Arda Cankat Bati

A53285400

CSE 285 Assignment 1 Report

Sorry for the quite long report, I just wanted to express everything I did. Mostly for future reference by myself, in case I forget. Thanks!

Task1 Read Prediction

For this task, I first included the data from the rating predictions task to my overall data, making 210K total datapoints. I shuffled the data, and split the train and validation sets as 200K train samples, 10K validation set. Then I doubled the size of each set by sampling books each user hasn't read. (so that in the end it was 400K train, 20K validation set size). I created feature mappings for each data point (user, book) pair as follows: 0-bias term, 1- maximum Jaccard book similarity, 2- average Jaccard book similarity 3- maximum Jaccard user similarity 4- average Jaccard user similarity 5- maximum cosine book similarity, 6- average cosine book similarity, 7- book in popular books set 1/0 (the set from HW3, with threshold ratio of 0.64).

Then, I put the train set through a logistic regressor to get the best feature weights. I used different C values between 10^-5 to 10^5, and used the validation set to decide on the best regularization parameter C (the one that gave the highest validation accuracy and BER). Mostly the best valued were around 0.001 – 0.05. Finally, for the actual test predictions, for each user, I sorted the predictions according to their confidence score (from the logistic regressor). For each user's read books, I predicted 1 for the first half of the books (with higher confidence scores) and 0 for the lower half. I did this because the test set was prepared in such a way. (10K initial samples, than 10K generated samples with the same users. So, each user has 2n books, n read books, n un-read books). This final prediction is what I uploaded to Kaggle.

To calculate the Jaccard similarities I used what we already implemented in class. The only difference I made is that while training, I excluded the Jaccard similarity of a pair if the two pairs were the same. I did this to not overfit the trainset. I didn't to this while testing, because it would be a disadvantage. To calculate the cosine similarities, I set each rating to 1 if it is greater than or equal to the current user's average, -1 otherwise. I tried working with Pearson similarities as well but didn't find them very successful. Therefore, I excluded it in the final model. I realized there are around 8K 0 scores, but you can't actually give 0 score on Goodreads, so they probably mean something else. I tried taking advantage of this, but couldn't find a good way to do so.

Finally, this is my idea on how I could improve: I think in my implementation I had to push C too low, which means higher regularization. I think this was caused by my train and validation sets not being big enough. I could have generated more negative test points to make them bigger. To keep the resulting model balanced, I could increase the weights of positive examples (as there would be fewer of them compared to the negatives). I believe this would give higher accuracy for me, probably it is what people above me in the leaderboard did.

Task 2 Rating Prediction

For this task I used a latent factor model as outlined in lecture 8 slides, with the following formula:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

Here, alpha is an offset term, beta_u represents user u's rating tendency, while beta_i gives a book i's rating tendency, gamma_u is user u's latent factor vector, gamma_i is a books latent factor vector, both two are k units long. From the data given to us, I used 195K as train and 5K as validation points to calculate the MSE on. I sed the loss function as the one below as outlined in lecture notes 8:

$$\arg\min_{\alpha,\beta,\gamma} \underbrace{\sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2}_{\text{error}} + \underbrace{\lambda\left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2\right]}_{\text{regularizer}}$$

The lambda value I used was lambda = 2.8. To decide in this I used my best testing results on HW3. I chose the size of the latent factor vectors as k = 4 at the end, although I believe higher k's are possible to use. This is just the value with which I was able to converge with the highest accuracy. (I tried integer k's between 1 to 10, higher values I believe are too much for our sparse dataset). For the updates, I had a descent loop. In each iteration all values alpha, beta_u, beta_i, gamma_u, gamma_i got updated **once,** in that order. I used coordinate descent as outlined in lecture 8 for alpha, beta_u and beta_i updates. I included gamma_i and gamma_u in the coordinate update equation, as shown below.

$$\alpha^{(t)} = \frac{\sum_{u,i\in\text{train}}(R_{u,i}-(\beta_u+\beta_i))}{N_{\text{train}}}$$

$$\beta_u^{(t+1)} = \frac{\sum_{i\in I_u} R_{u,i}-(\alpha+\beta_i)}{\lambda+|I_u|} + \text{gamma\_i * gamma\_u}$$

$$\beta_i^{(t+2)} = \frac{\sum_{u\in U_i} R_{u,i}-(\alpha+\beta_u)}{\lambda+|U_i|}$$

For finding the gamma_i and gamma_u values, I used gradient descent (simple solution as outlined in lecture 8 slides). The update equation derived in class is as follows (it is for gamma_u, but gamma_i is similar):

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2+\lambda\left[\sum_u \beta_u^2+\sum_i \beta_i^2+\sum_i \|\gamma_i\|_2^2+\sum_u \|\gamma_u\|_2^2\right]$$

$\gamma_u\cdot\gamma_i = \sum_k \gamma_{uk}\gamma_{ik}$

$$\frac{\partial obj}{\partial \gamma_{uk}} = \sum_{i\in I_u} 2\gamma_{ik}\left(f(u,i)-R_{ui}\right) + 2\lambda\gamma_{uk}$$

I didn't use any gradient or coordinate descent libraries, since I wanted to do the implementation on my own. For the starting values, I set alpha = 0, beta_u = beta_i = average_score. For gamma_u and gamma_i, I set each element of the vectors to a random value between –0.25 to +0.25. To keep track of how the algorithm is converging and possibly overfitting, I calculated the current MSE on the validation set at each iteration. The MSE starts decreasing as the gradient descent begins, but after a certain point it overfits the trainset and validation MSE starts increasing. At that stage I stopped the model. I also had an alternative model trained which just had alpha_simple, beta_u_simple, beta_i_simple, derived with coordinate descent, no latent factor vectors. I always tried to get better MSE than this simple model with my latent factor model (not so easy).

Finally, after finding the right alpha, beta_u, beta_i, gamma_u, gamma_i values, I predicted the ratings with the **f(u, i)** function I outlined in the beginning of this section. As the final step, I realized some predictions go out of the boundaries for ratings which is 0 <= rating <= 5. Since this is not possible for Goodreads, in such cases I reverted back to the simple model's prediction using just: alpha_simple, beta_u_simple, beta_i_simple. If that value was not in the correct range as well, (which almost never happened), I predicted the closest value to it, either 0 for f(u, i) < 0, or 5 for predictions f(u, i) > 5.

Overall, I found this task to be much more straightforward than Task 1.