGD: *Stochastic Gradient Descent*
  - ALS: *Alternating Least Squares*
- Respone
  - Prediction

3. Methodology

3.1. CONTENT-BASED FILTERING

Content-based recommendation system is the most commonly used item-item recommendation system.
It predicts users' preferences through the attributes of the items. Before the prediction, we obtain users' evaluations or ratings of previous items. This technique can produce different kinds of models to find similar types of items. The model has some advantages: No other users' information is required, and the characteristics of the recommended items are listed, benefiting the users with special interests.

Simple explanation of model CBF construction

model	Basic Steps
CBF	(1) Obtain the genres of each movies and from a genre matrix by applying TF-IDF. (2) Calculate the cosine similarity between each movie by the genre matrix. (3) Using deep learning to train the CBF model (4) Predict the potentially interested movies and their ratings, then compare to the actual (5) Repeat STEP 5 with more batches (6) Evaluation

3.1.1. TF-IDF

"Term Frequency-Inverse Document Frequency" is the numerical representation of a corpus. In this case, TF-IDF transforms the genre matrix into a numeric "genre" matrix. At each entry, one of {0,1} is documented to show either this movie belongs to this genre type or not (1 for yes, 0 for no).

3.1.2. Cosine Similarity

Cosine Similarity Matrix: Firstly, every row of the TF-IDF matrix is considered as a vector. If all vectors are placed on a vector space, the cosine value of each two vectors can be easily obtained by the angle formed by those. Every cosine similarity will be recorded in the corresponding entry. The form of the resulting matrix is similar to the correlation matrix.

3.2. COLLABORATIVE FILTERING

Collaborative filtering is one of the traditional methods for building a recommender system. It recommends items based on one user's past behavior. There are two types of collaborative filtering. One is user-based; it measures the similarity between users. Another one is content-based; it measures the similarity between items a user rated and other items. Collaborative filtering usually performs better than another traditional method, a content-based approach. The key intuition behind this method is that similar items will be liked by users and that similar users tend to have the same taste of items.

3.3. MF MODEL

3.3.1. Matrix Factorization

Matrix Factorization is a traditional recommender system, gaining its popularity because of its relatively good results. The technique was widely introduced by Simon Funk's blog in a Netflix Price competition back in 2006 and has since become a staple in the recommender system world. Matrix factorization recommendation systems often outperform classic nearest-neighbor approaches since MF can incorporate additional information other methods struggle with, such as implicit rating (Koren et al., 2009). We would explain the concept of MF and demonstrate it in python using the package "Surprise."

We would first process the data and create the unitarity matrix, and then we decompose it to the User Matrix and the Item Matrix. Matrix Factorization technique means that we would simply multiply these two matrices to reconstruct the unitarity. Just as other traditional methods, matrix factorization use RMSE (Root Mean Square Error) as a metric to compute the loss. Here we would explain the latent factor.
During the decomposing process, we would project the user and item into two k-dimension latent vectors.

$$\hat{y}_{ui} = f\left(u, i | \mathbf{p}_u, \mathbf{q}_i\right) = \mathbf{p}_u^T \mathbf{q}_i = \sum_{k=1}^{K} p_{uk} q_{ik}$$

Each dimension would be a feature of user or item, for users, they could be gender, characteristics, for items, for example, the movie, they could be genres, orientation to children or adults. The interaction is modeled as the inner product of the latent vectors, and the different number of dimension K would have different results.

3.3.2. Singular Value Decomposition

The most basic matrix factorization model is the multiplication of two latent factor matrices, W^T and H. However, the model is always being developed to encounter more complicate situations. For example, in Huang's post (Huang, 2018), he mentioned singular Value Decomposition (SVD), which was invented independently by Eugenio Beltrami and Camille Jordan. In SVD, the utility matrix, R, can be expressed as

$$\hat{R} = W^T \Sigma H$$

where W^T is the latent factor matrix of user with dimension $(n*k)$ and H is the latent factor matrix of item with dimension $(k*m)$. Σ is a diagonal matrix indicating the strength of each latent factor. SVD can be used for dimension reduction since it has the minimal reconstruction Sum of Square Error(SSE), which is monotonically related to Root Mean Square of Error(RMSE). SVD also breaks the limitation of only using explicit feedback. However, SVD has a disadvantage that it has little explanation to why the system recommends an item to a user (Huang, 2018). For example, according to the documentation of surprise.prediction algorithms.matrix factorization.SVD (Hug, 2015), the prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

where b_u and b_i are the biases from user and item, respectively. It also introduces learning rate α and regularization term λ to reduce dimension of latent factor. b_u, b_i, p_u, and q_i can be updated during the iteration:

$$\begin{aligned} b_u &= b_u + \alpha\left(e_{ui} - \lambda b_u\right) \\ b_i &= b_i + \alpha\left(e_{ui} - \lambda b_i\right) \\ p_u &= p_u + \alpha\left(e_{ui} q_i - \lambda p_u\right) \\ q_i &= q_i + \alpha\left(e_{ui} p_u - \lambda q_i\right) \end{aligned}$$

3.4. NCF MODEL

NCF (X. He, 2017) models the user and item interactions in the latent space effectively with a Neural Network. It is a combination of Matrix Factorization and Neural Network based on the idea of collaborative filtering - to predict the future preference based on the existing rating to the items given to the user.

3.4.1. Matrix Factorization

This part has been demonstrated in the MF model.

3.4.2. Multi-Layer Perceptron(MLP)

Multi-Layer Perceptron is the deep learning network which enjoys the advantages of flexibility because it can handle the non-linearity transformation. Like other deep learning methods, the MLP uses backpropagation to do the optimization. For the hidden layers and output layer of MLP, the details are:

$$\phi_l\left(z_l\right) = a_{out}\left(W_l^T z_l + b_l\right), (l = 2, 3, \ldots, L-1)$$

where W_l, b_l, and a_{outt} denote the weight matrix, bias vector, and activation function for the l -th layer's perceptron. We have multiple choices of activation functions of MLP layers, such as the sigmoid, Tanh, ReLU.
We can search for an economical way to construct the MLP network by taking the number of hidden layers and the dimension of latent factors into account, and finally, achieve a preferable result and reasonable computing time. Here, we would apply different activation functions and different number of hidden layers to see the difference in results.

3.4.3. NeuMF

NeuMF is the fusion of Multiple-layer Perceptron (MLP) module and the Generalized Matrix Factorization (GMF) layer. GMF and MLP would learn seperate embeddings, and two models would be combined by concatenating their last hidden layer.
Firstly, We obtain ϕ^{GMF} from GMF:

$$\phi_{u,i}^{GMF} = p_u^{GMF} \odot q_i^{GMF}$$

where p_u^{GMF} denotes the latent factors of users in GMF, and q_i^{GMF} denotes the latent factors of items.
Secondly, we obtain ϕ^{MLP} from MLP.

$$\phi_{u,i}^{MLP} = a_{out}\left(W_L^T\left(a_{out}\left(\ldots a_{out}\left(W_2^T\begin{bmatrix} p_u^{MLP} \\ q_i^{MLP} \end{bmatrix} + b_2\right)\ldots\right)\right)\right) + b_L$$

where p_u^{MLP} denotes the latent factors of users in MLP, and q_i^{MLP} denotes the latent factors of items.
Lastly, fuse output from GMF and MLP. (ϕ^{GMF} and ϕ^{MLP} are the last output of each process)

$$\hat{r}_{u,i} = \sigma\left(h^T\begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}\right)$$

Simple explanation of model construction

model	Basic Steps
MF	Get latent space of users and items, create the unitarity matrix. Reproduce ratings by calculating the dot product(element wise product). Calculate the mean squared errors or other metric to analyze the model.
MLP	Get latent space representation of users and items Do the concatenation of the vectors to get the first MLP layer Apply the deep learning model, choosing different number of hidden layer and activation function
$NeuMF$	Create latent space for user and items for both MF and MLP process. For MF, do the element-wise product of vectors. For MLP, do the concatenation of vectors, apply the standard MLP model. Combine the two models by concatenating their last hidden layer.

3.5. Evaluation Method

Unlike the regression model and classification model, which would have specific evaluation metric, the recommender system would not have clear indicators to show the effectiveness of the model; given the off-line situation, we can not reach live feedback from the users so instead we apply MAE, RMSE and nDCG to evaluate the model.

3.5.1. MAE

"Mean Absolute Error" is the measure of the average magnitude of the errors in a set of predictions and the difference of direction is not considered, i.e. all difference are weighted equally.

$$MAE = \frac{1}{|\hat{R}|} \sum_{\hat{r}_{ui} \in \hat{R}} |r_{ui} - \hat{\boldsymbol{r}}_{ui}|$$

where rating r_{ui} is the true rating and $\hat{r_{ui}}$ is the estimated rating. $\hat{R}$ is a test set of user-item pairs (u, i)

3.5.2. RMSE

Root Mean Squared Error is another metric, similar to the MAE.

$$\text{RMSE} = \sqrt{\frac{1}{|\hat{R}|} \sum_{\hat{r}_u \in \hat{R}} (r_{ui} - \hat{r}_{ui})^2}$$

3.5.3. NDCG

NDCG is the normalized version of DCG given by

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}}$$

where IDCG is the ideal DCG. DCG is a measure of ranking quality. Assuming each user u has a "gain" g_{uij} from being recommended an item i, the average Discounted Cumulative Gain (DCG) for a list of J items is defined as

$$\text{DCG} = \frac{1}{N} \sum_{u=1}^{N} \sum_{j=1}^{J} \frac{g_{uij}}{\max(1, \log_b j)}$$

Usually, the free parameter would set to 2, and there is an alternative form which places a stronger emphasis on retrieving relevant documents:

$$\text{DCG}_\text{J} = \frac{1}{N} \sum_{u=1}^{N} \sum_{j=1}^{J} \frac{2^{g_{uij}} - 1}{\log_2(j+1)}$$

3.6. Approaches to Avoid overfitting

Koren et al. also addressed that Matrix Factorization models aim to take a closer prediction to the actual utility matrix, which means to minimize the loss function $(R - \hat{R})^2$. The model trains on the observed ratings and then predicts unobserved ratings. Therefore, the issue of overfitting should be taken into account. There are two classical approaches to minimize overfitting: Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS) (2009).

3.6.1. Stochastic Gradient Descent

This approach is popularized by Simon Funk in his blog (2006). The approach iteratively picks observed rating r_{ui} and get the prediction $\hat{r}_{ui}.q_i$ and p_u are updated by optimizing the loss function with learning rate α and regularization term λ, which is showed as below:

$$p_u = p_u + \alpha * \left(\left(r_{ui} - p_u^T * q_i\right) * q_i - \lambda * p_u\right)$$
$$q_i = q_i + \alpha * \left(\left(r_{ui} - p_u^T * q_i\right) * p_u - \lambda * q_i\right)$$

3.6.2. Alternative Least Squares

W and H is unknown, so the loss function with regularization term

$$\Sigma_{(u,i)\in\kappa} \left(r_{ui} - p_u^T * q_i\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2\right)$$

is not convex, where κ is the set of (u,i)'s from observed ratings. ALS approach can solve the optimization problem by fixing p_u to compute the gradient of q_i, and then fixing q_i to compute the gradient of p_u. The procedure rotates to update p_u and q_i iteratively until convergence.

Compared to ALS, SGD is more user-friendly and takes less time. However, when the system allows parallelization or uses implicit data, ALS is more favorable (Koren et al., 2009).

4. Data Analysis

- MovieLens:100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. Last updated 9/2018.
- LastFM: 92,800 artist listening records from 1892 users.

4.1. Data Processing

- LastFM: The original rating is not available because people seldom grade the songs(not so common as movies), but they do listen to the songs more if they like the songs. In the original file, the weight is given, so we transform it into the rating and then proceed with the analysis.

  Here is the detail about the transformation:
  - Normalization: Consider that every users spend a different amount of time listening to music, we would normalize the weight according to every user. We would apply the idea of min-max feature scaling, in this case:

$$r_{ui} = \frac{w_{ui} - min(w_{u\_})}{max(w_{u\_}) - min(w_{u\_})}$$

where $w_{u\_}, w_{ui}$ is the weight of user u to artist(i)

- bias: We multiply the normalization result by 5, and then plus 0.5 to make the data more operable.

4.2. Model Performance

4.2.1. CBF Model

Table 1:
output example:Recommendation for movie: Something to Talk About (1995)

- Movies:
  - 11: *American President, The (1995)*
  - 48: *Mighty Aphrodite (1995)*
  - 53: *Postman, The (Postino, Il) (1994)*
  - 84: *Beautiful Girls (1996)*
  - 166: *Something to Talk About (1995)*
- Genres: Comedy,Drama,Romance

Table 2: TF-IDF Matrix

Movie ID	Comedy	Action	Drama	Romance	Animation
166	1	0	1	1	0
11	1	0	1	1	0
48	1	0	1	1	0
53	1	0	1	1	0
84	1	0	1	1	0

Instead of listing all the genres of each movie, the TF-IDF matrix shows the binary result.

Table 3: Cosine Similarity Matrix

Movie ID	166	11	48	53	84
166	1	1	1	1	1
11	1	1	1	1	1
48	1	1	1	1	1
53	1	1	1	1	1
84	1	1	1	1	1

All have the same genre, which causes all entries are 1.

Table 4: Performance of CBF model with different dataset

Dataset with model	MovieLens	Last.fm
MSE	0.675820088	0.475776424
MAE	0.660784175	0.466895859
RMSE	0.822082774	0.689765485

4.2.2. MF Model

A comparison between SVD and NMF (non-Negative Matrix Factorization) model is shown below:

Table 1: MovieLens: RMSE and MSE by number of Latent Factors

Latent space dimension Metric	k=2	k=5	k=20	k=50	k=100	k=150
NMF_RMSE	1.29	1.07	0.94	1.06	1.17	1.23
NMF_MSE	1.65	1.15	0.88	1.12	1.37	1.52
NMF_MAE	1.11	0.88	0.71	0.80	0.91	0.97
SVD_RMSE	0.87	0.87	0.87	0.86	0.86	0.86
SVD_MSE	0.75	0.75	0.75	0.75	0.75	0.74
SVD_MAE	0.67	0.67	0.67	0.66	0.66	0.66

Table 2: last.fm: RMSE and MSE by number of Latent Factors

Latent space dimension Metric	k=2	k=5	k=20	k=50	k=100	k=150
NMF_RMSE	1.00	1.00	0.99	0.98	0.98	0.98
NMF_MSE	1.00	0.99	0.97	0.96	0.96	0.96
NMF_MAE	0.65	0.65	0.63	0.63	0.63	0.63
SVD_RMSE	0.96	0.96	0.96	0.96	0.96	0.96
SVD_MSE	0.92	0.92	0.92	0.92	0.92	0.92
SVD_MAE	0.64	0.64	0.64	0.64	0.65	0.65

4.2.3. NCF Model

Table 1: Performance between different models
Latent Space= 4,Layer_Size = (16,8,4)

Dataset with model	Last.fm.quick neuMF	Last.fm.quick GMF	Last.fm.quick MLP	movie.quick neuMF	movie.quick GMF	movie.quick MLP
Training Time(sec)	802.480	771.969	969.574	522.382	600.154	503.132
Testing Time(sec)	68.781	69.576	69.786	7.946	12.587	13.55
RMSE	2.145	**2.134**	2.178	3.260	3.223	**3.114**
NDCG	0.135	**0.183**	0.151	0.111	**0.131**	0.111
Precision@K:	0.127	**0.162**	0.144	0.107	**0.127**	0.110
Recall@K:	0.106	**0.137**	0.122	0.026	**0.029**	0.028

Table 2: Performance with Pre-training
Latent Space= 4,Layer_Size = (16,8,4)

Dataset	Last.fm.deep without PreTrain	Last.fm.deep PreTrain	movie.deep without PreTrain	movie.deep PreTrain
GMF Training Time(sec)		1158.667		786.785
MLP Training Time(sec)		1266.599		885.265
NeuMF Training Time(sec)	802.48	1565.28	522.382	985.687
Prediction Time(sec)	68.781	66.166	9.212	8.206
RMSE	2.145	**2.144**	3.27	**3.26**
NDCG	0.135	**0.144**	0.099	**0.116**
Precision@K:	0.127	**0.136**	0.093	**0.109**
Recall@K:	0.106	**0.114**	0.044	**0.05**

Table 3: Performance between different models
Latent Space = 10. Layer_Size = (16,8,4).

Dataset with model	Last.fm.quick neuMF	Last.fm.quick GMF	Last.fm.quick MLP	movie.quick neuMF	movie.quick GMF	movie.quick MLP
Training Time(sec)	418.083	360.320	365.808	311.073	258.5	265.68
Testing Time(sec)	53.715	50.905	51.012	7.489	7.544	7.459
RMSE	2.156	**2.148**	2.175	3.281	**3.244**	3.248
NDCG	**0.173**	0.168	0.139	0.112	**0.122**	0.11
Precision@K:	**0.167**	0.165	0.136	0.107	**0.119**	0.107
Recall@K:	**0.07**	**0.07**	0.06	**0.026**	**0.026**	0.025

Table 4: Performance with Pre-training.
Latent Space = 10. Layer_Size = (16,8,4)

Dataset	Last.fm.deep without PreTrain	Last.fm.deep PreTrain	movie.deep without PreTrain	movie.deep PreTrain
GMF Training Time(sec)		360.320		258.5
MLP Training Time(sec)		365.808		265.68
NeuMF Training Time(sec)	418.083	437.59	311.073	306.868
Prediction Time(sec)	53.715	57.808	7.489	7.817
RMSE	**2.156**	2.157	**3.281**	3.298
NDCG	**0.173**	0.13	**0.112**	0.108
Precision@K:	**0.167**	0.129	0.107	**0.108**
Recall@K:	**0.07**	0.054	**0.026**	0.025

4.3. Interpretation

4.3.1. CBF model

By obtaining the results of MSE and MAE, the CBF shows a good ability to predict the rating. The MSE and RMSE in both datasets are in a relatively low range. The absolute average difference between predictions and actuals is in 0.4669-0.6608, comparing to their scale, which is around 8.49% -13.22%. However, the method is not flawless, and the shortage is noticeable. For some users, they may only watch the movies with the best overall rating or the movies of their favorite actors. These types of users encourage database engineers to create more and more attributes to fulfill those needs, such as "Top-Rating", "Johnny Depp" (actor's name), which can be time-consuming and not practical. To solve this, the collaborative Filtering should be applied.

CBF will recommend items that users are typically interested in, although the satisfaction rating may be lower or higher by 10 percent depending on the scenario. In other words, the CBF has space to be improved. The most common way is to generate item-user Collaborative Filtering, and for a new user or user with less information, user-user collaborative Filtering will be a better alternative. However, when the features of items become the main factor of recommendation decision, the CBF is the irreplaceable option(Aggarwal 2013).

4.3.2. MF Model

SVD has better performance than the NMF model regarding RMSE and MSE. In MovieLens, for the NMF model,k = 20 latent factors has the lowest RMSE. For SVD, the model has the lowest RMSE with k = 150 latent factors. These results show that the SVD model provides a more accurate estimate of user movie recommendations, with the square root average rating error being less that one star. In LastFM, NMF model with k =100/150 and SVD model with k = 2/5 have the best performance.

4.3.3. NCF Model

Table 1 shows the performance of timing and some evaluation metrics for two datasets. The training time and testing time for the music dataset are generally more than the movie dataset since the number of users and the number of items of music dataset is more than the movie dataset. The difference between each model is not significant. However, GMF shows the best performance in most metrics except the RMSE for the movie dataset. This result is being quick fit data for each model; we don't train the GMF and MLP models before training the neuMF model.

Table 2 demonstrates the difference between NeuMF with and without pre-training with two datasets. For both datasets, we do find NeuMF with pre-training achieves a better performance according to each metric. The average improvement for RMSE is 0.5%; the average improvement for NDCG is 1.3%; the average improvement for Precision is 1.25%; the average improvement for Recall is 0.7%. From the result table, despite the longer training time, we can verify that the pre-training of GMF and MLP layer might help our NeuMF model perform better.

From Tables 3 and 4, we can tell that the performances do not get improved as we expected as we change the dimension of latent space. In our sense, we may first think that a more advanced and complicated model would demonstrate better results, but our test results show something different. The uncertainty of users' preference, the serendipity, and other random effects can explain these. We can see from the table that when we choose the number of latent factors as 100, the caculation consumption is high, but the performance is not as good as the dimension 4 one. Also, we experiment with a different number of layers and find that as the number of layers increase, the results do not improve either.

We also compare the pre-train performance with larger latent space,i.e., the case with k=10(where k is the dimension of latent space), out of our expectation, the overall performances are still not so good as the case with k=4. The pre-trained performances do not reach the expectations, either. These results indicate that although sometimes the combination of models could have better performance, it might not satisfy every dataset. Sometimes, the GMF model outperforms the NeuMF, and sometimes the MLP would gain better results, which corresponds to the idea we mentioned earlier, No Free Lunch Theorems.

We also choose k = 100 for our model at the first time. The result show the worst among all the result in the tables above. In a word, our final goal is to obtain a relatively accurate and simple model.

5. Discussion

Content-based Filtering is a useful method, while the dataset includes the classifications or the attributes of each item. The other user has less effect in content-based systems comparing to others (Aggarwal 2013). By analyzing the dataset, the CBF method shows a good result by recommending similar types of movies like the one user already likes.

Matrix factorization model is a good example of a cold-start problem (D. Kluver and J.A. Konstan, 2014) because the basic MF models are only predicting the use/item interaction. However, some models, like SVD++, are not a model-based algorithm, which may cause to retrain the whole model when a new user appears. More complicate MF models are then developed to improve the performance, such as sparse linear model (SLIM) and contextual sparse linear model (CSLIM) (R. Burke et al., 2014).

NCF combines the advantages of linear and non-linear, applying both MF techniques and neural network to obtain a better model performance. Nowadays, deep learning techniques are welcomed in many fields and achieve great results. Although we only used the simple multi-layers network and did not achieve great advancements in model performance as well as the model building process, we do learn about how to develop future research directions and benefit a lot.

Besides, we do find the training time, and the testing time is pretty slow when we were running it locally. We can improve this by using cloud computing to train the model, which will be faster and allow us to train larger datasets. Furthermore, to help model perform better. We construct our work using what we learned in advanced data analysis, from interpreting the dataset, searching for instructive materials to choosing proper evaluation methods, and presenting the final results. We do realize our restrains and would keep improving ourselves. Methods such as cross-validation, Precision, Recall, and parameter tuning do help us a lot in the analysis. But what we feel instructive the most is the way of thinking to conduct research.

In the future, we would think about applying other fields' ideas to the recommendation system. The idea of neighbor was widely used in many applications such as dimensionality reduction area so that we could compare the similarity between two fields and use some of their techniques to achieve better results of the recommendation system. We can try to realize using the method as a conditional probability to find the neighbor as they do in the t-SNE. Moreover, maybe develop tree-based algorithms suitable for recommendation systems.

Contributions

At the very beginning, we came up with an idea of building a music recommendation system, since it is an entirely new topic to us, we planned to use the deep learning network to build our model. After obtaining the related base knowledge, we start to consider using multiple methods to do the analysis. We do cooperate during the process, and everyone gives their works.

The articles present three models, Kechen Xiao and Jiachen Yan are working on the CBF model; Kechen Xiao also helps to explain the evaluation methods. Jia Liang and John are working on the MF model, contributing to the optimization algorithms, dataset introduction, and finalizing the article. Xinyi Zeng and Ruiying Zhang work on NCF model, select and process(cleaning) the dataset, explain the evaluation methods, summarize the teamwork, propose our future expectation and finalize the article.

Here is the contribution within the group.

(1) Kecheng Xiao writes the code and literature part of the content-based Filtering. Jiachen Yan writes the literature part of the content-based Filtering.
(2) Jia Liang writes the code and literature part of Matrix Factorization. John Dominic Tomaszewski writes the literature part of Matrix Factorization and Data Introduction.
(3) Xinyi Zeng and Ruiying Zhang work together on the NCF model construction. We first processed the dataset to make it more convenient to reach. Since our model combines the MF model and MLP model, we also contribute to the literature part of MF model. Our codes also contain the MF model but of the different environment of the previous one. And we also analyze the dataset from differing point of view, hope they would make the whole results more interesting! We present the core of our model in the appendix, hope it would help with interpretation.

References

(1) Reference of the real dataset.
MovieLens https://grouplens.org/datasets/movielens/
lastFM https://grouplens.org/datasets/hetrec-2011/

(2) He, Xiangnan, et al. ”Neural collaborative filtering.” Proceedings of the 26th international conference on worldwide web. 2017.

(3) Koren, Yehuda. ”Factor in the neighbors: Scalable and accurate collaborative filtering.” ACM Transactions on Knowledge Discovery from Data (TKDD) 4.1 (2010): 1-24.

(4) Shani, Guy, and Asela Gunawardana. ”Evaluating recommendation systems.” Recommender systems handbook. Springer, Boston, MA, 2011. 257-297.

(5) Y. Koren et al, ”Matrix Factorization Techniques for Recomender Systems”, Aug. 2009;

(6) S. Funk, ”Netflix Update: Try This at Home,” Dec. 2006; http://sifter.org/-simon/journal/20061211.html

(7) Huang, ”Introduction to Recommender System. Part 1 (Collaborative Filtering, singular Value Decomposition)”, Jan. 2018;

(8) N. Hug, ”Matrix Factorization-based Algorithms”, 2015
https://surprise.readthedocs.io/en/stable/matrix_factorization.html

(9) Aggarwal, C. C. (2018). Recommender Systems The Textbook. Cham: Springer International Publishing.

(10) R. Burke et al, “CSLIM: Contextual SLIM Recommendation Algotithms”, Oct. 2014; https://dl.acm.org/doi/pdf/10.1145/2645710.2645756

(11) D. Kluver, J.A. Konstan, ”Evaluating recommender behavior for new users”, Oct. 2014; https://dl.acm.org/doi/10.1145/2645710.2645742

Appendix

Algorithm 1 Matrix Factorization

```
0: procedure MF(num_user,num_iterms,layers)

   {Get latent space of users and items, create the unitarity matrix.}
1: user_embedding_mf= Embedding(num_users, latent_dim)
   item_embedding_mf = Embedding(num_item,latent_dim)

   {Element-wise product of user and item embeddings, multiplication}
2: mf_vector =torch.mul(user_embedding_mf, item_embedding_mf))

3: prediction = Dense(1, activation='sigmoid')(predict_vector)
```

Algorithm 2 MLP

```
0: procedure MLP(num_user,num_iterms,layers)

   {Get latent space of users and items, create the unitarity matrix.}
1: user_embedding_mlp= Embedding(num_users, latent_dim)
   item_embedding_mlp = Embedding(num_item,latent_dim)

   {The 0-th layer is the concatenation of embedding layers}
2: vector =torch.cat(user_embedding_mlp, item_embedding_mlp))

   {MLP layers}
2:   for idx  in  xrange(1, num_layer) do
   layer = Dense(layers[idx], activation='relu')
   vector = layer(vector)

3: prediction = Dense(1, activation='sigmoid')(vector)
```

Algorithm 3 NeuMF

```
0: procedure MLP(num_uses,num_iterms,layers)

1: MF_Embedding_User = Embedding(num_users,latent_dim)
   MF_Embedding_Item = Embedding( num_item, latent_dim)
2: MLP_Embedding_User = Embedding(num_users, output_dim = $layers[0]/2$)
   MLP_Embedding_Item = Embedding(num_items, output_dim = $layers[0]/2$)

   {MF layers}
3: mf_vector = torch.mul(MF_Embedding_User, MF_Embedding_Item)
   {MLP layers}
   mlp_vector =torch.cat(user_embedding_mlp, item_embedding_mlp))
3:   for idx in xrange(1, num_layer) do
   layer = Dense(layers[idx], activation='relu')
   mlp_ vector = layer(mlp_vector)
   mlp_vector = torch.cat([MLP_Embedding_User, MLP_Embedding_Item], dim=-1)

   {Final prediction layer}
4: vector = torch.cat([mlp_vector, mf_vector], dim=-1)
```