Your Name: Yan Zhao

Your Andrew ID: yanzhao2

Homework 6

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

1. Training Set Construction (5 pts)

Construct the training set for LETOR as instructed and report the following statistics.

Statistics	
the total number of observed ratings in R	820367
the total number of training examples in T	3520432
the ratio of positive examples to negative examples in T	1/1
the number of training examples in T for user ID **1234**	196
the number of training examples in T for user ID **4321**	64

2. NDCG of Memory-based methods (10 pts)

2.1 User-user similarity (Do the same imputation first as in HW4)

Rating Method	**Similarity Metric**	**K**	**NDCG@20**	**Runtime(sec)***
Mean	Dot product	10	0.9381	66
Mean	**Dot product**	**100**	**0.9433**	**308**
Mean	Dot product	500	0.9421	1320
Mean	Cosine	10	0.9316	75
Mean	Cosine	100	0.9424	313
Mean	Cosine	500	0.9428	1334

*runtime should be reported in seconds.

2.2 Movie-movie similarity (Do the same imputation first as in HW4)

Rating Method	Similarity Metric	K	NDCG@20	Runtime(sec)
Mean	Dot product	10	0.9347	83
Mean	Dot product	100	0.9419	301
Mean	Dot product	500	0.9408	1267
Mean	Cosine	10	0.9381	28
Mean	**Cosine**	**100**	**0.9452**	**270**
Mean	Cosine	500	0.9420	1152

3. NDCG of PMF (6 pts)

Report the best results you can get on dev set from training PMF with different dimensions of latent factors.

Num of Latent Factors	NDCG@10	NDCG@20	NDCG@30	Runtime(sec)*
10	0.9469	0.9509	0.9513	5206
20	0.9453	0.9492	0.9496	5126
50	0.9466	0.9506	0.9510	3416
100	**0.9480**	**0.9519**	**0.9523**	**3258**

4. RankSVM (12 pts)

a. After the optimum w is learned by SVM on the training set T, given a testing user, how to generate the ranked list of unrated items and why? (Please be concise and clear; use formulas to express your idea when possible) (2 pts)

For a given test user, I collect all of its queried movies and compare each movie pairs to generate test data. Then I use trained model to do binary classification labeling each pair as 1 or -1.

When doing re-ranking, I do topological sort based on pairwise labels with movie *i* and j. When label is 1, it means movie *i* ranks higher than movie *j*, so I add *i* -> *j* to graph; when label is -1, similarly I will add *j* -> *i* to graph. Then I do topological sort for the directed graph to generate the rank of movies for given user.

b. Report the best results you can get on dev set from training RankSVM with features from PMF (10 pts)

Num of Latent Factors from PMF	NDCG@10	NDCG@20	NDCG@30	Runtime(sec)*
10	**0.9468**	**0.9508**	**0.9512**	**8**
20	0.9445	0.9485	0.9489	8
50	0.9458	0.9499	0.9503	14
100	0.9467	0.9507	0.9511	16

Command line used: ./train -s 3 -c 0.1 -e 1e-8 rank_train.data model
./predict rank_dev.data model rank_result

5. LR-LETOR (12 pts)

a. After the optimum w is learned by LR-LETOR on the training set T, given a testing user, how to generate the ranked list of unrated items and why? (Please be concise and clear; use formulas to express your idea when possible) (2 pts)

For each user/movie pair in test data, I generate feature vector based on U and V by $x_{ui} = [a_u .* b_i]$, then I calculate score of user/movie pair by $\text{score} = w^T . x_{ui}$, as my prediction for given user/movie pair.

b. Report the best results you can get on dev set from training LR-LETOR with features from PMF (10 pts)

Num of Latent Factors from PMF	NDCG@10	NDCG@20	NDCG@30	Runtime(sec)*
10	0.9459	0.9498	0.9502	0.8
20	0.9445	0.9485	0.9489	1.9
50	0.9459	0.9499	0.9503	3
100	**0.9468**	**0.9507**	**0.9511**	**19**

Parameters used in training: learning rate = 0.05
regularization factor = 0.01
momentum = 0.4
epsilon = 1e-15 (20 iterations when latent factor = 100)
batch size = 3500

*Runtime only includes training time of LR, without runtime of reading data.

6. Analysis of results (20 pts)

6.1 Algorithm Analysis

User-user vs. Movie-movie Similarity

From the experiments, we can find that user-user similarity and movie-movie similarity have quite similar performance, with NDCG@20 approximately 0.94.

Also, both methods can get best performance with nearest neighbors K equals 100. This results are similar with homework 4, 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 small set of similar users for prediction will underfit training examples, and getting a large set of similar users will not help us generalize information given by similar users, and this is true for movies also.

As for similarity metric, we can find that dot-product and cosine have quite similar performance. 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.

PMF

From the experiments, compared with different latent factors, we can find with latent factors equal to 100, performance is the best. Then, surprisingly, we get second best performance with latent factors equal to 10. I think the reason is during the PMF training process, I set a training error threshold, to make training break when training error is smaller than it. I decide the threshold by lots of experiments in homework 4. So technically, I can get same training error whatever latent factor is. From this point of view, with 10 latent factors, the model can be more flexible, less easily to get overfitting; with 100 latent factors, the model can have better representation on training data, which will contain more information. Thus, the performance with restrict to latent factors is like a curve, and can be good with reasonable settings like either 10 or 100.

Rank SVM vs. LR-LETOR

From the experiments, we can find Rank SVM and LR-LETOR have quite similar performance. This result is reasonable cause SVM and LR are both doing binary classification on the same training data. However, we can find Rank SVM has best performance with 10 features while LR has best performance with 100 features. Based on the fact that PMF performs good with either 10 or 100 latent factors, this result did not happen beyond my expectations. I think with custom set of learning rate, epsilon and reasonable iterations, LR can be better trained with more number of features; while for SVM using liblinear, it may be more easily to get overfitting.

Collaborative Ranking vs. Collaborative Filtering

Firstly, we can find that memory-based method has lowest NDCG among all the approaches. This is easy to understand, as in homework 4, memory-based method performed less than PMF approach. Also, this method is designed to do collaborative filtering, which give an inferred score for given user/movie pair. However, in the collaborative ranking task, we want to get top-k movie for given user and measure in NDCG, so memory-based method is not quite fit for this task.

Secondly, we can find that PMF actually outperforms two ranking approaches Rank SVM and LR-LETOR. I think the reason is dataset for this assignment is designed to do collaborative filtering, so using re-rank and evaluating with NDCG may not be appropriate for user/movie pairs in this dataset. Also, when I do LR, I found that the algorithm get to converge very fast with only a few iterations. I think this is because we have more than $3 * 10^6$ training instances while we only have no more than 100 features, so the number of instances and number of features are quite unbalanced. As a result, the model cannot be successfully trained due to overfitting issue. This is also one reason why I did not get improvements with Ranking approaches.

6.2 Predicting Method

For each user/movie pair in test data, I generate feature vector based on U and V by $x_{ui} = [a_u .* b_i]$, then I calculate score of user/movie pair by score $= w^T . x_{ui}$, as my prediction for given user/movie pair.

I do this because in the training process, we actually learn a relationship between different movie pairs for a given user, and since sigmoid function is monotonic increasing, in test data, we do not need to calculate relationship of each movie and re-rank again. Instead, we can directly calculate dot product of weight matrix and feature vector.

By doing this, I can reduce running time in prediction significantly. Moreover, when doing topological sort, there are some cases like cycle that will influence re-ranking process. By only calculating scores, I can exclude errors in topological sort and thus improve performance. Results shows I can get a little bit improvements on NDCG with this predicting method (e.g. NDCG@20 improves from 0.9503 to 0.9507 when using 100 latent factors).

7. The software implementation (15 pts)

7.1 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.

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.

For LR, cause we are doing binary classification, I modified loss function to make it $\mathrm{l}(\mathrm{w}) = \sum_{i=1}^{n}\{y_i ln\sigma(w^T x_i) + (1 - y_i)\ln\,(1 - \sigma(w^T x_i))\}$, in this way, weight matrix can be just a row vector and I can significantly reduce training time for LR. Also, I shuffle training data before gradient descent, so that I can converge faster in training process.

In addition, at each step, I save the result so that I can reuse them when doing experiments. In this way, I can speed up the following process of parameter tuning. For example, after I tuned parameter for PMF, I save the latent matrix, and when I tuned parameter for LR, I can just read latent matrix to construct training data matrix.

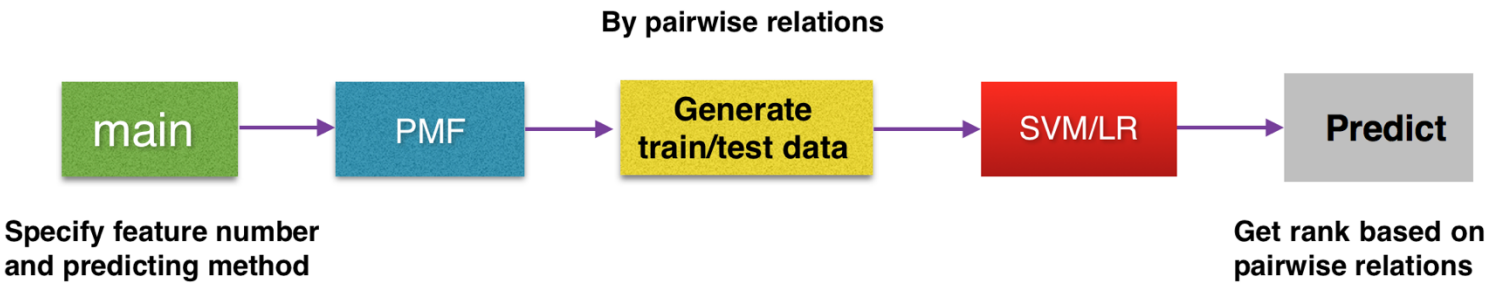
7.2 Data structures, programming tools or libraries

For CF, 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.

For LR, I use sklearn to shuffle training data to make my training process converge faster.

When doing re-ranking, I do topological sort based on pairwise labels as discussed above, to generate order of movies for a given user.

Work flow of my system is as follows.



7.3 Strengths and weaknesses

Strengths:

Based on design of previous assignments, I can easily modify them into the new pipeline and make predictions. I can train with LR and generate predictions very fast.

Weakness:

1. When generating training data T, in order to make file smaller, I only save feature value with 4 significant digits. I think this will influence the performance of training with LR and SVM.

2. In order to do experiments, I separate each step in the above work flow, and hard-coded input and output file paths. The extension ability for my code may not be very good. In the future, I may add file path as arguments to make my code more robust.

4.4 Run code

The only two parameters you need to set is number of features (-f) and LR predicting method (-m, 1: pairwise labeling and topological re-rank; 2: get test data score by using weight matrix directly).

To run code, cd to src folder and type command like $ python main.py –f 50 – m 1