Neural Collaborative Filtering for Music Recommendation

Narisu Tao
taonarisu@gmail.com

March 11, 2021

Abstract

Nowadays, thanks to the wide adaptation of smart phones and popularity of online content providers, we have access to huge variety of content within a few clicks. This turns out to be both a blessing and a curse - a curse because, for users, it is similar to try to find a needle in a haystack. Recommendation system are great tools to mitigate this pain with accurate suggestion for the next item might be of the users' interest. In this report we try to improve the music recommendation system by taking into account meta information of user and item with deep neural network models.

***K*eywords** Recommendation system · Deep neural network · Collaborative filtering

1 Introduction

Not long before, it used to be expensive and time consuming to find music one likes from a cabinet full of cassettes. With the wide adaptation of new devices (such as MP3 and smart phone) and availability of all kinds of online content provider, nowadays, we are having a hard time to find what we really like from millions of songs and music online.

To assist the user to better navigate through the huge number of items (such as songs), all kinds of data driven methods, also called Recommendation Systems, are proposed. E.g. the well known Netflix price[1] competition was an event where researcher tried to come up with better algorithm given user and movie interaction data. Today, Recommendation system algorithms are not only used for music or online product recommendation, but also for other resources, such as YouTube videos and app store applications.

The classic recommendation algorithms can be divided into two categories: collaborative filtering algorithms and content based algorithm. The collaborative filtering algorithms include matrix factorization based algorithms and k Nearest Neighbor inspired algorithms. The input data for these algorithm are user-item interaction matrix, where the matrix entry can be binary value and integer value (number 1, 2, 3, 4, 5). The content based algorithms take the feature extracted from the meta information of user and item as input and turn the recommendation system as a classification or regression task.

In the recent few years, deep learning models has revolutionize many fields, including computer vision, natural language processing and speech recognition. Similarly, deep learning has also been introduced to recommendation system research and many new algorithms has been proposed[1]. Among those deep learning based approaches, the Neural Collaborative Filtering(NCF)[2] is a framework where classic matrix factorization based algorithm and content based algorithm can be merged seamlessly. In this report, we re-implement and extend the NCF algorithm in the context of a music recommendation system. we observe that the NCF model can outperform classic recommendation system algorithm.

In the following part of the report, in section 2, we give the notation and formalization of the problem. In section 3, we introduce the classic recommendation system models. In section 4,we provide in details the NCF

[1]http://www.netflixprize.com/

model. In section 5, all the empirical study details, such as data-set, model configuration and experiment result are given.

2 Problem Statement

In this section, we will first provide the relevant notation of the user and item in the recommendation system. And then, we give the formalization of the problem we are trying to solve.

2.1 Notation

We assume that there are p number of users and q number of items within a system. The user set and item set are denoted by the following:

- Users: $U = \{u_1, u_2, ..., u_p\}$
- Items: $I = \{i_1, i_2, ..., i_q\}$

Each user and item also have their own meta information, which can be denoted as the following:

- Each user u has j different meta information:

$$\begin{aligned} U_{meta} = \{&u_1 : (u^1_{meta_1}, u^1_{meta_2}, ..., u^1_{meta_j}),\\ &u_2 : (u^2_{meta_1}, u^2_{meta_2}, ..., u^2_{meta_j}),\\ &...\\ &u_p : (u^p_{meta_1}, u^p_{meta_2}, ..., u^p_{meta_j})\} \end{aligned} \tag{1}$$

- Each item i has l different meta information:

$$\begin{aligned} I_{meta} = \{&i_1 : (i^1_{meta_1}, i^1_{meta_2}, ..., i^1_{meta_k}),\\ &i_2 : (i^2_{meta_1}, i^2_{meta_2}, ..., i^2_{meta_k}),\\ &...\\ &i_q : (i^q_{meta_1}, i^q_{meta_2}, ..., i^q_{meta_k})\} \end{aligned} \tag{2}$$

In this report, we only study the system which only provide binary recommendation. The system which provide integer number (e.g. 1 to 5) recommendation are not covered. The interaction event between a user and a item can also include meta information, which can be denoted as the following:

- User item interaction event meta information:

$$\begin{aligned} E = \{&e_1 : (u_{i_1}, i_{j_1}, e^1_{meta_1}, e^1_{meta_2}, ..., e^1_{meta_l}),\\ &e_2 : (u_{i_2}, i_{j_2}, e^2_{meta_1}, e^2_{meta_2}, ..., e^2_{meta_l}),\\ &...\\ &e_n : (u_{i_n}, i_{j_n}, e^n_{meta_1}, e^n_{meta_2}, ..., e^n_{meta_l})\} \end{aligned} \tag{3}$$

By merging the meta information of users and items into the interaction even data, the final dataset can be denoted by the following:

$$\begin{aligned} E = \{&e_1 : (u_{i_1}, u^{i_1}_{meta_1}, u^{i_1}_{meta_2}, ..., u^{i_1}_{meta_j}, i_{j_1}, i^{j_1}_{meta_1}, i^{j_1}_{meta_2}, ..., i^{j_1}_{meta_k}, e^1_{meta_1}, e^1_{meta_2}, ..., e^1_{meta_l}),\\ &e_2 : (u_{i_2}, u^{i_2}_{meta_1}, u^{i_2}_{meta_2}, ..., u^{i_2}_{meta_j}, i_{j_2}, i^{j_2}_{meta_1}, i^{j_2}_{meta_2}, ..., i^{j_2}_{meta_k}, e^2_{meta_1}, e^2_{meta_2}, ..., e^2_{meta_l}),\\ &...\\ &e_n : (u_{i_n}, u^{i_n}_{meta_1}, u^{i_n}_{meta_2}, ..., u^{i_n}_{meta_j}, i_{j_n}, i^{j_2}_{meta_1}, i^{j_n}_{meta_2}, ..., i^{j_n}_{meta_k}, e^n_{meta_1}, e^n_{meta_2}, ..., e^n_{meta_l})\} \end{aligned} \tag{4}$$

2.2 Problem definition

Given historical interaction data E between users and items, we can denote the probability of observing the event as the following:

$$\begin{aligned}&P(e|E,\Theta)\\&=P((u_i,i_j,e_{meta_1},e_{meta_2},...,e_{meta_l})|E,\Theta)\\&=P((u_i,u^i_{meta_1},u^i_{meta_2},...,u^i_{meta_j},i_j,i^j_{meta_1},i^j_{meta_2},...,i^j_{meta_k},e_{meta_1},e_{meta_2},...,e_{meta_l})|E,\Theta)\end{aligned}\tag{5}$$

What we really want to do is to forecast whether there could be an interaction between certain user and certain item.

$$\frac{P(e|E,\Theta)}{P(e|E,\Theta)+P(\neg e|E,\Theta)}\tag{6}$$

3 Benchmark Models

In this section, we provide detailed introduction of two examples within classic recommendation system algorithms: matrix factorization based algorithm[3] and content based algorithm.

3.1 Matrix factorization

The interaction data between users and items can be represented as a matrix M, where

$$M_{i,j}=\begin{cases}1 & \text{if } (u_i,i_j,...)\in E\\ 0 & \text{otherwise}\end{cases}\tag{7}$$

In real world scenarios, lots of interaction matrix has low rank property, which can be used to efficiently reconstruct the matrix with small percentage of the observation in the matrix.

$$M=U^T*V,\tag{8}$$

where U is $k*p$ matrix and V is $k*q$ matrix, $k<<p$ and $k<<q$.

The matrix U can be seem as a k dimensional embedding for users and the matrix V can be seem as a k dimensional embedding for items.

The matrix factorization algorithm only needs the user and item information event data-set E. Iterative numeric algorithm, such as Alternative Least square algorithm [4], can be used to learn the matrix U and V.

In this report, we use PyTorch[5] to build the matrix factorization algorithm. Specifically, we define embedding for user and item, similar to matrix U and V. The model prediction is the following:

$$\hat{M}_{i,j}=\sigma(U_i^T*V_j+B_i^{user}+B_j^{item})\tag{9}$$

where σ is the Sigmoid function.

$$\sigma(x)=\frac{1}{1+e^{-x}}\tag{10}$$

And loss function is defined as the following:

$$loss=\sum CE(M_{i,j},\hat{M}_{i,j})\tag{11}$$

where CE is binary cross entropy function.

$$CE(y,\hat{y})=y*log(\hat{y})+(1-y)*log(1-\hat{y})\tag{12}$$

The model weights are obtained with Adam optimizer with batch size 512 and learning rate 0.001.

3.2 Multi-layer perception

For content based algorithm, we use the 4 layers Multi-Layer Perception (MLP). The input of the MLP is the concatenation of the embedding of user, the embedding of item and the embedding of meta information of user, item and event.

The model prediction is the following:

$$\hat{M}_{i,j} = \sigma(W_4 * max(0, W_3 * max(0, W_2 * max(0, W_1 * X + b_1) + b_2) + b_3)) \tag{13}$$

where

$$\begin{aligned} X = [&Embed_{u_i}, Embed_{u^i_{meta_1}}, Embed_{u^i_{meta_2}}, ..., Embed_{u^i_{meta_j}}, \\ &Embed_{i_j}, Embed_{i^j_{meta_1}}, Embed_{i^j_{meta_2}}, ..., Embed_{i^j_{meta_k}}, \\ &Embed_{e_{meta_1}}, Embed_{e_{meta_2}}, ..., Embed_{e_{meta_l}}] \end{aligned} \tag{14}$$

The loss function is the same as defined in 11. The model weights are obtained with Adam optimizer with batch size 512 and learning rate 0.001.

4 Solution Statement

In this report, we re-implement and extend the Neural Collaborative Filtering (NCF) model[2]. As shown in the Figure 1[2], the user and item have two copies of embedding: one for collaborative filtering input and one for MLP input, the output of collaborative filtering and MLP are concatenated into a flat layer which are used to generate the prediction.

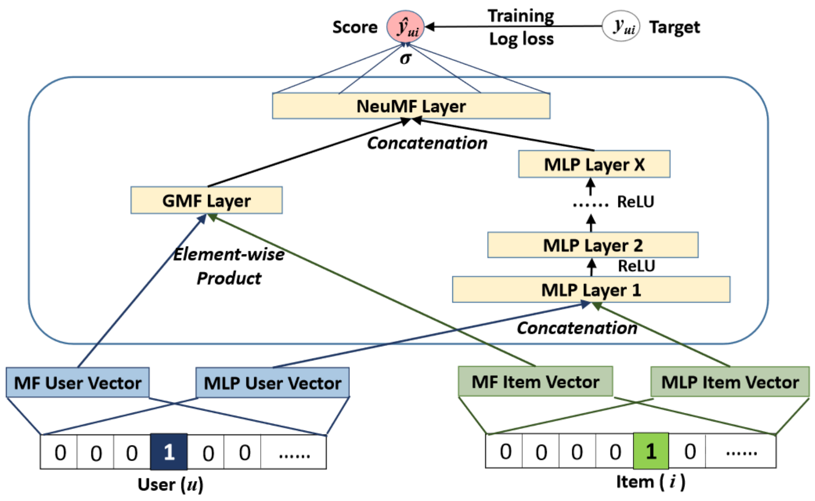


Figure 1: Neural collaborative filtering framework

By having separate input embedding and model components, NCF is able to capture both the linear and the non-linear interaction between user and item.

Furthermore, in this report, we extend the original NCF model by taking into account other meta information of user and item, and the meta information of user-item interaction. In this way, we hope the new NCF model can also leverage the content based algorithm. Specifically, the embedding of all meta information are concatenated to form the input of the MLP Layer 1 in the Fig. 1.

The model is implemented as the following PyTorch lightning module:

```
import numpy as np
```

[2]The figure is the Fig. 3 in [2]

```python
import torch
from torch import nn
import torch.nn.functional as F
import pytorch_lightning as pl

class LitNCF(pl.LightningModule):
    """
    This is a custom implementation of Neural Colaborative Filtering
    """
    def __init__(self,
                 n_user,
                 n_user_meta_1,
                 n_user_meta_2,
                 n_item,
                 n_item_meta_1,
                 n_item_meta_2,
                 n_event_meta_1,
                 n_event_meta_2,
                 n_dim=10):
        """
        Initialize the model.
        """
        super(LitNCF, self).__init__()

        self.user_nn_embed = nn.Embedding(n_user, min(n_dim,n_user))
        self.user_mf_embed = nn.Embedding(n_user, n_dim)
        self.user_mf_bias = nn.Embedding(n_user, 1)

        self.item_nn_embed = nn.Embedding(n_item, min(n_dim,n_item))
        self.item_mf_embed = nn.Embedding(n_item, n_dim)
        self.item_mf_bias = nn.Embedding(n_item, 1)

        self.user_meta_1_embed = nn.Embedding(n_user_meta_1,
                                              min(n_dim, n_user_meta_1))
        self.user_meta_2_embed = nn.Embedding(n_user_meta_2,
                                              min(n_dim, n_user_meta_2))

        self.item_meta_1_embed = nn.Embedding(n_item_meta_1,
                                              min(n_dim, n_item_meta_1))
        self.item_meta_2_embed = nn.Embedding(n_item_meta_2,
                                              min(n_dim, n_item_meta_2))

        self.event_meta_1_embed = nn.Embedding(n_event_meta_1,
                                               min(n_dim, n_event_meta_1))
        self.event_meta_2_embed = nn.Embedding(n_event_meta_2,
                                               min(n_dim, n_event_meta_2))

        self.concat_dim = min(n_dim,n_user)\
                        + min(n_dim,n_item)\
                        + min(n_dim,n_user_meta_1)\
                        + min(n_dim,n_user_meta_2)\
                        + min(n_dim,n_item_meta_1)\
                        + min(n_dim,n_item_meta_2)\
                        + min(n_dim,n_event_meta_1)\
                        + min(n_dim,n_event_meta_2)

        self.l1 = torch.nn.Linear(self.concat_dim, self.concat_dim*2)
        self.l2 = torch.nn.Linear(self.concat_dim*2, self.concat_dim)
        self.l3 = torch.nn.Linear(self.concat_dim, n_dim)

        self.l4 = torch.nn.Linear(2*n_dim, 1)
        self.sig = nn.Sigmoid()

    def forward(self, x):
        user = x['user'].long()
        item = x['item'].long()
        user_meta_1 = x['user_meta_1'].long()
        user_meta_2= x['user_meta_2'].long()
        item_meta_1 = x['item_meta_1'].long()
        item_meta_2 = x['item_meta_2'].long()
        event_meta_1 = x['event_meta_1'].long()
        event_meta_2 = x['event_meta_2'].long()

        concat_embed = torch.cat((self.user_nn_embed(user),
                                  self.item_nn_embed(item),
                                  self.user_meta_1_embed(user_meta_1),
                                  self.user_meta_2_embed(user_meta_1),
                                  self.item_meta_1_embed(item_meta_1),
                                  self.item_meta_2_embed(item_meta_2),
                                  self.event_meta_1_embed(event_meta_1),
                                  self.event_meta_2_embed(event_meta_2)), 1)

        h1 = torch.relu(self.l1(concat_embed))
        h2 = torch.relu(self.l2(h1))
        h3 = torch.relu(self.l3(h2))

        mf_embed = torch.mul(self.user_mf_embed(user),
                             self.item_mf_embed(item))

        logit = self.l4(torch.cat([mf_embed, h3], dim=-1))

        user_bias = self.user_mf_bias(user)
        item_bias = self.item_mf_bias(item)

        logit = logit + user_bias+ item_bias

        return self.sig(logit)
```

Listing 1: Model example

5 Experiments

To empirically show that the extended NCF model can outperform classic recommendation system algorithm, we have implemented the NCF model together with the above mentioned classic recommendation system algorithm in section 3.

In this section, we first introduce the data-set use for the study. Then we give the definition of the metrics for the model evaluation. And, finally, we have present the experiment result in different charts.

5.1 Data-sets

During the 11th ACM International Conference on Web Search and Data Mining (WSDM 2018), one challenge on building a better music recommendation system using a donated data-set from KKBOX was announced. The competition was hosted as a Kaggle competition as well[3]. In this report, we use of one of the downloadable kaggle dataset[4].

In this report we only consider the categorical variables appeared in the meta information. All the numeric variable, such as year of registration, length of songs, are ignored.

For the MLP and NCF model, the number of unique element in each embedding has to be provided. With some analysis, we observed the following statistics of the users, the items and the meta information:

- The number of unique songs and users are the following:
  - Unique number of songs: 2296869

[3]https://www.kaggle.com/c/kkbox-music-recommendation-challenge
[4]https://www.kaggle.com/bvmadduluri/wsdm-kkbox

  - Unique number of users: 34403

- The song has the following meta information and corresponding unique number of options:
  - Composer: 360936
  - Lyricist: 128374
  - Language: 12
  - Country: 201
  - Genre: 192
  - Artist: 239750

  The user has the following meta information and corresponding unique number of options:
  - City: 22
  - Gender: 3
  - Registered via: 7
- The event has the following meta information and corresponding unique number of options:
  - Source system tab: 9
  - Source screen name: 22
  - Source type: 13

5.2 Evaluation metrics

To evaluate the performance of models, we use the following metrics:

- Rooted Mean Square Error (RMSE)
- Mean Absolute Error (MAE)
- Accuracy in Top k recommendation
- Precision in Top k recommendation
- Recall in Top k recommendation
- F1 score in Top k recommendation
- Area Under the Receiver Operating Characteristic Curve (ROC AUC) in Top k recommendation

For the top k recommendation, we have studied the cases where $k = 10$ and $k = 50$.

5.3 Experiment setup

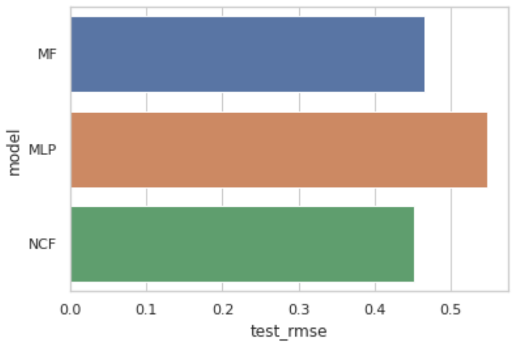
We only use the training data-set of the KKBox data-set, since there is no label information in the test data-set and we cannot evaluate the model with the incomplete test data-set.

The data-set is first split into train_valid data-set (95%) and test data-set (5%).Then the train_valid data-set is further split into train (85.5%) and validation data-set(9.5%).The validation data-set is used for early stop training mainly. The test data-set is used to generate the evaluation metrics of the models.

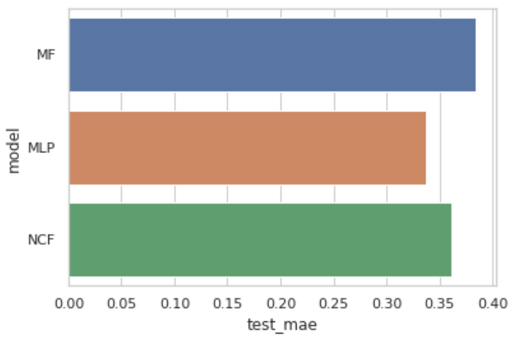
5.4 Result

In this result section, we show the model performance of matrix factorization based algorithm, content based algorithm (MLP) and the extended NCF algorithm.

In Fig 2, the RMSE and MAE value of model evaluation on the test data-set are shown. We can see that the NCF model reached the smallest RMSE score on the test data-set.



(a) RMSE



(b) MAE

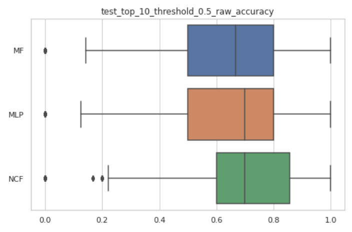
Figure 2: RMSE and MAE on the test data-set

In Fig 3, the metric value for binary classification problem on the test data-set are shown. We use the following formula to turn raw prediction (floating number in the range of [0,1]) into binary recommendation

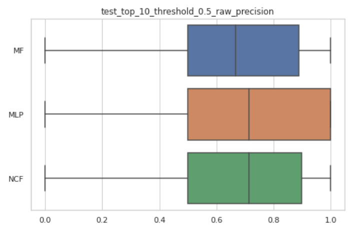
$$\hat{y} = \begin{cases} 1 & \text{if } \hat{M_{i,j}} > threshold \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where the $threshold = 0.5$.

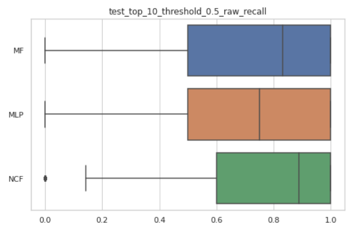
Please note that, we had grouped the metrics based on user. And Each plot includes three box plot of the metrics of all user appeared in the test data-set.We can see that the NCF model reached the highest accuracy on the test data-set. However, it is hard to see which model is the best when it comes to recall and precision.



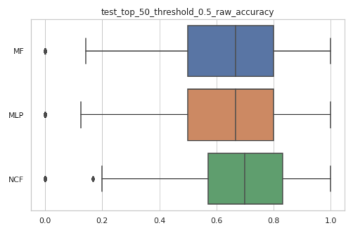
(a) Accuracy in the Top 10 recommendation.



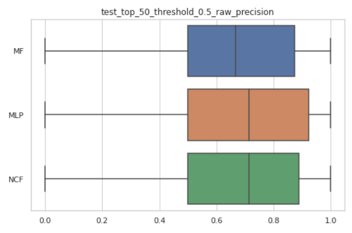
(b) Precision in the Top 10 recommendation.



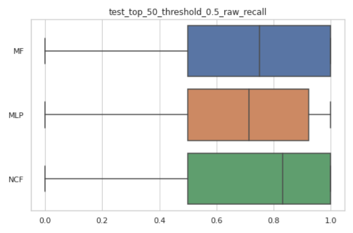
(c) Recall in the Top 10 recommendation.



(d) Accuracy in the Top 50 recommendation.



(e) Precision in the Top 50 recommendation.

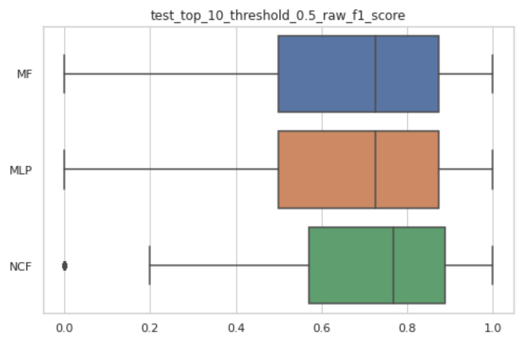


(f) Recall in the Top 50 recommendation.

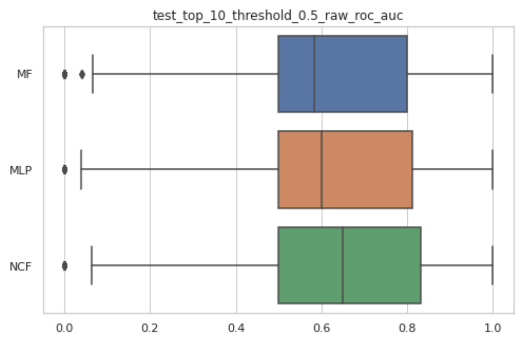
Figure 3: Accuracy, Precision and Recall

The $threshold$ value has direct influence on the recall, precision and accuracy metrics. To get rid of the impact of the $threshold$ value on the model evaluation, one natural idea would be to look at the F1 score and ROC AUC score, which is independent of which value is used to set the threshold. In Fig 4, we have shown the F1 score and ROC AUC score box plot of top 10 and 50 recommendation on the test data-set.

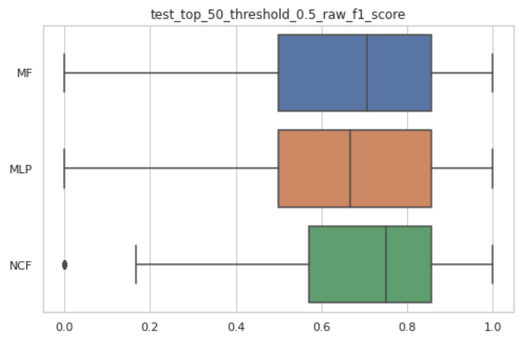
It is easy to see that the NCF model outperforms the other model with clear margin.



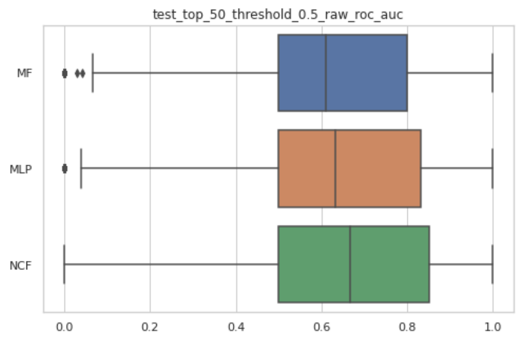
(a) F1 score in the Top 10 recommendation.



(b) ROC AUC score in the Top 10 recommendation.



(c) F1 score in the Top 50 recommendation.



(d) ROC AUC score in the Top 50 recommendation.

Figure 4: F1 score and ROC AUC score

6 Conclusion

In this report, we have studied the extend NCF model on music recommendation problem. With extensive experiments, we have shown that the NCF model can outperform classic recommendation system algorithm.

In next step, one can improve the model performance by including other numeric features, such as length of the song, to further improve the model performance within the NCF model framework.

References

[1] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system. *ACM Computing Surveys*, 52(1):1–38, Feb 2019.

[2] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017.

[3] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, Oct 1999.

[4] Trevor Hastie, Rahul Mazumder, Jason Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares, 2014.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.