Matrix Factorization for Recommender Systems
CS 383
Chris Uzokwe
6/9/21

Abstract

In the advent of the digital age goods, services, and content have become the cornerstone of connecting customers to small and large businesses alike. Platforms like YouTube, Netflix, Amazon, and many others host massive catalogues of “items,” for lack of a better word. To ensure that users do not get lost and can find what they’re looking for, as well as new things they like, many corporations turn to powerful recommender systems. These systems can generate competitive advantages in their market place, or improve the customer experience, and have become an important part in many industries. The likelihoods of user-item or item-item matches can be predicted using the large amounts of data gathered, aiding in selection. There are many algorithms which exist under these systems, but I am exploring a popular variant -- matrix factorization. I will see how content and collaborative methods can be used through this variant to create predictions.

Background

Collaborative Filtering is the cornerstone of many recommendation systems -- proving to be invaluable for companies such as Amazon, Netflix, and Spotify. By recording the past predictions of their millions of users, these systems can form a correlation between users with similar preferences, or items with similar content. With these user-user or item-item methods , collaborative filtering systems are able to generate new recommendations for items that a user has yet to rate.

The primary algorithms used in collaborative filtering are neighborhood methods, and latent factor models. Neighborhood methods, like the popular k-means, focus on establishing ratings between neighboring items or users. A neighborhood algorithm might recommend items

to a user based on the ratings of similar items a user has rated. On the other hand, latent factor models work to establish some underlying feature vectors about the data, which can be used to extrapolate predictions to new users -- using a combination of the latent vector information. The most popular implementation of this model is in matrix factorization.

Matrix Factorization methods can break down matrices into their latent representation, and we can use these latent representations to reconstruct missing predictions in a system. Obviously following, we would need a matrix which we can break down. We can create a user-item matrix, which holds the available ratings for each item by each user. An example is below:

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

Figure 1: Example user-item matrix

Matrix factorization would break this user-item matrix into the latent space of dimensionality D, such that each item is now associated with the elements of q, and each user is associated with the elements of p. The dot product of the two elements gives us an approximation of rating (i,j) -- the rating of item I by user J.

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Figures 2 and 3: Latent vector matrix and recommendation rating calculation

Matrix Factorization has several methods for implementation, but this work will use stochastic gradient descent to zero in on the correct representations. Stochastic gradient descent scales

well to larger datasets, while other methods like singular value decomposition are better for one-shot predictions. In SGD, our error can be calculated as the difference between a user's previously rated items, and the predicted rating. Using this, we can figure out our update rule for the latent vector weights.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Figures 4 and 5: Error function and update function for SGD

This work implements a vanilla version of the matrix factorization algorithm, with the inclusion of a regularization parameter that controls the rate of our descent (along with hyperparameters). The final update rule is below:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij} q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij} p_{ik} - \beta q_{kj})$$

Figure 6: SGD update function with regularization

We also use the movielens 100k dataset as our user-item matrix. In this dataset, over 900 users have rated movies on a scale of 1-5 (with a lot of the ratings missing), for over 1600 movies. The data is available online, and after some manipulation using pandas, I was able to properly load the data. The main exploration of this work is to understand on a small scale how the number of users and items predicted would affect training times or predictions times. As such, I run several models which use a varying number of users, and items. I compare the resulting recommendations with each other to see how well the system performs.

Experiments

5-user, 50-item Factorization

We start with 5-user, 50-item factorization. This implementation converges the fastest (23 seconds), seeing as it has the least amount of overall data in our implementations. In a similar way, although it converges, the overall error is less than most of the other implementations. We use an additional metric of a top 5 error to rate the first 5 users. In this run, it's all of the users we are considering, and the final error is 0.7.

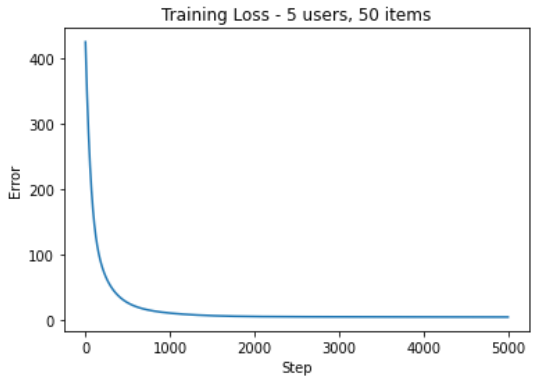

Figure 7: Training Loss (5-users, 50-items)

We also pull a few ratings and recommendations from the system. We will be using user 1 as an example for each implementation. You can see that we have similar years between the movies, as well as a lean toward more dramatic or serious movies.

	movie_id	Title	Year
47	48	Hoop Dreams (1994)	01-Jan-1994
0	1	Toy Story (1995)	01-Jan-1995
31	32	Crumb (1994)	01-Jan-1994
18	19	Antonia's Line (1995)	01-Jan-1995
15	16	French Twist (Gazon maudit) (1995)	01-Jan-1995

	movie_id	Title	Year
11	12	Usual Suspects, The (1995)	14-Aug-1995
19	20	Angels and Insects (1995)	01-Jan-1995
38	39	Strange Days (1995)	01-Jan-1995
43	44	Dolores Claiborne (1994)	01-Jan-1994
9	10	Richard III (1995)	22-Jan-1996

Figure 8: 5-user, 50-item ratings and recommendations (respectively)

5-user, 150-item Factorization

Moving to 5 users and 150 items, we can see that the system converges similarly (step-wise), with a larger starting error, from the calculation over all ratings. (5*150 = 750). This system takes 45 seconds to train, and has a top 5 error of 2.2.

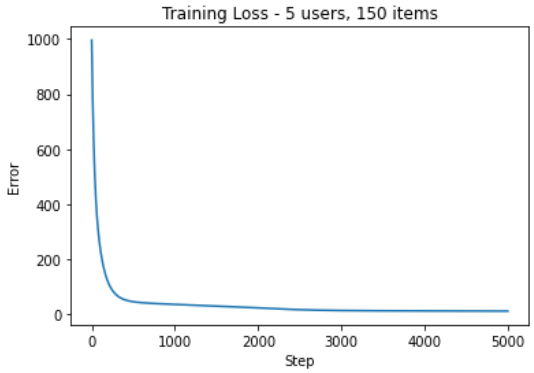

Figure 9: Training Loss (5-users,150-items)

For our new recommendations for user 1 we can see that their ratings have changed since there are more options. There is also a slightly greater variety of years -- the furthest being the Wizard of Oz. I would assume here that that recommendation was so popular that it ended up in user 1s list. In general, the movie list is still a rather serious selection.

	movie_id	Title	Year
123	124	Lone Star (1996)	21-Jun-1996
58	59	Three Colors: Red (1994)	01-Jan-1994
118	119	Maya Lin: A Strong Clear Vision (1994)	01-Jan-1994
86	87	Searching for Bobby Fischer (1993)	01-Jan-1993
114	115	Haunted World of Edward D. Wood Jr., The (1995)	26-Apr-1996

	movie_id	Title	Year
38	39	Strange Days (1995)	01-Jan-1995
127	128	Supercop (1992)	26-Jul-1996
79	80	Hot Shots! Part Deux (1993)	01-Jan-1993
16	17	From Dusk Till Dawn (1996)	05-Feb-1996
131	132	Wizard of Oz, The (1939)	01-Jan-1939

Figure 10: 5-user, 150-item ratings and recommendations (respectively)

30-user, 50-item Factorization

With 30 users and only 50 items, our general training loss increases again. The model stepped through 5000 iterations in 71 seconds. The final top 5 loss, however, was 1.4, which could be an indicator that more users is better than more items.

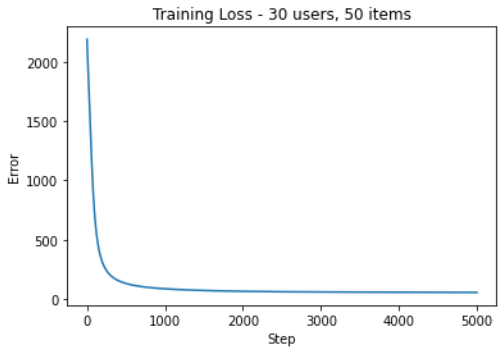

Figure 11: Training Loss (30-users, 50-items)

The predictions show that we can get a slightly different result with only 5 users, as the recommendations are similar, with similar years and content types.

	movie_id	Title	Year
47	48	Hoop Dreams (1994)	01-Jan-1994
0	1	Toy Story (1995)	01-Jan-1995
31	32	Crumb (1994)	01-Jan-1994
18	19	Antonia's Line (1995)	01-Jan-1995
15	16	French Twist (Gazon maudit) (1995)	01-Jan-1995

	movie_id	Title	Year
38	39	Strange Days (1995)	01-Jan-1995
30	31	Crimson Tide (1995)	01-Jan-1995
13	14	Postino, Il (1994)	01-Jan-1994
46	47	Ed Wood (1994)	01-Jan-1994
48	49	I.Q. (1994)	01-Jan-1994

Figure 12: 30-user, 50-item ratings and recommendations (respectively)

30-user, 150-item Factorization

30 users and 150 items greatly increases the loss incurred on the system. This run trained in 160 seconds, with a top 5 prediction rate of 2.18. This is similar to the 5-user, 150-item factorization, which implies that it may be more difficult to create relevant predictions with greater items.

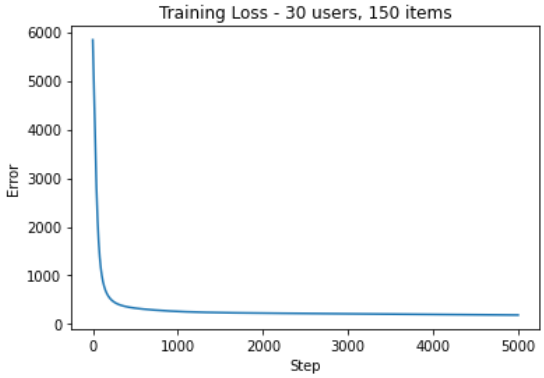

Figure 13: Training Loss (30-users, 150-items)

We can see that with more users and items, our ratings move back to a tighter bound of predictions. With 5 users and 150 items, we see popular yet off-genre selections like the wizard of Oz get recommended, vs. here, the years are more tight with similar genre.

	movie_id	Title	Year
123	124	Lone Star (1996)	21-Jun-1996
58	59	Three Colors: Red (1994)	01-Jan-1994
118	119	Maya Lin: A Strong Clear Vision (1994)	01-Jan-1994
86	87	Searching for Bobby Fischer (1993)	01-Jan-1993
114	115	Haunted World of Edward D. Wood Jr., The (1995)	26-Apr-1996

	movie_id	Title	Year
95	96	Terminator 2: Judgment Day (1991)	01-Jan-1991
96	97	Dances with Wolves (1990)	01-Jan-1990
139	140	Homeward Bound: The Incredible Journey (1993)	01-Jan-1993
149	150	Swingers (1996)	18-Oct-1996
124	125	Phenomenon (1996)	29-Jun-1996

Figure 14: 30-user, 150-item ratings and recommendations (respectively)

Conclusions & Future Work

We can see how the recommendations vary from system to system. The big systems that run in large companies have to consider this trade off when they are making predictions, but more data means that the systems are able to generalize to new users. Generally, the more items*users we had, the longer training time and the greater overall error that was incurred. It is also worth noting that in terms of predictions, we saw greater variance in predictions with more users.

Future work on this implementation might include a more in depth analysis on how similar the selections are. Currently we just have movie title and date, but it would be more advantageous to see how the genres group up based on these movies, or something similar. On the flip side, we could use the subsequent ratings to group the users by demographic. Generally, there is a lot more inferences that could have been made from the system, but time was the constraint in this work.

Sources

Takács, Gábor, et al. "Matrix factorization and neighbor based algorithms for the netflix prize problem." Proceedings of the 2008 ACM conference on Recommender systems. 2008.

Patrick Ott (2008). Incremental Matrix Factorization for Collaborative Filtering. Science, Technology and Design 01/2008, Anhalt University of Applied Sciences.

Yeung, Albert Au. "Matrix factorization: A simple tutorial and implementation in python." (2010).