**Solution Approaches:**

Matrix Completion

As previously discussed, SVD is an algorithm which decomposes a matrix into 3 matrices and reconstructs the original one using the product of matrices. For this to happen, however, SVD requires a completed matrix to being with. As previously discussed, the matrices examined in recommender systems are extremely space and their entries must be filled in. After experimenting with SVD, it was quickly apparent that the method of completion of the sparce matrix with significantly impact the result. For this, we constructed different methods which include completion using zeros, average user ratings, average movie ratings, and a combination of user and movie ratings.

**Implementation:**

**Correlation**

Throughout this method, we aimed to find the correlation between the user behavior across different movies. To do this, we extracted all ratings for each movie into their own vector, then obtained person’s and spearman’s correlations coefficients between each movie and all other movies.

The assumption is that spearman’s correlation would better understand the dataset since it can capture non-linear correlations more effectively.

**Synthetic Dataset Generation**

Before constructing any algorithms and implementing them on the MovieLens 1M dataset, it is important to build a concrete understanding of some factor which can significant impact the performance of a recommendation system. For this, we constructed a synthetic dataset througha set of setps described below, then altered different parameters to understand the variation in performance with each parameter.

* Generate a random vector of IID integers of size d
* Obtain the outer product of the vector to generate a rank 1 matrix
* Generate a zero dxd matrix with 1’s populating r entries of the diagonal
* Get a pointwise sum between the matrices above resulting in a matrix with rank approximately r
* Rescale all entries in the matrix using the max and min points to make all entries between 0.5 to 5
* Round all entries to the nearest 0.5 to simulate a rating system

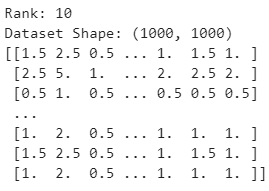


Figure 1: Synthetic Dataset of rank 10

**SVD**

Matrix Completion

To fill all entries in the matrix, we first obtained the mean of every user across all their ratings, as well as the mean of every movie across all users, and used these two vectors to complete the matrix, as required.

Regular Reconstruction

Implementing SVD was done through using the function provided by scipy.sparse.linalg in python.

This function takes the completed dataset, as well as a parameter which decides the number of terms to keep in the resultant diagonal matrix, sigma. This is important as it essentially decides the rank of the resultant matrix. If set too high, computational complexity would exponentially increase, if set too low, the model will lose important information required for reconstruction. This parameter was chosen using a validation set.

Once the function has been called, the reconstruction is computed by multiplying the resultant matrices. Next, all entries are rounded to the nearest 0.5.

Iterative Reconstruction

Once the reconstruction has been completed, we decided to return all original entries in their position, yielding us the original matrix with a “better estimate” of the unknown entries, this matrix now essentially serves as a more accurately completed matrix for the next iteration. Then the SVD function is called again and this process is repeated iteratively until some tmax.

**Regression**

This approach is fundamentally different to SVD in that it utilizes user and movie metadada. Here, we first loaded all user and movie information which include user id, movie id, age, occupation, gender, genre, and rating. We then onehot encoded gender and multihot encoded the movie genre. Occupation was already provided as number ranging from 0 to 23, so it was used as is. We then computed the average user rating and average movie rating and appended these values to all entries of our vectors.

To understand the importance of the features above, we then ran multiple OLS and Ridge models, each using a subset of the features described. In three cases can be seen below:

Case 1: User Features

* UserID
* MovieID
* Age
* Occupation
* Gender

Case 2: User & Movie Features

* UserID
* MovieID
* Age
* Occupation
* Gender
* Genre

Case 3: All Features

* UserID
* MovieID
* Age
* Occupation
* Gender (OneHot)
* Genre (MultiHot)
* Average user rating
* Average movie ratings

In all cases, the labels used were the entry ratings.

The OLS and the Ridge regression functions used can be found in sklearn.linear\_model in python.

**Combined Approach**

Since SVD and Regression work very differently, we were curious to find out if we could find a way to combine both methods. After some discussion, we decided to implement iterative SVD to complete the matrix, then use the completed matrix as the starting point for linear regression. This, in essence serves as some form of data augmentation to the linear regression models since the number of entries is exponentially increased as compared to the initial density of 4%.

**Experimental Results:**

**Correlation**

One of the important things to understand which could provide some understanding for the problem at hand is detailed in figure 2. This plot denotes the average rating of each movie vs the rating count. Each of these parameters have their own histogram shown on the respective axis as well. The key conclusion from this graph is that there seems to be some sort of positive correlation between rating count and average rating i.e. the more people rate a movie the higher its mean rating is likely to be

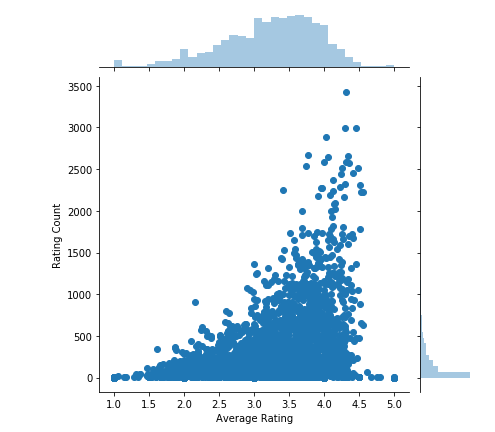


Figure 2: Rating Count vs. Average Rating

After implementing the discussed procedure, the following plot was generated. This plot describes the correlation coefficient of one of the movies with all the others. Although this could technically work as a means of recommending movies to users, it has many downfalls. Firstly, most values are near 0 which means no correlation. This is due to the scarcity of the matrix it is highly unlikely that multiple movies have been watched and rated by all the users which means most ratings are being ignored in this computation. Next, assuming the sparsity issue was somehow taken care of, all users who liked this movie will get the same recommendation, the next most correlated movie. This system delivers recommendations based on aggregate user behavior, ignoring variations in user to user preferences.

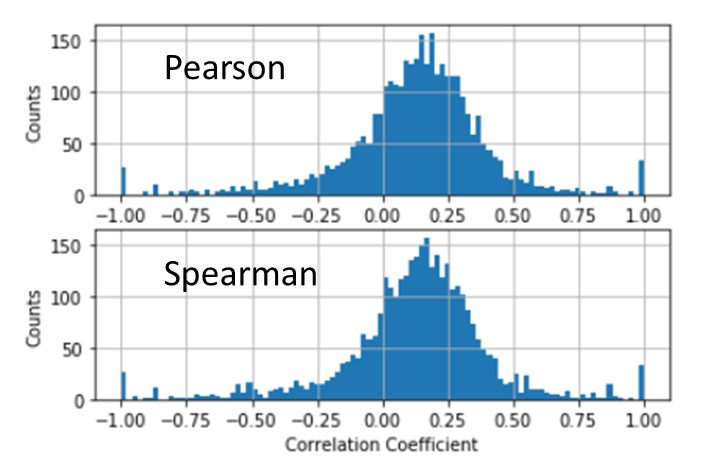


Figure 3: Correlation Coefficients of one movie with respect to all others

**SVD**

Synthetic Data

After constructing the data as explained, we ran the iterative SVD algorithm on it to better understand the performance of the algorithm with respect to changes in initial matrix density as well as rank. The figures illustrated below each describe the resultant MAE with respect to these parameters. These results are averaged over 50 iterations with tmax = 100 on each run.

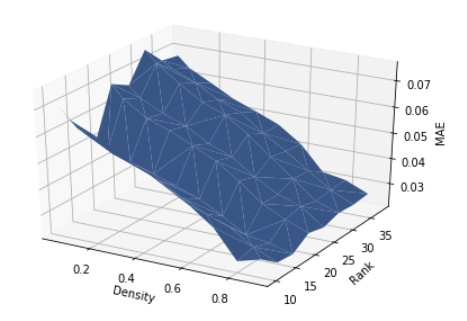


Figure 4: SVD MAE vs. Rank & Density (Synthetic Data) v1

From the above there is a clear improvement in MAE with increases in matrix density; however, the relationship between rank and MAE is not clear. For this, the whole process was repeated another time yielding the figure below. The two figures follow relatively the same behavior with respect to matrix density; however, there still doesn’t seem to be a clear direct relationship between rank and MAE. This could be due to the difficulties involved in generating a low rank matrix which make the resultant rank not equal to the required ones. It could also be due to the fact that the original matrix is generated from only a couple vectors which makes capturing its information using SVD fairly simple with even very low k values.

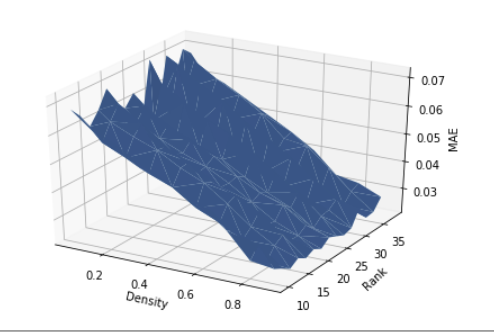


Figure 5: SVD MAE vs. Rank & Density (Synthetic Data) v2

Moving on from that, we then explored the variation of MAE with density for a set rank. This can be seen in the figure below. As expected, it is essentially the same graph as the ones above, as viewed from the side.

Dataset Shape: (100, 100)

Dataset rank: 10

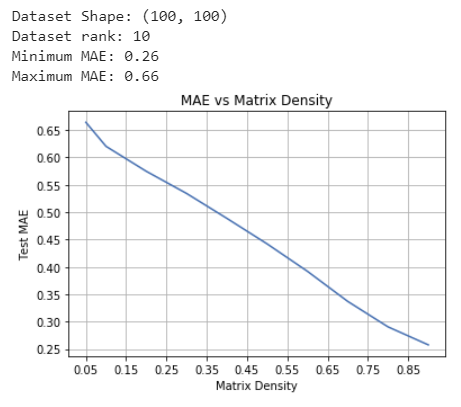


Figure 6: SVD MAE vs. Density (Synthetic Data)

Finally, we wanted to confirm the validity of iterative SVD, at least in theory. The plot below illustrates the improvement in MAE of Iterative SVD after every iteration with a set rank and density.

Dataset Shape: (100, 100)

Dataset rank: 11

Density: 0.2

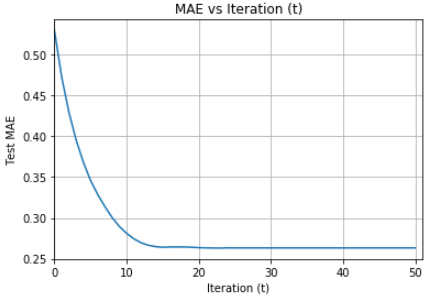


Figure 7: SVD MAE vs. Iteration (Synthetic Data)

Real Data

To properly understand the results of SVD on the real dataset, we first obtained the resultant MAE after the initial matrix completion step. These values will serve as a method of comparison for the results of iterative SVD, as well as an understanding of the impact of the starting point for SVD on its final results.

Four different matrix completion approaches were followed and their results can be seen below.

Table 1: Intial MAEs of different matrix completion approaches

|  |  |  |
| --- | --- | --- |
| **Method** | **Initial MAE** | **Final MAE** |
| Zeros | 3.581 | 1.083 |
| Average user rating (AUR) | 0.829 | 0.714 |
| Average movie rating (AMR) | 0.782 | 0.701 |
| Average of AUR and AMR | 0.771 | 0.699 |

Clearly, the initial method, filling with zeros, yielded the worst performance. This makes perfect sense since it incorporates the least information on the problem. Similarly, although through a very marginal amount, completion using user and movie means together, last table entry, yielded the best performance as it captured the most information on the problem. One important thing to point out is that although the last method performed better than the others, the final MAE the algorithm converges to in all cases, excluding the zero approach, was almost the same.

Finally, is interesting to see that completion using AMR alone yielded better results that using AUR alone. This means that the average rating of each movie better captures the behavior of the entire matrix. This goes hand in hand with our next result which can be seen below in which we validate the weighting parameter lambda between AMR and AUR.

MME: missing matrix entry

AUR­u: Average user rating for user u

AMR­m: Average movie rating for user u

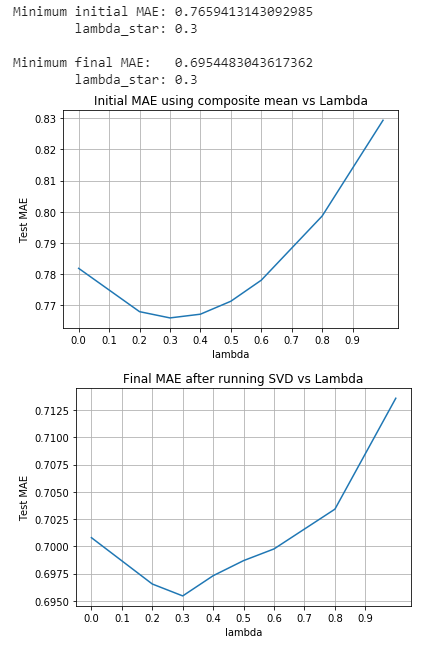
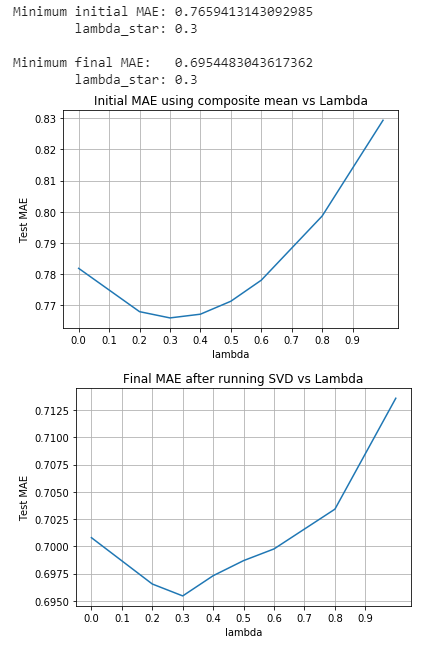


Figure 8: MAE on composite means vs Lambda (

From the above, and yields an initial MAE of 0.766. and a final MAE of 0.695