NCF-based Recommender System

Artificial Intelligence - Project Final Report

Abhi Mohnani, Chikai Shen, Earl Wu

Introduction

Recommender systems have become an essential part of user experience nowadays–be it online merchants, streaming services, or social media, they all depend on recommender systems to attract users' attention to more products, or to simply keep them engaged. A good recommender system can be very beneficial for the business, and potentially for the users as well.

There are different methods out there for implementing a recommender system. For our project, we decided to implement a deep-learning-based system following James Loy's tutorial. The full tutorial can be found on Kaggle:
https://www.kaggle.com/code/jamesloy/deep-learning-based-recommender-systems/notebook.

More specifically, it's a recommender system based on a technique called neural collaborative filtering, or NCF.

Collaborative Filtering

The key problem in a recommender system is collaborative filtering. Essentially, it is a method of making automatic predictions about users' interests by collecting preferences from other users. One of the standard techniques is matrix factorization, or MF, which is a way to generate latent features when multiplying two different types of entities. MF has been the state of the art for collaborative filtering for years.

On the other hand, although deep neural networks (DNN) have had many successful applications on speech recognition, computer vision, and natural language processing, there was not as much work on its application on collaborative filtering before NCF. l Although there has been some work on using deep learning for recommendation, DNN was mostly used to model auxiliary information such as textual descriptions, and the central interaction between user and item features was still modeled by matrix factorization.

Neural Collaborative Filtering

NCF is a general framework proposed by He et al. in a 2017 paper of the same name. It is an architecture that can be used to construct powerful recommender systems. As input, the network takes both an item and a user, and outputs a predicted score for how much the user will like the item. The format of the input of both the item and the user is a sparse binary vector, where exactly one value is 1 and all others are zero. These vectors are then mapped to k-dimensional embeddings, where k is a hyperparamter that we can tune to achieve optimal results. These embeddings represent learned information about the user and the item.

For example, in a book recommender system with k = 4, the architecture may learn 4 values about each book corresponding to some linear combination of how close the book is to mystery, drama, fantasy and sci-fi. These genres or meanings of these embeddings are not chosen by the builder of the architecture; they are learned by the network through the training examples. The value of k is an important hyperparameter in NCF, and can be changed to affect the model's bias and variance. Increasing k can lead to overfitting, while decreasing k may lead to a model without enough complexity to model the underlying function.

After the embeddings, the architecture is fairly simple and flexible. In the paper and in many other examples, a multilayer perceptron is used to map the embeddings to a prediction of how much the user will like the item. Essentially, NCF turns the problem of recommendation into the problem of binary classification. As with most neural networks, the objective function is not convex. Therefore, there is no guarantee of finding the global optimum value. Consequently, as with other neural networks, random initialization and multiple trials are useful for achieving a good result.

Data Processing

Implicit Data Conversion

```
train_ratings.loc[:, 'rating'] = 1
```

Since NCF is developed to focus on implicit feedback, such as watching video or clicking videos, whereas the dataset we have only has explicit feedback in the form of user ratings, we need to convert the explicit data in such a way that it can be interpreted as implicit data. This is simply achieved by normalizing all the ratings scores to 1, indicating an interaction between the user and the item.

Train Test Split

```
ratings['rank_latest'] = ratings.groupby(['userid'])['timestamp'].rank(method='first', ascending=False)

train_ratings = ratings[ratings['rank_latest'] != 1]
test_ratings = ratings[ratings['rank_latest'] == 1]
```

As for the train test split, we used a leave-one-out split method to make the most sense for a recommender system, where the most recent reviews were used as the test set as the leave-one-out methodology indicates, and the rest of the dataset were used as training data. The timestamps in the pre-processed dataset were used to produce a 'rank_latest' column to identify the most recent reviews.

Factorization Encoding

```
ratings['userid'] = pd.factorize(ratings['userid'])[0]
ratings['ASIN'] = pd.factorize(ratings['ASIN'])[0]
```

Since training a NCF learning model takes in numeric value inputs, we converted user id and ASIN, which is the product id, to the equivalent integer values, since the original forms contain special characters.

Results

We tried different parameters for the training and eventually settled down on 10 epochs which gave us the best performance on hid ratio. As a result, we have a 54% hit ratio. Since hit ratio is implemented to be a strict and efficient way for measurement, the result is acceptable.

Instead of analyzing results with an accuracy or RMSE which are most commonly used for classifications and regressions, we used a hit ratio to indicate the performance of our model. The hit ratio evaluation process is implemented by first, for each user, randomly selecting 99 books that the user has not interacted with, and combining the 99 books with a target book (an actual book that the user interacted with lately) to acquire a list of 100 books. Then, run the trained NCF model on the 100 books, and sort the books according to their predicted probabilities from the model. Finally, select the top 10 books from the list of 100 books, and the presence of the test book within the top 10 books indicates a hit. The above process is repeated for all users, eventually getting an averaged hit ratio.

Due to the limitation of Google Colab, we did not get to train the model with the entire dataset given. For future improvement, more entries could be included in training for a more holistic learning process, and increasing the number of layers could also be potentially helpful to better interpret implicit feedback. Another direction could be to collect and train with actual implicit data instead of converting them from the explicit data, essentially with books that a user browsed/interacted instead of reviewed.