Your Name: Yan Zhao

Your Andrew ID: yanzhao2

Homework 4

0. Statement of Assurance

a) I did not receive help of any kind from anyone in developing your software for this assignment.
b) I did not give help of any kind to anyone in developing their software for this assignment.
c) I am the author of every line of source code submitted for this assignment.

Except: I refer to some open source code (matlab) given by Ruslan Salakhutdinov, author of PMF, to finish my SGD method. I only read the idea of SGD, but write every line of my code in python myself.

d) I am the author of every word of your report.

1. Corpus Exploration (8%)

1.1 Basic statistics (4%)

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	3.3806

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	3.1507

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 (4%)

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

2. Basic Rating Algorithms (40%)

In all experiments, runtime measures by prediction on development and test data together.

Runtime is reported in seconds.

2.1 User-user similarity

Rating Method	**Similarity Metric**	**K**	**RMSE**	**Runtime(sec)**
Mean	**Dot product**	**10**	**1.0026**	**74**
Mean	Dot product	100	1.0067	284
Mean	Dot product	500	1.0430	1227
Mean	Cosine	10	1.0631	69
Mean	Cosine	100	1.0620	279
Mean	Cosine	500	1.0754	1221
Weighted	Cosine	10	1.0683	68
Weighted	Cosine	100	1.0622	297
Weighted	Cosine	500	1.0741	1241

2.2 Movie-movie similarity

Rating Method	Similarity Metric	K	RMSE	Runtime(sec)
Mean	Dot product	10	1.0207	80
Mean	Dot product	100	1.0468	302
Mean	Dot product	500	1.1109	1202
Mean	**Cosine**	**10**	**1.0173**	**25**
Mean	Cosine	100	1.0639	234
Mean	Cosine	500	1.1183	1153
Weighted	Cosine	10	1.0211	31
Weighted	Cosine	100	1.0572	231
Weighted	Cosine	500	1.1023	1175

In experiment 1 and 2, I use two metrics calculating cosine similarity: dot product and cosine; and I use two methods to combine ratings: mean average and weighted mean average, as instructed.

When combing ratings, if the movie has not been rated by the user, I make it 3 instead of using average rating of the given user or movie. My reason is simple, if I use mean rating of given user or movie, I will actually introduce user/movie bias into my model, and thus my prediction will depend on training data. With a moderate rating 3, I actually give a query independent prediction, which can give a more stable and meaningful rating considering robustness of training data. I'm not saying user/movie bias is not important, actually in the next experiment, I will consider them to find KNN and predict, as I will state below. So, in these two experiments, I didn't use bias.

2.3 Movie-rating/user-rating normalization

Rating Method	Similarity Metric	K	RMSE	Runtime(sec)	Bias
Mean	**Dot product**	**10**	**1.0471**	**68**	**user**
Mean	Dot product	100	1.0498	295	user
Mean	Dot product	500	1.0764	1355	user
Mean	Cosine	10	1.0897	96	movie
Mean	Cosine	100	1.0837	319	movie
Mean	Cosine	500	1.0929	1186	movie
Weighted	Cosine	10	1.0705	108	movie
Weighted	Cosine	100	1.0812	340	movie
Weighted	Cosine	500	1.0912	1279	movie

Based on experiments 1 and 2, I find that memory-based CF using user-user similarity has better performance, so I decide to normalize rating bias based on memory-based CF.

In this experiment, I consider 2 biases: user and movie, I try to eliminate each of them and observe performance on development data. I use below formula to normalize weight. For user bias, μ is average rating in training data over same user, σ is standard deviation of rating in training data over same user. Similarly, for movie, μ is average rating in training data over same movie, σ is standard deviation of rating in training data over same movie.

$$\text{rate}_{\text{norm}} = \frac{rate - \mu}{\sigma}$$

The reason I normalize in this way is that it can keep the sparse property of rating matrix, to speed up my program. Also, it's meaningless to standardize over entire row for user or entire column for movie, cause most of the ratings are just moderate scores 3 (after imputation). If we normalize like that, the average rating will always be close to 3 and standard deviation will always be close to 0.

In this experiments, I fill in the table with the better bias reduction method. As shown in the table above, when using dot-product, user bias reduction will get a lower RMSE, while when user cosine, movie bias reduction performs better. Also, if we compared with memory-based CF without bias reduction, it can be found that with bias reduction system will get higher RMSE. These observations will be analyzed in the following session.

2.4 Matrix Factorization

a. Briefly outline optimization algorithm for PMF

I implement 2 optimization algorithm, GD and SGD, trying to compare them and get best performance. Gradient is derived by error function given in the paper.

$$\text{E} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{M} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda}{2}\sum_{i=1}^{N} ||U_i||^2 + \frac{\lambda}{2}\sum_{j=1}^{M} ||V_j||^2$$

In this experiment, N is number of users, M is number of movies, I_{ij} indicates whether user i rates movie j, R_{ij} is the rate given by user i to movie j, which we can get from training data. For the regulation items, I set latent user matrix U and latent movie matrix V with equal $\boldsymbol{\lambda}$ = 0.0001. And I **initialize U and V** with random values between 0 to 0.05.

GD: In gradient descent mode, I derive the gradient of error function and use it directly w.r.t the whole data set.

$$\mathrm{dV} = -\mathrm{U} * (\mathrm{I} .* (\mathrm{R} - U^T\mathrm{V})) + \lambda * \mathrm{V}$$

$$\mathrm{dU} = -\mathrm{V} * (I .* \ (\mathrm{R} - U^T\mathrm{V}))^T + \lambda * \mathrm{U}$$

In each training iteration, I use whole rating matrix R to update U and V separately. I set very small **learning rate** equals to 0.0001 to avoid skipping global minimum. And to make convergence faster, I use **momentum** equals to 0.4 to accumulate previous gradients.

SGD: In stochastic gradient descent mode, I read training data by batches, with number of batches equals to 200. Cause we have more than 80000 training data in total, each batch will contain nearly 500 training examples.

$$\mathrm{dV} = -U_{\mathrm{batch}} * \left(R_{batch} - {U_{batch}}^T\mathrm{V}_{\mathrm{batch}}\right) + \lambda * V_{\mathrm{batch}}$$

$$\mathrm{dU} = -V_{\mathrm{batch}} * \left(R_{batch} - {U_{batch}}^T\mathrm{V}_{\mathrm{batch}}\right) + \lambda * U_{\mathrm{batch}}$$

In each training iteration, I read training data by batches to get rating R, and update U and V separately. I set a higher **learning rate** equals to 0.001, because SGD can have training errors decreasing more stable than GD. And to make convergence faster, I use **momentum** equals to 0.5 to accumulate previous gradients.

b. Describe stopping criterion

The first stopping criterion is **number of iterations**. Because training rating matrix R is a very large matrix, it takes a lot of time to run gradients. Also considering overfitting issues, I limit the number of iterations for both optimization algorithms. For GD, I make it **1000 iterations**, and for SGD, I make it **500 iterations**.

The second stopping criterion is **training MSE**, it's sum of squared error of training data at each iteration. After a number of experiments over development data with latent factor D equals to 50, I find that with training **MSE equals to 0.5655**, the model can have best performance. Higher SSE will cause underfitting and lower SSE may cause overfitting. So whenever training SSE gets below 0.5655, I will break training iterations.

As shown in the table, SGD can converge faster with less iterations than GD. And although they have similar performance, GD performs better on development data with same training MSE. So for latent factor 2, 5, 10, 20, I run experiments with GD. These observations will be analyzed in the following session.

Num of Latent Factors	RMSE	Runtime (sec)	Num of Iterations	Optimize Algorithm	Training Error
2	0.9456	5283	1000	GD	0.8089
5	0.9277	5098	1000	GD	0.7221
10	0.9284	5206	1000	GD	0.6455
20	0.9436	5126	916	GD	0.5655
50	0.9949	4411	100	SGD	0.9194
50	0.9549	9527	200	SGD	0.7128
50	0.9387	14532	300	SGD	0.6483
50	1.0134	23520	500	SGD	0.4189
50	0.9679	1547	300	GD	0.8476
50	0.9417	2273	400	GD	0.7523
50	0.9272	2791	500	GD	0.6544
50	**0.9252**	**3416**	**600**	**GD**	**0.5655**
50	0.9358	3935	700	GD	0.4997
50	0.9835	5739	1000	GD	0.3919
50	1.1150	19501	2000	GD	0.2954

3. Analysis of results (20%)

3.1 Algorithm Analysis

Memory-based vs. Model-based

From the experiments, we can find that memory-based CF approach has a better performance over model-based approach on average. In memory-based approach, we use user-user similarity to find nearest neighbors, while in model-based approach, we use movie-movie similarity. I think this is because in this training data, user similarity is a factor more important than movie similarity. Thus, if two users have more similarity, that they watch similar movie and give similar ratings, they are possible to give similar rating to another unwatched movie. However, if two movies have more similarity, that they are watched and rated with similar scores, it is less possible that they will be given same rating with a specific user, considering user bias. Also, according to corpus exploration, we have unique number of users almost 2 times of unique movies in training data, so user may be a more general and robust feature than movie, that it is easier for us to find users with similar bias than to find movies. As a result, user similarity can better indicate ratings.

Another interesting point is if we compare two approaches using cosine similarity, when K is very small like 10, model-based approach performs a little bit better. I think this is because there are less number of unique movies, so when K is smaller, movie similarity is important feature, while when K gets larger, this feature will lose information and perform close to random guess.

As for runtime, we can find model-based approach has less runtime on average. This is easy to understand, because model-based approach calculate similarity between movies offline, while memory-based calculate KNN for each test example on the fly. If we enlarge number of examples in test data, this advantage will be more obvious; if we reduce number of examples in test data, memory-based approach may run faster instead.

Memory-based vs. Bias-reduction

From the experiments, we can find memory-based approach has a better performance than bias-reduction based on memory-based. The reason is when reduce bias, we actually lose information of user and movies. For example, if two users have similarity in rating movies, they are highly likely to have similar bias, and they tend to give same rating to a given movie. For user bias, if both users tend to give relatively higher ratings, they will be possible to give relatively high rating to another movie. For movie bias, it is the same, like if both users love fantasy movies, they are likely to give higher rating for fantasy movies and lower rating for the others. If we reduce these user or movie bias, we will lose this information in calculating nearest neighbors of a given user, and the results will be more general instead of user specific, which in our case, may not be a good idea.

Also, we can find reducing movie bias seems to have better performance than reducing user bias. This results correlates with my previous analysis, that if two users have more similarity, that they watch similar movie and give similar ratings, they are possible to give similar rating to another unwatched movie. When considering user similarity, if we reduce this bias information, we will lose more information and give less accurate prediction. But if we reduce movie bias when calculating user similarity, we can give user bias more weights in giving predictions, and thus get better performance.

Dimension-reduced

From the experiments, we can find dimension-reduced approach PMF has much better performance than the others. According to corpus exploration, only 1.4% ratings have been given in the training data, while we need to predict the other. It is kind of “cold start” issue in CF, and using normal approaches will definitely have a bad prediction. When using PMF, we use latent user feature matrix and latent movie feature matrix to represent rating matrix, perform gradient descent to get squared error on training data as low as possible and use it to give predictions, although it takes more time to train the model, the model can actually be able to generalize considerably better for users with very few ratings.

3.2 Number of Nearest Neighbors K

In all memory-based, model-based and bias reduction approaches, K = 10 always give lowest RMSE. This is easy to understand, when K is small, system has low bias, as K grows, the method becomes less flexible and produces a decision boundary close to linear. So it is a bias-variance trade-off issue. In our experiments, getting a large set of similar users for prediction will not help us generalize information given by similar users, and this is true for movies also.

3.3 Similarity metric

In all memory-based, model-based and bias reduction approaches, using mean average method and given same K, dot-product has a better performance in general. The only difference between cosine similarity and dot product is that cosine similarity has a normalization of vectors. In our experiments, training matrix is quite sparse, so normalization won’t give help for the prediction. On the other hand, as I analyze above, user bias will give helpful information in our prediction, like 2 users are both likely to give relatively higher ratings, and with dot-product, we can better catch this information in our prediction. Thus, dot-product has a better performance.

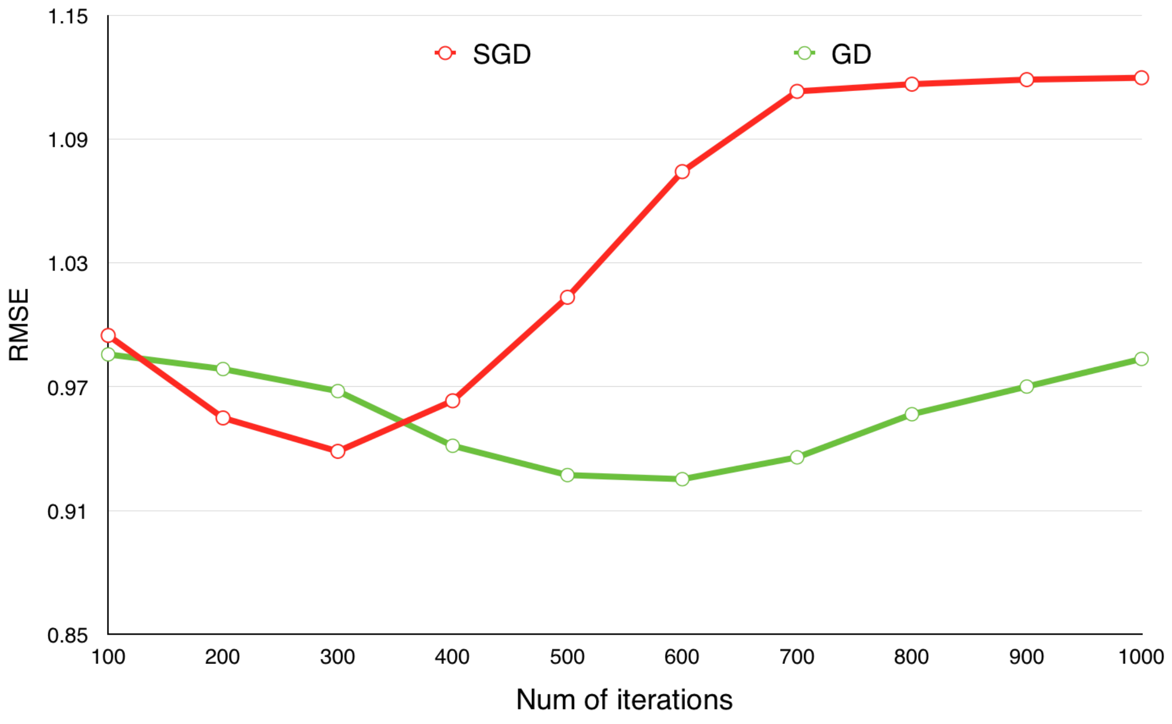
3.4 Weighting method

In all memory-based, model-based and bias reduction approaches, given same K, weighted mean has a better performance than mean average in general. The intuition is obvious, users with higher similarity between given query should have higher weights in helping prediction. And the same for movies. This observation also proves that KNN method is a good way to help predict ratings cause with higher similarity, user/movie can give a higher possibly reference for given query.

3.5 Optimization algorithm for PMF

Comparing two optimization algorithm GD and SGD, from picture shown below, we can find they have similar performance, that with the increase of iterations, RMSE on development data goes down first, and then increase, suffering from overfitting issues. However, with incremental mode using SGD, system can converge faster, with batch mode using GD, system can get more stable with iterations and get better performance. Also, in order to get similar performance, system runs faster using GD. This observation shows that SGD often converges much faster compared to GD but the error function is not as well minimized as in the case of GD. And the close approximation in SGD for the parameter values are enough because they reach the optimal values and keep oscillating there.

Since in this experiments, the data set we use is not quite large, if training data is so large that we could not even load into memory, then GD will not be feasible and the advantage of SGD will be obvious.



4. The software implementation (15%)

4.1 Preprocess dataset to make implementations easier or efficient

For memory-based, model-based and bias-reduction, I read training data into a sparse matrix, stored as csr_matrix by scipy. I use option 2 to do imputation, by minus 3 to each given rates, to maintain sparsity of rating matrix. In this way, when calculating similarity, all I do is normalize or dot-product on sparse matrix, making my system run faster.

For PMF, I do the same thing, read rating matrix R as csr_matrix. However, when calculating error matrix $(\mathrm{R}-U^T\mathrm{V})$, it will become a dense matrix cause U and V are dense. To solve this, I introduce a matrix I which has same size of R, and if user i rates movie j, I will set corresponded element in I to be 1, otherwise 0. Then, when calculating error matrix, I do an element wise multiplication of I and $(\mathrm{R}-U^T\mathrm{V})$, to make it a sparse matrix again. In this way, I can speed up for the following process of calculating gradients.

4.2 Data structures, programming tools or libraries

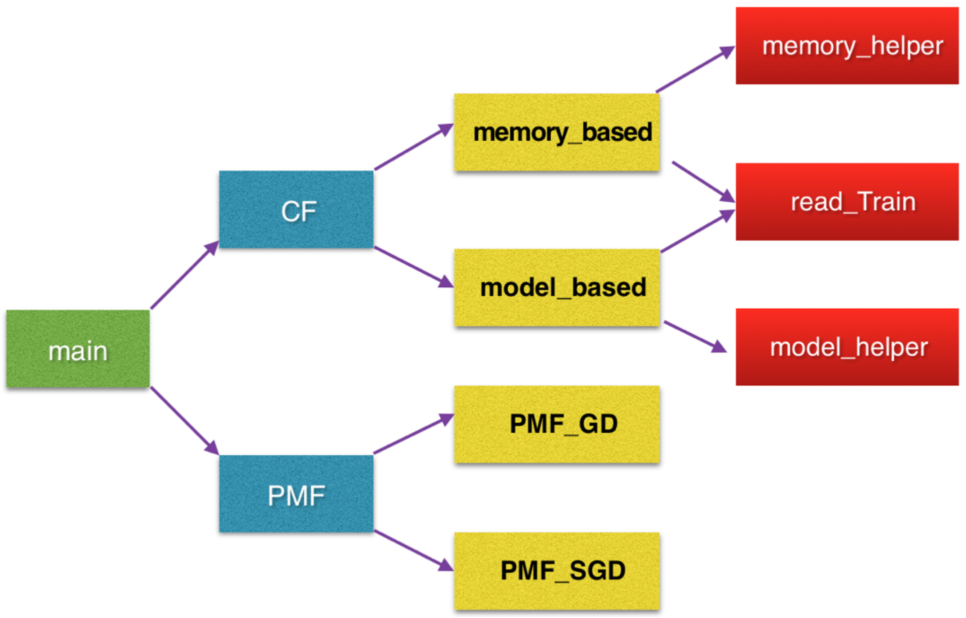
In memory-based approach, every time I find KNN of given query, I will store the result in a dictionary hashed by user ID. In this way, next time when I come across query with same user ID, I don't need to calculate KNN again, instead, I can retrieve from dictionary directly. In this way, I can reduce time when making predictions.

In model-based approach, I use sparse matrix dot product to get model A, movie-movie similarity pair.

In bias-reduction approach, I calculate average and standard deviation of user/movie vector only based on given ratings in training data, and standardize vector by only modifying non-empty elements. In this way, I can keep sparsity of standardized rating matrix and speed up following steps.

I use scipy to store training data into csr_matrix, and do matrix operation using property of sparse matrix to speed up. I use sklearn to normalize the vector for computing cosine similarity, to avoid nan value when the norm of a vector is 0.

Structure of my system is as follows.



4.3 Strengths and weaknesses

For memory-based, model-based and bias-reduction CF, my program can run very fast. This is because I use sparse matrix operation at every step, and I store KNN at query time.

The weakness of my program lies on PMF by SGD. Although SGD can converge faster than GD, but it takes much longer to run SGD at each iteration. I think the reason is when using batches, I have to expand user/movie feature matrix by latent factor D, and U_batch, V_batch will be dense matrix, so I do a lot more computation in calculating gradients. In the next step, I will try to optimize my algorithm to make it run faster.

4.4 Run code

There is a "parameter.txt" in src folder, where you can choose algorithm (memory-based, model-based, PMF), set number of nearest neighbors (K), change similarity metric (cosine, dot-product), change weighting method (mean, weighted-mean), choose whether to use bias reduction (PCC) and number of latent factors in PMF (D). After setting parameters, to run program, just cd into src folder and type "$ python main.py".