Compare Models of Recommendation on Netflix Dataset

Name:Yijun Zhou
Email: yiz209@pitt.edu

ABSTRACT

Even though numerous recommendation technologies have been proposed and used, how to model users' performance better and get more precise evaluation results is still challenging. This paper aims to build recommendation models including matrix factorization model, neural collaborative filtering model, graph-embedding neural network and some improved models. Then this paper compares the performance of each model based on Netflix Dataset, which is a sparse dataset with more than 100 million records of ratings from anonymous users. Finally, this paper proposes several potential research directions in the future.

Keywords

Recommendation system; Matrix Factorization; Deep Learning; Neural Network; DeepWalk; Node2Vec

1. INTRODUCTION

With the development of e-commerce and stream media, it is hard for traditional advertising to satisfy the needs of personal advertising recommendations. Many recommendation technologies have been proposed and many works have been done to improve the performance of recommendation systems.

However, choosing and identifying products that users need most from massive amounts of information is still an important and challenging problem.

There are two fundamental patterns in recommendation system: one is predicting the ratings from users to items, and the other is recommending most related items to users. Both patterns are widely used in industry, but current recommendation technologies require more improvement to make the prediction result more precise and effective.

In part2, I introduced some related recommendation technologies, all of them are classic models, but still suffer from some challenging problems like cold start and sparse problem. Then in part3, I brought a brief introduction of the Netflix Dataset which is used in the Experiment part to compare the performance of different models. In order to be more comprehensive, I will use both explicit feedback dataset and implicit feedback dataset to do evaluation. In part4, I talked about some basic concepts of recommendation models used in the Experiment from the perspective of model principles, hyper-parameters and evaluation methods. In part5, I compared the results in the experiment. Finally, I made the conclusion of all works and proposed several potential research approaches in the future.

2. RELATED WORK

Nowadays, many popular recommendation models are widely used on many platforms. The most traditional method is collaborative filtering, which could be divided into user-by-user similarity-based collaborative filtering and item-by-item similarity-based collaborative filtering.

2.1 Collaborative filtering

User-based collaborative filtering assumes users have a high rating score on the item rated highly by a similar user. While item-based collaborative filtering assumes users have a high rating score on the item, which is similar to the item highly rated by the user. Figure 1 shows the processing of both patterns of collaborative filtering. For user-based collaborative filtering, it's more possible for user_B to give high score on item_A and item_B because user_A gives high score on item_A and item_B and these two users are similar to each other. For item-based collaborative filtering, user_A tends to give high score to item_C because user_A has high score on item_B, and item_B is very similar to item_C.

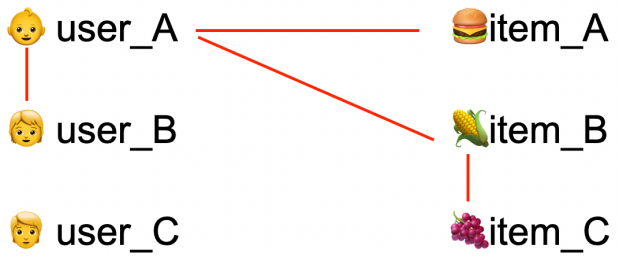

Figure 1: example of collaborative filtering

However, the main disadvantage of collaborative filtering is that it is hard to make precise predictions on the sparse dataset.

2.2 Matrix Factorization Model

Paper [1] introduced a model that combines the latent-factor and neighborhood models. THE Latent-factor model uses embedding to represent users and products directly, while the neighborhood model analyzes the similarities between products or users. The main idea of this method is to divide the rating matrix R into the product of the user matrix and item matrix. By doing this, the item that hasn't been predicted by the user could get a predicted score. However, If we do not gather sufficient information about a user or an item, it is difficult to get a precise score, which is called the cold start problem. It is impossible to solve the cold start problem completely, but some methods could help provide somehow reasonable rating values and update the rating values to be more precise.

2.3 Probabilistic Matrix Factorization Model

Paper [2] proposed PMF(probabilistic matrix factorization) model, which could perform well on the sparse dataset. Based on the matrix factorization model, PMF adds a Gaussian probability function to ensure that users and items with few ratings could get rating scores close to the average value. Also, adding fixed and adaptive prior on user and item vectors could also improve the results. The technologies of matrix factorization perform well in general, but there are still some limitations of this method; one is the ranking loss. To be more specific, the latent vectors of similar users might not perform similarly.

2.4 Neural Network Model

Besides, WEEK9'S paper employed the neural network to model the interaction between user and item features. The general framework of neural network is showed in figure2, in which the input layer is the embedding layer with user latent vectors and item latent vectors, and the output layer is the predicted score from user to item. Under this creative framework, the author implemented three models: Generalized Matrix Factorization(GMF) model, Multi-Layer Perceptron(MLP) model, and the fusion of GMF and MLP model(the combination of the first two models). The only difference between these three models is the Neural CF layers part. GMF model uses the product of corresponding vectors instead of the dot product of matrix factorization model. The MLP model uses the fully connected layer as the multi-layer perceptron.

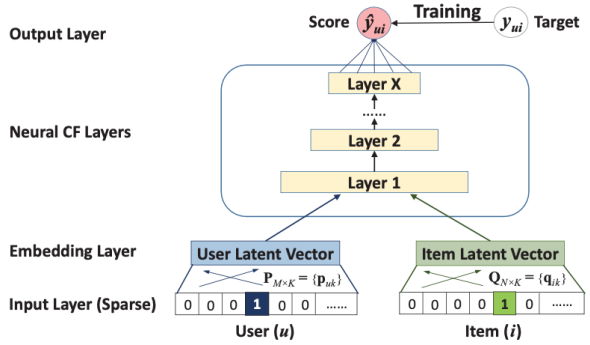

Figure2: Neural collaborative filtering network [2]

As figure3 showed, the third model combines the GMF model and MLP model to get better performance.

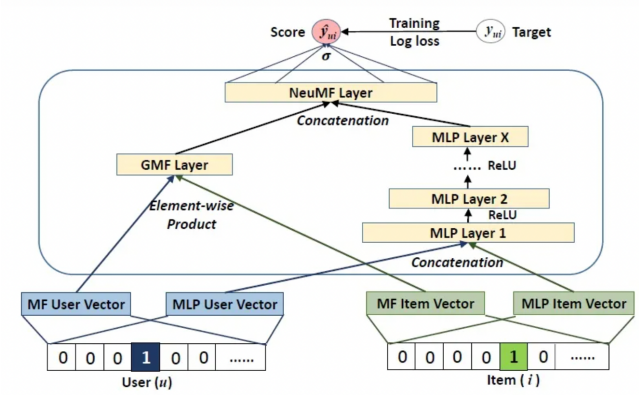

Figure3: neural matrix factorization model [2]

3. DATASET

The Netflix dataset has more than 100 million movie ratings performed by anonymous Netflix customers from 1998 to 2005. Based on this original dataset, I created a subset dataset with more than 5 million movie ratings and divided the training dataset and test dataset according to the ratio of 80 to 20. To validate models' performance on both explicit feedbacks and implicit feedbacks, I created explicit feedback dataset with ratings showed in Table1 and implicit feedback dataset shoed in Table2 where each entry is stored as 0 or 1 to identify it the user rated the movie [2].

There are two important methods used when creating an implicit feedback dataset. The first method is the leave-one-out method, and the second is negative sampling. I will talk about these two methods in part 4 in detail.

4. METHODS

I used matrix factorization model and neural network model as baseline models, and then combined the graph embedding technologies and neural network method to do recommendation. I introduce these 5 models from 4.1 to 4.5, and then in 4.6 and 4.7 I talk about the methods I use to create the implicit dataset.

4.1 Matrix factorization model

According to paper [1], matrix factorization in recommendation could work well in the sparse dataset combining the latent factor model and the neighborhood model. Matrix factorization is to represent users and items in the latent space, so that we could predict ratings based on the latent vector of each pair of user and item. Based on the formula1, P is the user matrix with latent vector for each user, while Q is the item matrix with latent vector for each item. By calculation of the dot product of the two vectors, we could predict the rating of the item from the user. However, some tendencies might affect the effectiveness of this model. For example, some users tend to give high ratings, so that the movies rated by these users tend to get higher rating scores than others. One way to solve this problem is adding the bias to users and items as formula2 showed. The parameter represents the overall average meeting, bi represents the observed deviations of user u and item i [1]. By doing this, the deviation caused by users' personal tendency could be reduced. Combing the formula1 and formula2, we could get the rating formula as formula3.

$$R \approx P * Q^T = \widehat{R} \quad (1)$$

$$b_{ui} = \mu + b_i + b_u \quad (2)$$

$$\widehat{r_{ui}} = b_i + b_u + p_u^T * q_i \quad (3)$$

To learn the best latent embeddings of users and items, the loss function of formula (4) could be used in gradient descent optimization part.

$$min \sum_{(u,i)} r_{ui} - \widehat{r_{ui}} \quad (4)$$

Table 3 shows the processing of matrix factorization model.

Table1. matrix factorization(with bias) model

Algorithm: SVD+bias

INPUT: train_df, test_df, dimensional k, steps, gamma, Lambda
Initialize: pu, qi, bu, bi
shuffle(train_df)
Definite pred_rating function
For step = 1 **to** steps **do**
    For all (user, movie, rating) ∈ train_df **do**
        Pred_rating =pred_rating(user,movie)
        Upgrade bu, bi, pu qi
    For all (user, movie, rating) ∈ test_df **do**
        Pred_rating = pred_rating(user, movie)
    Calculate metrics
Return metrics

4.2 Neural Matrix Factorization Model

The neural network model is the fushion of the GMF model and MLP model [3], so the input layer is the embedding layer with user latent vectors and item latent vectors, and the output layer is the predicted score from the user to item. For the Neural layer, the GMF layer and MLP layers will train the same input embedding separately and concatenate the result in the NeuralMF layer, as figure 3 showed.

TABLE 2: Neural Matrix Factorization Model

```
Algorithm: GMF+MLP

INPUT: train_df, test_df, dimensional k, steps, gamma, Lambda
Initialize: GMF_model, MLP_model
shuffle(train_df)
Define predict function
For step = 1 to steps do
    For all (user, movie, rating) ∈ train_df do
        Pred_rating =pred_rating(user,movie)
        Upgrade GMF_model, MLP_model
    For all (user, movie, rating) ∈ test_df do
        Pred_rating = pred_rating(user, movie)
    Calculate metrics
Return metrics

Predict(user, movie, GMF_model, MLP_model)
Define GMF_model.predict function
Define MLP_model.predict function
Create a layer to concatenate the result of two models
Calculate result
Return result
```

4.3 DeepWalk Neural Network Model

Based on the neural matrix factorization model of 4.2, which uses the neural network model to learn the latent embeddings of users and items, this model aims to use two different models to take responsible of the part of learning embedding and the part of training the neural network separately.

The first step is creating a network based on the interaction of users and items. The nodes are users and items, and the edges are the interactions of users and items. For the Netflix Dataset, there is overlapping between users and items, so adding "U" in front of the users' id and adding "M" in front of the items' id as the nodes' name.

For the embedding part, I use the DeepWalk algorithm to learn the embedding of this model. The DeepWalk model uses truncated random walks first to get walks of each node, and then uses word2vec model to learn the representations of each node [4].

After learning the latent embeddings of all nodes, the next step is a recommendation based on the neural network model. As table6 showed, the input layer of the neural network is the concatenating vector of the user's latent vector and the item's latent vector with dimensionality twice the latent vector.

TABLE 3: DeepWalk Model

```
Algorithm: DeepWalk + Neural Network

LearnFeatures(Graph G=(V,E), Dimensions d, Walks per node r, Walk
length l, Context size k)
Initialize walks to Empty
For iter = 1 to r do
    For all nodes u ∈ V do
        Walk = RandomWalk(G,u,l)
        Append walk to walks
Return Word2vec(walks)

RandomWalk(Graph G'=(V,E), Start node u, length l
Initialize walk to [u]
For walk_iter =1 to l do
    Curr = walk[-1]
    Vcurr = GetNeighbors(curr,G')
    S = random.sample(Vcurr)
    Append s to walk
Return walk
```

TABLE 4: Neural Network Model

```
NeuralNetwork(representations, users, movies,k, batch_size)
Set model = {
Input layer with dimension ((2*k,1));
hidden layer with dimension ((128,48));
ReLU layer();
output layer with dimension ((48,1));}
Return model

Recommendation(model, train_df, test_df)
shuffle(train_df)
For step = 1 to steps do
    model(train_df)
    model(test_df)
    Calculate metrics
Return metrics
```

4.4 Node2vec Neural Network Model

The concept of combining the node2vec and neural network models is similar as 4.3. According to paper [5], node2vec also uses truncated random walks to get walks of each node, but the difference between deep walk and node2vec is that node2vec uses a biased random walk. Node2vec method seeks to preserve local neighborhoods of nodes by developing a family of biased random walks based on both BFS and DFS. BFS explored only the limited neighborhoods, while random walk is a DFS method, which aims to go far away. In Node2vec, the search bias is controlled by the hyper-parameter p and q.

TABLE 5: Node2vec Model

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent($k, d, walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to [u]
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr, G'$)
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$

After learning the representations of nodes, the other steps are same as 4.3.

4.5 Weight-Deepwalk Neural Network Model

This model is generated from the deep walk model. Aiming to take graph characters into account, this model uses the degree centrality of each node as the bias to guide the process of random walk. For nodes who are possible to be the next node, it is easier for node with highest degree centrality to be chosen in the next step. After the process of embedding, the other steps are same as 4.3 and 4.4.

TABLE 6: Weight-DeepWalk Neural Network Model

Algorithm: weightDeepwalk + Neural Network

LearnFeatures(Graph G=(V,E), Dimensions d, Walks per node r, Walk length l, Context size k)
Initialize walks to Empty
Calculate degree centralities of each node
For iter = 1 to r do
  For all nodes u ∈ V do
    Walk = WeightWalk(G,u,l,degree_centralities)
    Append walk to walks
Return Word2vec(walks)

WeightWalk(Graph G'=(V,E), Start node u, length l, degree_centralities
Initialize walk to [u]
For walk_iter =1 **to** l **do**
  Curr = walk[-1]
  Vcurr = GetNeighbors(curr,G')
  S = node for node ∈ Vcurr with highest centrality
  Append s to walk
Return walk

4.6 Negative Sampling

Negative sampling is a method used to add negative samples to the train dataset so that the train dataset could include both positive and negative samples. In this experiment, for training dataset, I chose 10 negative samples for each user to create the implicit dataset; while for test dataset, I choose 100 negative samples for each user when creating implicit dataset.

4.7 Leave-One-Out

Leaving-one-out is a common technology used in machine learning. By splitting dataset into 10 folders, using one folder to do validation is the key concept of leave-one-out method. In this situation, I keep the latest movie of each user in the test dataset.

5. EVALUATION RESULTS

5.1 Metrics

To evaluate the results of the experiment, I use RMSE and MAPE for explicit dataset, and use HR and NDCG for implicit dataset.

RMSE is the root mean square error, and MAPE is the mean absolute percentage error. Both of RMSE and MAPE are used to measure the loss between true values of predicted values. For explicit dataset, using RMSE and MAPE could measure the difference between true ratings and predicted ratings.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}}, \tag{5}$$

$$\mathrm{M} = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right| \tag{6}$$

HR@k is the hit ratio among top k items which are recommended and NDCG@k is the normalized discounted cumulative gain which used to rate the quality of search results. Based on the formula7, the denominator is all test sets, and the numerator represents the sum of the number of test sets in each user's top-k list.

$$HR = \frac{|U_{hit}^{L}|}{|U_{all}|} \tag{7}$$

$$DCG(k) = \sum_{i=1}^{k} \frac{G_i}{log_2(i+1)} \tag{8}$$

$$IDCG(k) = \sum_{i=1}^{|I(k)|} \frac{G_i}{log_2(i+1)} \tag{9}$$

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)} \tag{10}$$

5.2 Experiment Results

After comparing performance of the 5 models, the results are showed in this part. In the figures below, B1 is the matrix factorization with bias model, which is the best model of HW1; B2 is the fusion of MLP and GMF model, which is the best model of HW2; A1 is DeepWalk Neural Network model; A2 is Node2Vec Neural Network model; and A3 is the weight-DeepWalk Neural Network model.

The parameters set for each model are as below:

For all models, the dimensional of latent vector is 32, the number of epochs to train model is 20, and learning rate is 0.001.

For B2, the number of layers in MLP model is 4, with 1 input layer, 2 hidden layer and 1 output layer. The dimensional of input layer is 64, which is twice of the dimensional of latent vector

(32), the dimensional of first hidden layer is 50, the dimensional of the second hidden layer is 25, and the dimensional of NeuMF layer is 25.

For A1, the number of layers in neural network model is 3, with 1 input layer, 1 hidden layer and 1 output layer. The dimensional of the hidden layer is 128, the dimensional of the output layer is 48.

The hyper-parameters setting of A2 and A3 are same as the setting of A1.

Explicit dataset records values of user's rate on movies. Based on the figure 4 and figure 5, the matrix factorization model and fusion of GMF and MLP models perform best. For the value of RMSE, the values of all models are in the range of [1,1.1], which means the performance of all models based on explicit dataset has small difference.

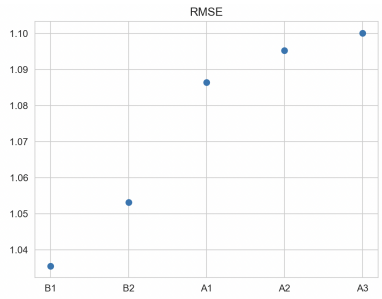

Figure 4: RMSE result

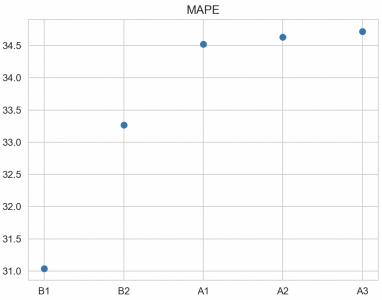

Figure 5: MAPE result

Implicit dataset records whether there is interaction between each pair of users and items. Based on the figure6 and figure7, the DeepWalk Neural Network model performs best among all 5 models.

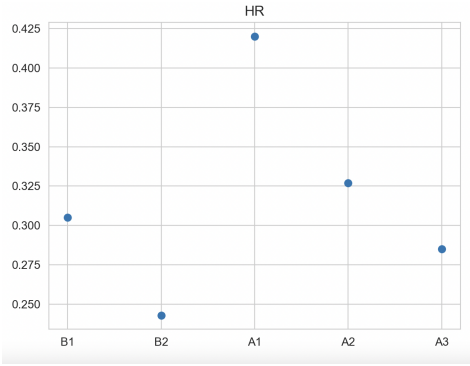

Figure 6: HR@10 result

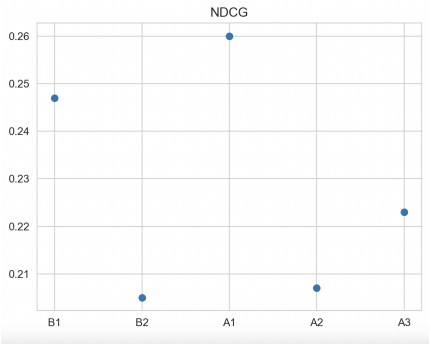

Figure 7: NDCG@10 result

6. DISCUSSIONS

In this paper, I compared the performance of matrix factorization with bias model, Neural Network model, DeepWalk Neural Network model, Node2vec Neural Network model and weight-DeepWalk Neural Network model on both explicit dataset and implicit dataset.

Matrix Factorization model(B1) performs best on explicit dataset, while DeepWalk Neural Network model performs best on implicit dataset.

The explicit dataset requires recommendation models to predict specific rating values; this is an absolute value instead of the comparing values in the implicit dataset. In this case, compared with neural network models, matrix factorization has a smaller number of hyperparameters. The latent embeddings of users and items could be trained and updated directly. Due to computing resources and time limitations, the matrix factorization model could get a result that is more approximate to the actual ratings.

Neural network with graph embedding models like DeepWalk neural network model, Node2Vec neural network model, and weight_deepwalk neural network model are combined with the embedding part and the neural network part. Therefore, when the embedding is terrifying, it is impossible to get good results by training the neural network with terrible input embeddings.

Besides, for users and movies who have few ratings, graph-embedding methods might not work well to find the latent vectors of these samples. Since these nodes have few opportunities to be walked through, their embeddings are not accurate enough. The weight-DeepWalk Neural Network designs to make nodes tend to go to nodes with highest degree centrality, which will increase the probability that those nodes with few ratings (interactions with other nodes) not be walked. Therefore, changing the design of weight-DeepWalk Neural Network to make nodes tend to go to nodes with lowest degree centrality might be a research direction in the future.

On implicit feedback dataset, graph-embedding neural network models perform better than the other two models in general. Since the embeddings of A1, A2 and A3 model are learned based on graph, and graph is created based on the interactions between users and items, we could make the conclusion that graph-embedding neural network models are effective on implicit feedback dataset.

In general, the results on both explicit feedback dataset and implicit feedback dataset are not good enough, there are many possible reasons:

- The scale of dataset. Due to the limitation of computing resources, I reduce the dataset from 100 million records to 5 million records, so some important interaction information might be loosed.
- The choice of hyper-parameters is too randomly, so that the robustness of these models needs to be verified
- Even though the weight-DeepWalk neural network model does not have good performance, taking the graph characters into account is still an interesting perspective to think of.
- The hyper-parameters of neural networks need multiple attempts. With more time, grid-search is a good way to find more suitable hyper-parameters.
- For node2vec model, I set p = 0.5 and q = 2 in this experiment, so the nodes tend to go back to the start node. Since deep walk model has a better result, setting p to be larger and q to be smaller might improve the performance of node2vec neural network model in this experiment.

7. CONCLUSIONS

This paper presents some classic recommendation technologies in Introduction and Related Works and briefly introduces the Netflix Dataset. After that, this paper explains the concepts of Matrix Factorization with the Bias Model, Neural Network Model, DeepWalk Neural Network Model, Node2Vec Neural Network Model, and Weight-DeepWalk Neural Network Model. On Netflix Dataset, this paper compares the performance of these models based on both explicit and implicit feedback datasets. Finally, this paper analyzes and discusses the experiment results.

Acknowledgements. I am really thanksful to Yuru Lin, Xizhi Wu and Arun Balajiee.

8. REFERENCE

[1] Koren, Y. (2008, August). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 426-434).

[2] Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

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

[4] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 701-710).

[5] Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 855-864).