Your Name: João Coelho
Your Andrew ID: jmcoelho
Your Nickname on Leaderboard: Joao

Homework 2

0. Statement of Assurance

1. Did you receive any help whatsoever from anyone in solving this assignment? No

2. Did you give any help whatsoever to anyone in solving this assignment? No

3. Did you find or come across code that implements any part of this assignment? I read the PMF paper. I read pytorch documentation for optimization which contains an SGD usage example (https://pytorch.org/docs/stable/optim.html). I read the piazza hint by the TA of creating U and V as nn.Embeddings.

1. Corpus Exploration (10)

Please perform your exploration on the training set.

1.1 Basic statistics (5)

Statistics	
the total number of movies*	5353
the total number of users*	10858
the number of times any movie was rated '1'	53852
the number of times any movie was rated '3'	260055
the number of times any movie was rated '5'	139429
the average movie rating across all users and movies	3.3805

For user ID **4321**	
the number of movies rated	73
the number of times the user gave a '1' rating	4
the number of times the user gave a '3' rating	28
the number of times the user gave a '5' rating	8
the average movie rating for this user	3.1506

For movie ID **3**	
the number of users rating this movie	84
the number of times the user gave a '1' rating	10
the number of times the user gave a '3' rating	29
the number of times the user gave a '5' rating	1
the average rating for this movie	2.5238

1.2 Nearest Neighbors (5)

	Nearest Neighbors
Top 5 NNs of user 4321 in terms of dot product similarity	980 551 2586 3760 90
Top 5 NNs of user 4321 in terms of cosine similarity	8497 9873 7700 8202 3635
Top 5 NNs of movie 3 in terms of dot product similarity	1466 3688 3835 4927 2292
Top 5 NNs of movie 3 in terms of cosine similarity	4857 5370 5391 4324 5065

2. Rating Algorithms (50)

Note: Runtime entry contains [knn_search]+[similarity_computation]. Each similarity computation was done only once and reused.

2.1 User-user similarity (10)

Rating Method	**Similarity Metric**	**K**	**RMSE**	**Runtime(sec)**
Mean	Dot product	10	1.0024	16.0764 + 2.634
Mean	Dot product	100	1.0067	18.0394 + 2.634
Mean	Dot product	500	1.0430	25.5187 + 2.634
Mean	Cosine	10	1.0631	20.9723 + 2.480
Mean	Cosine	100	1.0620	21.8964 + 2.480
Mean	Cosine	500	1.0754	25.4327 + 2.480

Weighted	Cosine	10	1.0627	18.4175 + 2.480
Weighted	Cosine	100	1.0615	18.9409 + 2.480
Weighted	Cosine	500	1.0740	21.1282 + 2.480

2.2 Movie-movie similarity (10)

Rating Method	Similarity Metric	K	RMSE	Runtime(sec)
Mean	Dot product	10	1.0204	10.6550 + 0.486
Mean	Dot product	100	1.0469	16.8292 + 0.486
Mean	Dot product	500	1.1109	35.6159 + 0.486
Mean	Cosine	10	1.0173	11.5502 + 0.424
Mean	Cosine	100	1.0639	16.2195 + 0.424
Mean	Cosine	500	1.1183	29.1351 + 0.424
Weighted	Cosine	10	1.0147	8.8374 + 0.424
Weighted	Cosine	100	1.0566	11.0887 + 0.424
Weighted	Cosine	500	1.1022	17.9204 + 0.424

2.3 Movie-rating/user-rating normalization (10)

Rating Method	Similarity Metric	K	RMSE	Runtime(sec)
Mean	Dot product	10	1.0245	13.4064 + 13.81
Mean	Dot product	100	1.0703	17.1092 + 13.81
Mean	Dot product	500	1.1218	27.8020 + 13.81
Mean	Cosine	10	1.0245	13.4362 + 13.88
Mean	Cosine	100	1.0703	17.5200 + 13.88
Mean	Cosine	500	1.1218	27.7944 + 13.88
Weighted	Cosine	10	1.0216	10.6960 + 13.88
Weighted	Cosine	100	1.0625	12.8552 + 13.88
Weighted	Cosine	500	1.1060	20.6158 + 13.88

Add a detailed description of your normalization algorithm.

I'm using movie rating normalization. Interaction matrix M is (movies, users). Each row is centered around its mean. Each row is normalized by its variance. I then compute a similarity matrix through M@M.T. Given the centering+normalization, this matrix holds PCC-based similarity values. Before standardization, the matrix is imputed, i.e., zero rows are unchanged and others are subtracted 3. This allows the mean/variance to be taken over all row cells. I tried another approach where I would not impute the matrix, and take the mean/variance only on non-zero cells, but the results were worse than the imputed approach.

2.4 Matrix Factorization (20)

a. Briefly outline your optimization algorithm for PMF

I implemented the PMF paper. First, the U and V matrices are initialized following a Gaussian~(0,1) distribution. The R interaction matrix is normalized between 0 and 1 following $g(x) = (x-1)/4$. B is the binary mask computed from the original R. The following loss computation was implemented:

$$L = \frac{1}{2}(B \odot (R - sigmoid(UV^T))^2) - \frac{\lambda_U}{2}(||U||_{fro})^2 - \frac{\lambda_V}{2}(||V||_{fro})^2$$

Stochastic gradient descent is used to optimize U and V. For easy reproducibility of the table, I set a random seed. All the experiments in the table follow the same hyperparameter scheme, except for the number of latent factors: $\lambda_U = 0.3$, $\lambda_V = 0.3$, learning rate = 0.005, and sgd momentum = 0.9. The values were initially set to the ones provided in the paper, and further tuned considering the results in validation.

b. Describe your stopping criterion

I set a fixed number of iterations (500) and add an early stop mechanism based on validation RMSE after each iteration. If for 15 straight iterations the RMSE increases, the algorithm stops. Best set of validation U and V are kept if early stop happens.

I did not use GPUs for this assignment. Run time is on Mac m2 cpu.

Num of Latent Factors	RMSE	Runtime(sec)*	Num of Iterations
2	0.9501	316	182
5	0.9110	227	227
10	0.9085	589	418
20	0.9325	237	140
50	0.9739	233	113
7 - leaderboard	0.9061	434	272

3. Analysis of results (15)

Discuss the complete set of experimental results, comparing the algorithms to each other. Discuss your observations about the various algorithms, i.e., differences in how they

performed, what worked well and didn't, patterns/trends you observed across the set of experiments, etc. Try to explain why certain algorithms or approaches behaved the way they did.

The User-User/Item-Item algorithms behaved rather similarly, although the best score was achieved by User-User interaction with K= 10, no weighting, and dot product for similarity. Overall, dot product outperformed cosine similarity in most of the experiments. This may be due to cosine similarity “ignoring” the magnitude of the vectors, which seems to be important in this case. Using weighted averages instead of uniform averages did not yield significant differences. This leads me to hypothesize that ratings among neighbors may be similar (I did not confirm this empirically). Regarding time, all experiments run in < 40 seconds on my machine. As expected, it is faster to run with a smaller K. Obtaining the item-item similarity matrix is slightly faster than the user-user one, probably because number of users > number of items. For K=10, 100 the item-item experiments are faster than their user-user counterpart, but for K=500 they are slower. I don’t have a good explanation for this. If there is a tie in NN score, I followed the statement, using ascending id to solve ties. I tried with descending but results were the same.

Regarding standardization, I was expecting this method to outperform the previous ones, because user/item bias usually have a meaningful impact (i.e., users’ habits and magnitudes of rating are different). I picked movie-wise standardization because, overall, it achieved better results than user-user. Still, the best RMSE achieved by my standardization strategy (1.0216) is worse than the best vanilla KNN (1.0024 on user-user). This may be due to the fact that I impute the matrix before standardization by subtracting 3 from non-zero values (justified above). If rows are very sparse then the middle value will dominate the average. So, a better imputation strategy could improve this result. I also tried without imputing the matrix, and computing the mean/variance only on non-zero values, but did not help. It is also worth noting that the similarity matrix computation of a standardized matrix is more costly than the previous ones, probably due to losing some of the sparsity upon standardization. KNN search times are comparable to the other two approaches, which is expected in this implementation. Regarding cosine similarity versus dot product, in this case they are equivalent, since after standardization all rows have the same norm, the dot product is directly proportional to the cosine similarity. While the similarities will be different, they will have the same 'scale', so the neighbors will be the same.

As expected, the PMF-based methods achieved the best results, which I believe is due to these models ability of better capturing patterns and relationships in the underlying data through the learning of dense representations for users and items. For larger number of factors (~50) the results showed some signs of underfitting. This is because all entries in the table were run with the same hyperparameters, except the number of latent factors. For larger number of latent factors, the model has more complexity, i.e., more parameters. Hence, the more complex models should benefit e.g. from a lower learning rate, but these experiments were not conducted because two approaches achieved results < 0.93 RMSE under this scenario. So, I performed hyperparameter tuning around those approaches, achieving the submitted leaderboard score of 0.9061 RMSE using 7 latent factors. Hyperparameter tuning was very important, as I was able to drop RMSE that, on initial tests, were on the 0.93XX range, to 0.90YY just by adjusting hyperparameters. Regarding time, this method is the slowest one to experiment with as it involves training. But once U and V are computed, getting the predictions is almost automatic. As such, I consider this method to bring the most advantages if implemented in a production system – has the best results; training can be done periodically and offline; and run-time for users is very fast.

4. The software implementation (5)

Add detailed descriptions about software implementation & data preprocessing, including:

1. A description of what you did to preprocess the dataset to make your implementations easier or more efficient.

I did not change anything in the data. I used the provided function to load the matrices from the train.csv file. I chose data structures and used matrix operations appropriately to deal efficiently with each type of exercise. I did not feel the need to preprocess the data as my KNN experiments were already running in under 40s each.

2. A description of major data structures (if any); any programming tools or libraries that you used;

Libraries:

- NumPy: matrix operations
- SciPy: sparse matrix format
- Scikitlearn: metric implementations

- PyTorch: optimization

Data structures:

Initial representation of interaction matrix is csr_matrix if I'm computing user X user similarity matrix; csc_matrix if I'm computing item X item similarity matrix. It is faster to compute the similarity matrices in a single multiplication (S = MM.T or S=M.TM), rather than iterate per query vector. I also compute each type of similarity matrix only once and reuse it for different neighbor numbers and weighting functions. It is also faster to convert the similarity matrix to dense format rather than keep it sparse after multiplication.

3. Strengths and weaknesses of your design, and any problems that your system encountered.

The code is clean and easy to extend. It implements all the needed functionalities for the assignment. The code runs in acceptable time for the provided data. But further optimizations can be done which can help extend to larger matrices. For instance, parallelization can be used during matrix standardization since rows are independent. The slowest experiments are the PMF ones, which can be further optimized by leveraging GPU support, and trying the sparse tensors provided by pytorch to represent the R matrix and the mask.

As the major weakness of the design, I underline that all methods output float ratings (e.g., 4.2). I believe that an optimal method for this dataset should return integer values only, since those are the only ones available for Netflix. However, rounding the float values with python's rounding function resulted in much a lower RMSE in all experiments. This leads me to believe that there is an optimal rounding strategy, which I leave as future work.

Finally, I note that all the work in this report can be reproduced by running the script I uploaded. It will print validation RMSE and wall clock times for all methods.