Simple Recommender System Using Machine Learning Algorithms

Student Name: EMIL HARUTYUNYAN

Student ID: 521030990007

**Abstract**

As mentioned in my previous project plan, recommendation systems are critical parts of our lives, that is to say no matter what we do in our life everything can be made more efficient and productive, by simple recommendations from others or even the ones generated by computer. So, my project focuses on one tiny example of such recommendation systems, which is movie recommendations based on users’ reviews.

The usefulness of movie recommendation system lies in its ability to build those recommendations on the existing data. In other words we have users, items which are movies and we get a dataset which is all the users’ ratings of those movies on the scale of 0 to 5. After getting that dataset, we employ Machine Learning algorithms to achieve our predictions, recommendations so to say.

The possible benefits of this project may include: helping users get recommendations of movies they will most probably like, hiding or not even showing the movies with low recommendation score so to say at all on the recommendation table, showing the movies with not low and not high scores once in a while just in case user’s taste has changed. In this case it may result in a more exhaustive and useful recommendation system

1 Introduction

1.1 Background

There are many Machine learning algorithms that can be employed to build a recommendation system: clustering models, user-based k-nearest neighbors, matrix factorization, and Bayesian networks, etc. However my approach is matrix factorization with Neural Collaborative Filtering.

1.2 Problem Statement

Using Neural Collaborative Filtering and Matrix Factorization give probability scores for each of the items in the dataset, whether the user is likely to want to watch the movie or not.

1.3 Objectives

* Develop Movie recommendation system
* Use the MovieLens dataset 100K
* Implement Neural Collaborative Filtering
* Optimize model accuracy
* Evaluate the model using different metrics

2 Literature Review

For literature review I picked “Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems.” It introduces the concept and gives thee example of Netflix recommendation system project, as well as other examples of course recommendation system. Further in my project I will be using the algorithms described in the paper. It also explained how to deal with explicit and implicit feedback in the matrix factorization model. As for the learning algorithm I will be employing SGD (Stochastic Gradient Decent), possibly taking into consideration user and item biases, etc. As for temporal dynamics, currently I am not sure if it is possible for the dataset I am going to be working with, but I will try to include it too to improve model accuracy. For this paper I used several ideas from it including the optimization methods, and the overall structure and approach.

The next paper I read was “Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms.” This work presented item-based collaborative filtering, offering an alternative to user-based approaches and demonstrating its effectiveness. It is addressing the issue of scalability of traditional collaborative recommendation systems, by introducing a concept of similarity algorithms, basic idea behind which is computing all the similar items and then training the model on those. I will try to improve my model using this after I am done with the regular model. This idea although simple, but was hard to integrate to the user-based model I wrote. So, it was nice getting introduced to it but in the end I din’t use it much

Additionally, as I was unfamiliar with libraries and what should I use I also did a quick tutorial on tensorflow and how to use it. All the links will be in the appendix.

3 Methodology

Down Below I give the methods of data collection, the training and evaluation methods and the ML algorithms that will be used in the process of developing the recommendation system, the training and evaluation methods. So in a nutshell the overall pipeline includes:

* Number of epochs and top prediction number definition.
* Data preprocessing :

- Converting the interaction matrix into binary ratings.

- Creating a long-format dataset for model training.

- Handling missing values or outliers.

- Splitting the data into training and testing sets.

* NCF model definition with layers and embeddings.
* Compiling the model with all the metrics and Adam optimizer. (Adam optimization method employed)
* Extraction of top predictions’ table.
* Model evaluation using different metrics and plots.

3.1 Data collection

Here I will be using already collected MovieLens100K dataset, which is already cleaned and ready to use. The dataset includes user ratings, movie metadata, and user profiles making it a suitable choice for the collaborative filtering based approach. This dataset can be fetched form a python library lightfm, so I directly imported and used the dataset.

3.2 Machine Learning Algorithms

The main approach is Neural Collaborative Filtering (NCF) it is a hybrid recommendation approach that combines the strengths of matrix factorization and neural networks. It has shown promising results in improving recommendation accuracy. Its implementation will consists of the following key components:

3.2.1 The Overall Structure

The architecture of the Neural Collaborative Filtering (NCF) model and LightFM model.

* Embedding layers for users and items.
* Matrix factorization (MF) and Multi-Layer Perceptron (MLP) components.
* Regularization techniques used.
* Configuration of dense layers and activation functions.

All of the above written can be summerized in Figure 1

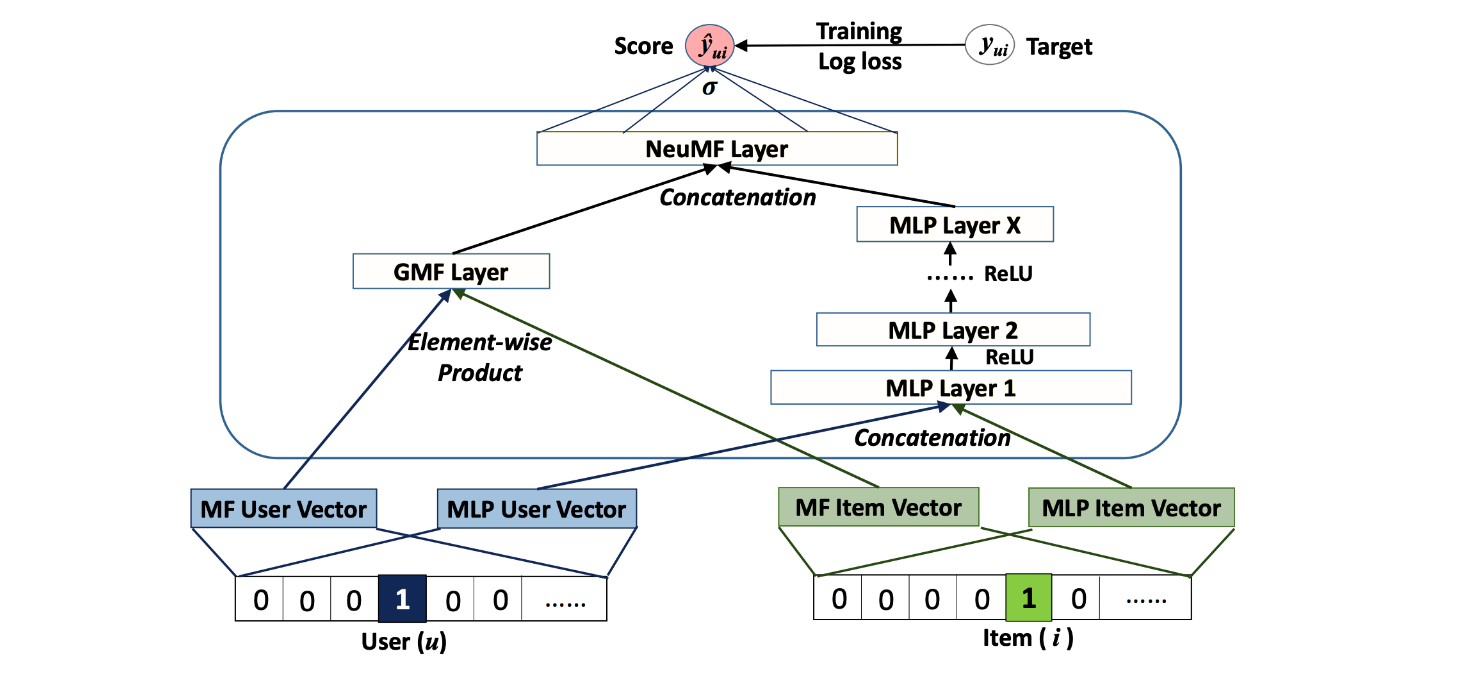


Figure 1

3.2.2 Models used

For this I used two: an already designed hybrid model LightFM which combines matrix factorization and NCF, and another self-designed NFC.

3.2.2.1 LightFM

LightFM employs matrix factorization techniques to decompose the user-item interaction matrix into latent factors. This enables the model to learn underlying patterns in the data.

It is also designed for both explicit and implicit feedback as mentioned in literature review.

One of its properties is that it can both capture user-item interactions and latent features of features of users and items.

The model uses embedding layers for users and items, similar to collaborative filtering models. These embeddings capture the latent features of users and items in a lower-dimensional space.

LightFM employs matrix factorization techniques to decompose the user-item interaction matrix into latent factors. This enables the model to learn underlying patterns in the data.

It also considers not only user-item interactions but also incorporates additional features that may be available for users and items. This makes it well-suited for scenarios where auxiliary information, such as user demographics or item characteristics, is available.

The model supports different loss functions, including logistic loss for binary classification tasks (implicit feedback) and mean squared error for explicit feedback.

3.2.2.2 NFC using keras

Model Architecture:

* Defines a Neural Collaborative Filtering (NCF) model using Keras.
* The model combines matrix factorization (MF) and multi-layer perceptron (MLP) components.
* Embedding layers are used for both user and item features in both MF and MLP branches.
* The model is compiled using the binary crossentropy loss and various evaluation metrics.

Data Preparation for TensorFlow:

* Converts the training data into a TensorFlow dataset using the make\_tf\_dataset function.

Training the NCF Model:

* Trains the NCF model using the TensorFlow dataset.
* Uses early stopping based on validation loss.

Prediction and Evaluation:

* Generates predictions on the test set using the trained NCF model.
* Evaluates the model using precision, recall, accuracy, and AUC metrics.
* Normalizes the predictions and compares them with ground truth values.

3.3 Model Training

I will train the NCF model using the MovieLens dataset. The dataframe we get from fetching the lightfm has “train” and “test” already, so we can use those. However we first need to define our NCF and its embeddings that we are going to use further in the code on the dataset. The piece of the code is shown below in Figure 2. It only shows user MF (matrix factorization) embedding as shown in Figure 1, but others are also in the same fashion for user MLP(multi-layer perceptron), item MF and item MLP embeddings.

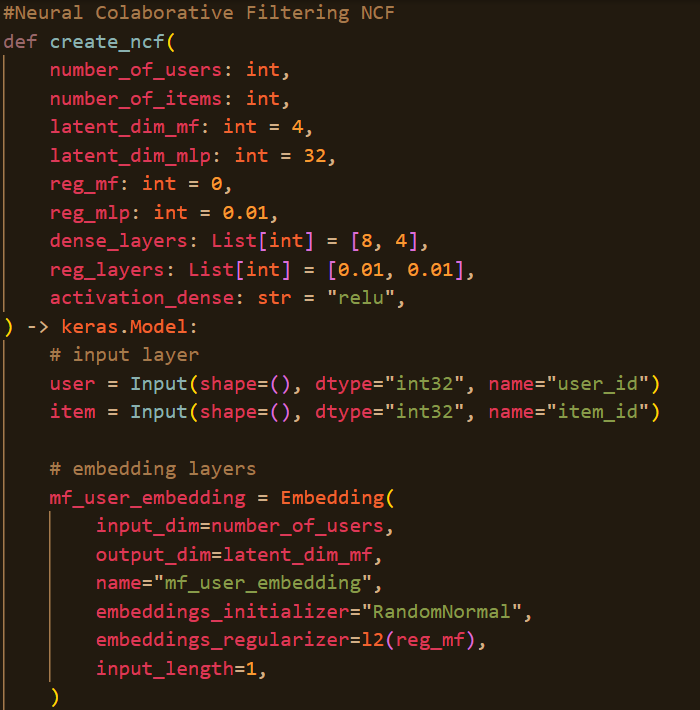


Figure2

3.4 Evaluation Metrics

The performance of the NCF-based recommendation system will be assessed using a set of standard recommendation system metrics, which include:

- Mean Absolute Error (MAE): MAE quantifies the average absolute difference between predicted and actual values, providing a measure of recommendation accuracy.

- Root Mean Square Error (RMSE): RMSE calculates the square root of the average squared differences between predicted and actual values, offering a measure of recommendation error.

- Precision at K: Precision at K evaluates the fraction of relevant recommendations among the top K recommendations. It measures the system's ability to provide high-quality suggestions.

- Recall at K: Recall at K quantifies the proportion of relevant items found among the top K recommendations. It indicates how effectively the system captures all relevant items.

- F1 Score: The F1 Score combines precision and recall into a single metric, offering a balanced measure of recommendation quality.

- Area Under the Receiver Operating Characteristic curve (AUC-ROC): AUC-ROC assesses the model's ability to distinguish between relevant and non-relevant items. It measures the trade-off between true positive rate and false positive rate.

All of the above mentioned is available down below.

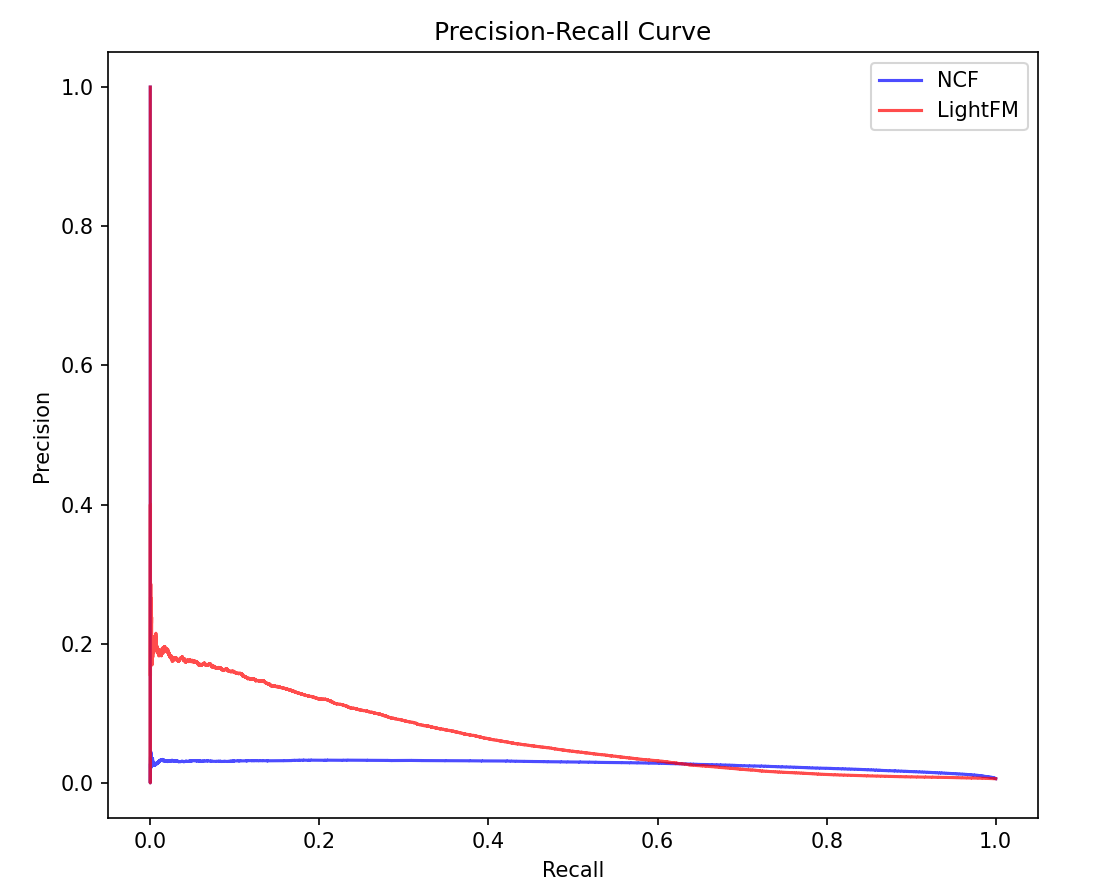


Figure 3

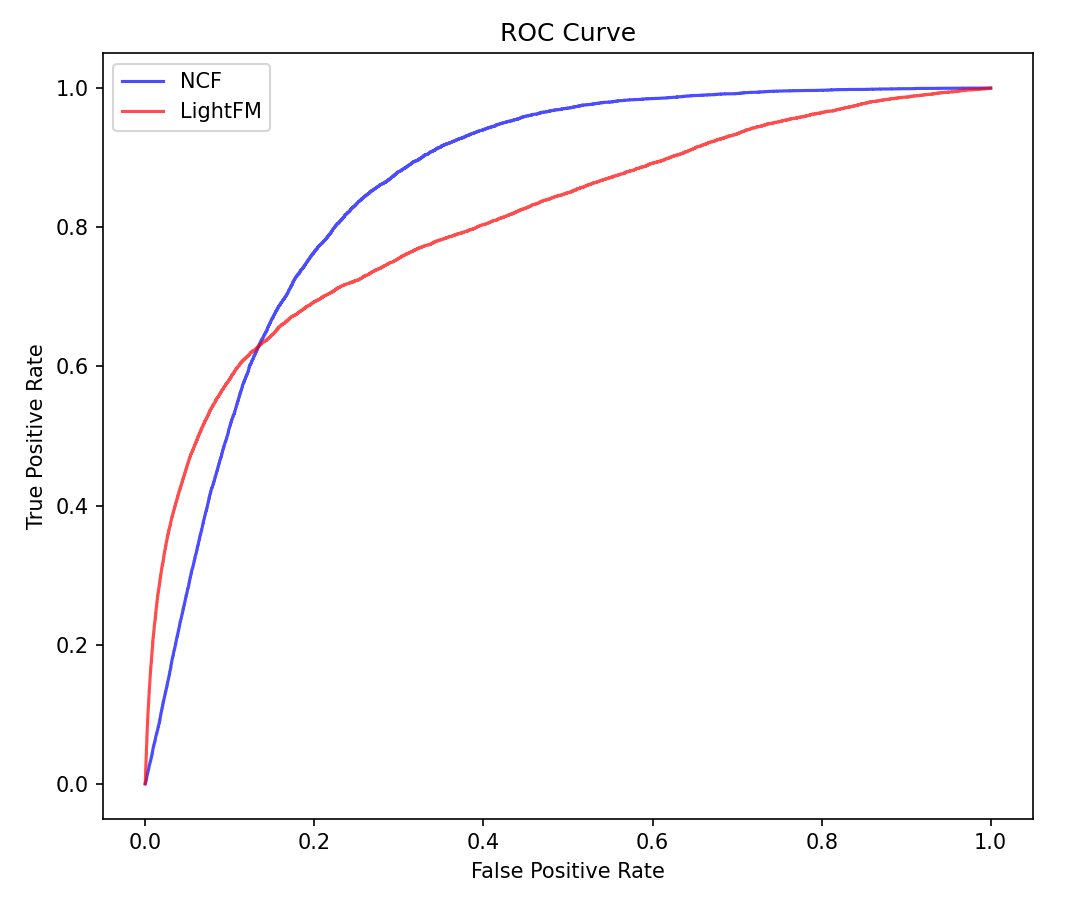


Figure 4

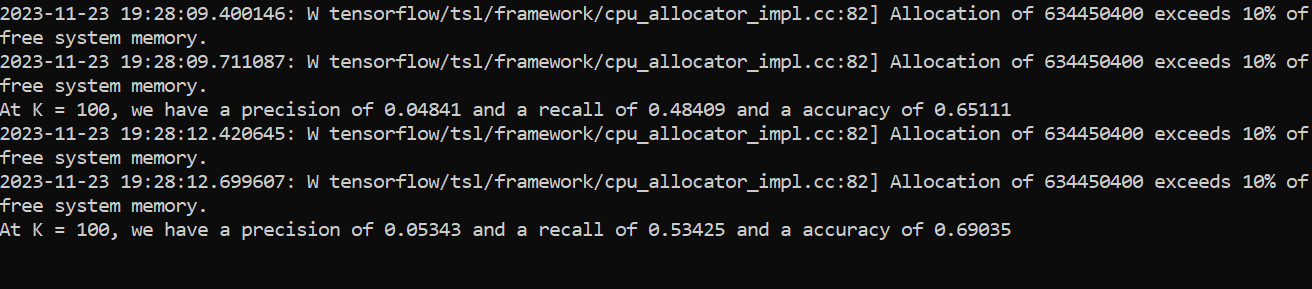


Figure 5

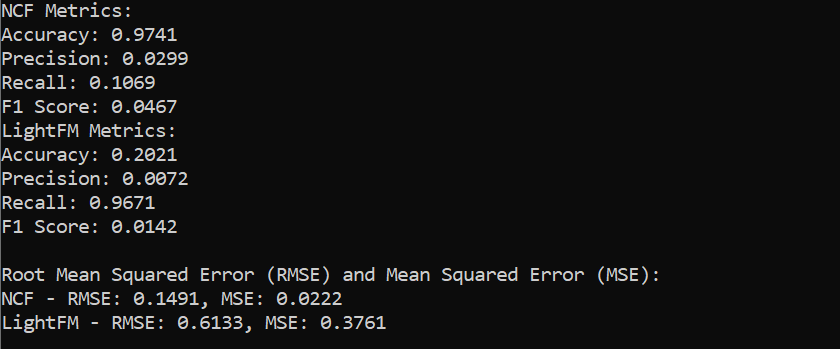
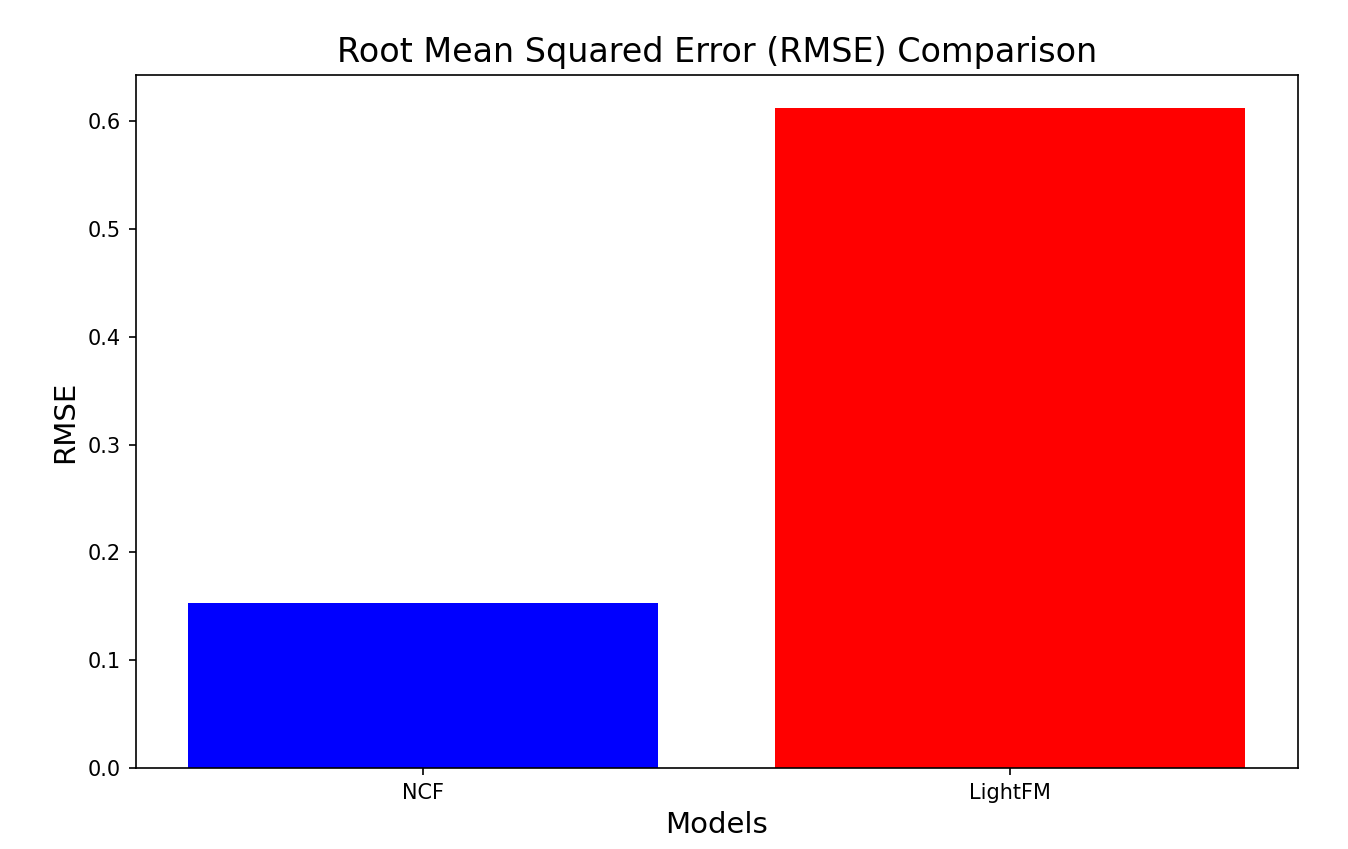


Figure 6

****

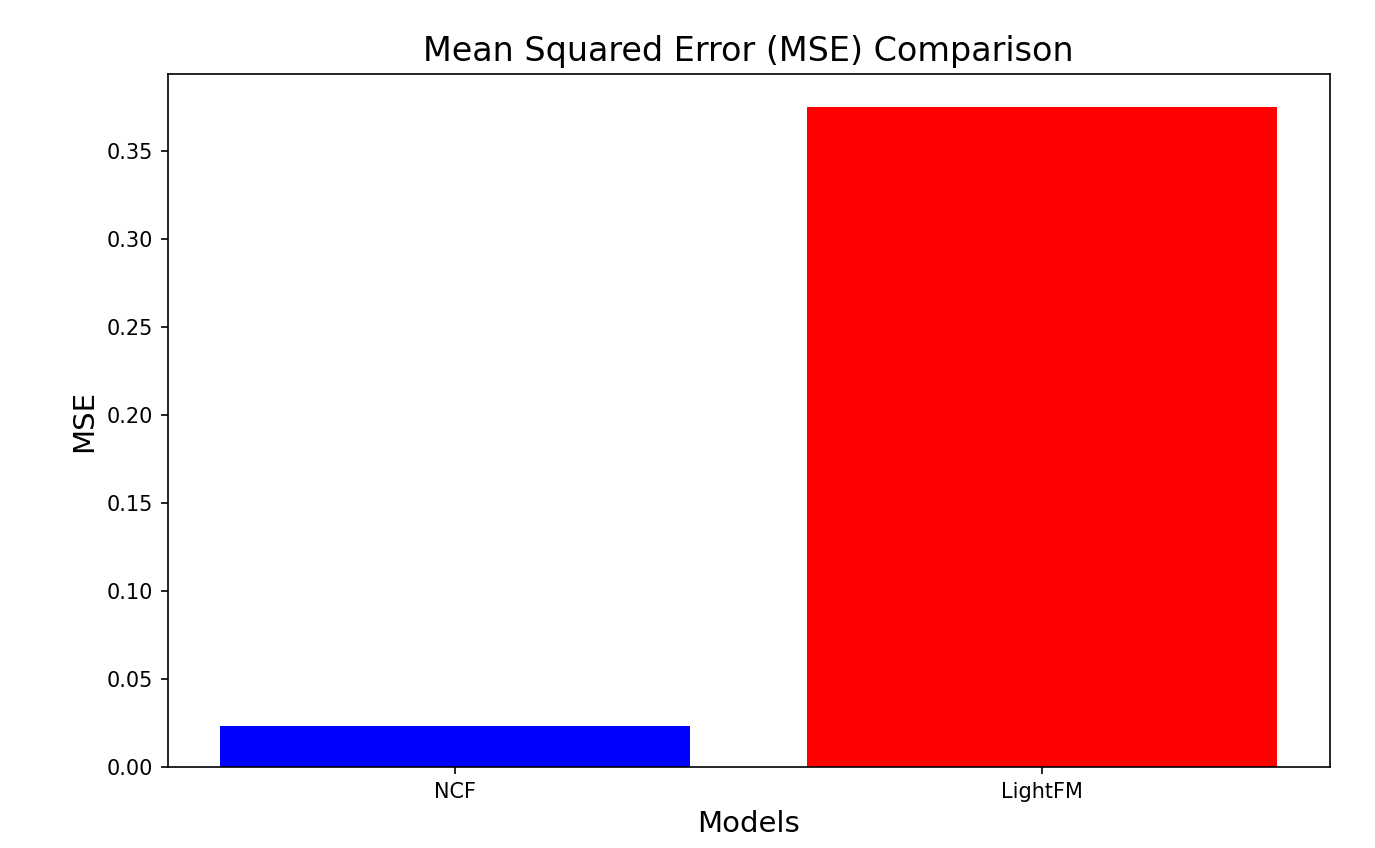
****

Figure 7

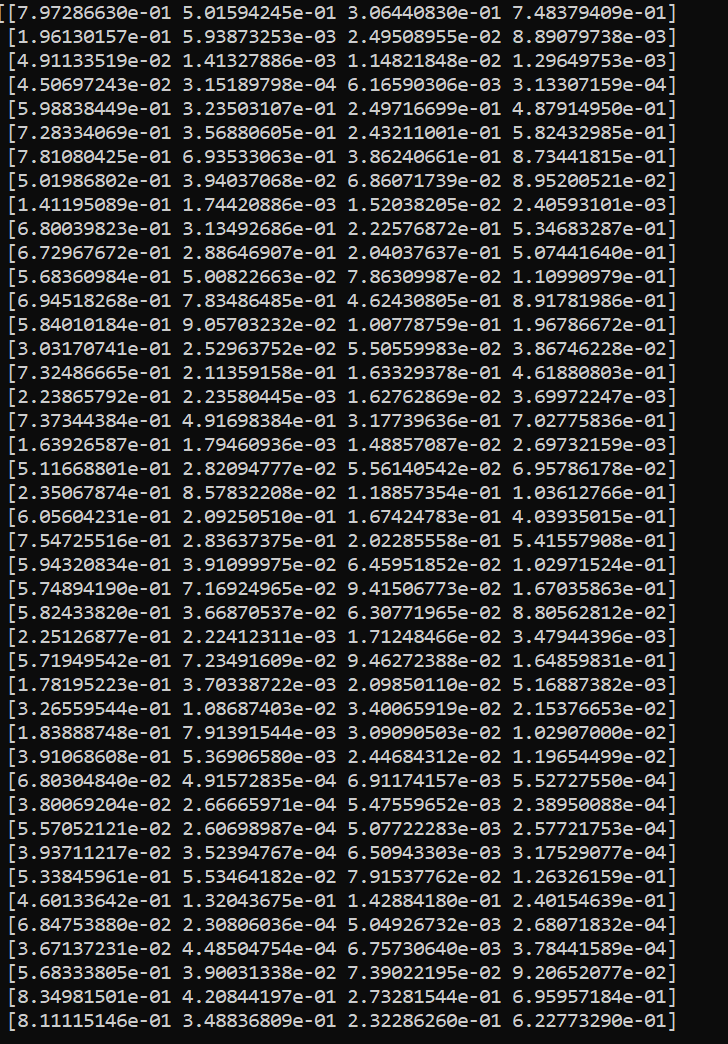
**Figure 3** - gives us the recall-precision relation in NCF and Lightfm, where we can see that precision in NCF doesn’t change much as recall grows while in Lightfm it drops.

**Figure 4** - shows the ROC curve of NCF and Lightfm, where NCF evidently shows better performance.

**Figure 5** - shows the precision recall and accuracy scores for NCF and Lighfm while making the predictions and the reason why it’s different from that in Figure 6 is because for Figure 6 the code calculates the metrics of binary classification whether the prediction is above 0.5 probability or not.

4 Results and Conclusion

In conclusion, the overall model I have built manages to make decent recommendations with 64% accuracy and 50% recall on average for the top 100 predictions. As the model outputs top 100 predictions i.e. recommendations in the format shown in the figure below.



And there is more to it. Overall, my model manages to captures all the latent dimensions and provide somewhat useful recommendations.

5 Limitations

Possible limitations might include:

* Giving recommendations for new users
* Introducing new items to the users
* Bias towards popular items, when users are only exposed to popular items and rate only those.
* Time as I mentioned before, should be taken into consideration because user preferences might change over time
* It is vulnerable to fake reviews

6 Appendix

Here are the links to the papers I have done literature review on

* [“Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems.”](https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf)
* [“Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms.”](https://www.academia.edu/80905729/A_hybrid_user_item_based_collaborative_filtering_model_for_e_commerce_recommendations)
* [Tensorflow Tutorial](https://www.tutorialspoint.com/tensorflow/tensorflow_tutorial.pdf)

The libraries I have used:

* Tensorflow - Models
* Lightfm - Dataset
* Numpy
* Pandas
* Scipy