

CS6850 Final Report - The Construction of Recommendation Networks Using Network-Based Inference

Jasdeep Hundal
jsh263@cornell.edu

Sohan Jain
sj346@cornell.edu

Harry Terkelsen
het32@cornell.edu

May 13, 2011

Abstract

Many websites put significant effort in setting up recommendation networks for their users to discover new items which may interest them. These recommendations are based on knowing what sorts of things the user already likes. From real-world data sets, one can build a graph between users and objects, where an edge between a user and object means that the user interacted in some way with the artist. This bipartite graph can be compressed into a graph consisting just of objects, in which objects have links if they are similar. This graph can be interpreted as a recommendation network, and we look at some ways to generate the most effective recommendation network. Typically, a method called collaborative filtering is used, but there do exist other algorithms for this purpose.

In this paper, we evaluate the performance of one of these algorithms, known as network-based inference, against Amazon.coms well-known item-based collaborative filtering approach. We test the two basic algorithms, along with modified versions of network-based inference, on two data sets. Specifically, we focus on a music network in which users download songs from artists. We then provide possible explanations for the performance differences. Ultimately, we find a modification to network-based inference that performs better than all other examined algorithms.

1 Introduction

As a consumer, you conceivably want to discover new things (music, food, gadgets) that you have a good probability of liking. For many web businesses, such as Amazon.com, giving useful recommendations to users increases the likelihood that the users will make a greater number of purchases, thereby improving their business. The process of making these recommendations is certainly nontrivial. Is it enough to recommend similar objects that the user has heard of before? By what dimensions should you define similarity?

In general, companies like Amazon.com or a music distribution hub like Yahoo! Music use recommendation networks to help their users discover items they like. In a general recommendation network, a node represents some sort of object, like an artist or a song, and a directed edge from X to Y translates to X is similar to Y . Note that Y is not necessarily similar to X . Another way of phrasing the directed edge is, if you like X , then you will like Y . The weight of this edge should be a function of the artists similarities. Based on such a network, one could examine a particular users past interactions with objects and recommend to them artists they can be expected to like.

There are two major ways to construct these networks: using human experts or using algorithms. Human experts can identify the most similar artists and draw connections between them, and are particularly good at connecting users to more obscure artists that the users will like but would not have discovered without the experts help. However, the range of a human experts knowledge is limited (e.g., the expert is limited to a single genre for a music network), and constructing a network for thousands of objects in this way would be a cumbersome process. For those reasons, many companies choose to use computer-generated recommendation networks.

The two goals for any algorithm used to create recommendation networks are to be fast and to reliably return individual recommendations for each user to which they will enjoy listening. The reason the algorithm must be fast is that the recommendations need to be updated quickly when presented with new information (such as a user downloading a song and indicating that they like it), and it must be accurate to give the user incentive to make a purchase, spend more time on the website, or whatever else would benefit the business.

The data that the algorithm has at its disposal is usually a record of user actions on the website. This data could simply be a a binary value for each object indicating whether or not a user interacted with object. It could also possibly indicate how many times the user has interacted with the object, or it could describe the rating that the user assigned to the object. In all cases the input to the algorithm is a matrix with a row for each object and a column for each user, which could be flipped and is usually sparse, that indicates some value for each user-object combination. Generally, the procedure does not analyze any inherent properties of the object, such as a songs genre or content, as a human expert would likely do. Rather, the procedure just analyzes how users have already interacted with objects.

This set of vectors can in fact be considered as a bipartite graph with the objects on one side and the users on the other. A connection is made from a user to an object if the value v for the user-object connection in the object vector is nonzero, and the edge is given the weight v . Given this bipartite graph, the goal of a recommendation network algorithm is to project it onto a one-mode object network. This network may in fact never be explicitly constructed (and usually is not for computational ease), but this is the essential operation of recommendation algorithms.

There are many algorithms that exist for this, but the class that is most widely used is that of collaborative filtering algorithms, which come in two varieties: user-based and item-based. User-based collaborative filtering items recommend artists to users by looking for what similar users listened to. Item-based collaborative filtering (IBCF, or just CF) recom-

mends objects that are deemed similar to objects with which a user has already interacted. This latter algorithm was developed by Amazon.com and is used in its recommendation engine.

As IBCF is well-known and well-regarded, we will use it as a standard to evaluate the effectiveness of a newer algorithm for collaborative filtering, known as network-based inference. Like IBCF, it calculates similarity between objects to make recommendations, but there are significant differences between the algorithms and we would like to determine if either is conclusively better than the other.

2 Item-Based Collaborative Filtering

Item-based collaborative filtering seeks to predict a users preferences by examining the objects that a user has interacted with in the past and then recommending objects similar to those that they have liked.

2.1 Overview

For each object, the algorithm is given a list of users that have interacted with the object, along with a value for each user. This value could be a variety of things, including a simple binary indication that the user has interacted with the object, the number of times the user has interacted with the object, or the rating the user has assigned the object. For each object pair, IBCF examines the lists for both objects and computes a similarity score. Then, when generating a recommendation for a user, IBCF adds together the object similarity lists that have been computed for each object the user interacted with, perhaps weighted by the interaction value. It then recommends the objects with the highest similarity scores in the sum of lists that the user has not already interacted with.

2.2 Calculating a Weight Matrix and Making Recommendations

In our implementation, we framed the above through matrix/vector operations, though sparse matrices/vectors were used for speed. A more precise explanation of IBCF as we implemented it follows. It is framed about as a music recommendation network, as we will be examining one such network later in the paper. Users can download from particular artists, so an “artist” is comparable to an object in our previous formulation.

Suppose we will construct a matrix D from raw collected data, which has n columns, one for each artist, and m rows, one for each user. D_{ij} represents the interaction value for user i and artist j . $D_{ij} = 0$ if i and j have never interacted, otherwise we use the given interaction value. For our implementation, this value is always 1 if the user downloaded from an artist.

The first step in the algorithm is to calculate the artist-to-artist similarities to construct a similarity weight matrix W . This matrix has n rows and n columns. For each artist j , we compare it to every artist k , including j itself. Let a_j represent the column j in D . We

set W_{jk} to a value according to a comparison function $f(a_j, a_k)$. In our implementation, we calculate f , the cosine between the vectors (as suggested by Amazon):

$$f(a_j, a_k) = \frac{a_j \cdot a_k}{|a_j||a_k|}$$

After setting all W_{jk} , we normalize the matrix across its columns (we continue to use columns to represent artists, which has the effect of giving each artist equal weight in future computations). This is not typical and takes longer to run on our data sets, though its worst case runtime is no worse than the expected runtime of computing the weight matrix. We did find that it performed better on our data sets, which will be explained later in the paper.

Precomputing the matrix W will allow us to quickly and efficiently calculate recommendations on request. When we need to make a recommendation for user i , we take a weighted sum of the vectors in the W corresponding to the artists that have a nonzero value in u_j , where u_j is row j in D (in our case, the nonzero value is always 1). We can frame this as a matrix computation which results in the recommendation vector R .

$$W \cdot u_j = R$$

Now, the value R_j indicates the recommendation score for artist j . We order the artists by this score, tossing out those for which the user already has some interaction value. This gives us a list of artists in the order they will be recommended to the user [1].

3 Network-Based Inference

Network-Based Inference (NBI) is a recommendation algorithm that uses a one-mode projection of the bipartite user-artist graph to create personal recommendations. The algorithm was discovered by Zhou et al. (2007) [3].

3.1 Overview

The following discussion is also framed around a music network, with users and artists, but it can be generalized by replacing “artists” with “objects”. The algorithm works by first projecting the bipartite user-artist graph onto a graph where the only nodes are the artist nodes. In Network-Based Inference, the new graph has weighted directed edges of the form $e(i, j)$ where the weight on the edge is meant to represent the “similarity” between node i and node j . As Zhou et al. point out, previous one-mode projections have used the number of users that i and j are both connected to as the edge weight, but this has been shown to lead to biased results in predicting the original bipartite graph. Also, most previous edge weighting methods have the property that $w_{ij} = w_{ji}$, meaning that the similarity of item i to item j is the same as the similarity of item j to item i . Clearly, this is not always the case, as it is possible for a relatively unknown artist to be most similar to a very popular artist, but the popular artist is not necessarily very similar to that unknown artist.

The weighting method proposed by Zhou et al. is as follows:

1. Initialize every artist with unit resource.
2. Have every artist distribute its resource evenly to every user that has an edge to that artist in the user-artist interaction graph.
3. Have every user evenly distribute its resource back to the artists it has an edge to in the user-artist interaction graph.
4. For every artist i , w_{ij} is the amount of final resource that artist i has that came from artist j .

We now have a weighted directed graph consisting of just the artist nodes. Note that the weight matrix is column-normalized since every artist started with unit resource and that $w_{ij} \neq w_{ji}$ generally.

We can use this graph to construct personal recommendations for a user u . In the original NBI algorithm, this is done by giving every artist that user u has interacted with unit resource. Using the weight matrix computed above, one can determine the final resource that each artist j will have if only the artists that user u has interacted with are given initial resource. To make the recommendation list we retrieve every artist the user has not yet interacted with and we sort them in decreasing order of final resource.

3.2 One-Mode Projection of the User-Artist Interaction Graph

In this section the bipartite graph projection is explained in more detail.

Consider the User-Artist Interaction graph $G(X, Y, E)$. The node set X represents the artists, and $|X| = n$. The node set Y represents the set of users, and $|Y| = m$. E represents the edge set.

Let $k(x_j)$ be the degree of node x_j and let us define A , the adjacency matrix of size $n \times m$, where a_{ij} has a value of 1 if user j has interacted with artist i , and has a value of 0 otherwise.

Also, in our discussion of NBI above we initially assigned every artist unit resource. However, we are not forced to give each artist an initial resource of 1. Let us define $f(x_j)$ to be the initial resource on artist j .

Then in the first step of NBI, every artist distributes its resource evenly amongst all users who have interacted with it. Thus, the resource on user j after the first step is given by the formula:

$$f(y_j) = \sum_{i=1}^n \frac{a_{ij} f(x_i)}{k(x_i)}$$

Now, all the resource flows back from the users to the artists. Each artist distributes their resource uniformly among all artists it has interacted with, and so now let us define $f'(x_k)$ to be the final resource on node x_k . Then we have:

$$f'(x_k) = \sum_{j=1}^m \frac{a_{kj}f(y_j)}{k(y_j)} = \sum_{j=1}^m \frac{a_{kj}}{k(y_j)} \sum_{i=1}^n \frac{a_{ij}f(x_i)}{k(x_i)}$$

We can rewrite this in terms of our weight matrix W :

$$f'(x_i) = \sum_{j=1}^n w_{ij}f(x_j)$$

Where we have defined w_{ij} as:

$$w_{ij} = \frac{1}{k(x_j)} \sum_{k=1}^m \frac{a_{ik}a_{jk}}{k(y_k)}$$

The weight matrix W gives the weight of all the directed edges in our one-mode projection onto artists. And w_{ij} represents the final resource on node i that came from node j [3].

3.3 Personal Recommendation

Once we have created the weight matrix above, creating a recommendation list for a user i is quite simple using the algorithm described in the NBI overview. We simply give every artist that the user has interacted with an initial value of 1, and we give every other artist an initial resource of 0. We can define an initial allocation vector \mathbf{f} where \mathbf{f}_j represents the initial resource on artist j . Then, to find the final resource on artists, we can simply perform the matrix-vector multiplication $\mathbf{f}' = W\mathbf{f}$. Now the vector \mathbf{f}' holds the final resource allocation for all artists, and we can simply sort all artists by final resource and return the sorted list after filtering out already collected artists.

4 Related Work

Zhou et al. introduced the network-based inference algorithm. Recall that a one-mode graph has one set of nodes, as opposed to a two-mode graph like a bipartite one. Zhou et al. improved upon the projection of a two-mode graph onto a one-mode graph by adding a weighting between edges in the one-mode graph. To clarify, consider the user-artist bipartite graph in music. The one-mode projection of this graph is the network where nodes are artists and the edges signify similarity between artists. The one-mode network clearly loses information from the original graph, but by adding weights to the edges, Zhou et al. encoded additional information, such as the magnitude of similarity between artists [3].

Zhou et al. are the original creators of network-based inference and furthermore showed how to create recommendation lists for users based on the final weight matrix of the recommendation network. To test the performance of network based inference, Zhou et al. empirically tested the recommendation network constructed by NBI on the MovieLens (www.grouplens.org) dataset. The MovieLens dataset has 1682 movies, or objects, 943 users, and 100,000 ratings on a scale of 1-5, which define an interaction between a user and

a movie; for sake of comparison to music recommendation networks, we will use the terms “object” and “artist” interchangeably. In the bipartite user-object graph, there is an edge between a user and object if the user gave the movie a rating ≥ 3 , which signifies that the user actually had a neutral/positive interaction with the object. The resulting graph has 85.25% of the original edges. They then partition the graph into a training set of the data from which to build the recommendation network, and a probe set on which to test the recommendations.

Zhou et al. compare the performance of NBI to two other recommendation networks: one generated by collaborative filtering and one generated by a global ranking method (GRM). The method of collaborative filtering is described above. The global ranking method recommends the same object list to each user, and that list of artists it those the user has not already seen, sorted in descending order of degree. A recommendation list generated by the GRM is similar to Billboards Top 100 Music Hits.

NBI gave the best recommendations of the three algorithms, according to tests by Zhou et al. Users in the probe set on average collected objects higher on the recommendation lists generated by NBI than by the other two algorithms. We will discuss the exact methodology for comparison later in the paper. Furthermore, NBI has about the same run-time complexity as CF. Let k_u denote the average degree of users and k_o denote the average degree of objects in the user-object bipartite graph. Let m be the number of users, and n be the number of artists. Then CF has a running time of $O(n^2k_o)$. NBI has a run-time complexity of $O(mk_u^2 + mnk_u)$, where the first term accounts for creating the weight matrix, and the second term accounts for generating the final weights. Because clearly $k_u^2 < nk_u$, NBI's runtime is $O(mnk_u)$. Note that in most cases in real life networks, the number of users can be orders of magnitude more than the number of objects, and we will look at such a data set later on. In general, CF looks at similarity among users, while NBI just looks at similarity among artists, so NBI should run faster if the number of users is magnitudes of orders greater than the number of artists.

However, the MovieLens data set is in general rather small and feels somewhat contrived. There are more objects than users, which again is unlikely in real-life recommendation networks like the music one, in which users should outnumber artists. Furthermore, each user has rated at least 20 movies, so there is a lot of information about each user, which may be missing in other real-world networks. In fact, users with less than 20 ratings or those with incomplete demographic information were sanitized from the data set. In this paper, we will look at another real-world data set to see if network based inference can be adapted to the general case of networks.

Liu et al. (2009) extended NBI to encode even more information in the weight matrix [2]. Recall that constructing the weight matrix involved use of the $n \times m$ user-object adjacency matrix A , in which $a_{ij} \in A$ was 1 if object i was collected by user j and 0 otherwise. However, as in the case of the MovieLens data set, we often can assign more than just a binary value for the weight of the relationship between object i and user j . In the instance of the MovieLens data set, a user j collects an object i if he gives it a rating ≥ 3 , in which case $a_{ij} = 1$. Instead, suppose we let a_{ij} be the rating the user gave. Then we are also encoding the magnitude

to which the user liked the object. In terms of the method of NBI, we are giving additional initial resource to the j th user node after the first step of distributing resource (i.e., when the resource flows from the objects to the users), because the i th object node that is giving them resource has additional weight. Intuitively, because we are encoding more information about users preferences, we would expect better recommendations.

To clarify, suppose we have the “weighted” adjacency matrix R , where r_{ij} is the weighting of the edge $e(i, j)$ if the edge exists, and 0 otherwise. In the MovieLens data set, r_{ij} is the rating that user j gives to object i . Let weighted network-based inference (NBI_w) be the network-based inference algorithm modified to use the weighted adjacency matrix R . Liu et al. found that the recommendation lists built by NBI_w performed better than those constructed by CF and unweighted NBI. The computational complexity of the weighted and unweighted versions of NBI are the same, and so one should use additional information about the magnitude of preference when a user collects an object when that information is available.

However, not every user-object graph will necessarily have more than a binary relation between a user and an object as given by the adjacency matrix. For example, consider a music site in which users can only download songs and not rate them. Then we can still build the bipartite user-object graph, where an edge $e(i, j)$ signifies that the user j collects, or downloads, song i . However, it is hard to give a weight other than a binary value to the edge $e(i, j)$. In this paper, we will modify NBI to account for situations like this one and even try to improve upon the performance of NBI by looking at other weighting methods.

5 Modifications

5.1 Data sets

We make two significant additions to the procedure used by Zhou et al; the first of these is that we test the ICBF and NBI algorithms on a much different data set than MovieLens. We use data collected from Hulkshare (www.hulkshare.com), a file-sharing website which focuses on hip hop music. The data is in the form of a list of downloads, with each entry giving the name and artist of a song, and an IP address that requested the download. We designate each unique IP as a user, and since the data reflects a very short time frame this measure of a user is fairly valid. One major difference between this data and the MovieLens data is that there are far more users than artists in the Hulkshare data. We feel this better reflects real world user-object interactions. For example, there are many artists and films out there, but the number of people listening to music and watching movies is far greater. Another potentially significant difference is that there are no ratings in the Hulkshare, and hence no way to confirm that the downloading user liked the song. But on a site like Hulkshare, users are searching for specific artists or songs, so the vast majority of downloads are of music that the user will like.

5.2 Resource Allocation

The second addition we made was that we ran several modifications of the NBI algorithm. Each modification involved setting the initial resource in a different manner, as Liu et al. (2009) [2]. We chose to run unweighted NBI along with five different initial resource allocations for each artist node. These are described below, with the following notation: $f(j)$ is the initial resource allocation function, d_{ij} is either 0 or 1 depending on if the user i interacted with artist j , q is the number of interactions the user had with the artist, and $k(j)$ is the number of users that have interacted with artist j .

- $f(j) = d_{ij} \cdot q$: Each artist has an allocation of resources that is equal to the number of times that a user has downloaded from the artist. We thought this would improve regular NBI due to the the artists that the user likes receiving more weights in the final calculation.
- $f(j) = d_{ij} \cdot \ln(q)$: Each artist has an allocation of resources that is equal to the log of the number of times that a user has downloaded from the artist. The natural log calculations is another experiment designed to make each additional download of an artist worth less to the user. This is done to try to give a more diverse artist selection by preventing any single artist from dominating the recommendation calculation.
- $f(j) = d_{ij} \cdot q^2$: Each artist has an allocation of resources that is equal to the square of the number of times that a user has downloaded from the artist. We thought this would improve regular NBI, because the artists that a user likes receives more weight in the final calculation. However, this initial resource would probably improve regular NBI less than the first algorithm presented since this one seemed likely to give too much weight to artist downloads.
- $f(j) = d_{ij} \cdot k^\beta(j)$, with $\beta = 0.5$: We gave each artist an allocation of resources that is equal to the square root of the number of times that the artist has been downloaded across all users. The square root lowers extreme values. The intention of this was to examine the influence of artist popularity. This weighting would perform better if there were a few artists that were downloaded significantly more than all others. We expected it to perform on par with IBCF and unweighted NBI.
- $f(j) = d_{ij} \cdot k^\beta(j)$, with $\beta = -0.5$: We gave each artist an allocation of resources that is equal to the inverse of the square root of the number of times that the artist has been downloaded across all users. Unlike above, the goal here was to bring obscure artists to the forefront. The users may in fact like these better, but we expect that the quality of recommendations found by this scheme will be lower than average.

6 Methodology

We will now explain how we measured the performance of each recommendation network algorithm. First, for either data set, we partition available data into a training set and a

probe set. At a high level, the recommendation lists are generated by the training set and tested on the probe set for accuracy.

The MovieLens data set consisted of 943 users and 1682 movies, or objects, with 100,000 ratings by users for movies. A user collected a movie if he gave it a rating ≥ 3 , to show some degree of liking. So the bipartite user-object graph ended up with 85,250 edges. We said that the top five rated artists for each user constituted the probe set, and we broke ties between ratings arbitrarily. All other data made up the training set.

We sanitized the Hulkshare data set a bit to ease computation without loss of generality. We removed all artists who only had one download, because by network based inference, they could not be deemed similar to any other artist and would have no use in recommendations. For the training set, we looked at the downloads during March 1, 2011. The resulting user-object (where artists are objects) bipartite graph had 199,782 users, 14,926 artists, and 649,268 edges, where an edge $e(i, j)$ means that user j downloaded a song by artist i . Our probe set consisted of data from March 2, 2011, and we randomly sampled 1000 users who downloaded songs on both days for ease of computation. The probe graph consisted of the bipartite graph of these probe users and artists from whom they downloaded songs. There were 4,642 edges in the probe graph.

From the training set, we sought to build recommendation lists for each user. A recommendation list is a list of objects that the user has not yet collected, sorted in descending order of likeliness for preference. That is, the first element of the list should be the object that is most recommended for the user, or the one that the user is most likely to like the most. All of the algorithms are able to produce such a recommendation list for a user.

To compare the effectiveness of the recommendation lists built by each algorithm, we will look at the “ r -value”, which was a metric used by Zhou et al. The r -value is defined as follows. For some user u_j , consider an object he collected in the probe graph o_i that the user has not already collected in the training graph; i.e., the edge $e(i, j)$ exists in the probe graph, but does not exist in the training graph. Look at the position of o_i in the users recommendation list. The r -value for this edge is the position of the object in the users recommendation list divided by the length of the recommendation list. For example, if a user in the probe graph downloaded a song he never downloaded previously, and that song was the third highest recommended song on his recommendation list of length 12000, then that edges r -value is $3/12000 = .00025$. The r -value for an algorithm is the average r -value for all the edges in the probe graph that meet the above criteria.

Intuitively, you can think of an r -value of a measure of how well the recommendation list predicts a users next object collection. Therefore, it is a measure of the effectiveness of a recommendation network. A lower r -value means that the recommendation list was more effective, because the recommendation list suggested a song that the user would have a higher chance of liking more.

We compared the r -values for NBI, CF, GRM, and each of the above modifications to NBI to try to find the most effective algorithm for computing recommendation networks. We tested the old algorithms (GRM, unweighted NBI, CF, and the weighted NBI by Liu et al.) first on the MovieLens data set to show that our findings were consistent with Zhou

et al. and Liu et al. Next, we compared the r -values for each algorithm on the Hulkshare data set to test the algorithms effectiveness on another real-world data set and to try to find improvements to NBI.

7 Results

The r -values for each algorithm are detailed in Table 1. The modifications to NBI are described by what their initial resource $f(j)$ is.

Data set	GRM	CF	Unweighted NBI	$d_{ij} \cdot q$	$d_{ij} \cdot \ln(q)$	$d_{ij} \cdot q^2$	$d_{ij} \cdot k^\beta(j), \beta = .5$ (JTH-NBI)	$d_{ij} \cdot k^\beta(j), \beta = -.5$
MovieLens	0.15335	0.089451	0.13964	0.12261	0.12227	0.13019	0.086697	0.68041
Hulkshare	0.16336	0.23593	0.27830	0.29886	0.41490	0.336726	0.19688	0.68228

Table 1: The r -values for each algorithm and data set. A lower r -value signifies that the algorithm produced a more effective recommendation list. In the MovieLens data set, JTH-NBI gave the lowest r -value. In the Hulkshare data set, GRM gave the lowest r -value, and JTH-NBI did second best.

We also used as a performance metric the “hitting rate” of the recommendation list. The hitting rate for a recommendation list of length k is defined as the proportion of objects that a user in the probe set collects that are in that users recommendation list. So, for example, if we restrict the size of the recommendation lists to be length $k = 100$ then if 75% of the artists that a user downloads in the probe set are within the first 100 artists in that users recommendation list, then the hitting rate is 0.75. We graph the hitting rate for each algorithm and data set for all lengths k in Figure 1.

8 Discussion of Results

In general, the modification of NBI that does best is when the initial resource is $f(j) = d_{ij} \cdot k^\beta(j)$, with $\beta = 0.5$, which we will refer to as the Jain-Terkelsen-Hundal modification to NBI, or JTH-NBI for short. This version of NBI does better than Liu et al.s modification and also better than standard item-based CF. In the MovieLens data set, this algorithm performed best. Not only did it have the highest average hitting rate, but also it provided the best recommendations.

JTH-NBI gives extra weight to popular artists, i.e., those with more edges, and the rate at which it gives addition weight is decreasing the more popular the artist is. This factor takes the edge of the scaling, so to speak. Because this algorithm produces the best recommendations, we can say that users are more likely to interact with popular objects, and this should be factored into recommendations. However, it is not the biggest factor, because GRM, which is entirely based on artist popularity, actually performs worst.

In the Hulkshare data set, we found that the global ranking method actually produced the best results, and our version of NBI performed second best. We think that GRM did best due to the nature of Hulkshare.com itself. The structure of the site encourages people to download the most popular songs, and there is no real venue for discovery of other songs.

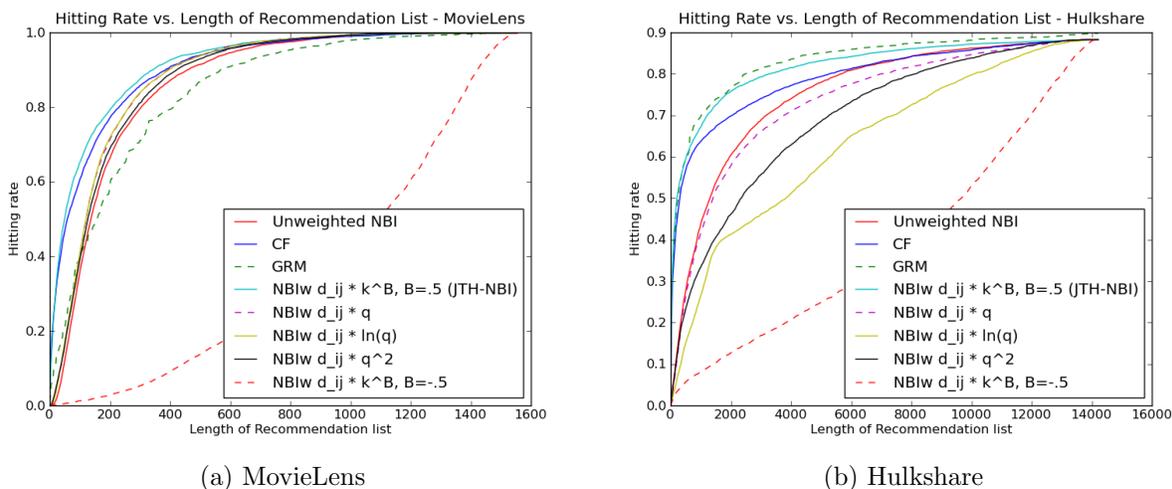


Figure 1: The hitting rate for each data set and algorithm. The hitting rate signifies what percentage of objects a user collected in the probe graph were in their recommendation list. In the MovieLens data set, the JTH-NBI algorithm gave the highest hitting rate. In the Hulkshare data set, the GRM algorithm gave the highest hitting rate, and JTH-NBI second highest.

Since GRM performed best for the Hulkshare data set, we have shown that JTH-NBI may not perform best in every case. Even in the Hulkshare data set, though, it performed second best. The point is that the best type of recommendation algorithm ultimately depends on the nature of the underlying user-object bipartite graph. If most users just have incentive to download the most popular items, then intuitively GRM will produce the best recommendation lists. However, if the site exposes users to many objects, as is the case with sites like Amazon.com, then a method like the JTH-NBI will produce the best performing recommendation lists. Ultimately, the more popular artists should be recommended more, as users are most likely to want to interact with them.

It should be noted that β is a mutable parameter of our algorithm, and it can be tuned to better fit a particular dataset. Anyone who wanted to employ our algorithm to create recommendation lists could tune β to fit their user-object interaction graph and give them the best results.

9 Conclusion

In this paper, we looked at the projection of a two-mode user-object bipartite graph onto a one-mode network via the network-based inference algorithm by Zhou et al. From the one-mode network, we generated recommendation lists, which for each user suggest an ordered list of uncollected objects with which that user should next interact. We examined three core algorithms for producing the recommendation lists: network-based inference, collaborative

filtering, and the global ranking method. We experimented with modifications to the initial resources placed on objects in the NBI algorithm. After generating the recommendation lists, we computed an r -value as an indicator of performance; an algorithm that produced a lower r -value gave better recommendations. Our results on the MovieLens data showed that the JTH-NBI produced the lowest r -value and had the highest hitting rate among all algorithms; thus it is an improvement upon CF, Zhou et al.s unweighted NBI, and Liu et als linearly weighted NBI. Although JTH-NBI accounts enough for artist popularity on the Hulkshare data set, the global ranking method actually produced the best recommendation lists, but this phenomenon occurs because of the nature of Hulkshare.com, in that users are most often only exposed to popular artists on the site. They are seeking specific songs and they have no options for exploration on Hulkshare.

We have shown that the performance of a recommendation algorithm is largely dependant on the dataset on which it is evaluated. In future work, one could evaluate these algorithms on other datasets, such as the Netflix dataset, which is a popular dataset for recommendation algorithms. In addition, one could also study ways to find an optimal β parameter for the JTH-NBI algorithm for their dataset. Furthermore, we would like to evaluate these algorithms on metrics other than the r -value. It does a decent job at determining the effectiveness of recommendation algorithms; we can see this as Amazons ICBF algorithm, which is strong in r -value tests, performs well for Amazon in practice. However, the goal of recommendation networks is to inform about what a user will think of an object that he has not interacted with before. r -values can perform relative comparisons between algorithms well, but this does not necessarily mean that low r -values indicate that the recommendations were as good as possible for the user, especially as the length of lists that are returned grows.

References

- [1] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.
- [2] J. Liu, M. Shang, and D. Chen. Personal recommendation based on weighted bipartite networks. In *Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. Sixth International Conference on*, volume 5, pages 134–137, 2009.
- [3] T. Zhou, J. Ren, M. Medo, and Y. Zhang. Bipartite network projection and personal recommendation. *Physical Review E*, 76(4), 2007.