CS272 HW2 Report

Anthony Liu | hliu35@ucsc.edu
05/27/2021

INTRODUCTION

For our dataset, we chose the video game rating dataset from Amazon. This takes around 18 GB of memory when running. Therefore, be aware of memory insufficiency.

We are using Jupyter Notebook in this assignment to fully utilize its experimental abilities. To efficiently clean memory after every section, we divided entire workflow into 4 consecutive notebooks. Running the code in the notebooks requires a complete setup of Anaconda environment. Packages such as NumPy, SciPy, Scikit-learn, and Numba must be installed before running.

GitHub URL: `hliu35/cs272-hw2: Recommender System`

More detail available in README.md .

DATA PREPARATION

part1.ipynb

The raw data comes in form of JSON entities. Therefore, we read the file line by line and store them into a HashMap. After storing each line, we processed them into a Pandas data frame. Then we created a NumPy 2-dimensional array where rows are users and columns are items. Values in each cell represent a rating of a particular user-item pair.

PREDICTION

part2a.ipynb, part2b.ipyb

In this section, we attempted 2 ways of predicting ratings. First is item-based Collaborative Filtering, and second is Matrix Factorization.

Item-based Collaborative Filtering

part2a.ipynb

There are different ways to implement item-based CF. One of the ways is to perform column-wise cosine similarities straight on the user-item rating table without removing 0s. The formula is shown as below.

$$\text{Similarity}_{cos}(r_i, r_j) = \frac{r_i \cdot r_j}{|r_i||r_j|}$$

Where $r \in$ Ratings R . This creates a $n \times n$ matrix, where n is the number of items, and each row i in the matrix is a list of cosine similarities of item i and item $1 \leq j \leq n$. Since we do not want similarities to self. We perform subtraction of

$$\text{S} = \text{Similarity}(r_i, r_j) = \text{Similarity}(r_i, r_j) - I$$

Where I is the identity matrix of size $n \times n$.

After the subtraction, we calculate the rating predictions R' using the weighted mean of the ratings from other products.

$$R' = R \cdot S^T$$

Alternatively, we created a modified version where we only compare nonzero pairs. However, this function became extremely slow given our huge dataset. Therefore, we decided to move on with the vectorized one above.

Numerical Results (vectorized)

MAE:	4.06 stars
RMSE:	0.01 stars

Result Interpretation

This result seems unreliable to normal understanding because of the difference between MAE and RMSE. However, the formula is correct and sound. No error was found in code lines. As such, it is possible that the collaborative filtering failed on this particularly sparse dataset.

Matrix Factorization

part2b.ipynb

First, it is important to stress that the data is sparse. Therefore, there exist many 0s in the ground truth (meaning users have not rated those items). This raises a debate on whether our rating predictions or recommendations should happen on those. A main issue is that we cannot measure the correctness of predictions on them.

The model fits one user and one item embedding for the user-item rating table. We set the embedding dimension to be 5 for experiment, but you can increase or decrease by altering the training code. The naïve Matrix Factorization works as followed:

$$\hat{R} = U \cdot V^T$$

where U is the user embedding, V is the item embedding, and $\hat{R}$ is the predicted ratings. In contrast to the naïve method, we also added a linear transform to each layer of embedding to address weights and biases.

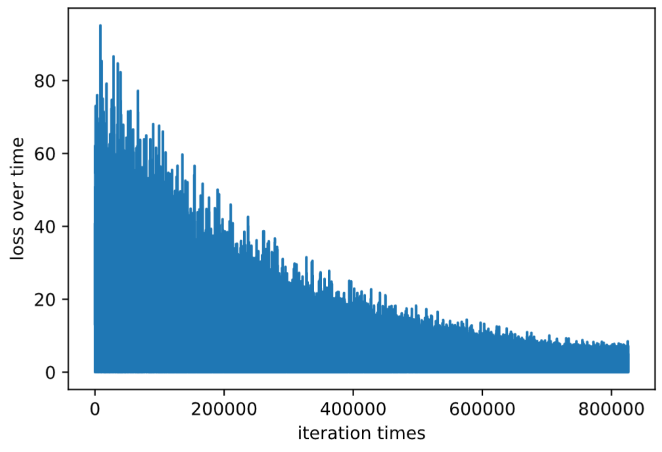
$$\min_{U \in \mathbb{R}^{m \times d},\, V \in \mathbb{R}^{n \times d}} \sum_{(i,j) \in \text{obs}} (R_{ij} - \langle U_i, V_j \rangle)^2$$

obs means observed cells. Notice we are not fitting on the unobserved parts, see FAQ for more information.

During the training process, we performed 15 epochs which took ~8 minutes using CPU. GPU may be faster, but the tables would probably require sufficient GPU memory.

Learning Curve | Training

```
Learning Curve
```



```
Final Loss
0.7368699908256531
```

Numeric Results:

MAE:	1.03 stars
RMSE:	1.60 stars

Result Interpretation

The result from matrix factorization is much better and normal than item-based CF, which encompassed us to use MF in the recommendation part.

FAQs

We listed some FAQs alongside for why we made our particular design choices.

Question 1: Why is the data so sparse?
Answer: This is either because when this dataset was prepared, the users may have not purchased some items, declined to purchase items for reasons, or may have purchased but have not rated.

Sparse data is difficult to work with because the sparser the data is, the less pattern/information we can retrieve from it.

Question 2: Are you predicting on ratings without ground truth as well?
Answer: Yes, because we think it is important to allow generations of new rating predictions beyond what is already available in the dataset. Otherwise, recommendations of not-purchased items cannot happen.

Question 3: How do you measure the correctness of the ratings you predicted?
Answer: For predictions that have corresponding ground truths, we can take their MAE and RMSE directly. In contrast, for predictions without any corresponding ground truths, we do not post limit or measure correctness on the respective predictions. They were all generated based on learned user and item embeddings.

<u>RECOMMENDATION</u>

part3.ipynb

As there are many **<u>controversial</u>** parts in the recommendation section. We are explaining the details in forms of FAQs, such as requirements to get recommended (in Question 3), and reasons for scores to appear in particular ways (in Question 2 and 4)

Question 1: What values is your recommender decisions based on?

Answer: **Predicted ratings**, which is a direct indicator of user satisfaction. Users rate them in extreme cases of either liking or disliking.

Using similarities on term embeddings have the following drawbacks:

a. Similarities only find similar items like “milk A” to “milk B”. However, for users that have been constantly drinking one brand, they are unlikely to switch to another brand.
b. Two items with 0 similarity can have the same user ratings. For example, assume a person likes car and milk. These two items have no linear relationship, but the user likes them both.

Using similarities on user embeddings can have the following drawbacks:

a. Poor at recommending personal items. Two users are in different genders, but their user embeddings failed to model that. Recommending hygiene products to male users can have adverse effects on user experience.

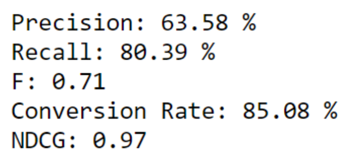
The predicted ratings - products of user and item embeddings - have the most intuitive and logical indication of user preferences. Although user and item embeddings may have their own drawbacks, the products produce the best indications at their best ability. Thus, the reasons why predicted ratings are used in recommendation.

Question 2: How exactly do you pick the top 10s?

This may be confusing to read.

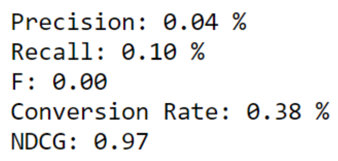
For every user, our recommendation system chooses items with top 10 predicted ratings above a given threshold. However, because we have more than 200k+ items to choose from, the probability landing on the items with ground truths available is exceedingly small. As such, we are providing 2 novel methodologies for measuring recommendation performances. Here we assume a Gaussian distribution for each user’s rating.

1. In `compute_scores()`, set `methodology=1`: We only choose from the <= 10 items that have ground truths, ignoring the generated values without ground truth. Set weight so that recommendation thresholds = $\mu + w \cdot \sigma$ (mean, weight, and standard deviation) for both ground truth and prediction, pick all ratings greater than thresholds as recommended items. The results for weight = -1 are listed below.



```
Precision: 63.58 %
Recall: 80.39 %
F: 0.71
Conversion Rate: 85.08 %
NDCG: 0.97
```

2. In `compute_scores()`, set `methodology=0`: We predicted all the 200k+ ratings with and without ground truths, and we choose the top 10 ratings among ALL of them. This may yield poor numeric results (shown below for w = -1); however, it is still a reasonable choice, as explained in question 4.



```
Precision: 0.04 %
Recall: 0.10 %
F: 0.00
Conversion Rate: 0.38 %
NDCG: 0.97
```

Question 3: In method 1, why did you intentionally pick the 10 items that have ground truths? That is cheating for good scores.

Answer: No. Not all items with ground truth should be recommended, as there must be other filtering strategies such as "minimum eligible ground truth rating", and "minimum eligible predicted rating". Choosing them intentionally is a way of measuring recommendation helpfulness.

Think about for a user, testing set only have items with ratings of 1-star, does your system recommend those as well?

Question 4: The scores of your method 2 are miniscule. Your system is useless.

Answer: No.

Since we have 200k+ items, users have not purchased and rated all of them. As such, even the ground truth has ratings missing and cannot fully model the preferences of a user. Therefore, missing a rating does not necessarily mean the user will not buy after our recommendations (projections).

Our reasoning thus stays, and that is, we still recommend what the users would give top ratings on. Also, to stress again, our predictions outside the dataset are not made up; they are mathematically generated based on learned user and item embeddings.