Andrew Gates

Professor Piya Pal

ECE 269: Linear Algebra and Applications

16 December 2017

Final Project: Recommender System Using Matrix Factorization

Objective:

The goal of this project is as follows –

A recommender system tries to model the preference of a user by analyzing the usage pattern and make appropriate suggestions. Recommendation systems have become ubiquitous in many areas such as movies, music, e-commerce. The data for a recommendation system is often stored as a matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ where tM users provide ratings for N items. Matrix factorization models aim to decompose this rating/preference matrix as a product of two matrices.

$$\mathbf{A} = \mathbf{PQ}, \quad \mathbf{P} \in \mathbb{R}^{M \times K}, \mathbf{Q} \in \mathbb{R}^{K \times N}$$

This factorization of the matrix allows us to map each user as well as each item to a common latent space of dimension $K \leq min(M, N)$. The resulting dot product, $p_i^T q_j$, captures the interaction between user-i and item j that will allow the recommender system to make predictions. In this project you are expected to do the following:

Background:

Since the explosion in popularity of movie streaming services such as Hulu and Netflix, as well as online music streaming services such as Spotify and Pandora, recommender systems have become exponentially popular. Anything that could possibly tie the user's interests to the interest of another movie, song, product and etc. can be utilized to recommend related products to the user. These Recommender Systems are fairly simple in nature, they aim to predict the rating of an item for the user based on how the user rates other similar items.

A) Read the papers listed in the reference and discuss different techniques used for designing a recommendation system

There are two main methods to designing a recommendation system, Collaborative Filtering and Content-Based Filtering. Collaborative Filtering aims to create a model based on the user's interest, and similar interests from other users as well. Using this model it can predict items that the user may have an interest in based on past behavior and the behavior of fellow users. While Collaborative Filtering uses data from the user and other users alike, Content-Based Filtering

attempts to use information about the item itself in order to recommend similar items to it. Both have their advantages and disadvantages. Collaborative Filtering may be more useful for a company like Netflix who wants to recommend to the user movies and shows that other users with similar models enjoy. Whereas a company like Amazon may be more interested in a Content-Based Filtering approach, where the user would want to see similar items to the ones that they purchased, rather than items purchased by another user who bought the same item as them.

Collaborative Filtering is the main method that I used for my Recommender System. When using Collaborative Filtering there are various approaches to creating the user model.

One approach is the Non-Personalized Model. This model is fairly simple, this model will present to the user a list of the top recommended items regardless of the user's preference. This model will not be heavily influenced based on the user's ratings of items. It will definitely change based on the majority of user's ratings, but an individual user will not have much control on the overall top recommended items. This method aims to present the user with the highest rated overall items, like how the top items on Amazon are displayed, or the top recommended movies in Netflix. These top items do not take into account the individual user, but rather the community as a whole.

Another approach to Collaborative Filtering is the Neighborhood Model. In this approach, the Neighborhood Model is used to find the relationships between individual users and individual items. It attempts to find a metric that will represent the relationship between a user (u) and an item (i), based on using the items that are most similar to item (i). Using this metric over all of the items it can then sort out which items the user will have the most interest in.

The final approach discussed is Latent Factor Models. In this approach the model attempts to take the matrix (A) that is composed of (x) user's ratings over (y) items. Using a method called Singular Value Decomposition it breaks the matrix (A) into two matrices (P and Q). These matrices can be used to index the current user (pi) and the current item (qj), and then taking the dot product of these vectors results in the interaction between user (i) and item (j). This method allows for a model that relates every single user to every single item, and allows for prediction of new user's recommendations based on those of previous users, as well as new recommendations for old users, based on the current user ratings.

Results:

B) Read (i) and (iii). Using real data, implement a recommendation system using the Singular Value Decomposition (SVD) based technique. In most cases, the matrix A is incomplete. Can you directly use SVD or do you need to make certain assumptions?

The data I used for my project was Jester Dataset 1_1, found at –

http://eigentaste.berkeley.edu/dataset/

By Professor Ken Goldberg. According to Ken Goldberg, This data has the following format –

1. 3 Data files contain anonymous ratings data from 73,421 users.
2. Data files are in .zip format, when unzipped, they are in Excel (.xls) format.
3. Ratings are real values ranging from -10.00 to +10.00 (the value "99" corresponds to "null" = "not rated").
4. One row per user.
5. The first column gives the number of jokes rated by that user. The next 100 columns give the ratings for jokes 01 - 100.
6. The sub-matrix including only columns {5, 7, 8, 13, 15, 16, 17, 18, 19, 20} is dense. Almost all users have rated those jokes (see discussion of "universal queries" in the above paper).

The problem with this data is that not every user rated every joke. There are many cases where a user rated 100% of jokes, but many where a user only rated 50%. This can lead to complications in SVD when our matrix (A) is incomplete. In my case I filled each missing rating with a 0. In most cases a 0 would not work because that would imply the user gave something the lowest rating possible. But since the range of ratings for Jester goes from -10 to +10, a 0 would imply that the user was neutral to the joke, similar to no rating but it will make it so that this joke has no impact on the overall model. Another way to get around this incomplete A is if you fill all missing ratings with the average user's ratings overall all of the jokes they have rated. That means that if they often give very good scores or very bad scores, these missing ratings will adhere to the same trend of the user's rating system.

As an aside I originally tried using the MovieLens data. This data was very interesting and showed how individual users rated various movies on a scale from 0 – 5. It showed multiple ratings for each individual user, roughly 10 for each user. Using these would have proved interesting, but the structure of the data that MovieLens released was not in the format usable by SVD. The way it was setup by MovieLens had only 3 columns, users, movies, and ratings. To put it into the same format, matrix (A) would be very sparse. Because there were 27,000 movies but each user only had ratings for roughly 10 out of these 27,000 movies, so each user rated roughly .037% of the total movies. I attempted to make some meaningful conclusions from the data before realizing I was doing it on an incorrect matrix (A), and decided to use the Jester data instead.

C) Compare the performance of your implementation of (b) with alternative approaches given in (a). Explain the performance evaluation metrics used.

The implementation I used in b, was that of Latent Factor Modeling. The implementation here that I decided to compare it against was the Non-Personalized Model. One of the best performance evaluation metric that can be used to compare the performance of different models in Recommender Systems is that of precision and recall. Precision is the proportion of recommended items in the top-N that are relevant, while Recall is the proportion of relevant items found in the top-N recommendations. I was not able to get these evaluation metrics working properly, but in the journal article "Performance of recommender algorithms on top-n

recommendation tasks." by P. Cremonesi, Y. Koren, and R. Turrin [1], they used the MovieLens data that I attempted to use earlier and found the following results for precision and recall.

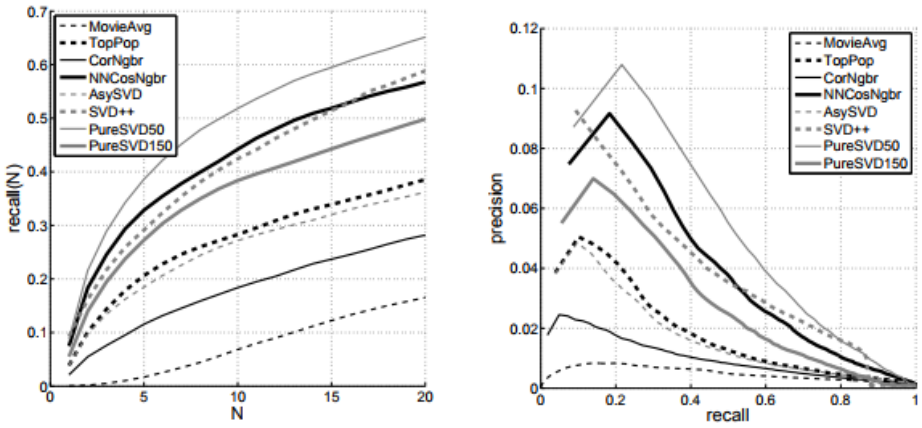

Figure 2: Movielens: (a) recall-at-N and (b) precision-versus-recall on all items.

Figure 2. MovieLens: (a) recall-at-N and (b) precision-versus-recall on all items. (From P. Cremonesi, Y. Koren, and R. Turrin, 2010)

This shows what is expected, MovieAvg and TopPop represents Non-Personalized Model, while AsySVD represents the Latent Factor Modeling approach that I used, and SVD++, PureSVD50, and PureSVD150 represent other approaches to Latent Factor Modeling. As expected the Non-Personalized Model is at the bottom in both recall and precision vs recall. This is due to the fact that it really does not take into account anything about the individual user, it only takes into account information about the community of users as a whole. Thus resulting in recommendations that, while may be accurate for the whole, are not as accurate for the individual. Also as expected, the Latent Factor Modeling approaches performed very well, AsySVD which I used is roughly in the middle of the pack, while more complex SVD methods result in greater recall and precision vs recall.

I also evaluated my two implementations in terms of runtime. I calculated runtime while ignoring the runtime for loading in the data, removing 99's in the data, and splitting the data into training and testing sets. Thus the runtime is just that of each implementation itself. For the Latent Factor Mode I only calculated the recommended jokes for one user (User 20,001), while for Non-Personalized Model approach I calculated the recommended jokes for the group of users as a whole. This resulted in the runtimes of –

Latent Factor Model - .204432 seconds

Non-Personalized Model - .014855 seconds

The Non-Personalized Model is roughly 14 times faster in this example, but again the Latent Factor Model is only calculating the recommended jokes for one user. If this were used over all 4,983 users in the test dataset, it would be much slower than this.

The recommended jokes for user 20,001 are as follows, with Non-Personalized Model on the left and Latent Factor Model on the right for the first 20 jokes–

1	0.6108	1	2.3752
2	0.1451	2	-5.1474
3	0.2118	3	0.5711
4	-0.9060	4	1.8413
5	0.4093	5	-3.3529
6	1.3498	6	-4.4687
7	-0.4491	7	-0.2070
8	-0.5639	8	0.5267
9	-0.3083	9	1.0108
10	1.1148	10	1.0563
11	1.6007	11	-4.4222
12	1.2757	12	-6.5897
13	-1.7168	13	-0.9697
14	1.2496	14	-4.1470
15	-1.6332	15	5.5950
16	-3.0847	16	5.1711
17	-1.1039	17	-0.6661
18	-0.5866	18	-0.4298
19	0.1576	19	0.5579
20	-0.8770	20	2.9138

As you can see there isn't much in common between these two recommendations. Again this is due to the Non-Personalized Model not taking into account the user's preferences.

D) Does the implementation in (b) take care of new users/items? Suggest a modification in (b) to incorporate addition of new items (and users) with proper mathematical justification. Compare the performance of this on-line system with the off-line version

Unfortunately, our implementation does not take care of new users/items. The SVD is calculated only once prior to doing any prediction on the (P) and (Q) matrices. In my example of 20,000 users it only takes roughly .2 seconds to calculate the SVD. But that means that for an example with 10 million users it would take 100 seconds to calculate the SVD. This means that whenever a new user or item is added you would have to take 100 seconds again to recalculate the SVD for this updated (A) matrix. This may not be that bad if new users or items are being added slowly, but if they are being added faster than once every 100 seconds, it would cause issues in the recommendation system.

A way to incorporate new additions of items is using the method proposed by Matthew Brand in "Fast online SVD revisions for lightweight recommender systems" [6]. In this journal article he proposes the solution as follows –

Given the SVD of $X = USV^T$, X' represents the updated SVD with the new data added where $X' = X + ab^T$. In this case a is the user vector (pi) and b is the item vector (qj) to be added. To do this, calculate the various values -

- $m = U^Ta$
- $p = sqrt((a-Um)^T(a-Um))$
- $n = V^Tb$
- $q = sqrt((b-Vn)^T(b-Vn))$

Then we can represent the updated SVD of X' as –

$$X' = U'S'V'^T = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} m \\ p \end{bmatrix} \begin{bmatrix} n \\ q \end{bmatrix}$$

This is much faster, now instead of recalculating the SVD every single time new users or items are added, we do the same by calculating the five steps above.

The time complexity of a full SVD on a single matrix of dimensions m x n is that of $O(m^2n + mn^2 + n^3)$, which is cubic in complexity Even for just small samples, recalculating this over and over with a cubic time complexity leads to runtimes that blow up in time as n gets very large. But using this method by Matthew Brand, the time complexity is only O(mnk). This is obviously still fairly slow, but comparing O(mnk) to $O(m^2n + mn^2 + n^3)$ shows just how much more efficient it is to use this on-line method rather than recalculating of SVD for an off-line method.

Discussion:

The method that I choose in this project is that of a Collaborative Filtering, Latent Factor Model, with AsySVD. This approach worked decently, it gave reasonable results in a reasonable runtime. This would not be a very good approach however for large data sets. AsySVD would be very time consuming as the number of users and items grew large. At this point, switching to SVD++ or PureSVD would greatly increase accuracy and reduce runtime. AsySVD is great though for small samples like in my case, and performed very well for the 25,000 users and 100 items in the Jester data set.

References:

[1] P. Cremonesi, Y. Koren, and R. Turrin. "Performance of recommender algorithms on top-n recommendation tasks." In Proceedings of the fourth ACM conference on Recommender systems, pp. 39-46. ACM, 2010.

[2] Y. Koren, R. Bell, and C. Volinsky. “Matrix factorization techniques for recommender systems.” Computer vol. 42, no. 8 (2009).

[3] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. “Application of dimensionality reduction in recommender system-a case study”, innesota Univ Minneapolis Dept of Computer Science 2000.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. “Incremental singular value decomposition algorithms for highly scalable recommender systems.” In Fifth International Conference on Computer and Information Science, pp. 27-28. Citeseer, 2002.

[5] M. W. Berry, S. T. Dumais, and G. W. O’Brien. “Using linear algebra for intelligent information retrieval.” SIAM review, vol.37, no. 4, pp. 573-595, 1995.

[6] M. Brand. Fast online svd revisions for lightweight recommender systems. In Proceedings of the 3rd SIAM International Conference on Data Mining, 2003.