How to Build a Recommender System

Junhua Chen
SCPD student
junhuac@stanford.edu

Abstract

The underlying goal of a recommendation system is to personalize content and identify relevant data for the audience. Given a user's past records, a ranked list of recommended items for that user is generated. We model the problem as a binary classification problem, where we learn a function to predict whether a user also likes other items. Four different models were studied, and their performances were measured and compared. The model called **Deep Learning to Collaborative Filtering Model** performed best among them.

1 Introduction

Recommendation systems suggest items of interest and enjoyment to people based on their preferences. They have been under development since the early 1990s[1]. During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys. Whether you are responsible for customer experience, online strategy, mobile strategy, marketing, or any other customer-impacting part of an organization, you're already aware of some of the ways recommendation technology is used to personalize content and offers.

In this project, we will be building a recommendation system by using the past record of users which can help the users discover their interests for the future. Specifically, given userID and itemID, for example, from Millions Song Data (MSD)[5], we need to generate a ranked list of songs for that user. We model the problem as a binary classification problem, where we learn a function to predict whether a particular user will like particular items or not as shown in equation(1).

$$f(userID, itemID) \rightarrow [0, 1] \tag{1}$$

2 Model studying

In this section we will build and test four different models.

2.1 Popularity Based Recommender Model

A simple approach could be to recommend the items which are liked by most number of users. This is a blazing fast and dirty approach and thus has a major drawback. The things is, there is no personalization involved with this approach. Basically the most popular items would be same for each user since popularity is defined on the entire user pool. So everybody will see the same results. It sounds like, 'a website recommends you to buy microwave just because it's been liked by other users and doesn't care if you are even interested in buying or not.

The naive approach is not personalized where the recommendations are the same for all users.

CS229: Machine Learning , Stanford, California, Fall 2019 .

2.2 Item Similarity Based Collaborative Filtering model

The second exercise of this project is to create a Machine Learning (ML) personalized recommender system by leveraging the item similarity based collaborative filtering model. In the algorithm, the similarities between different items in the dataset are calculated by using one of a number of similarity measures, and then these similarity values are used to predict ratings for user-item pairs not present in the dataset (see figure 1).

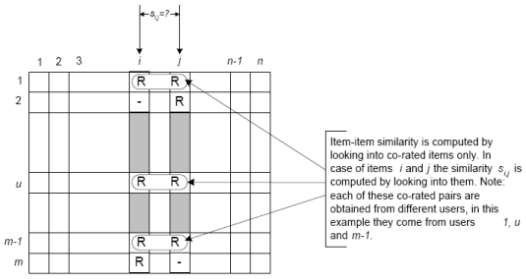

Figure 1: The similarity values between items[8]

With a co-occurrence matrix defined based on a item a user likes, we can predict the rating for any user-item pair by using the idea of weighted sum. First, we take all the items similar to our target item, and from those similar items, we pick items which the active user has rated. We weight the user's rating for each of these items by the similarity between that and the target item. Finally, we scale the prediction by the sum of similarities to get a reasonable value for the predicted rating as shown in Equation(2).

$$P_{u,i} = \frac{\sum_{allsimilaritem,N}(s_{i,N} * R_{u,N})}{\sum_{allsimilaritem,N}(|s_{i,N}|)} \tag{2}$$

To recap, the second model is personalized recommender leveraging the item similarity based collaborative filtering model to find a personalized list of song that a user might like based on what other similar user have liked. It is worth to note that this method is not Deep Learning but purely based on linear algebra

2.3 Matrix Factorization Based Model

In this part, we attempt Matrix Factorization (MF) based Recommender System [16]. This type of recommender system uses what is called a Singular Value Decomposition (SVD) factorized matrix of the original similarity matrix to build recommender system. In another word, we are vectorizing matrices in order to compute the distance between matrices. Just as its name suggests, matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix. From an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. (Of course, you can consider more than two kinds of entities and you will be dealing with tensor factorization, which would be more complicated.) And one obvious application is to predict ratings in collaborative filtering.

Here, we can use the song rating data to give an abstract explanation for the MF model. When a user gives feed back to a certain song they saw (say they can rate from one to five), this collection of feedback can be represented in a form of a matrix. Where each row represents each users, while each column represents different songs. Obviously the matrix will be sparse since not everyone is going to watch every songs, (we all have different taste when it comes to songs).

MF associates each user and item with a real-valued vector of latent features. Let p_u and q_i denote the latent vector for user u and item i, respectively; MF estimates an interaction y_{ui} as the inner product of p_u and q_i:

$$\hat{y}_{u,i} = f(u, i|p_u, q_i) = p_u^T q_i = \sum_{k=1}^{k} p_{uk}, q_{ik} \tag{3}$$

where k denotes the dimension of the latent spaces.

As we can see, MF models the two-way interaction of user and item latent factors, assuming each dimension of the latent space is independent of each other and linearly combining them with the same weight. As such, MF can be deemed as a linear model of latent factors. The possible limitation of MF caused by the use of a simple and fixed inner product to estimate complex user–item interactions in the low-dimensional latent space have been discussed[11]. We note that one way to resolve the

issue is to use a large number of latent factors K. However, it may adversely hurt the generalization of the model (e.g., overfitting the data), especially in sparse settings.

2.4 Deep Learning to Collaborative Filtering Model

The final goal of this project attempts to address the MF model limitation by applying Neural Collaborative Filtering (NCF) to recommend products based on a list of liked tracks from the user[15]. The general NCF framework will be presented in the figure 2.

The deep learning based model we are working on are deep multi-layer perceptron. The bottom input layer consists of two features v_u^U and v_i^I that describe user u and item i, respectively.

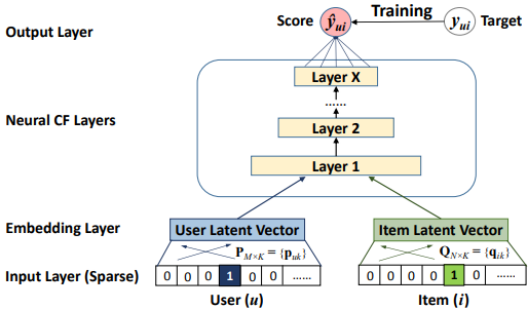

Figure 2: The similarity values between items[8]

The input to the model is same as before including userID and itemID, which is fed into an embedding layer. Thus, each user and item is given an embedding, which is a fully connected layer that projects the sparse representation to a sense vector. The obtained user embedding can be seen as the latent vector for user in the context of latent factor model. The user embedding and item embedding are then fed into a multi-layer neural architecture, which we term as neural collaborative filtering layers, to map the latent vectors to predict scores. Each layer of the neural CF layers can be customized to discover certain latent structures of user-item interactions. The dimension of the last hidden layer X determines the model's capability. The final output layer is the predicted score $\hat{y}_{ui}$, and training is performed by minimizing the pointwise loss between $\hat{y}_{ui}$ and its target value y_{ui}. There are multiple dense layers afterward, followed by a single neuron with a sigmoid activation. The output of the sigmoid neuron can be interpreted as the probability the user is likely to interact with an item. It is interesting to observe that we end up training a classifier for the task of recommendation. Our loss function is Binary Cross-entropy loss. We use Adam for gradient descent and L-2 norm for regularization.

The predictive model is:

$$\hat{P}_{u,i} = f(P^T v_u^U, Q^T, v_i^I u, i | P, Q, \theta_f) \tag{4}$$

Here, $P \in R^{M \times K}$ and $Q \in R^{N \times K}$ denotes the latent factor matrix for users and items, respectively; and θ_f denotes the model parameters of the interaction function f. Since the function f is defined as a multi-layer network, it can be formulated as:

$$f(P^T v_u^U, Q^T v_i^I) = \phi_{out}(\phi_X(\ldots \phi_2(\phi_1(P^T v_u^U, Q^T v_i^I))\ldots)) \tag{5}$$

ϕ_{out} and ϕ_x respectively denote the mapping function for the output layer and x-th neural collaborative filtering layer and there are X neural CF layer in total.

To learn model parameters, existing pointwise methods [12] largely perform a regression with squared loss:

$$L_{sqr} = \sum_{(u,i) \in \Upsilon \cup \Upsilon -} \omega_{ui}(y_{ui} - \hat{y}_{ui})^2 \tag{6}$$

where Υ denotes the set of observed interactions in Y, and $\Upsilon -$the set of negative instances, which can be all (or sampled from) unobserved interactions; and w_{ui} is a hyper-parameter denoting the weight of training instance (u, i). While the squared loss can be explained by assuming that observations are generated from a Gaussian distribution, we point out that it may not tally well with implicit data. This is because for implicit data, the target value y_{ui} is a binarized 1 or 0 denoting whether u has interacted with i. In what follows, we present a probabilistic approach for learning the pointwise NCF that pays special attention to the binary property of implicit data. Considering the one-class nature of implicit feedback, we can view the value of y_{ui} as a label $-$ 1 means item i is relevant to u, and 0 otherwise. The prediction score $\hat{y}_{ui}$ then represents how likely i is relevant to u. To endow NCF with such a probabilistic explanation, we need to constrain the output $\hat{y}_{ui}$ in the range of [0, 1], which can be easily achieved by using a probabilistic function (e.g., the Logistic or Probit function) as the

activation function for the output layer ϕ_{out}. With the above settings, we then define the likelihood function as:

$$p(\Upsilon, \Upsilon - |P, Q, \Theta_f) = \prod_{(u,i)\in\Upsilon} \hat{y}_{ui} \prod_{(u,i)\in\Upsilon-} (1 - \hat{y}_{ui}) \tag{7}$$

Taking the negative logarithm of the likelihood, we reach

$$L = -\sum_{(u,i)\in\Upsilon} log\hat{y}_{ui} - \sum_{(u,i)\in\Upsilon-} log(1 - \hat{y}_{ui}) \tag{8}$$

This is the objective function to minimize for the NCF methods, and its optimization can be done by performing stochastic gradient descent (SGD). Careful readers might have realized that it is the same as the binary cross-entropy loss, also known as log loss. By employing a probabilistic treatment for NCF, we address recommendation with implicit feedback as a binary classification problem.

3 Result and Performance Comparison

To evaluate the performance of item recommendation, we adopted the leave-one-out evaluation, which has been widely used in literature [13]. For each user, we held-out her latest interaction as the test set and utilized the remaining data for training. Since it is too time-consuming to rank all items for every user during evaluation, we followed the common strategy [14] that randomly samples 100 items that are not interacted by the user, ranking the test item among the 100 items. The performance of a ranked list is judged by Hit Ratio (HR)[7]. Without special mention, we truncated the ranked list at 10 for both metrics. As such, the HR intuitively measures whether the test item is present on the top-10 list. We calculated the metrics for each test user and reported the average score.

Figure 3 shows the HR performance of Top-K recommended lists where K ranges from 1 to 10. As can be seen, deep learning model demonstrates consistent improvements over other methods across positions from K=3 to K=10. The deep learning model outperforms baseline popularity model by about 35% at K=10. And deep learning model is also 12% better than item based model with K=10. For K<3, the Item based model outperforms that of deep learning model which may indicate that the deep learning model still needed more training and optimization. Note that in figure 3: Popu represents Popularity model; Item represents Item based model; Deep represents Deep Learning model.

Figure 3: The comparison using HR

4 Conclusion and Future works

4.1 Conclusion

- The popularity based model is the naive approach, therefore the HR is lowest. And this approach will be our baseline model.

- The item similarity based model is looking for the similarities between different items and then these similarity values are used to predict ratings for user-item pairs not present in the dataset. This model is purely based on linear algebra which also shows lower HR.
- The possible limitation of Matrix Factorization based model caused by the use of a simple and fixed inner product to estimate complex user item interactions in the low dimensional latent space. One possible way to resolve the issue is to use a large number of latent factors K. However, it may adversely hurt the generalization of the model (e.g., overfitting the data), especially in sparse settings.
- Through learning the interaction function using DNNs from data, the limitation from MF can be addressed. We present a neural network architecture to model latent features of users and items and devise a general framework Neural Collaborative Filtering for collaborative filtering based on neural networks. The performance of HR is highest with the deep learning model.

4.2 Future Works

- The model of deep learning is still under training. We expect the the performance will be even better if all of the parameters have been optimized.
- Will try to apply this deep learning model on other online dataset. The MSD dataset [5] will be the next step to test the model developed in this project.
- The recommender system should not be limited on the recommendation of songs, movies or books. Will try to apply on the industry level yield or error analysis.

Code Link

`https://1drv.ms/u/s!ArgNcBVbSyOuzH7o3IuothrkIbRn`

References

[1] Resnick, P. Varian, H.R., Recommender systems, sommunications of the ACM 40 (1997), 56-58.

[2] NetflixPrize. Retrieved from `http://www.netflixprize.com/`

[3] How dose AI work with product recommender system? Retrieved from `https://www.smarthint.co/en/ai-product-recommendation-engine/`

[4] An introduction to recommender engines. Retrieved from `https://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/`

[5] Million song dataset. Retrieved from `https://labrosa.ee.columbia.edu/millionsong/`

[6] MovieLens. Retrieved from `https://grouplens.org/datasets/movielens/`

[7] X. He, T. Chen, M.-Y. Kan, and X. Chen. TriRank: Review-aware explainable recommendation by modeling aspects. In CIKM, pages 1661-1670, 2015

[8] Recommender systems. Retrieved from `http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/itembased.html`

[9] Carlos Pinela, recommender systems user-based and item-based collaborative Filtering. Retrieved from `https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f`

[10] Kasper Van Lombeek, Finding similar names with matrix factorization. Retrieved from https://www.datacamp.com/community/tutorials/matrix-factorization-names

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua: Neural collaborative filtering. Retrieved from https://arxiv.org/abs/1708.05031

[12] X. He, H. Zhang, M.-Y. Kan, and T.-S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In SIGIR, pages 549-558, 2016.

[13] I. Bayer, X. He, B. Kanagal, and S. Rendle. A generic coordinate descent framework for learning from implicit feedback. In WWW, 2017.

[14] A. M. Elkahky, Y. Song, and X. He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In WWW, pages 278-288, 2015.

[15] Harshdeep Gupta, Recommender Systems using Deep Learning in PyTorch from scratch. Retrieved from https://towardsdatascience.com/recommender-systems-using-deep-learning-in-pytorch-from-scratch-f661b8f391d7 https://github.com/HarshdeepGupta/recommender_pytorch

[16] Eric Le, How to build a simple song recommender system. Retrieved from https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85