CS 5304 (Data Science) : HW3

Team Members: Jonathan Huang (jhh283@cornell.edu), Rohit Jain (rj288@cornell.edu)

Part 1:

Per the recommendation in the assignment, we chose to work primarily with the MovieLens 100k dataset (link). Although most of this assignment was completed with this dataset in mind, the implementation should be extensible to other Movielens datasets assuming proper configuration.

Note: Our implementation assumes that a "data/" folder is found in the root which contains the unzipped dataset as downloaded from the website (so … "data/ml-100k/").

Part 2:

The MovieLens dataset we looked at contains 100,000 ratings (from a scale of 1 to 5) from 943 users on 1682 movies where each user has rated at least 20 movies. For both users and movies, we have some metadata detailing demographic or categorical data. For users we have: age, gender, occupation, zip. For movies we have release date and genre.

Given the scope of the assignment -- proposing 3 different implementations of a recommendation system which uses *collaborative filtering* -- we came up with the following three approaches.

Approach 1: User-User Collaborative Filtering using Cosine Similarity

The first collaborative filtering approach we looked at was user-user similarity using cosine similarity (one of the basic methods we covered in class). In this approach, the goal is to find users who have rated movies similarly to the target user, and to use these similar users' ratings of the target to estimate the target user's predicted rating.

Similarity in this approach can be estimated using cosine similarity which is an approach for measuring the similarity between 2 vectors (in this case user-indexed rating vectors). Theoretically, cosine similarity measures the "cosine of the angle" between two vectors and judges the orientation rather than the magnitude. It can be described by the formula below (where A and B are the rating vectors for user A and user B). As we are only in operating in positive space (rating values are positive), cosine similarity provides the benefit of having values bounded from [0, 1] (assuming we ignore missing ratings).

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Specifically, this approach can be implemented in the following manner:

- For each user in the dataset, calculate cosine similarity with the target user using the array of ratings belonging to each user (ignoring 0 rated movies). This gives us a list of how similar the other users are to the target user.
- Taking the top N similar users to the target user, find the weighted score for the target movie given the neighbors' reviews of the movie. This can be calculated by taking a weighted sum of the neighbors' ratings (weighted by the user-user similarity score) and dividing by the sum of the similarities. This gives us the projected score.

- Recommendations can be made for the user by finding the top scoring movies which the user has not reviewed.

There are two primary issues with this approach. In our case with movies, user-oriented filtering may be unideal because of the complex tastes that humans have especially when compared to item-item filtering. It is harder to model human similarities in taste than the similarity in movies. Also, cosine similarity, although nicely bounded for positive values, has the downside of penalizing non-contributions -- something that is inherent for this problem considering the sparse nature of a review-oriented dataset. It also doesn't take scale into account.

Approach 2: Item-Item Collaborative Filtering using Pearson Similarity
The natural alternative to user-user cosine similarity filtering is item-item pearson similarity filtering. In the case of item-item filtering, instead of calculating the similarity between users, we find the similarity between items (in this case movies) and use these similarities to estimate the ratings for similar items.

The proposed similarity measure for this approach is the more canonical Pearson similarity measure which is given by:

$$r = r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Pearson correlation similarity is a measure of the linear correlation between two variables X and Y, giving a value between +1 and −1 inclusive. In our case, it has the added benefit of taking scale into account by normalizing against the mean and standard deviation.

Using this similarity measure, we can get recommendations using a similar process as user-user filtering:

- For each movie in the dataset, isolate users who have reviewed both movies (target and items being iterated on) and calculate pearson similarity between the user ratings of the two movies. This gives us a list of how similar each of the global movies are to the target movie.
- Using these similarity values, calculate the sum of ratings given by the user on the items similar to the target movie (top N or all of them). Each rating is weighted by the corresponding similarity between the two items. This amount is normalized against the total similarity value. Like the user-user case, this gives us the predicted rating

$$P_{u,i} = \frac{\sum_{j \in N}(s_{i,j} * R_{u,j})}{\sum_{j \in N}(|\ s_{i,j}\ |)}$$

- Using these ratings, we can make recommendations by finding the top scoring movies which the user has not reviewed.

Although this technique (as covered in literature) is relatively robust, it is pretty general and has room for improvement considering the specific MovieLens dataset we are working with. We think that, by explicitly looking at deviations from an "average baseline," we can remove biases (on both movies and users) and better handle sparse data (where similar items may not be available). Also, as a lot of the complaint on collaborative filtering-based recommendations is a lack of diversity, we believe that, by incorporating genre information in the final step, we can make more genre-diverse recommendations to improve the end experience (when it comes to application).

Approach 3: Baseline/Deviation Based Item-Item Collaborative Filtering (Pearson) with Genre-Diverse Recommendations

The approach that we ended up implementing and the method we propose in this section is an extension on Approach 2. Specifically, we build on some of the shortcomings we encountered while implementing approach 2 with our dataset.

Because of the extremely sparse nature of our dataset, we found that many items (depending on the training/validation split) may not have any "similar" items for generating a score estimate -- resulting in score estimates of 0. In order to solve this, we chose to implement a baseline and deviation approach ("weighted average"). Specifically, we generated rating estimates using the following formula:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

$$b_{xi} = \mu + b_x + b_i$$

- μ = overall mean movie rating
- b_x = rating deviation of user x = (*avg. rating of user x*) − μ
- b_i = rating deviation of movie i

Using this method, we calculated a baseline estimate for the target movie using the formula: global average rating for dataset + (average rating for target user - global average) + (average rating for target movie - global average). We then add this baseline estimate of the target movie to a weighted sum of the difference between actual rating and baseline estimates for all similar movies where this difference is multiplied by the pearson similarity score. This weighted sum is then normalized by the sum of all similarity scores. This results in the predicted rating.

In the case of movie recommendations, the common use case is to find and display "top recommendations" for a user. Normally, this would be the movies with the top predicted ratings which the user has not yet reviewed. The issue with this approach is that this often gives very homogeneous and non-diverse recommendations (for example, all movies of one genre) which, in practice, is not ideal. In order to improve on this, we propose making recommendations based on a combination of predicted rating combined with genre diversity. Specifically, after collecting the top predicted score movies (which the user has reviewed), we recommend the top 5 movies by score as well as 5 additional movies from the top score set which have "different" genres from the top 5 movies. We do this by iterating over the review

set and first storing the genre classifications [as specified by our dataset] of the top 5 movies. We then iterate through the rest of the sorted list of movies (by rating) looking for genres that are not captured in the current recommendation set and adding these movies as we find them. If no movies of different genres are found, we accept repeat genres maintaining as much diversity as possible.

To recap the following steps outline our approach for generating recommendations:

- For each movie in the dataset, isolate users who have reviewed both movies (target and items being iterated on) and calculate pearson similarity between the user ratings of the two movies. This gives us a list of how similar each of the global movies are to the target movie.
- Using these similarity values, calculate a predicted rating by adding the baseline estimate score of the target movie to a weighted sum of the difference between actual and predicted baseline scores for all similar movies. Each baseline score is weighted by the pearson similarity value between the two items and is normalized against the total similarity value. This gives us the predicted rating.
- Using these ratings, we can make recommendations by finding the top scoring movies which the user has not reviewed. We make 10 recommendations -- 5 of which are the top predicted scores, 5 of which are the next highest scoring scores which have different genres (see details above)

Part 3:

As discussed above, we implemented approach 3.

See attached for the source code of our implementation. The instructions to run the code are there in the readme file submitted along with the code. It can also be viewed here.

Part 4:

In order to evaluate our model, we first needed to partition our dataset into training and validation sets. We implemented this in two separate ways :

1. Validation set = random sampling of 30% of total reviews
2. Validation set = reviews belonging at the intersection of a random sampling of 40% of users and 70% of movies

Both of these approaches give us a ~30%/70% split between validation set and training set. The primary difference is that in approach 1, we are guaranteed a consistent sized validation set (where we are not guaranteed consistency of having “perfect information” users and movies), while approach 2 doesn’t guarantee a consistent size but does guarantee having complete information for some users and movies (in order to effectively train our recommendation engine). After some consideration, we decided that approach 2 would be better for this reason -- we wanted to make sure that there was some baseline of information to find similar elements.

The evaluation metric we decided to go with was Root Mean Square Error where RMSE was calculated by (for each point in the validation set) squaring the difference between predicted rating (using the classifier) and the true rating and taking the square root.

Team Members: Jonathan Huang (jhh283@cornell.edu), Rohit Jain (rj288@cornell.edu)

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}.$$

Admittedly, there are some issues with RMSE. Specifically, it performs poorly at taking into account context and diversity. Because it takes the sum of the squared errors (where every point matters the same amount), it penalizes methods that performs well on providing high ratings (practically most of what we care about for a movie recommendation engine) but performs poorly for low ratings. One proposed solution for accounting for this would be to weight the error value proportionally to the true rating that it corresponds with (where high ratings are highly weighted).

Even considering this; however, we feel that RMSE does have its merits. As covered extensively throughout the Netflix study (source), RMSE performs well at improving the relative “quality” when suggesting the top K movies (what we have implemented here -- and much of how relative recommendations would be used in practice). Even though optimizing RMSE may introduce errors in the prediction, it does help refine the relative predicted order values (making it closer to the true state). According to the post in the Netflix competition: “small improvements in RMSE translate into very significant improvements in quality of the top K movies. In other words, a 1% improvement of the RMSE can make a big positive difference in the identity of the "top-10" most recommended movies for a user.”

In the case of our model, we observed a **RMSE** of: **1.45**. This is an improvement over the initial implementation of Approach 2 which had an RMSE 3.49 (a shockingly bad score considering us operating on a scale of 5).