STAT 542 Final Project

Group 4

2023/05/12

1 Data set and research goal

Our research goal for the final project is to predict and complete the restaurant feedback matrix using different matrix completion algorithms. Before we could achieve this we first needed to train and compare the improvement over the 'naive' guess for both training datasets for each of our algorithms. The algorithms were trained using both the TripAdvisor and MovieLens datasets. The TripAdvisor dataset contained a total of 980 users and 10 different items (art gallery, beach, theatre, etc.). This dataset is a complete matrix in which the average ratings (0-5) for a given item is used for each user.

art_gallery	dance_club	juice_bar	restaurant	museum	resort	park	beach	theatre	religious_institution
0.93	1.80	2.29	0.62	0.80	2.42	3.19	2.79	1.82	2.42
1.02	2.20	2.66	0.64	1.42	3.18	3.21	2.63	1.86	2.32
1.22	0.80	0.54	0.53	0.24	1.54	3.18	2.80	1.31	2.50
0.45	1.80	0.29	0.57	0.46	1.52	3.18	2.96	1.57	2.86
0.51	1.20	1.18	0.57	1.54	2.02	3.18	2.78	1.18	2.54
0.99	1.28	0.72	0.27	0.74	1.26	3.17	2.89	1.66	3.66

Figure 1: First 6 rows of TripAdvisor dataset

To use the matrix factorization methods, we want, we removed fifty percent of the values randomly using set.seed(100) in R. Our second training dataset, MovieLens dataset, describes 5-star ratings and free-text tagging activity from MovieLens (a movie recommendation service). The dataset contained ratings from 600 users for 9000 different movies. We compared the percentage of improvement over the naive guess each of our algorithms across both datasets and chose the matrix completion algorithms that performed the best overall. This leaves us with having the best possible algorithms to complete our restaurant feedback matrix.

The following shows the algorithms we chose to use in our initial testing using the TripAdvisor and MovieLens datasets:

POPULAR ALS LIBMF SVD UBCF IBCF

2 Algorithms Used

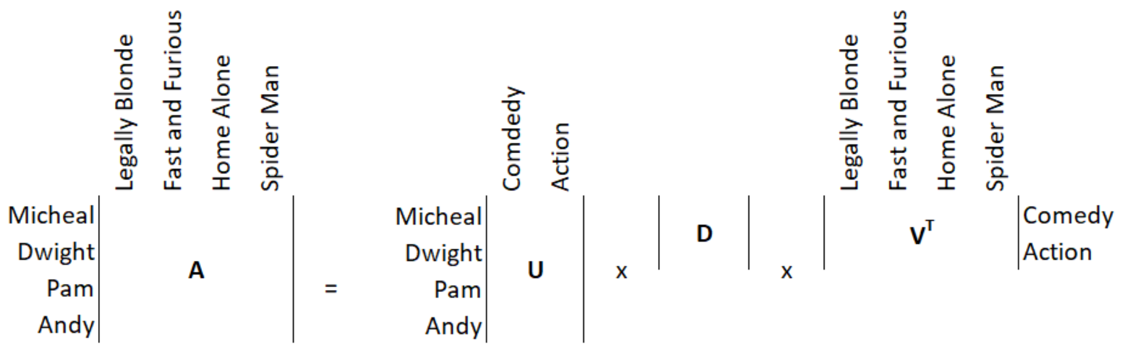
Out of a big number of algorithms that we had during this class (and more), the three that performed the best all happened to be matrix factorization algorithms. Matrix factorization is a family of algorithms that works by decomposing the original matrix of ratings into two (sometimes three) lower-ranked matrices and then using the product of those 2 (3) matrices to reconstruct the original matrix, and to fill as many missing entries as possible (in our case, we only used parameters that resulted in full matrix completion).

2.1 Singular Value Decomposition (SVD)

SVD is a type of matrix factorization technique that decomposes a given $m \times n$ matrix, A into the product of three matrices U, D, and V, where U is a $m \times r$ orthonormal matrix, V is a $r \times n$ orthonormal matrix, and D is a $r \times r$ diagonal matrix with positive real values sorted in descending order.

$$A = UDV^T$$

For example, let us consider a matrix A, which consists of the user's movie preferences, where m represents the number of users and n represents the number of movies and the values inside the matrix consist of the user's ratings of the movies. Using SVD, the matrix A decomposes to U, V, and D. The U matrix represents m users and r genres and the values inside the matrix represent the genre preference of each user. Similarly, the V matrix represents r genres and n movies, where the values inside the matrix represent the score of each movie belonging to a particular genre. Finally, D consists of the "strength" of each genre.



SVD was implemented in R using the `recommenderlab` library. Since the algorithm cannot decompose the matrix in the presence of missing values, `recommenderlab` imputes the missing values by column means before decomposition. Furthermore, it also takes in a parameter k that controls the rank of matrix D which results in an approximated matrix, $B = U_k D_k V_k^T$. Grid search technique was used to identify the most optimal rank k, that determined the best approximation matrix B with respect to mean square error.

2.2 Alternating Least Squares (ALS)

$$\text{Factorization: } R \approx X^T Y \text{ , where } X \in \mathbb{R}^{k\times n},\ Y \in \mathbb{R}^{k\times m},\ R \in \mathbb{R}^{n\times m}$$

$$\text{Objective: } \min_{X,Y} \sum_{r_{ui}\ observed} \left[(r_{ui}^2 - x_u^T y_i)^2 + \lambda\left(||x_u||_2^2 + ||y_i||_2^2\right)\right]$$

$$\text{where } x_u \in \mathbb{R}^k \text{ - user, } y_i \in \mathbb{R}^k \text{ - item.}$$

Notice that here the objective function isn't convex because of $x_u^T y_i$ term, so, the direct approach (e.g., using gradient descent) will be largely ineffective (slow, costly, rough approximate). But if X is fixed, the objective function becomes convex in Y, and vice versa. The 'know-how' of ALS is that at every iteration, first Y is fixed and the gradient descent is applied to X, and then vice versa. Below is the step-by-step ALS algorithm for this particular objective function.

Algorithm 1 ALS for Matrix Completion

```
Initialize $X, Y$
repeat
    for $u = 1 \ldots n$ do
        $$x_u = (\sum_{r_{ui}\in r_{u*}} y_i y_i^\intercal + \lambda I_k)^{-1} \sum_{r_{ui}\in r_{u*}} r_{ui} y_i$$
    end for
    for $i = 1 \ldots m$ do
        $$y_i = (\sum_{r_{ui}\in r_{*i}} x_u x_u^\intercal + \lambda I_k)^{-1} \sum_{r_{ui}\in r_{*i}} r_{ui} x_u$$
    end for
until convergence
```

2.3 LIBMF

LIBMF is a high-performance C++ library for large scale matrix factorization. LIBMF itself is a parallelized library, meaning that users can take advantage of multicore CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance. (Chin, Yuan, et al. 2015). To minimize the objective function, LIBMF uses stochastic gradient descent. The algorithm parallelizes the computation by griding the data matrix into nr_bins^2 blocks (parameter tuning is not needed for nr_bins in most cases, according to the authors). The objective function that LIBMF minimizes is the following:

$$\min_{P,Q} \sum_{r_{ui}\ observed} \left[f(p_u, q_i; r_{ui}) + \mu_p ||p_u||_1 + \mu_q ||q_u||_1 + 0.5\lambda_p ||p_u||_2^2 + 0.5\lambda_q ||q_i||_2^2 \right]$$

For `recommenderlab` package in R, the options are limited to the following:

$$f(p_u, q_i; r_{ui}) = ||r_{ui} - p_u^T q_i||_2^2;\ \mu_p = \mu_q = 0$$

While in C++ for real-valued matrix factorization one could choose between squared error (L2-norm), absolute error (L1-norm) and generalized KL-divergence (NMF is required). `recosystem` package in R is a wrapper of LIBMF, hence it inherits most of the features of LIBMF, and allows to tune a lot more parameters rather than just λ_p and λ_q, such as L1 regularizations (μ_p, μ_q), SGD learning rate, number of iterations etc. We, however, stuck with `recommenderlab` because it was much simpler to use in R and we only were missing out on L1 regularizations since we evaluate our models on MSE, which inherently means we are bound to use L2-norm as $f(p_u, q_i; r_{ui})$.

3 Results: % improvement over 'naive' guess

Below is the table that shows, by how much the MSE has improved when the predictions using each of our algorithms replaced the 'naive' prediction of just filling all of the missing values of the matrix with its mean.

Algorithm	POPULAR	ALS	LIBMF	SVD	UBCF	IBCF
Tripadvisor	58%	**74%**	***78%***	64%	61%	-
MovieLens	23.52%	24.12%	***26.84%***	**25.81%**	25.31%	24.60%

Note that IBCF didn't manage to complete the matrix for TripAdvisor dataset, there was simply no rank such that the decomposition yielded a full matrix reconstruction. As we can see here, LIBMF performed the best for both datasets, none of which were particularly large scale (especially the TripAdvisor one), which signalizes that LIBMF algorithm might be the best one for smaller datasets too, like our Feedback dataset. Hence, we decided to complete the restaurant ratings matrix using LIBMF. But before that, here are the parameters that we chose for the two datasets we tested our algorithms on:

Algorithm	ALS	LIBMF	SVD
Tripadvisor	$\lambda = 0.02$	$(\lambda_p, \lambda_q) = (0.015, 0.035)$	$r = 2$
MovieLens	$\lambda = 0.1$	$(\lambda_p, \lambda_q) = (0.1, 0.1)$	$r = 10$

4 Challenges

We list the challenges we faced and our solutions:

- High Compute Time: ALS takes a while to compute, which is especially costly during parameter tuning (186 sec for 20-iter. single loop on TripAdvisor). We found that using LIBMF, a parallelized algorithm similar to ALS, was significantly faster (29 sec for 961-iter. double for loop on TripAdvisor) while producing similar results.

- MovieLens Dataset Sparsity: The MovieLens dataset is sparse, with 100,000 reviews across 600 users and 9000 movies. Furthermore, the distribution of ratings per user and ratings per movie follow a power law, such that some users have very few ratings and some movies have very few ratings. To reduce computation time and make our results more meaningful, we removed users with less than 50 ratings, as well as movies with less than 50 ratings.

- Hyperparameter Tuning: Matrix factorization methods include many hyperparameters including the rank for SVD, number of users/items for UBCF/IBCF, and regularization terms for ALS/LIBMF. To determine the best hyperparameters we use a cross-validation approach where we generate multiple validation sets by randomly removing half the elements in the matrix. Each validation set has a different random set of elements removed, and we measure the prediction error on the removed elements.

- Few items for TripAdvisor dataset: Since we only had 10 items for the TripAdvisor dataset, IBCF was unable to complete most of the matrix. As such, we were unable to compare it with other algorithms for TripAdvisor.

5 Feedback Predictions

As breefly mentioned in Section 3, we used the LIBMF algorithm with hyperparameters costp_l2 = 0.1 and costq_l2 = 0.1 which we determined to the be the best algorithm and hyperparameters from the TripAdvisor and MovieLens datasets. Performance on the MovieLens dataset was sensitive to hyperparameters while performance on TripAdvisor wasn't, so we chose to use the hyperparameters from MovieLens.

After completing the matrix, we rounded values to the nearest integer to fit the original rating format. We chose to discretize our output by rounding to the nearest integer to minimize mean squared error, which is the loss function of our algorithm.

Our final predictions can be found in the Appendix. The same exact prediction matrix, but in .csv format, will be attached to the project

6 Appendix

6.1 Code

Please find the code, along with the presentation and datasets used for training, inside the .zip archive with this report you're reading.

6.2 Project Report/Presentation Contributions

Name	Contribution
Lomasov, Ilia	Section 2 (ALS & LIBMF algorithms), Section 3 (TripAdvisor results, comparison tables)
B, Chidharth	Section 1 in Presentation (Dataset and research goal), general project discussion
Samantula, Charumeghana	General project discussion
Edwards, Grant	Section 1 (Dataset and research goal), Section 4
P, David	Section 3 (MovieLens results), Section 4, Section 5 (full)
Kota, Yasomitra Sampat	Section 2 (SVD algorithm), Presentation (SVD algorithm)

6.3 Feedback Predictions

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
3	3	3	3	3	3	3	3	3	3	3	3	4	3	3
4	3	3	3	4	3	3	3	4	3	3	4	5	4	3
3	3	3	3	3	3	3	3	3	3	3	4	3	3	3
4	4	3	3	3	3	3	3	4	2	3	4	4	4	3
5	4	5	5	5	5	5	5	5	4	4	5	5	5	5
1	3	4	2	4	2	4	2	4	2	3	3	2	2	2
4	2	3	4	3	3	2	3	3	3	1	5	4	4	3
2	3	3	2	4	2	3	2	1	2	3	3	1	2	2
3	3	3	3	3	5	3	3	3	4	3	3	3	4	3
5	4	4	5	5	5	4	4	5	4	4	5	5	4	5
4	3	3	3	5	3	3	3	4	3	3	4	4	3	3
2	2	3	2	4	2	3	2	3	2	2	2	4	2	2
4	3	5	5	3	5	4	5	4	4	3	4	5	3	4
4	3	3	3	4	3	3	3	4	3	3	4	4	3	3
2	2	2	2	3	2	2	2	2	2	2	2	2	2	2
4	1	1	1	2	1	1	1	3	1	1	2	3	2	2
4	2	2	2	3	2	2	2	4	1	1	3	5	3	2
3	3	3	3	3	5	4	5	3	4	3	3	3	3	4
4	4	4	3	5	3	4	3	3	3	5	4	3	3	3
4	3	4	4	4	4	3	3	2	2	3	5	4	4	5
2	1	3	3	1	4	2	5	1	5	1	2	3	4	3
5	5	5	5	5	5	5	5	4	5	5	5	5	5	5
3	4	4	2	5	1	5	2	3	2	3	3	3	2	2
4	2	2	2	3	2	2	2	2	2	2	3	3	3	3
3	3	3	3	4	3	3	3	5	3	2	5	4	4	3
1	1	1	2	1	1	1	1	1	1	1	1	1	1	1
5	3	3	3	4	3	3	3	4	3	3	4	5	3	3
2	3	3	1	4	4	5	3	2	4	4	3	3	2	3
5	3	3	4	3	4	3	4	5	5	3	4	5	4	4
4	3	3	3	3	3	3	3	3	2	3	3	3	3	3
2	2	4	2	3	2	2	2	1	2	3	3	2	2	2
5	3	3	3	4	3	3	3	4	3	2	4	5	4	3
2	3	4	2	4	3	4	3	2	3	5	3	2	2	3
5	3	3	3	5	4	3	2	5	2	2	4	5	4	3
4	4	3	4	5	3	4	2	5	2	4	4	3	3	3
5	3	3	4	4	4	3	3	5	3	3	5	5	5	3
5	3	3	3	4	3	3	3	5	3	3	4	3	3	3
4	3	3	3	4	2	3	3	4	2	2	4	5	3	3
5	3	3	4	5	3	3	3	5	3	3	4	5	4	1
5	2	1	3	3	3	2	1	4	2	1	3	4	4	2
5	3	3	3	5	4	3	3	4	3	3	4	3	3	3