EE239AS Project 1

Collaborative Filtering

Winter 2015

Introduction: Recommendation systems and "suggestions" have become an essential part of a lot of web applications. Examples of these applications include recommended items in online stores, pages or people in social networking sites, movies or songs in streaming websites, etc. What has enabled these systems to expand widely to so many different applications is the rapidly growing collective user data and the use of Collaborative Filtering. "Collaborative Filtering" refers to methods of predicting a user's opinion on an entity using other users' opinion. These methods are based on the notion that there exist other users with similar behavior as the target user and finding them and observing their actions will reveal information that we could use to predict the target user's behavior.

We can view this as a matrix problem by putting all users on rows and all items on columns of the matrix. Then the entry (i, j) of the matrix will be the rating user i has given to item j. The problem will now become estimating the empty entries of the matrix to predict what items a user will most probably like other than the ones they have rated.

In this project we are going to use a popular method for collaborative filtering called "Alternating Least Squares" to make a recommendation system on the MovieLens[1] dataset that contains movies and user ratings.

1) Download the dataset with 100k ratings and create a matrix named R containing user ratings with users on rows and movies on columns. Note that a lot of the values in the matrix will be missing and only a small fraction of the matrix entries will contain known data points. We want to run a matrix factorization job to get matrices U and V such that $R_{m\times n} \approx U_{m\times k}V_{k\times n}$ in known data points. For this purpose, we calculate matrices U and V such that the squared error is minimized:

$$\min \sum_{known\ i,j} \left(r_{ij} - (UV)_{ij}\right)^2$$

This can be implemented by putting 0 s where the data is missing and creating a weight matrix to calculate the squared error. Assume that the weight matrix $W_{m\times n}$ contains 1 in entries where we have known data points and 0 in entries where the data is missing. Then if R contains 0 in missing entries, we can formulate the above problem as:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}\left(r_{ij} - (UV)_{ij}\right)^2$$

[1] http://grouplens.org/datasets/movielens/

You can run least squares factorization using the `wnmfrule` function in the Matrix Factorization Toolbox[2] in Matlab. Choose k equal to 10, 50, 100. What is the total least squared error in each case?

2) Try changing the weight matrix and use rating values as weights, instead of 1, for the known data points. Turn R into a 0-1 matrix where $r_{ij} = 1$ for known ratings and $r_{ij} = 0$ for unknown entries. Run matrix factorization again with the same values of k. Is there a change in the total squared error?

3) In order to test the recommendation system we have designed, we use the classic machine learning approach of splitting the data. That is, we take 90% of the known ratings for training and intentionally make the other 10% unknown for testing. After training the model we compare the predicted value of the 10% testing data with their actual values. If we split the data into 10 equally-sized parts and test 10 times, each time testing for one of these 10 parts while training on the other 9 parts, we would achieve "10-fold Cross-validation".

In our model this can be achieved by putting zero weight on the data we want to make unknown. Note that you shouldn't remove rows or columns of the matrix entirely because that would jeopardize the idea behind this method. A way of avoiding this problem is taking random points for testing, so that the data points we remove are dispersed throughout the matrix. However, we want the 10% testing data in each test to not overlap with the other 9 tests.

Assume that we assign an index 1 to N to the N known data points. Create a random ordering of the numbers 1 to N and split this list of numbers into 10 parts. Now each time take one of these 10 parts for testing and train on the rest.

Train your system as in part 1 and 2 and find the average absolute error over testing data (prediction error $\left|R_{predicted} - R_{actual}\right|$). What is the average error? (Average among 10 the tests for each system, and for each test among all entries)

What are the highest and lowest values of average error among the 10 tests? (Average over the testing entries in each test and find which of the 10 tests has highest or lowest average error)

4) Now assume that if a user has rated a movie 3 or lower we conclude they didn't like the movie, and if a user has rated a movie 4 or higher they have liked it. Based on the estimated values in the R matrix in parts 1 and 2, we can use a threshold on the estimated entry to predict if the user will like the movie or not. In your tests with the method in part 3, find out the number of entries where your system predicted the user will like the movie. Out of these, for what percentage did the user actually like the movie? (This is called *precision.*) Now find the entries in the testing data where the user did actually like the movie. Out of these entries, for what percentage did your algorithm predict the user will like the movie? (This is called *recall.*) Plot *precision* over *recall* for different values of the threshold for each system from part 1 and part 2 (Each threshold value will give you a point on the plot.)

[2] https://sites.google.com/site/nmftool/

5) Now we modify the cost function to add a regularization term λ:

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}\left(r_{ij} - (UV)_{ij}\right)^2 + \lambda\left(\sum_{i=1}^{m}\sum_{j=1}^{k} u_{ij}^2 + \sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}^2\right)$$

This will make the matrix calculations more stable. You can find out how to implement the regularized version of ALS in a set of slides found in the reference link[3] . Run your code for $\lambda = 0.01, 0.1, 1$ and compare the results with parts 1 and 2.

6) Create a recommendation system in the following manner:

Convert R to a 0-1 matrix where the entries are 1 for available data points and 0 for missing data points. Use ratings as weights and run your algorithm to find the top L movies recommended to each user. Using the cross-validation method described in part 4, find out the average precision of your algorithm for $L = 5$. While calculating precision, ignore the picked entries for which you don't have the actual rating.

Find your algorithm's hit rate (what *fraction* of the test movies liked by the users are suggested by your system) and false-alarm rate (what *fraction* of the test movies not actually liked by the user, are suggested by your algorithm). Start from $L = 1$ and while increasing L measure hit rate and false-alarm rate for different values of L. Plot these values as points in a two-dimensional space with hit rate on the y axis and false-alarm rate on the x axis. Note that you do not need to run the factorization every time you change L, only the number of results you pick for recommendation will be different.

Submission: Please submit a zip file containing your codes, any output files and a report to ee239as.winter2015@gmail.com . If your zip file is too large to email you can send it to the dropbox account associated with this email. If you had any questions you can send an email to the same address.

[3] http://www.slideshare.net/srowen/big-practical-recommendations-with-alternating-least-squares