Isai Mercado Oliveros
April 3, 2016
Lab 11 - Recommender System

Exploratory data analysis

Since I chose the matrix factorization algorithm for my recommender system, I did not explore the data too much because in order to implement matrix factorization, we only need to know userID and movieID to construct the observed matrix.

Description of technical approach

As mentioned before, I implemented a matrix factorization algorithm for my movies recommender system, and I used gradient descent to find the best solution. Matrix factorization is an algorithm that constructs a matrix based on two features. In this movie recommender system, the features where userID, and movieID. The entries in the matrix contain the rating that user X gave to movie Y. However, in this matrix there are several unknown entries, since users watch a movie and do not rate the movie, or user have not watch a movie so there is no rating for that user-mavie pair. The goal of matrix factorization is two find two smaller matrices that by their dot product can construct the observed ratings, and also can predict values of the unknown entries in the original observed matrix.

The way I implemented the algorithm is the following. First I read the csv file that contained the observed ratings. Then I made a matrix with userID as rows and movieID as columns that I called the observed matrix. Then I created two dictionaries that mapped the userID, and the movieID to their corresponding row index and column index. These dictionaries are used to go from the prediction-set's usersIDs and moviesIDS to the indexes of the observed matrix. Second, I initialized 2 random smaller matrices. The users matrix was size length(users) times N, and the movies matrix was size length(movies) times N where N is and arbitrary number that represents features that they have in common in order to predict unobserved values. Third, for each known entry(unknown entries are skipped) in the observed matrix, I compared the observed rating, to a prediction computed by dot multiplying a row from the users matrix to a column from the movies transverse matrix. Then, by subtracting the target rating by the predicted rating, we get a local error. This error is then used to calculate a partial derivative of the SSE function, which then gives use direction to minimize the error between the observed rating and the predicted rating. Then, the row of the users matrix and the column of the movies transverse matrix are updated by following the direction of the derivative. This process is repeated until the rows of the user matrix and columns of the transverse movies matrix can predict the observed values correctly. Then these factorization smaller matrices can predict unobserved data by dot multiplying the row with the userID in the users matrix and the column with the movieID in the movies matrix

Analysis of performance of method

I ran an instance of my program in the BYU super computer with all the users and all the movies in the training data set, but the matrix is so big that my python program did not even finish after 30 hours. While waiting for that instance to finish, I started another instance with a subset of 10,000 random entries from the training data set, but it did not finish even after 20 hours. At the same time, I started another instance with a subset of 5000 random entries from the training data set which finished after 13 hours. However, the factorization matrices cannot predict all the instances of the predictions data set because there might be missing users or missing movies since the observed matrix was made out of a random subset of the training data set. So, I just predicted a subset of the predictions data set.

I think that matrix factorization is very useful because it lets you predict unobserved data.

The training phase is very slow because gradient descent is ran on every entry of the observed matrix which is very big, and on top of that, you have to ran the entire matrix several times in order to converge to an optimum solution. Compared to other methods like deep neural networks. It might not be so slow after all since deep neural networks also need a long time to train. In addition, the spacial complexity is n^2 since the observed matrix is users times movies.

For this lab, I wanted to understand matrix factorization as a recommender system, so once I created the algorithm, I did not try to change it because I wanted it to be pure.

Finally, My model did not overfit because it stopped very early since the stopping criteria of my algorithm compared the overall error of the observed matrix and the predictions made by the dot product of the users matrix and the movies matrix, but the error threshold was not very tight to help it finish early, so the factoring matrices are not very optimized. Thus, the algorithm did not overfit.