Optimizing Social Recommendation: Efficiency vs. Accuracy

Steven Rutledge[1]

Abstract

Social recommendation plays a huge part in our lives today. Everything from personalized ads on Instagram, to what else you should buy on Amazon. However, applications like these have a large amount of users and products that can lead to a computational nightmare. We explored a few different articles about different aspects of weighted recommendation to understand the different use cases including political bias and emergency-like events. We ended up deciding to explore a Discrete Social Recommendation [2] algorithm that make use of user ratings and trust values upon specific products.
The original DSR [2] paper expressed their approach and sudo-functions to implement a faster, more computationally friendly social recommendation algorithm. We then took the outline of the approach, worked to re-implement these methods and test in our own manner. In order to cope with computational complexity required by a large n $\times$ m user-item matrix, we decompose the matrix into real valued latent vectors, then encode these values in a binary representation. Like many social recommendation algorithms, the users preference is determined by the dot product of the vectors in latent space. However, the niche aspect to this is how we calculate the inner product through bitwise computations to save space and time. We compared the resulting algorithm to the industry standard approach and compared the accuracy of the two methods.

1. Review and Critique

We read a few papers about how recommendation algorithms are used in social media to help understand the particular use cases and approaches to this concept.

1.1. Tracking Disaster Footprints with Social Streaming Data [1]

1.1.1. Summary [1]

This paper dives into the theme of social media tracking by focusing on topic detection from the specific content of each post. The proposed use case for this algorithm is in the modern world, the quickest way to seek relief efforts for natural disaster and events alike is often through social media such as twitter. An example they used was that after Hurricane Harvey there were over 21 million Tweets in less than a week. These tweets will have news about future dangerous, human struggles, evacuation routes, missing people, ect. that all can be relevant useful information for the disaster relief process. The problem is how to condense and sort related and unrelated tweets, into common and distinct topics.

The proposed solution measures topics over time to efficiently identify common and distinct topics. This solution, Tracking Disaster Footprints (TDF), uses a Non-negative Matrix Factorization (NMF), to store information about the existing tweets. The matrix reduction used in the algorithm converts the original data matrix X, into a latent factor H, to speed up computation. The algorithm with take two points in time, t and t+1, to aim to identify the number of common topics among tweets. After identifying the common topics, they perform linear transformation to create a new feature space where distinct topics are discovered from knowing which topics are in common and which are not. They use two provided data sets from hurricanes in both 2017 and 2018, to test their model. There are models to calculate the "commonness" score and the "difference" score and then used for comparisons to validate their evaluation of tweets with common or distinct topics. The general application is to filter out the unnecessary tweets and make sure in a time of crises the most relevant information is linked together and presented to create an efficient recovery effort.

1.1.2. Review [1]:

The focus of this algorithm is so specific that it makes me wonder if this can be used across other topics? For

example can this replace Twitters existing algorithm that may place ”GANs” and ”Machine Learning” under the same topic? The concept that seems most important is out of the ”21 million tweets”, to provide the most useful and necessary information. The algorithm seems to focus on using matrix reduction to link tweets with different word choice and topic identification, but would it be a separate function that decides strength and priority of this information? The idea of matrix reduction seems like a good one, and I do enjoy the practicality of the method, including recent relevant data sets. A drawback seems to be mentioned in the conclusion where in the future they will attempt to recompute k_c (the number of common topics) for each input, along with distance measures for gauging connected topics. What are the drawbacks of using a set k (number of topics) at 10 topics when testing unknown data sets. Can this lead to a less than maximized output capable by this algorithm? Will some topics be forgotten? Will distinct topics be merged under a common one? To end on a positive, the references are well done, and the formulas for the algorithm are clearly displayed for re-use and captioned in detail.

1.2. Discrete Social Recommendation [2]

1.2.1. SUMMARY [2]:

Social recommendation has taken over the world through our form of interactions with each other and many online decisions. This paper describes how in order to provide the item suggestions for each individual user, there exists very expensive algorithms in terms of storage and computational costs, and then proposes a method to reduce these costs.

The proposed solution is a Discrete Social Recommendation (DSR) algorithm that uses bit-wise comparison of latent vectors to allow for a more compact relationship between user and items. The original $n \times m$ user-item matrix are decomposed into real latent vectors, then a binary comparison using the Hamming formula is used to estimate the inner product between these latent vectors. Items may be Instagram pictures, movie ratings, news articles, ect. They use 3 real world data sets in their testing, and got results that solved both the efficiency and storage issues at hand. In testing the change in accuracy was nominal while, the speed increase 5x and took up 1/37th of the memory. The algorithm is what they refer to as a two-staged binary method, that eliminates the need to compare each item of the original matrices containing all users and items. The approach has complexity $\Omega(mnr + mn \log k)$ to compute and sort the preferences in descending order. The use of Hamming distance to compute the dot product between users and items is shown by $b_i d_j = 2H(b_i, d_j) - r$. In general this article proposes a more efficient Discrete Social Recommendation algorithm by reducing the number of comparisons, in order to save space and time costs.

1.2.2. REVIEW [2]:

The authors do a strong job of getting the reader hooked by explaining how used social media recommendation is used, and then immediately throw out their algorithms extreme statistical success with 5 times as fast and 1/37th of the memory. This relates to the other papers we read, as the back-end recommendation algorithm is the responsible piece for insuring readers get both a balanced scope of issues, and what would assign the highest priority to a post for an emergency professional. The part that seemed most necessary was their comparison to DCF and CFSR which are competitive algorithms that do not use bit-wise comparison in the latent vectors. The results against these ”state of the art” algorithms was that their DSR was more accurate than competitor DCF, and more efficient than CFSR. In order to test the accuracy of these functions they evaluated the algorithms based on the Normalized Discounted Cumulative Gain (NDCG), which compares the position and weight of rankings to output a score between 0 and 1. They were able to use the optimized parameters for each algorithm and compare the results. The use of related works helps explain the need and use for a new algorithm in the same industry, and sells the reader/user on its superiority. We plan to test this theory to see if the no-loss in accuracy claim holds true.

1.3. Co-exposure maximization in online social networks [3]

1.3.1. SUMMARY [3]:

The general purpose of this report is to propose a solution to the idea that social media is just giving people what they want to see rather than a balanced set of posts. This becomes an issue in the political world, where users feeds become echo chambers and only further polarize viewpoints. The idea is to form a new social media algorithm that insures full exposure to both sides rather than re-enforcing existing views, hence the title, ”Co-exposure maximization in online social networks. Instead of using just user preferences and trust ratings like in DSR [2] this algorithm uses two disjoint sets from the political spectrum and attempts to maximize exposure to both

The authors describes some related work and gives examples of how other functions will classify articles as $[-1, 1]$ to provide an article suggestion of equal and opposite classification in order to balance user exposure.

The proposed solution uses a graph algorithm of directed edge (u, v) from graph $G = (V, E)$ where user u, can see posts v. In a given example of two politicians campaigning on opposite sides of an issue, the goal is to expose the user to the maximum number of nodes in two disjoint sets from each side. They use two seeds S_r and S_b for political campaigns r and b each given a "set budget" to insure the amount of nodes traversed result in equal number from each campaign. The sets will be NP-hard and a greedy algorithm will be used to trace the directed graphs described above. In their evaluation measuring the balanced of co-exposure models the CoEM [3] model outperformed their existing competitor BalanceExposure.

1.3.2. Review [3]:

The CoEM method [3] is another approach in the space the DSR algorithm [2] attacks into the decision of what is displayed for each user. However it goes into a very different aspect of it, Discrete Social Recommendation [2] is focused on efficiency and computational costs, while this does not set that as a priority. The main focus here is to ensure a balanced political exposure to all of its users in order to avoid the existence of "echo chambers." I would ask the authors why they chose the graph approach and if the opposite political views would need to be in disjoint sets? My thoughts would be that with a left/right political spectrum they could use something like the binary algorithm in the "Discrete Social Recommendation" [2], to make sure the user is exposed to an even $[0, 1]$ instead of the need for a graph solution. I think using a binary comparison would save in computational costs. I also want to know more about current use and if existing social media algorithms have some attempts in place to balance the content from both political sides. I think this article and method has great potential and could help soften the divide and prevent some arguments occurring on social media.

2. Implementation goal

Our goal is to use the ideas presented in the social recommendation articles described above to re-implement the Discrete Social Recommendation algorithm using the algorithm described in the DSR paper [2]. We plan to use a subsection of the DSR suggested data sets to a size compatible with the Google Colab computing power, to represent user ratings and trust among various social items. Though my 3 selected papers were all social media algorithms, the user recommendation stood out above balancing users political feeds [3] and sorting useful emergency information [1] demonstrated by the other reviewed papers. We wanted to explore an algorithm focused on efficiency and what we believed would be a real world application's main goal.

In social recommendation, a users preference for an item is calculated by the dot product between the user and item preferences, and well as the social factor latent vectors. The goal of this algorithm and our implementation is to be able to use bit-wise comparisons to estimate this inner product using Hamming distance represented by $b_i d_j = 2H(b_i, d_j) - r$. Calculating the product for a very large $n \times m$ user-item matrix can be extremely expensive and a bitwise comparison of the latent vectors contents have the potential to be much more efficient. We are going to implement this method and test the accuracy of this estimation using a discounted cumulative gain on our sorted user preferences values.

3. Implementation plan

Our plan is to re-implement the compact binary comparison through Hamming distance described in the Discrete Social Recommendation paper. The proposal claims to be 5x the speed of many industry standard matrix based algorithms. We could not find this implementation in an existing repository so our plan is to re-implement the methods and optimizations algorithms described in the DSR paper [2]. However, there are some repositories and sudo-code algorithms we will use to help establish the preliminaries that are listed below:

- Matrix Decomposition in order to form latent vectors `https://tinyurl.com/m943ca9x`
- Hamming Distance Algorithm to compare binary strings `https://en.wikipedia.org/wiki/Hamming_distance`

There are a series of steps in this implementation that we will be coding on our own, on Google Colab as part of the re-implementation effort. Though we may use an online resource as a guide for matrix decomposition, most of the code will be written ourselves and the specific steps are listed below:

- Matrix Decomposition of User and Item Preferences to form real valued latent vectors
- Create Trust relationship latent vectors through matrix decomposition
- Minimize Squared loss between Observed ratings and Social relationships
- Replace user and items with binary codes

- Use hamming distance bit-wise comparisons to estimate inner product of latent vectors
- Perform objective function of DSR model
- Evaluate over suggested data-sets using Discounted Cumulative Gain to assess accuracy of results

These steps above are a high level overview of the re-implementation process we followed. For testing the implementation we are going to use the suggested FilmTrust data-set. The size of this dataset was too large for our computing resources, so we will truncate to 200 user-item relationships.

4. Implementation

4.1. Preliminaries

We started the process of implementing the DSR algorithm [2] from scratch. In our initial implementation attempts, were able to work through the preliminary setup but ran into some computing problems. The initial step for the preliminaries was to decompose the user (R) and trust (S) matrices into real-valued latent vectors. We decomposed the 2 original panda dataframes into 3 latent vectors of user preferences P, item preferences Q, and social factor T. However when testing the latent space vectors we found this was only partially working due to overflow issues due to the pure size of the FilmTrust dataset. The found solution was to truncate the dataset to 200 user-item values, and that adjustment is represented in our implementation steps above.

After decomposing user-item matrix R, and the social matrix S, the goal is to minimize squared loss between the original matrices and the latent vectors P, Q, and T. This allows us to find minimum squared loss between the user preferences and social trust factor. The objective function for Matrix R with latent vectors P and Q is:
$min \sum_{i=1}^{n} \sum_{Q_j \in N_i} (R_{ij} - P_i^T Q_j)^2$

We sampled this calculation by taking the first 200 entries of the FilmTrust user-item matrix and the result of the 200x3 matrix was a total squared loss of 138.196. This means that on average each item of the matrix was off by a little more than 0.23. We then graphed the 3 columns of matrix R; User ID, Item ID, and Rating respectively. Below R we visualized the same columns calculated from the dot product of latent vectors P^T and Q to help understand the error.

Though the values were unique for each column of the decomposed matrix, the resulting shape was the interesting. The user id and item rating resulting from the dot product of P^T and Q seemed to almost mirror each other as shown by the graph. Our best result was that the decomposed item id matched the original item id values very closely. We implemented this same method for the Social Matrix S, with P^T and T, and well as graphing the results to compare the error. Once we had identified the 3 real value latent vectors and used the hyper parameters to identify the minimum squared error we were ready to convert these to binary values for efficient comparison

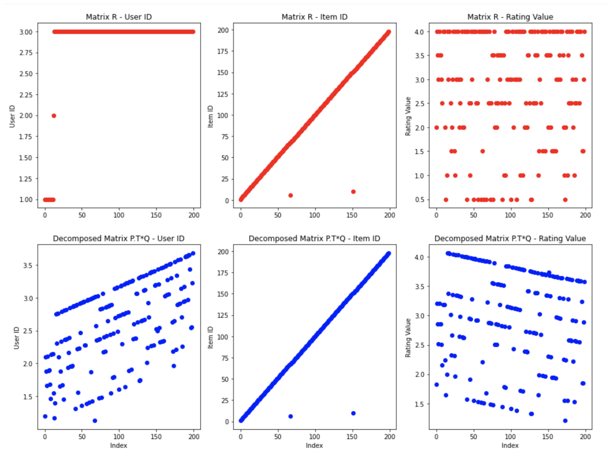


Figure 1. Matrix R vs Latent Vectors P^T and Q

4.2. Establishing Hamming Distance

As we mentioned before the user preference for a specific item is calculated from the inner product of the latent space user and item vectors. However, this calculation can be computationally expensive for a large set of data. Instead, we use a bitwise comparison algorithm, Hamming Distance, and an estimator function to achieve the dot product. The first step we took was to write a function to convert a 32 bit float to a binary string. Each float in the latent vectors, P, Q, and T would be converted into a 32 character binary string made up of the float bits.
$P_i^T \rightarrow b_i, Q_j \rightarrow d_j, T_k \rightarrow f_k$

Next, we created a hamming function which compares 2 binary strings (b_i, d_j) or (b_i, f_k). But instead of a complicated comparison it just returns the number of bits that are the same. If two strings have no binary bits in common the result of the function will be 0, $H(b_i, d_j) = 0$, and if they have all the bits in common it returns the size of the binary string r, $H(b_i, d_j) = r$

The dot product estimator is formed using the above steps by the equation: $b_i d_j = 2H(b_i, d_j) - r$. We take the number of matching bits between two binary strings and double it, then subtract the length of the string,

which in our case is 32 bits. The idea of this is that if half the bits are matching than we are at 0. If there are more than half in common we have a positive recommendation, and if less it is a negative recommendation. This allows us to successfully compute a value for the dot product without executing $n \times m$ matrix multiplication.

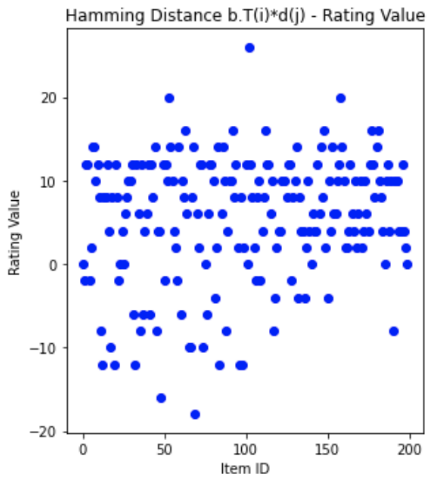


Figure 2. Rating Approximation vs Item ID

In our sample of 200 items we are able to approximate the rating value by comparing the latent vectors through $b_i d_j = 2H(b_i, d_j) - r$. The results of each items ratings are shown graphed below. A higher value symbolizes and stronger rating for the user-item combination. It becomes clear that item 102, has the highest approximated rating of 26.0.

4.3. DSR Objective Function

Once we have established the rating values, we used the idea of DSR Objective Function to minimize loss between the original matrices R and S, and the new found approximated inner product, and well as implemented a weighted coefficient a_0 to adjust the weight of the high trust-trustee id values. The DSR Objective function we followed is:
$argmin_{(B,D,F)}$ **for**
$\sum_{(i,j)\in\Omega}(R_{ij} - b_i^T d_j)^2 + a_0 \sum_{(i,k)\in\Psi}(S_{ik} - b_i^T f_k)^2$

The goal is to achieve the minimum loss between the original matrices, and the decomposed bitwise comparison estimate. However, the nature of our bitwise comparison based on the number of matching bits provides us with a rating scale in the range $[-32, 32]$. We are not sure the specifics of the original DSR authors code for the Hamming function but our implementation adjusts the rankings to this range. In order to then provide an accurate difference between the original values and our approximated values we normalize the original values to a mean of 0, and adjust the range to be the same $[-32, 32]$. The resulting total squared loss found by the equation $\sum_{(i,j)\in\Omega}(R_{ij} - b_i^T d_j)^2$ for our 200 user-item sample was 83730.37 with the graph of the column differences shown in figure 3.

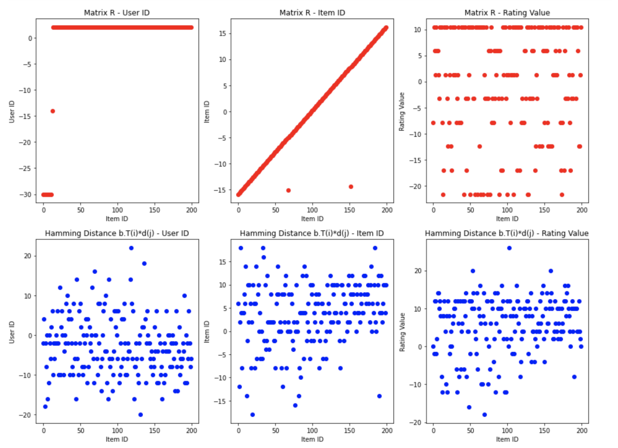


Figure 3. Adjusted R vs. Rating Approximation

This seems very high to us. Even after the normalization of the matrices we still on average have a difference of 12 for each data point among users, items, and then rating score. Our understanding for the cause of this seemingly large error is the need for normalization among the original matrices. Another reason for this error may be the decomposition of the user matrix at such a small scale of sample. Our sample of 200 items only includes users 1,2, and 3, so when adjusted to a $[-32, 32]$ range scale this can lead to unnecessary squared loss. We hope to identify the errors and further optimize the DSR objective function in the future.

5. Results

5.1. Evaluation Method

The goal of the Discrete Social Recommendation algorithm is to provide a more computationally efficient method to rank social items. In common recommendation algorithms, item rankings are calculated through the inner (dot) product between a user-item matrix in latent space. However, after we decomposed the matrix, we used the hamming distance between the binary strings of the contents of the latent vectors. We figured the best way to evaluate this approximation was to also compute the dot product, while calculating and visualizing the squared loss between the two approaches.

The second method is to use the DSR authors suggested accuracy approach by calculating the Normalized Discounted Cumulative Gain or NDCG. This method evaluates the effectiveness of a recommendation algorithm by scoring the results based on a sorted list of relative relevance. This function will provide us with a score between 0 and 1 based on the effectiveness of our rankings. The SKLearn library provides us with this function that will be used to assess the accuracy.

5.2. Approximation of Inner Product

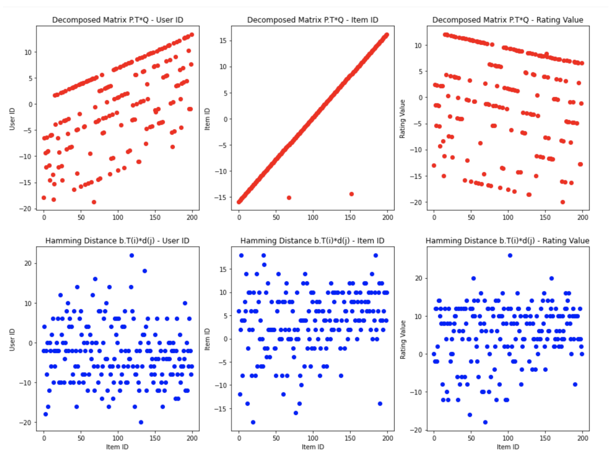
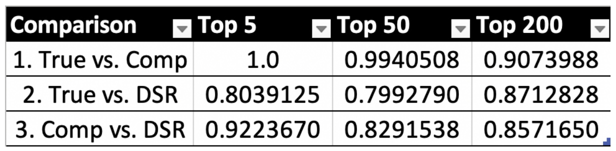


Figure 4. Latent Vectors vs. Rating Approximation

We calculated the dot product of the user-item matrices which is our competing algorithm's method, then graphed that against the result of our computationally friendly dot product approximation in figure 4. The total squared loss between these 2 methods for the 200×3 item subset from FilmTust, was 82,195.88. When breaking the loss down for the 600 values in the dataset and then square rooting the loss we find that on average we 11.7 units off from the competing algorithms values. In a range of 64 units $[-32, 32]$ we find that the average difference was 19.5 percent. The given data set had rankings for items on a scale of [0,4], or a 25 percentile ranking system. Though the error may seem large due to the re-adjustment of the scale, we were on average within 1 point of the original ranking scores. We believe this means that this means the hamming distance would serve as a viable replacement to the dot-product computation.

5.3. NDCG Evaluation

Our second evaluation for accuracy was to use the Normalized Discounted Cumulative Gain or NDCG for short. This method allows you to compare the true rankings with the estimated rankings, and provides a score between $[0, 1]$ based on the weighted relevance by position and accuracy. The weighted relevance means that the first scores have more weight then the last. Such that the 2nd most suggested is more important than the 197th. We sorted the lists and performed 3 evaluations to score the top K item rankings. The evaluations were the 3 comparisons shown below over top {5,50,200} items for our K value.



Comparison	Top 5	Top 50	Top 200
1. True vs. Comp	1.0	0.9940508	0.9073988
2. True vs. DSR	0.8039125	0.7992790	0.8712828
3. Comp vs. DSR	0.9223670	0.8291538	0.8571650

Figure 5. NDCG Comparison Chart

1. True Scores vs. Competition Dot Product Scores
2. True Scores vs. Our DSR Hamming Distance Scores
3. Competition Dot Product Scores vs. Our DSR Hamming Distance Scores

The closer the score is to 1 the more accurate the ranking algorithm is. We can see that across all selected K values our hamming distance method was out performed by the original dot product solution. The dot product of the latent space seemed to be near perfect when selecting the top 5, or top 50 items, however it did slip to 0.9 when testing the entire dataset. Our approximation algorithm was consistently less accurate and posted scores around 0.8 when comparing our sorted rankings to the true values, but seemed to improve when ranking all items. The last comparison we checked, was how our item rankings matched up directly to the pure dot product approach. We found that for small values, we received a very high NDCG score as the top results proved to have a 0.922 score, and not shown in figure 5 but an even higher 0.978 score when selected just the best 3 items. As we tested more items the scores strayed further from 1, but proved to be some similarities for ranking the very best items.

6. Conclusion

After implementing the Discrete Social Recommendation Algorithm [2] and testing the recommendation by comparing it to an industry standard method and evaluating the item scores, we found that though more efficient, we sacrifice some accuracy. In an attempt to approximate the inner product with a hamming distance estimate, we find it increases loss, and creates a larger discrepancy between the true rankings and our calculated scores. Though it does increase the error, it is still widely efficient and received an NDCG score of about 0.8/1 or higher for all the tests. For the most part, it

provides an effective ranking system for the items and is an algorithm that would thrive with a large data set in a case where efficiency is more valued than accuracy.

References

[1] Lu Cheng et al. "Tracking Disaster Footprints with Social Streaming Data". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.01 (Apr. 2020), pp. 370–377. DOI: 10.1609/aaai.v34i01.5372. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5372.

[2] Chenghao Liu et al. "Discrete Social Recommendation". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 208–215. DOI: 10.1609/aaai.v33i01.3301208. URL: https://ojs.aaai.org/index.php/AAAI/article/view/3787.

[3] Sijing Tu, Cigdem Aslay, and Aristides Gionis. "Co-exposure Maximization in Online Social Networks". In: 33 (2020). Ed. by H. Larochelle et al., pp. 3232–3243. URL: https://proceedings.neurips.cc/paper/2020/file/212ab20dbdf4191cbcdcf015511783f4-Paper.pdf.