**Introduction**

Recommender systems are found everywhere and have become an essential component in helping companies guide their users throughout their products. As discussed throughout our summary, this project aims to compare different recommender system constructions and algorithms on the MovieLens dataset. At a very high level, there are two ways we hope to tackle this project. The first approach is to solely rely on previous users’ ratings and combine them in some sort of way to predict all other ratings, and the second approach is to incorporate user metadata such as user occupation and age.

As previously discussed, one of the biggest obstacles we expect to face is the matrix sparsity. As shown in table 1, our dataset (1M Movielens) has a density of 4.26%.

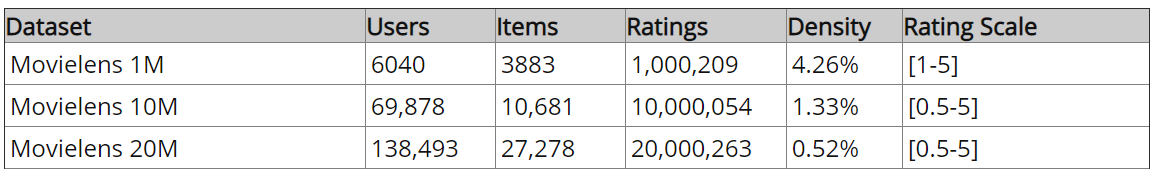


Table 1: Dataset Overview [1]

The first thing to consider is choosing our loss function and performance metric. Our initial approach was treating this as a classification problem, adopting 0-1 loss and measuring our performance using CCR. However, such a method would consider it wrong if a ground truth label is 3 and our prediction was 3.5, even though such a case is relatively correct. For this, we decided to use mean absolute error in measuring performance and treat this as a regression problem with the following constraint. Since ratings are only allowed in 0.5 increments from 0.5 to 5, we will be rounding our predictions to the nearest 0.5.

The plan is to approach the problem of recommendation system on the MovieLens dataset in 3 ways. The first is a very simple approach where we consider just the user ratings to build a model that recommends movies. Here we plan to build a synthetic dataset and use algorithms like correlation, Matrix Factorization to predict ratings and recommend movies and then apply the algorithm on MovieLens 1M. The second approach would be to consider the metadata of the users to group them into clusters and use that information to recommend movies. Finally, the third approach would be the combination of the first two, where we use both movie ratings and user metadata to predict ratings of movies that the user has not watch and recommend movies.

**Correlation**

We first looked into the property of the rating data. For each movie, we extract the average rating among all users (average rating) and the number of users who rate the movie (rating counts). We plotted the correlation (shown in Figure 1) between rating counts and average rating and found that: (1). Most of movie are not rated (rating count is low), which means sparsity will be a problem during the next step. (2) Some of the movie has a positive correlation between rating counts and average rating, which means a highly rated movie will have more rating count.

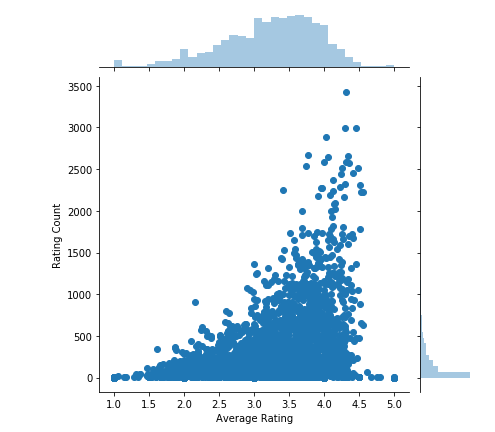


Figure 1. Property of the rating data. Average rating versus rating count.

Our initial approach is to check the correlation between different movies including linear correlation (Pearson correlation) and nonlinear correlation (Spearman correlation). We calculated the Pearson correlation and Spearman correlation between every two movies. For each movie, we have a vector containing the rating from different users. The results are shown in Figure2. We can see that most of the movie do not have any correlation (coefficient = 0) and we cannot correlation to predict the behavior between movies. This is because: (1) Due to the sparsity of the data, the result of process two vectors will be zero and thus most of the correlation will be zeros. (2) This correlation only considers the rating of all users and fails to consider the preference of each user.

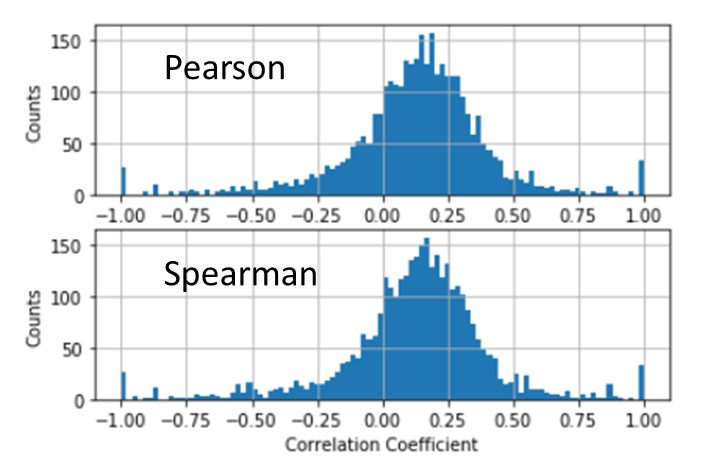


Figure 2. Histogram of correlation between every two movies.

**Matrix Factorization using Single Value Decomposition (SVD)**

One of the main things to consider when building this dataset is that although the matrices involved in recommender systems are extremely large, their rank is usually relatively small. It is for this reason that many people suggest the implementation of matrix factorization and reconstruction on such problems.

As shown in [2] and [3], a matrix (R) can be deconstructed into smaller matrices which can then be multiplied together reproduce the original matrix.

As discussed in [3], the intuition behind the deconstruction of our matrix is as follows:

* this deconstruction of the ratings matrix R, will yield some insight into user preference and movie features, U and V respectively. The matrix is diagonal matrix which will then somehow weight different users and movies accordingly to restore the original ratings.

After discussing with Dr. Ishwar, we have concluded the following:

* This deconstruction requires the presence of a complete ratings matrix R. For this, we will initially fill the matrix with the mean value of each movie, then iteratively deconstruct and reconstruct our matrix while reasserting the training values to their original ones throughout each iteration.

**Synthetic Dataset**

Before applying this algorithm to the movielens dataset, we decided to construct a synthetic dataset which we could use to first build and test our algorithms in a controlled manner before tackling the real-world data. This would help us understand our best-case scenarios when testing each algorithm on the actual dataset. Keeping this in mind, we constructed a randomized, low-rank matrix, whose values range from 0.5 – 5, as can be seen below.

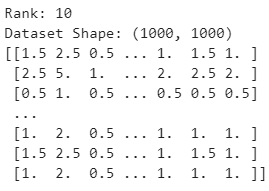


Figure 3: Synthetic Dataset

We wanted to visualize the effect of matrix sparsity on our algorithm discussed above, so we varied the density of our matrix and plotted the MAE while keeping all other variables constant. This is shown in Figure 4. From here, it is quite clear that, as expected, the MAE is indeed largely affected by matrix density dropping as low as 0.25 when at 90% density.

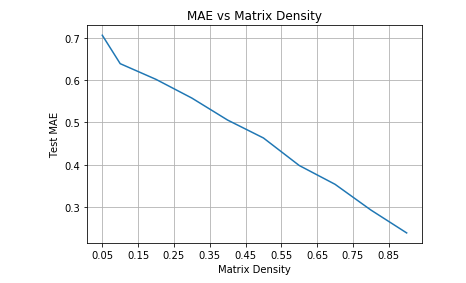


Figure 4: Test CCR vs. Density

Now, we set our matrix density to 20% to run the algorithm and recorded our mean absolute error after each iteration. However, as shown in Figure 5, unlike what is expected, the Mean Absolute Error increased after every iteration but the change in MAE is 0.008 which is almost negligible.

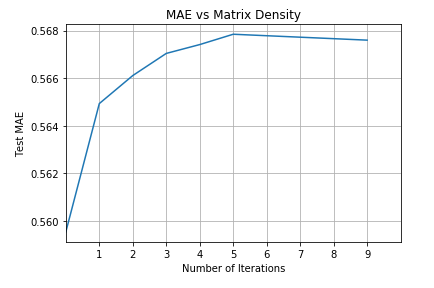


Figure 5: CCR vs Iteration

**MovieLens 1M**

From the learnings of applying the algorithm on synthetic data, now moving to MovieLens 1M, a real world dataset with 96% of data sparsity, after running the algorithm with an 80-20 split, we were able to achieve a MAE of 0.66.

Similar to what we observed with the synthetic data, as shown in figure 6, the MAE grew for every iteration but the delta change was 0.05, again almost negligible.

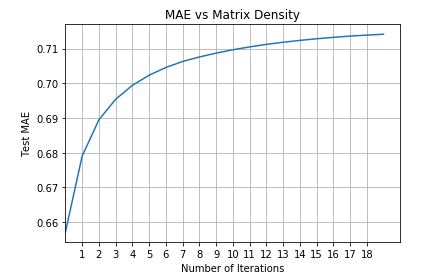


Figure 6 Mean absolute error versus matrix density.

**Work that needs to be done**

1. We proceed to the second approach where we and cluster the users based on their metadata and then recommend movies to the users based on their clusters.
2. We then proceed to the third approach where we plan to run an SVM on the ratings of the user and the metadata to predict the ratings of user for all movies. Using that rating information we plan to cluster the users and then use that cluster information as an additional feature and run the data through algorithm to make predictions and recommend movies.

**References**

[1] Gude, A. (2019). *9 Must-Have Datasets for Investigating Recommender Systems - KDnuggets*. [online] KDnuggets. Available at: https://www.kdnuggets.com/2016/02/nine-datasets-investigating-recommender-systems.html [Accessed 26 Oct. 2019].

[2] Koren, Y., Bell, R. and Volinsky, C., 2009. Matrix factorization techniques for recommender systems. *Computer*, (8), pp.30-37.

[3] nick becker. (2019). Matrix Factorization for Movie Recommendations in Python. [online] Available at: https://beckernick.github.io/matrix-factorization-recommender/ [Accessed 22 Nov. 2019].