Collaborative Filtering on Massive Sparse Datasets

Jay Patel
Rutgers University
New Brunswick, New Jersey, United States
jsp202@scarletmail.rutgers.edu

Michael Yen
Rutgers University
New Brunswick, New Jersey, United States
mjy37@scarletmail.rutgers.edu

Patrick Hickey
Rutgers University
New Brunswick, New Jersey, United States
ph402@scarletmail.rutgers.edu

ABSTRACT

Collaborative filtering is an approach to predict item ratings given other known ratings. There are two main types of collaborative filtering: user-user and item-item. Using collaborative filtering, we can create a recommender system to provide users with the top item choices most relevant to them. In this paper, we propose a recommender system that works for massive and sparse datasets, a common situation that may occur for any large selection of items and users in e-commerce. To do this, we contribute our own implementation for collaborative filtering, tailored to this scenario. We found that for our data set, user-user collaborative filtering generally out-performed item-item collaborative filtering, and that making top n item prediction is particularly challenging with large, sparse, data sets.

KEYWORDS

Collaborative Filtering, Recommender Systems, Massive Datasets, Sparse Datasets

1 INTRODUCTION

Recommender systems have become increasingly prevalent in society, growing alongside e-commerce and targeted advertising. Companies aim to show the most relevant products to their customers to increase sales, even when they have little information. In our project we developed a recommendation system using a large, sparse dataset in order to explore the challenges that a large entity would face when implementing recommendations in such an environment. We focus on user-user and item-item collaborative filtering to make recommendations. We used two types of similarity measure and compared their performance for each collaborative filtering method.

2 DATASET AND PRE-PROCESSING

We used the full "Digital Music" subset from the public Amazon e-commerce dataset provided by UCSD [1]. This provided us with a real-world example of a large, sparse dataset that a company would use to make recommendations. Our first step was filtering the dataset, as many items and reviewers had only one review, which isn't helpful for collaborative filtering. For our user-user collaboration, we removed all users who gave fewer than five reviews, and all items with less than three ratings. This resulted in a dataset with 23,958 users, 33,083 items, and 305,453 total ratings. Even after filtering the lowest review count users and items, our final user-item matrix sparsity was 0.9996. For item-item collaborative filtering, we set the item threshold at five reviews, and user threshold at three reviews. The distribution of users and items by review count also skewed heavily towards lower review counts, which can be seen in Figures 1 and 2:

Additionally, review values were heavily skewed towards 5 stars, which makes recommendation more difficult as many items are likely to be recommended, as can be seen in Figure 3:

After analyzing and filtering the data, our first attempt was to create a standard user-item matrix. Representing the data in this form makes it simple and convenient to perform numerous linear algebra operations utilized in collaborative filtering. What we found was that the data was not able to fit in memory using this data structure due to the scale of the dataset. Our solution was to represent the data as a dictionary, with 'user' as the key, and an array of (item, rating) tuples as the value for our user-user collaborative filtering. For item-item collaborative filtering, we created a separate dictionary where the position of the item and user values were switched. This allowed us to save an immense amount of room in memory by not having to store the 99.96% of null ratings values. The downside of this approach was that we had to manually implement many standard linear algebra operations, which will be discussed in the next section. Lastly, we split the data into train and test sets. We used an 80/20 split per user for the train and test sets. Since the minimum review count was five, each user was guaranteed to have at least one item in the test set. Due to the sparsity of our matrix, we would have ended up with many users with no reviews in the test set if we had simply split the data 80/20 by items.

3 COLLABORATIVE FILTERING

The model we decided to use was collaborative filtering, both user-user and item-item. Because the methodology of the two is roughly the same, we reuse a lot of the logic from user-user to item-item. We begin with the implementation of user-user collaborative filtering.

User-User Collaborative Filtering. As mentioned in section 2, we represent the dataset as a dictionary where the user is the key and an array of item-rating tuples is the value. This is provided as input to our user-user collaborative filtering. The first step is to convert this input into a dictionary, where the key is the user and the value is another dictionary, where the item is the key and the rating is the value. Thus, we obtain a dictionary of dictionaries in the format {user: {item: rating}} for both the training and testing sets and call them the training and testing dictionaries, correspondingly.

Our next step was to predict an items' rating for a user given a dictionary of training ratings. To do this we subtracted the row means from every row in the training dictionary. We then take the sum of all the item ratings for a user, divide by the number of items that user has rated, and subtract this value from each of the

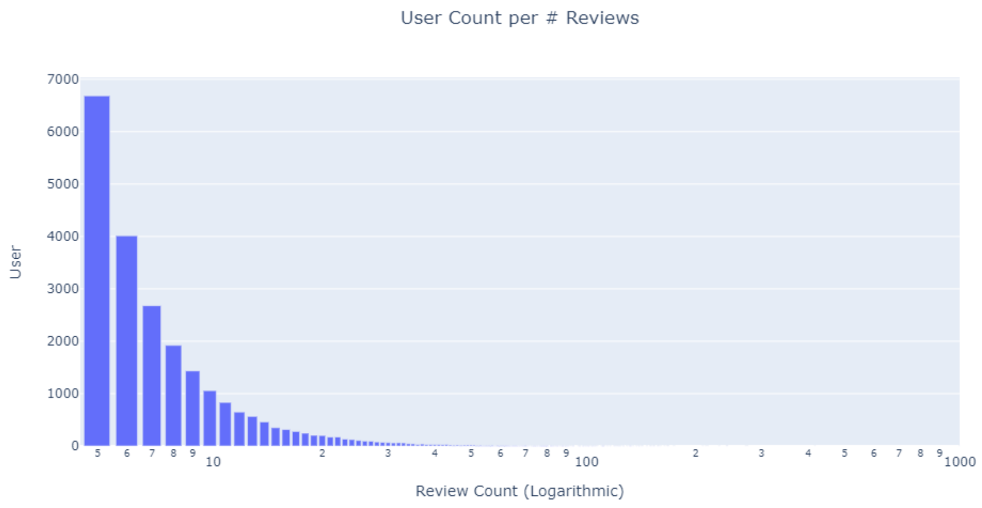

Figure 1: User Count per # Reviews

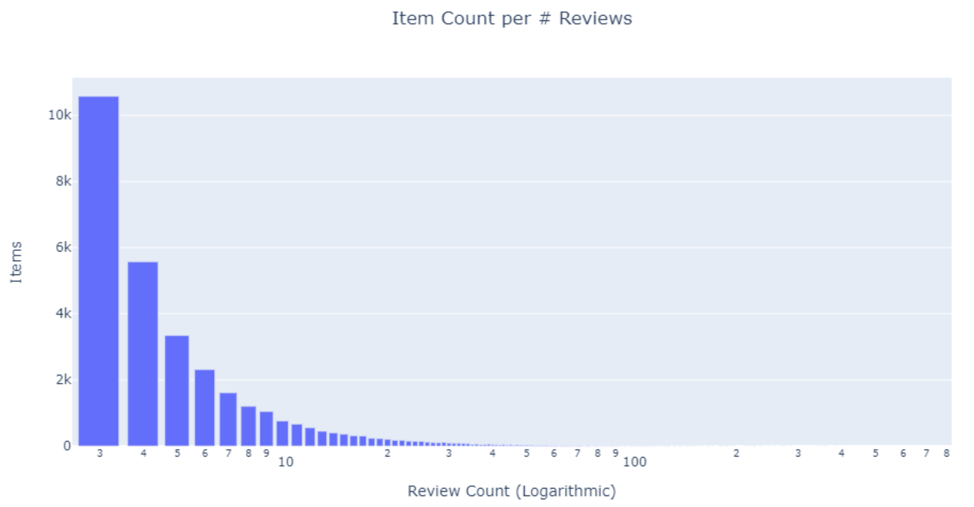

Figure 2: Item Count per # Reviews

item ratings. This is stored as the subtracted row means dictionary. Given the users in the testing dictionary, we then computed the cosine similarities ($cos(r_x, r_y) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$) between each testing user x and the other users y in the training set that are not the testing user. Because our data is represented in a dictionary instead of a matrix, we cannot use library defined cosine similarity, dot product, and norm functions. We implemented our own cosine similarity function and this is returned as a dictionary where the key is the user we are comparing to, and the value is the cosine similarity between the given user and the user in comparison.

Rating Prediction Option. We use both average predicted rating of the top k most similar users and similarity weighted rating of the top k most similar users to obtain rating predictions for a given user and compare them in the results.

The average predicted rating for item i of user x is given by:

$$r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi} \tag{1}$$

The similarity weighted rating for item i of user x is given by:

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} r_{yi}}{\sum_{y \in N} s_{xy}} \tag{2}$$

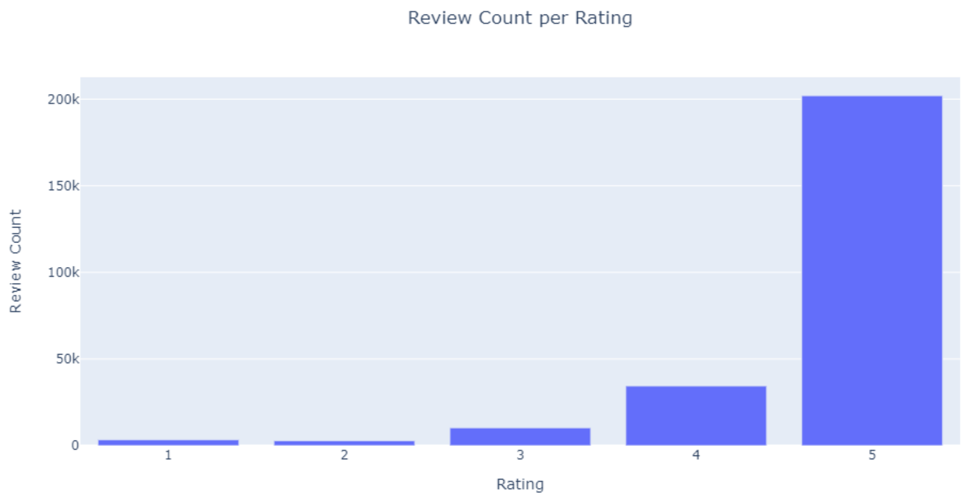

Figure 3: Review Count Per Rating

Like the cosine similarities, we have to implement a function to do this using our dictionary format. Given the ratings dictionary, the cosine similarity dictionary, k, the item in question, and the user, we search for the top k most similar users in the cosine similarity dictionary that are not the user and calculate the rating using either option.

For each user in the testing set we obtain their predicted ratings for the testing items and compute the Mean Average Error (MAE) and Root Mean Squared Error (RMSE) cumulatively using both rating prediction options. When all of the testing set users are done, we output the final MAE and RMSE among all users, for both average predicted rating and similarity weighted rating.

Item-Item Collaborative Filtering. With Item-Item Collaborative Filtering, we take almost the same approach that we did for User-User Collaborative Filtering. The key difference is the structure of the training and testing dictionaries, where the key is the item and the value is a list of user-rating pairs.

4 TOP K RECOMMENDED ITEMS

Our next challenge was to use the predicted ratings to make item recommendations for users. Due to the size and sparsity of our data set, it was difficult to make accurate predictions, because we can only recommend 10 items out of a potential 33,000+, hoping to correctly guess the item in the test set. In order to get any kind of usable results, we used only the 100 users with the most reviews to make our predictions. To get our top recommendations, we predicted the ratings for all unrated and testing set items, using both User-User and Item-Item Collaborative Filtering, and with average and similarity weighted rating predictions. The top k items with the highest ratings were taken as the recommended items results for that user. In our case, $k = 10$. With the recommended items, we can calculate the precision, recall, F-Score, and Normalized Discounted Cumulative Gain (NDCG) based on how many recommended items matches those in the test set for that user.

	MAE	**RMSE**
U-U Avg	0.30189	0.61650
U-U Sim	0.30600	0.65276
I-I Avg	0.34592	0.65495
I-I Sim	0.32841	0.67072

Table 1: Rating Prediction Results

5 RESULTS

The results for our ratings predictions can be seen in Table 1. In general, user-user collaborative filtering slightly outperformed item-item collaborative filtering. We speculate that this is because users have a higher amount of average ratings than items. Additionally, taking the average of the top k most similar users outperformed taking a similarity-weighted average. In an environment where users' similarity was closer and didn't depend on one or two overlapping items due to the sparsity of the dataset, we would expect that similarity-weighted approach would lead to better performance.

The results for our top $k = 10$ recommended items can be seen in Table 2. Our accuracy values for this category were very low across the board. As mentioned previously, this is largely due to the fact that we are attempting to make a correct prediction of a very few number of items in the test set among 30,000+ possible options. Additionally, the majority of those options have a rating of 5 stars, so it is difficult to differentiate by rating. In this category, ranking items by using the similarity-weighted average outperformed taking the simple average for both user-user and item-item collaborative filtering. This is closer to what we expected, however an opportunity for future work would be to investigate why the simple average rating recommendations were more accurate, but

	Precision	**Recall**	**F-measure**	**NDCG**
U-U Avg	0.00200	0.00070	0.00104	0.00964
U-U Sim	0.00300	0.00120	0.00171	0.01562
I-I Avg	0.00100	0.00011	0.00020	0.00301
I-I Sim	0.00300	0.00055	0.00091	0.019871

Table 2: Top 10 Recommendation Results

the similarity-weighted average items recommendations performed better.

6 CONCLUSIONS AND FUTURE WORK

By using a realistic large-scale data set, we found that performing relatively simple operations quickly become difficult at scale. For one, changing the data structure was required to process the data on a single machine. This change in data structure required manual implementation of many operations that a standard library would normally provide. Storing the data and implementing the algorithm in a distributed fashion may help alleviate this issue. We also found that making "top k" recommendations is difficult with a large, sparse data set because there are so many potential items and relatively few ratings available to match with. An idea for future work would be utilizing users' listening history rather than reviews, as that dataset would be much less sparse. Users' behavior provides much more insight than explicit feedback for applications such as this, so it is likely a better metric to use.

7 REFERENCES

[1] Amazon Review Data. UCSD. https://cseweb.ucsd.edu/ jmcauley/datasets/amazon_v2/