Faber Ventures Challenge
Proposed solution

Kishan Rama

May 12, 2017

Main Idea

Before I started building this simple recommendation engine I did some research on recommendation engines. I found that there are typically two types of algorithms - Content Based and Collaborative Filtering. I used the latter approach where the basic idea is to find look alike customers (based on similarity) and offer products which first customer's look alike has chosen in past. Giving a concrete example: if a person A likes item 1,2,3 and B likes 2,3,4 then they have similar interests and A should like item 4 and B should like item 1.

Algorithm

In order to implement the idea stated in the previous section I inspired in Bioinformatics where a very well-known algorithm, The Needleman-Wunsch Algorithm, is used to align biological sequences. In our specific problem we are going to align sequences of movies defined for each user where the similarity of each pair of user will be given by the score of the alignment.

Data preprocessing

From the provided dataset I will just use the entries correspondent to *productId*, *userId*, and *score*. This preprocessing step is necessary for building sequences for each *userId* of the type:

$$[score_1.productId_1, score_2.productId_2, ..., score_N.productId_N] \tag{1}$$

where $productId_1, productId_2, ..., productId_N$ are N different movies reviewed by some *userId* and $score_1, score_2, ..., score_N$ are the corresponding ratings given by that user.

Sequence Alignment

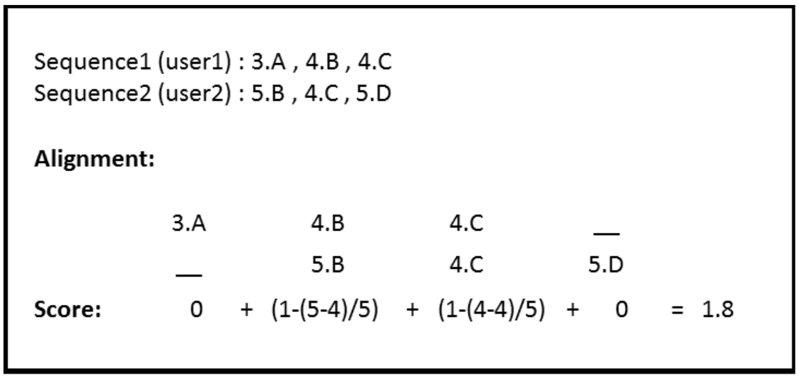
With the previous step we built sequences for each user. Now given a *userId* it is possible to align the sequence of this user with all the other users. This is done by using the Needleman-Wunsch Algorithm with a small modification in the way the score matrix D is built:

$$D(i,j) = max \begin{cases} D(i-1,j-1) + s(x_i,x_j) - \boldsymbol{f(r_{x_i}, r_{x_j})} \\ D(i-1,j) + g \\ D(i,j-1) + g \end{cases} \tag{2}$$

where $s(x_i, x_j)$ is the similarity between two pairs of movie, it is 1 if they are equal or 0 otherwise. The function $f(r_{x_i}, r_{x_j})$ is a rating penalty function $f(r_{x_i}, r_{x_j}) = |\frac{(r_{x_i} - r_{x_j})}{5}|$ where r_{x_i} and r_{x_j} are ratings given to movies x_i and x_j. The gap penalty g is defined as 0 in this problem, in this way we introduce gaps whenever a sequence has movies that some other sequence does not have.

For a better understanding of what we are trying to do here an example is given in Figure 1. In this figure we have the result of the alignment between two users where the letters A,B,C,D represent movies. It is possible to observe that movies that were reviewed by both users appear aligned while the other movies align with gaps.

If all the movies reviewed by both users had the same ratings it is easy to conclude that the score would be equal to the number of movies aligned.



```
Sequence1 (user1) : 3.A , 4.B , 4.C
Sequence2 (user2) : 5.B , 4.C , 5.D

Alignment:

              3.A          4.B          4.C          __
              __           5.B          4.C          5.D

Score:         0    +  (1-(5-4)/5)  +  (1-(4-4)/5)  +   0    =  1.8
```

Figure 1: Alignment of two sequences

What movies to recommend?

By performing alignment we have an idea of how similar are two users. What movies should we recommend given a *userId*? Well when performing alignment with other users we are trying to find users that had the same opinion in the movies that both saw. In the example given in Figure 1 user1 and user2 do not have exactly the same opinion. However if user2 rated movie B with 4 then these two users would have the same opinions and the score would be 2. Hence we are looking for these users that can be defined by having an score *opinion* given by:

$$opinion = \frac{\text{score of the alignment}}{\text{number of aligned movies}} \tag{3}$$

It is easy to see that if opinion is equal to 1 then we found an user that has the same opinion as our *userId* that we are trying to recommend movies to. In this case we should recommend movies that are aligned with gaps of *userId*, i.e. movies that *userId* did not reviewed yet. More specifically we should only recommend good movies, i.e movies that were rated 5 stars.

Another approach that was followed is to start aligning the sequence of *userId* with users with longer sequences. The intuition is that users with longer sequences are usually more knowledgeable about movies and if an alignment is performed with *opinion* = 1 then this user can give good recommendations.

Some observations

- I only tested my implementation with about 1 million reviews. It takes considerable amount of time if I increase this number so I did not read the entire file.
- Aligning longer sequences (more than 50 reviews) also takes considerable amount of time.
- In order to not recommend a huge number of movies, an early termination condition was defined to just recommend around 40 movies.
- In this implementation only the rating was used but we could easily add the helpfullness parameter to recommend movies.
- The code presented still has improvements to be made and probably there are bugs that I did not fix.