Graph-Based Recommendations of Amazon Products

Aaron Effron
Department of Computer Science
Stanford University
Stanford, CA 94305
aeffron@stanford.edu

Kelly Shen
Department of Computer Science
Stanford University
Stanford, CA 94305
kshen21@stanford.edu

Ryan Mui
Department of Computer Science
Stanford University
Stanford, CA 94305
ryanmui@stanford.edu

1 Introduction

There is no doubt that recommendation systems have become increasingly relevant in the modern consumer economy. Two main problems, however, exist in the construction of such systems: handling sparse user data, as noted by Shams and Haratizadeh [5], and a need for implicitly derived information, as recognized by McAuley and Leskovec [3]. To overcome the issue of sparse data, several past solutions have proposed neighbor-based collaborative filtering methods in which users and items are represented as a bipartite graph with links between users and items rated, which is then used to make recommendations. Others have sought to extract implicit information from data such as user reviews [3]. Combining the two strategies, we use the Amazon Product Dataset to construct a graph-based recommendation system supplemented by implicit information garnered from user reviews, ratings, and characteristics of their graph structure. The problem we aim to solve is to recommend an Amazon product that a target user will like, given a user profile of products bought and the corresponding review metadata of these products.

2 Related Work

2.1 Network-based recommendation algorithms: A review

Yu et al. surveys various network-based recommendation systems and outline differences between them, the impact of these differences, and their performance on three datasets. The authors further discuss the implications of time on recommendation systems; the ideal case, reflecting practical use cases, is to predict the most recent links based on past links rather than removing a random subset of the graph to predict back. We pursue this direction in our project by splitting our data into train and test by time, where edges before a certain year are in train, and the test set to predict are edges after that year.

2.2 Hidden factors and hidden topics: understanding rating dimensions with review text

McAuley and Leskovec explore a statistical way to use review text content to create effective recommendation systems. They develop HFT ("Hidden Factors as Topics"), a framework using review text to shed light on the underlying hidden structure of ratings. This framework consists of a latent-factor recommender system used to find a low-dimensional structure of users and items and a Latent Dirichlet Allocation (LDA) model which discovers underlying structure in review text. Once the model is trained, the authors evaluate their methods on various review datasets. As compared with a simple linear baseline, a latent factor recommender system alone, and an LDA-based product topic learner alone, HFT achieves the best performance on 30 of 33 datasets in terms of MSE on rating prediction.

2.3 Finding and evaluating community structure in networks

Newman and Girvan propose a set of community detection algorithms via a top-down approach in which the original graph has edges iteratively removed based on a "betweenness" measure, recalculated after each removal. This "betweenness" favors edges between communities over those within communities. One way to measure this is "shortest-path betweenness", essentially measuring how many shortest paths run along each edge; other ways include random walks and circuit theory. To determine the quality of a given community partitioning, the authors define a 'modularity measure', which measures the fraction of network edges connecting vertices within a single community, minus the expected value of this quantity in a random network with random edges but the same community divisions. This comparison to random ensures strong community structure for higher modularity.

3 Dataset

We use the Amazon Product Dataset [8] aggregated by Julian McAuley of UCSD, which includes 143 million reviews and corresponding metadata across 20+ domains between 1996 and 2014. Relevant fields to our recommendation problem include:

1. product information (product id, product type)
2. user/review information (user id, review helpfulness, product rating)

For a given product, we use the helpfulness of and ratings in its reviews to evaluate its attractiveness. We use the user id to aggregate products a given user has reviewed, hence creating a profile per user. Among the many domains provided, we use the Amazon Instant Video and Amazon Office subsets.

4 Network Methodology

We structure our data as a bipartite graph, where users are connected to products they reviewed (bought) and gave a favorable rating to (≥ 4). To create a set of recommended products for a user, we take the top n products from a ranked set of refined products, where we experiment with $n \in \{5, 10, 20, 30\}$. We explore two approaches to finding a refined product set: community detection (sections 4.1 and 4.2) and leveraging node2vec embeddings (section 4.3),

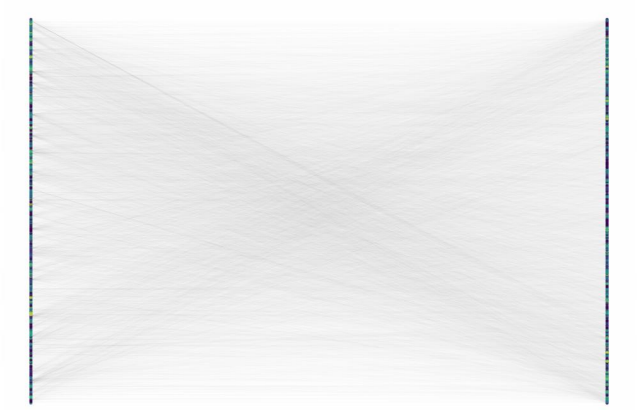

Figure 1: Full bipartite graph of users (left) and products (right) in the Instant Video domain. The full bipartite graph is similarly dense for several other domains.

4.1 Refined Product Set via Graph Folding and Community Detection

In our first approach to creating a refined product set, we:

1. Create a folded graph
2. Perform community detection and filtering

4.1.1 Folded Graph Creation

To create a folded user graph, we define a threshold T and only connect two users if they have co-reviewed $\geq T$ products, where we experiment with $1 \leq T \leq 4$. We found that a threshold value higher than 4 loses too much of the information from the original graph and is incapable of accurate prediction.

We additionally create a graph where edges are weighted by the following Jaccard similarity metric:

$$\frac{\text{(number of co-reviews)}}{\text{(size of the union of products both users have reviewed)}}$$

As the original graph is fairly dense, we hope that this weighting can distinguish between closely related pairs of users, and pairs where one or both users review many products but are not very similar to each other.

4.1.2 Community Detection and Filtering

We perform community detection on the folded user graph to find a cluster of users most similar to each target user. To do this, we use the Louvain Algorithm [1], as the algorithm performs well on large graphs, with $O(nlogn)$ run time. Furthermore, it supports weighted graphs, converges quickly, and produces communities with high modularity, where modularity is defined as

$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$

where A_{ij} is the weight of the edge (i, j), $k_i = \sum_j A_{ij}, m = \frac{1}{2} \sum_{ij} A_{ij}, \delta(u, v) = (u == v)$, and c_i is the community to which node i belongs.

As a point of comparison, we also use the Clauset-Newman-Moore community detection algorithm, as it performs well on large networks and similarly seeks to maximize modularity [7].

Once we have our clusters, given a user u, let S define the set of similar users in the cluster, and p_u denote the set of all products user u has bought and given a rating ≥ 4 (as constrained in the original graph creation). We define the refined set as the union of p_u across users in S, excluding products already bought by user u.

4.2 Recommending from the Refined Set

To choose a set of items to recommend from this refined set, we do the following:

```
set recSetSize (number of products to recommend)
for each product in the refined set:
    f = number of times this item was bought
    hu = product hubbiness (from original bipartite graph)
    for each user review r of this product:
        extract (h_r = helpfulness, p_r = product rating)
    product score = w_1 * 1/f * sum_r(h_r * p_r) + w_2 * log(f) + w_3 * hu

return the |recSetSize| items with highest score
```

The score of a product encodes:

1. Average rating weighted by helpfulness, such that more helpful reviews are trusted more and higher rated products get a higher score
2. Number of reviews, such that popular products get a higher score ($*log(f)$)
3. Product hubbiness, such that products that act more as "hubs" in the original graph should be rated more highly.

We optimize for the best combination of the above 3 considerations with a set of weights (w_1, w_2, w_3) as shown in the product score equation. For each graph folding threshold and recommendation set size, we do a grid sweep over the following range:

$0 < w_1 < 10$
$0 < w_2 < 10$
$0 < w_3 < 500$, step size 50 (0, 50, 100, ..., 450)

The weights for w_3 are an order of magnitude higher because we observe that hubbiness scores tend to be an order of magnitude smaller than the other two terms, and we want to allow all three terms to contribute proportionally.

Our scoring function as defined above is community-agnostic in its rating scheme. To encode community information, we also explore "community-specific" recommendations, where a product's score is calculated based only on reviews from the users in the community who bought the product. We find that "community-specific" recommendations consistently outperform the general case, and thus all reported results are using community-specific recommendations.

4.3 node2vec

We explore two additional node2vec approaches that run parallel to the two-step procedure defined in 4.1 and 4.2. The first uses node2vec embeddings as a proxy for

user profiles, from which a community of users is created per target user based on the similarity of their embeddings. This replaces the community detection method in 4.1, and continues downstream to use the refined set scoring described in 4.2. The second approach uses node2vec embeddings as a proxy for product profiles, and directly uses similarities to embeddings of a target user's past purchases to recommend potential products. In both approaches, node2vec is run on the original bipartite graph using 100 random walks, a p value of 1, and a q value of 2. We select $q > p$ to characterize a microscopic view of the graph, where nodes with similar network roles have more similar embeddings.

4.3.1 node2vec User Similarity

In this approach, the user node2vec embeddings are aggregated into a matrix, row normalized, and the user-pair cosine similarity scores are computed. The similarity scores per user u (each row of the similarity matrix) are then ordered from highest to lowest, and the top 20 most similar users are selected to be u's community. From here, community-specific product scoring as defined in 4.2 is executed with the optimal weight combination found through grid search. Note that though we construct a community for user u, we look for structural similarity in our random walks because we want u to be compared to nodes that share similar roles in the original graph, as opposed to those which are simply proximal (proximity makes more sense in the folded graph).

4.3.2 node2vec Product Similarity

In this approach, the product node2vec embeddings are aggregated into a matrix, row normalized, and the product-pair cosine similarity scores are computed. For each user u, the set of products p_u that u has reviewed (bought) previously is collected. For each product in p_u, we find the top 10 most similar products, and aggregate this across all products in p_u into $p_{u_{recommend}}$. Once $p_{u_{recommend}}$ is constructed, similarity scores for each element of $p_{u_{recommend}}$ are updated to be the sum of similarities to each product in p_u (as opposed to simply a similarity to a single product). $p_{u_{recommend}}$ (excluding products already in p_u) is then ordered by these updated similarity scores, and the highest $|recSize|$ products are returned as the recommendation set for u. Unlike all previously discussed methods, this pipeline directly recommends products using product similarity, rather than first finding similar users and then recommending products associated with those respective user profiles.

5 Evaluation

To evaluate our recommendations, we:

1. Split edges from the original graph into 80% train / 10% val / 10% test
2. Create product recommendations based on information from the train graph. To choose optimal score weights, we perform a grid sweep as described in 4.2, and select the weights that produce the best recommendations on the validation set of edges for the Instant Video graph. All prediction methods uses these weight values.
3. Perform predictions on the test set of edges.

We evaluate on a test set of edges generated through random splitting, as well as temporally based splitting (predict future purchases based on past behavior).

5.1 Metrics of Evaluation

Let $\{s_u\}$ = the recommended products list for user u, per user
and $test$ = the set of user-item edges we would like to predict.

We define the following metrics:

1. Recall: $\frac{|\text{ test edges (u, p) for which p appears in } s_u|}{|test|}$
2. Precision: $\frac{20 * |\text{ test edges (u, p) for which p appears in } s_u|}{|\text{num users in test set * rec set size}|}$
3. F1 Score: $\frac{2 * \text{Precision} * \text{Recall}}{\text{Precision + Recall}}$

Recall measures how often true test user-item relations are discovered by our algorithm, and high recall means that we recover most of the test edges.

Precision measures how many guesses we require to discover true user-item edges, in that a smaller recommendation set with the same number of test edges discovered as a larger recommendation set will get a better precision score. High precision means that our recommendations have high concentration of test edges.

We multiply by 20 in the numerator of precision for multiple reasons:

- This multiplicative factor makes precision and recall the same order of magnitude, allowing a proper F1-score.
- In a test set of all unique users, a recommendation set size of 20 per user that accurately predicted every edge would get a precision of 1 (precision should technically be the min of our expression and 1, but this issue never pops up in our analysis).

Note that we have removed average ranking score since our milestone, because we realized that this metric measures relative preference among reviewed products, which is not encoded in the data we are working with.

5.2 Baselines

To evaluate the significance and efficacy of our results, we compare to the following baselines:

1. Truly Random: For each user, choose a random set of products to recommend.
2. Score Random: For each user, choose a random set of products from that user's community, ignoring the score.
3. Global: For each user, choose based on the globally most popular products, ignoring community structure.

The truly random baseline measures whether our algorithm generally is able to make recommendations, the score random baseline measures if our scoring algorithm works well, and the global baseline measures if our community structure is important.

Table 1: Optimal weights as found by grid search

Rec Size	Weights 1	Weights 2	Weights 3	Weights 4	Weights Weighted
5	(1, 4, 100)	(1, 6, 0)	(1, 7, 100)	(1, 7, 100)	(1, 7, 150)
10	(3, 7, 200)	(1, 9, 200)	(1, 4, 0)	(1, 4, 0)	(1, 4, 150)
20	(1, 7, 200)	(3, 5, 200)	(1, 7, 400)	(1, 7, 400)	(2, 7, 400)
30	(1, 4, 350)	(1, 10, 300)	(1, 7, 350)	(1, 7, 350)	(1, 7, 250)

6 Results

We present our optimization results in Table 1, and recommendation results in Tables 2 and 3.

6.1 Optimization Results

To populate our recommendation results tables, we optimized parameters for all fold values of the instant video graphs, and then fixed these parameters when evaluating on the office products graph and the temporally separated version of the instant video graph.

We observe that a larger recommendation set size tends to correlate with a larger weight placed on the hubbiness of the product in the original graph. This may indicate that hubbiness is only good differentiator among recommendations further down a list.

6.2 Analysis

In tables 1 and 2, we observe that we are able to outperform every baseline across all datasets, meaning our parameter optimization produces meaningful performance and also is able to generalize. Due to the poor performance of node2vec methods relative to the community detection + scoring from refined set methods, node2vec results are not included.

We observe in table 2 that a higher folding threshold leads to a larger outperformance of the global baseline. One possible reason for this is that higher threshold values produce more and smaller communities, allowing community structure to differentiate scoring from a global score list. In addition, we observe that a lower folding threshold leads to a larger improvement over the score random baseline. Lower threshold values have few and large communities, such that score becomes the important differentiating factor.

We additionally measure the effect of recommendation set size on performance. Though recall will always improve with a larger recommendation set size, we see that this will not always be the case for F1-score: for example, Folded 1 achieves a higher F1-score for rec set size of 20 than 30, indicating that the precision drop outweighs the recall gain. However, we also see that Folded3 and Folded4 have very low F1 scores, likely because their low recall values singlehandedly drag down the F1-scores. Future optimization could involve examining if different weightings between precision and recall highlight different characteristics of the folding methods (e.g. the ability of high fold values to outperform the global random baseline).

Table 2: Recommendation algorithm performance on instant video graphs. Improvements are % improvements over the relevant baselines (Glob = Global, TR = True Random, SR = Score Random). The best values for each metric for each recommendation set size are bolded.

Graph Type	Rec Size	F1	Recall	Imp over Glob	Imp over TR	Imp over SR
Weighted	5	**0.1684**	**0.0999**	90.29	3600	**1858.82**
	10	**0.1951**	**0.134**	50.06	1761.11	**1085.84**
	20	**0.21**	**0.1834**	15.93	1960.67	896.74
	30	0.2106	0.2233	12.72	1419.05	**786.11**
Folded 1	5	0.1574	0.0934	75.57	2339.47	1498.28
	10	0.1807	0.1241	38.19	2027.59	849.23
	20	0.2088	0.1824	14.63	**2220.51**	**964.71**
	30	0.2032	0.2155	12.15	976.1	738.78
Folded2	5	0.1592	0.0944	123.17	**3833.33**	1211.11
	10	0.1832	0.1258	71.62	2187.27	957.14
	20	0.2049	0.179	33.28	1588.68	732.56
	30	**0.2142**	**0.2271**	34.86	1522.14	474.94
Folded3	5	0.0684	0.0406	101.99	2800	524.62
	10	0.0834	0.0573	54.03	**5630**	249.39
	20	0.0984	0.0859	**38.33**	1852.27	219.33
	30	0.099	0.105	**35.66**	**2286.36**	210.65
Folded4	5	0.0149	0.0089	**229.63**	2866.67	74.51
	10	0.0204	0.014	**154.55**	1900	70.73
	20	0.0203	0.0177	33.08	1164.29	52.59
	30	0.0212	0.0225	17.8	1223.53	37.2

Though our recall values generally are quite low (never exceeding .22), we are successfully able to outperform baselines in all areas, and generalize beyond the data trained on. Therefore, while our system may not work as a recommendation system on its own, it clearly is able to deduce meaningful connections between users and the products they enjoy.

7 Community Visualization

In order to better understand our graphs, we constructed visualizations using networkx show below in Figure 2. Nodes represent communities, and edges the existence of co-reviewed products between communities. The size of a node is proportional to the number of nodes in a community, and the width of an edge is proportional to the number of edges between two communities. This graph visualization method was chosen because it is infeasible to clearly depict all nodes and edges.

From these depictions, we can gain insight into the structure of our graphs, and consequently intuition as to how well our prediction algorithms perform. We observe that the Folded1 graph with Louvain communities is fully connected, as a low threshold allows for more connections both within and between communities. In comparison, higher threshold graphs display fragmentation, with larger central nodes and many small isolated satellite nodes. The central communities likely represent

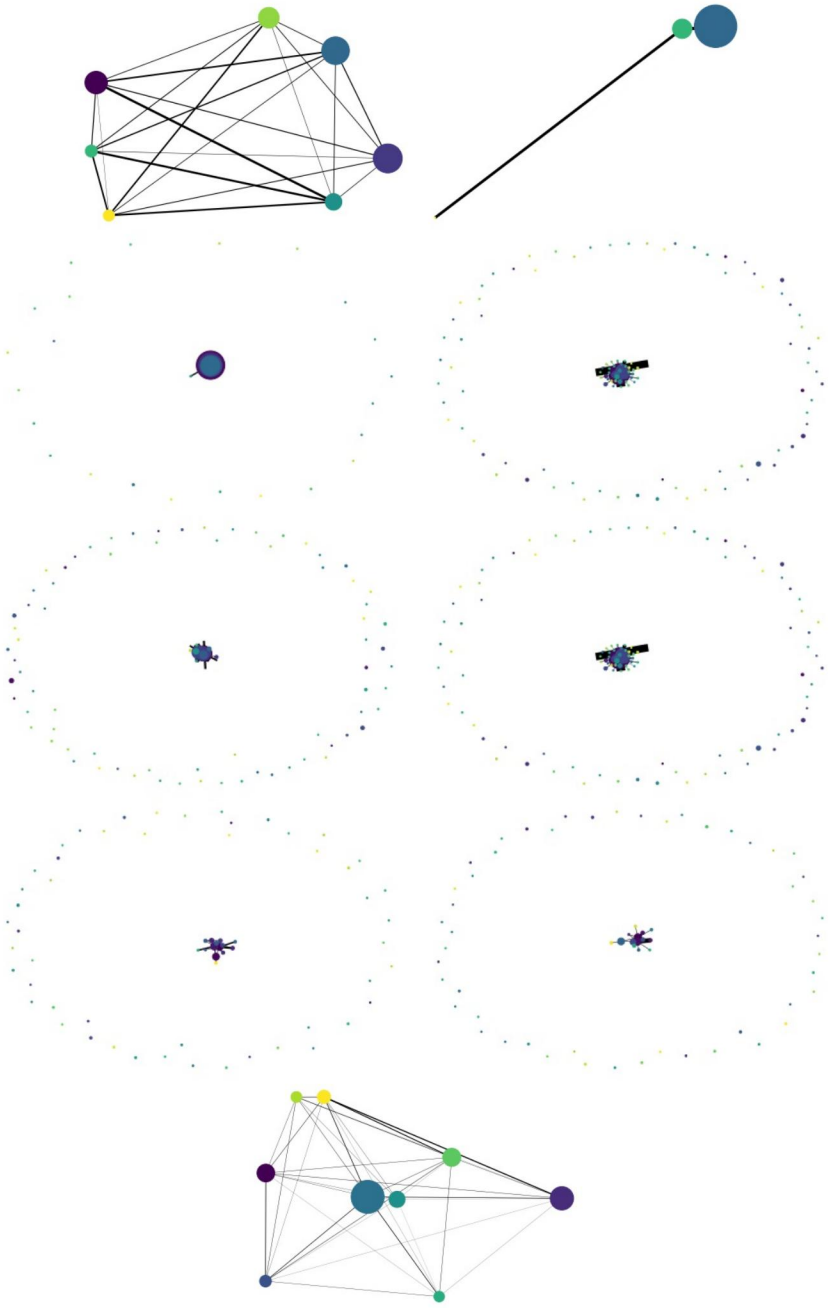

Figure 2: Community visualizations from the Instant Video data. Rows 1-4 depict fold1/2/3/4, the left/right column show Louvain/CNM detection, and Row 5 displays weighted Lovain.

Table 3: Recommendation algorithm performance on temporally separated instant video (TIV) and office products (OP). We dispplay only rec size of 10 and 30 to demonstrate generalization without cluttering.

Graph Type	Rec Size	Recall	Imp. over Global	Imp over SR
OP Weighted	10	**0.0545**	**32.28**	701.47
	30	**0.124**	**26.4**	**588.89**
OP Folded1	10	0.0527	29.17	602.67
	30	0.105	7.91	427.64
OP Folded2	10	0.0501	31.15	**922.45**
	30	0.1156	20.79	564.37
TIV Weighted	10	**0.013**	**550**	**154.9**
	30	**0.0365**	**38.78**	88.14
TIV Folded1	10	0.0123	156.25	123.64
	30	0.0338	28.52	**148.53**
TIV Folded2	10	0.0048	9.09	140
	30	0.0232	6.42	118.87

groups of users who review many popular products. some central communities have extremely high connectivity as evident by the thick edges. Satellite nodes can best be thought of as users who only reviewed the same niche products.

Supporting our visual observations, in table 4 we observe that the 1-folded and weighted graphs have large communities (>200 users, >450 products); in comparison, 2/3/4 folded graphs have many 2-user communities, and include some communities with large number of products as in 1/weighted, but also communities with a more desirable number of products (10-100). Additionally, modularity appears positively correlated with fold threshold, but independent from the number of communities.

Table 4: Community Structure for instant video and office product graphs. Q is the community modularity; |C| is the number of communities; U Extrema (P Extrema) are the minimum and maximum number of users (products) per community across all communities; CC is clustering coefficient.

Graph	Fold	Nodes	Edges	Q	|C|	U Extrema	P Extrema	CC
inst_vid	1	5054	609822	.382	7	[193, 1631]	[456, 1370]	.503
	2	4124	56885	.608	31	[2, 819]	[3, 911]	.492
	3	1709	5984	.721	88	[2, 254]	[4, 493]	.343
	4	426	643	.8	56	[2, 60]	[5, 394]	.218
	weighted	5054	609822	.399	8	[256, 1866]	[454, 1431]	.503
off_prod	1	4878	601018	.208	5	[224, 2025]	[850, 2078]	.411
	2	3703	81357	.238	66	[2, 763]	[4, 1468]	.417
	3	1829	17704	.289	46	[2, 394]	[6, 1247]	.408
	4	876	5231	.357	20	[2, 186]	[7, 967]	.420
	weighted	4878	601018	.214	5	[89, 2048]	[430, 2072]	.411

8 Future Work

There are multiple future directions that this work could take:

1. As user-product recommendations are a form of link prediction, we could use a stochastic block model (SBM) to predict the emergence of links given an initial graph. This would particularly be applicable in the temporal case, as the SBM could use the current state to predict the future graph state.
2. The current bipartite graph is dense; additional techniques beyond the current thresholding/weighting to eliminate less meaningful edges could involve network deconvolution, by which we separate direct and indirect connections.
3. Similarity metrics could incorporate review text. We implemented a preliminary version, in which a similarity score between two users was calculating by finding the max/avg cosine distance between all pairs of respective user reviews with stop words removed. This method only slightly improved performance, and more complex methodologies could be explored.

References

[1] Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." Journal of statistical mechanics: theory and experiment 2008.10 (2008): P10008.

[2] Kannan, Ravi, Santosh Vempala, and Adrian Vetta. "On clusterings: Good, bad and spectral." Journal of the ACM (JACM) 51.3 (2004): 497-515.

[3] McAuley, Julian, and Jure Leskovec. "Hidden factors and hidden topics: understanding rating dimensions with review text." Proceedings of the 7th ACM conference on Recommender systems. ACM, 2013.

[4] Newman, Mark EJ, and Michelle Girvan. "Finding and evaluating community structure in networks." Physical review E 69.2 (2004): 026113.

[5] Shams, Bita, and Saman Haratizadeh. "Graph-based collaborative ranking." Expert Systems with Applications 67 (2017): 59-70.

[6] Yu, Fei, et al. "Network-based recommendation algorithms: A review." Physica A: Statistical Mechanics and its Applications452 (2016): 192-208.

[7] Clauset, Aaron, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks." Physical review E 70.6 (2004): 066111.

[8] McAuley, Julian. 2014. Amazon Product Data: http://jmcauley.ucsd.edu/data/amazon/