Recommendation Systems for the Netflix Network

Aleksander Dash, Jason Lin, Nolan Handali
{adash, jason0, nolanh}@stanford.edu

December 2018

Abstract

In this project, we explore creating a recommendation system for Netflix users in a variety of ways, such as item-item and user-user collaborative filtering, as well as various ways to create projections to aid with recommender systems. Of all the methods tested, a hybrid approach involving a graph projection of movies based on the Pearson Correlation coefficient has the best performance, followed by user-user collaborative filtering. We also explore multiple ways of clustering nodes together that have ratings that correlate with each other, which could provide useful external information to improve the accuracy of future recommender systems.

1 Introduction

Recommendation systems and link prediction are classic problems when a wealth of content is provided to users and one needs to know what content users have not yet seen but are most likely to enjoy. They are some of the most common problems seen in today's world. Recommender systems are used to suggest friends on Facebook, products you might like on Amazon, the next video on Youtube, more shows on Netflix, etc.

Our project is to build a recommender system for users based on the Netflix network, which is a bipartite graph consisting of users and movies, with edges between movies that users rated (on a discrete scale from 1 to 5). Specifically, we will try to recommend movies a user has not seen that they may enjoy, based on the previous movies they saw and rated. We plan to use the methods from the papers we surveyed as a baseline and improve upon them, by taking advantage of the unique structures of our network. For instance, there is some granularity to the ratings users give movies instead of a binary recommend or not

2 Prior Work

A recommender system is a system that attempts to predict the rating a user would give an item, and suggest items that the user would enjoy, whether this be movies, books, music, or people. They have wide ranging usages, and show up from our Google searches, to our friends on Facebook.

In this section, we will discuss several papers related to recommendation system in order to understand what they discuss. We present a brief summary of the paper, and then a critique of the papers, leading into a discussion on directions for further exploration, and a brainstorm on the extensions we can pursue.

2.1 Analysing Bipartite Graphs

There has been significant work[1] that deals with projection of bipartite graphs, while being able to maintain information about the original graph, as a naive approach results in an undirected, unweighted network that loses significant information.

Tao Zhou et al.[1] deals with a better method of projection of a bipartite graph that invovles a weighting method that can be applied to extract the hidden information of networks, leading to a personal recommendation. In their approach, given a bipartite graph $G(X, Y, E)$, they assign each node in X (or Y, depending on which you are projecting) a weight, which then "flows" to the neighboring nodes in equal weight. Then, this weight flows back to the original set, giving it its weight in the projection graph. **Figure 1** shows how weight flows, leading to the final weight of the nodes.

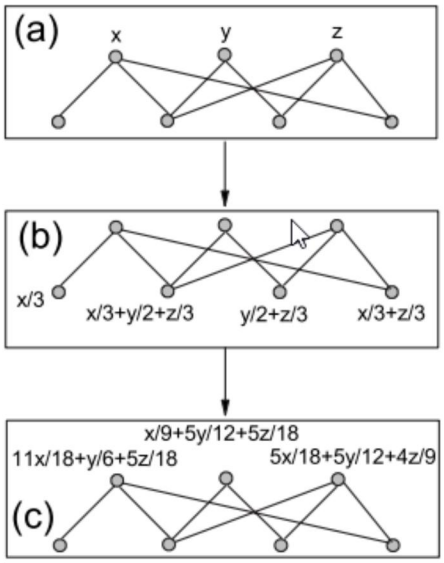

Figure 1: weight flow in bipartite network

The advantage of this method is that weights are not symmetric, and represent some internal structure of the bipartite graph. For example, in a collaboration network, the weight of a single collaboration paper between two researchers is fairly low if one scientist has published many papers, and vice versa.

Building off of this, Zhou and his team suggest a recommendation algorithm, which proceeds as follows. First, for a given user u_i, assign some weight on each object node as $f(x_j) = a_{ji}$, where a_{ji} is binary, representing if an edge exists between object x_j and user u_i. Then perform object-projection on the graph according to above. Then, sort all the objects that the user had not seen before, and recommend the ones with highest value. They demonstrate that this performed better than both collaborative filtering and global ranking method.

Critique: The proposed framework assumes equal weight to each node at the start, and disregards the case in which the original graph has weighted edges. Furthermore, this approach considers similarities between the users, but does not explicitly consider similarities between the objects themselves. Finally, this approach may suffer from the same issues as other recommender systems, as it may give stronger weight to more popular objects, as they have more incoming edges.

2.2 Building Recommender Systems with Node2Vec

Typically, when building a recommender system from a graph, there are two kinds of approaches you can use: The first, is to use the features of nodes in the graph to measure similarity between different nodes, and from that make a recommendation based on other nodes that are similar to the ones the user has already interacted with. The second approach is to look at the edges between nodes, and try to deduce with what probability there should be an edge between two pairs of nodes that currently are not connected. Node2Vec [3] is a scalable algorithm for networks that aims to accomplish both those objectives.

Enrico Palumbo et al. [2] used the first approach outlined above. They used a knowledge graph where for every node N they had a feature extractor which, in the case of users, was able to extract the movies they had given feedback on, and for movies, was able to extract details such as lead actors, production company, genre, and directors. Using the Movielens knowledge graph they were able to achieve a more than 10% increase over baseline performance.

Critique. Something they did not seem to explore in their paper was whether they were able to build a recommender system without a significant knowledge graph, instead leveraging the structure of the edge connections in the graph they had (i.e. which users recommended which movies, and which movies were watched by which users) to make recommendations. We wonder if it is possible to develop recommender systems that do not rely on that kind of specialised domain knowledge, and what the performance differences are between the two approaches, and will explore that in our project.

2.3 The Netflix Recommender System: Algorithms, Business Value, and Innovation

Unsurprisingly, Netflix themselves have put a large amount of work into building and optimizing their own recommendation system. Their recommendation system is very complex with several key components. **Carlos A. Gomez-Uribe and Neil Hunt** detail the various approaches. First is a Personalized Video Ranker (PVR) that gives recommendations based on genre, such as horror or comedy. It works by blending personalized recommendations with the popularity of content. Next, is a Top-N Video Ranker, which provides a user a ranking of all content Netflix has, across all possible genres. It recommends videos similarly to how the PVR gives recommendations. There is also a Trending Now section that ranks videos solely on temporal popularity. Finally, there is a Video-Video similarity ranker, which computes similarities between content a user has watched and other content in Netflix's catalog, and returns the most similar

other content. Each algorithm uses both statistical methods and machine learning techniques, including classification, regression, clustering, and matrix factorization.

This paper relates to topics taught in CS224W because it deals with the various ways in which Netflix deals with applying algorithms to their vast network of user data in order to create a effective recommendation system. This connects with other papers we are discussing because it gives concrete ways in which Netflix is currently attempting to tackle the recommendation problem. With this knowledge, we will be able to brainstorm better ideas to tackle the same problem using information from other papers, such as through bipartite graph analysis or using Node2Vec.

The strengths of the paper are how it breaks down the complex task of recommending Netflix content into four simpler categories. This allows us to comprehensively understand the various ways in which Netflix approaches this problem.

The weaknesses of the paper include how it doesn't detail ways in which their current approaches to recommendation can be improved; they only mention ways in which their approach to A/B testing can be improved, since they mention their current methodology of A/B testing can lead to skewed results. However, I argue that nominal differences in A/B testing results are overshadowed by improving the overall approach to recommendation.

3 Dataset

We will analyse the Netflix dataset [4] provided by Netflix in a competition to design a better recommender system for movies than they had managed internally. The dataset is a bipartite graph and consists of the following:

- **Users:** There are more than 480,000 users in the dataset.
- **Movies:** There are 17,770 movies in the dataset. For each movie, the English title and year of release is provided.
- **Ratings:** Ratings are represented as an edge between a user and a movie if the user rated that movie. The rating has a weight corresponding to the number of stars the user gave the movie, from 1 to 5 stars. There are over 100 million ratings in the dataset.

3.1 About the Dataset

We begin by analysing graphs showcasing the distribution of ratings, degree distributions of users and degree distributions of movies for the whole data set.

As we can see from **Figure 2**, most of the ratings are on the positive end of the spectrum. This suggests that users who watch movies they end up disliking often stop watching the movie before they finish it and get a chance to rate it.

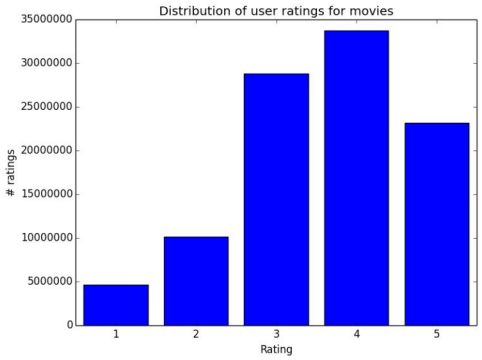

Figure 2: distribution of movie ratings

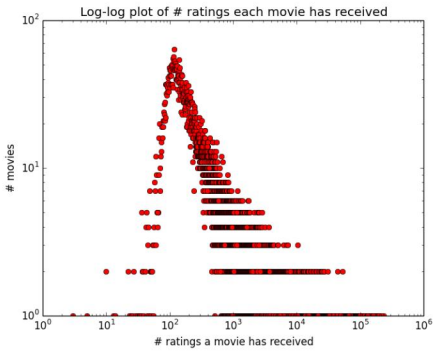

Figure 3: distribution of number of movies which have been rated a certain number of times

As displayed in **Figure 3**, we computed the degree distributions for users and movies. We did this to see if the Netflix network followed the power law, where the number of nodes with a given degree is inversely exponentially proportional to the degree of the nodes. We can see that is not quite true for the degree distribution for movies. After about 100 ratings, the degree distribution for movies visually appears to follow the power law, however, between 1 and 100 ratings, there is a relative dearth of movies. This suggests that the movies that receive less than 100 reviews are so unpopular that users actively decide not to engage with them, or that the existing Netflix recommender systems think these movies should not be recommended to any users. It is likely not

the case that these movies are particularly new, since Netflix always promotes new movies when they are made available on the platform. In any case, it suggests that it is probably not worth it to recommend these movies to other users.

By examining **Figure 4**, we can see that the distribution over the number of ratings each user has given compared to the number of users who have given that many ratings more closely follows the power law. Again, there is some irregularity for users who have rated less than 30 movies, so for algorithms like collaborative filtering, it is more useful to focus on users who have rated more than 30 movies - this also allows us to create a normal distribution over that user's specific preferences, so that we can compare users who rate movies differently with each other better. On the other hand, the users who have only rated a few movies might be newer to the service, which presents a unique opportunity as those users are the ones Netflix wants to spend most of their attention on retaining, so it is perhaps worth looking at algorithms to recommend movies to users when those users have not engaged much with the service yet.

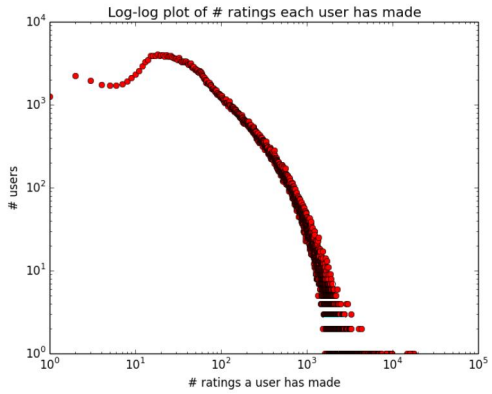

Figure 4: distribution of number of how many users have rated a certain number of movies

4 Algorithms Used

4.1 User-User Collaborative Filtering

To perform user-user collaborative filtering on the graph, we extract a vector v for each user where the i-th element of the vector is the user's rating for movie i. Since ratings for different users are drawn from different distributions, we then subtract the user's mean from each rating they have so as to be able to compare them better. If we let v be a user's rating vector, then we have that the user's adjusted rating vector $u_i = 0$ if the user did not rate movie i and $u_i = v_i - \bar{v}$ if the user rated movie i, where $\bar{v}$ is the mean of all the nonzero elements of v.

If we now have the adjusted rating vectors a and b for two different users, we can calculate the cosine similarity as follows:

$$\text{sim}(a, b) = \frac{a \cdot b}{\|a\|_2 \|b\|_2}$$

In order to now find the most similar users to a specific user we are interested in, we can take the cosine similarity between our user of interest and every other user in the graph, and pick the users with the top n cosine similarities to our target user in order to get a source to make movie recommendations from.

In order to test the results of our similarity algorithm, we picked a random user from the graph that had rated 100 movies and plotted a histogram of the cosine similarities between that user and the rest of the users in the network, which can be seen in **Figure 5**.

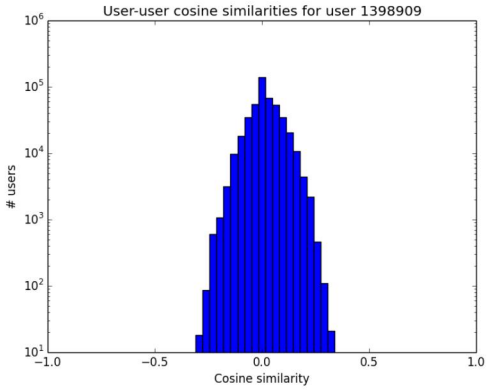

Figure 5: distribution of cosine similarity between user 1398909 and every other user in the network

The shape of this histogram almost perfectly resembles a bell curve. This means there are plenty of users who are both similar and dissimilar to our user of choice. This, in turn, means that there are users we can pick out whose movie ratings will correlate with our user's movie ratings.

4.2 Item-item collaborative filtering

Similar to user user filtering, we also attempted item-item collaborative filtering, using both cosine and pearson similarity. For pearson similarity, we defined the similarity between two movies i,j with common raters U as

$$sim(i, j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_i})(R_{u,j} - \bar{R_j})}{\sqrt{(R_{u,i} - \bar{R_i})^2}\sqrt{(R_{u,j} - \bar{R_j})^2}}$$

where $\bar{R}$ represents the average rating for a given item, and $R_{u,i}$ is the rating user u gave to item i. Cosine similarity was the same, except that we subtracted the average rating from a user u, rather than the average item rating. Finally, our prediction was given by

$$\frac{\sum_{allsimilaritems,N}(sim_{i,N} * R_{u,N})}{\sum_{allsimilaritems,N}(\|sim_{i,N}\|)}$$

4.3 Community detection

We used several community detection algorithms in an attempt to create communities of movies. The idea was to use these communities of movies to recommend similar movies to users, similar to Netflix's own "Because You Liked..." recommendation system, since related movies would likely be in the same communities. We approached the problem by first contracting the bipartite graph of movies and users to just a graph of movies, with pairs of movies being connected if a user watched both of those movies. Next, we used Louvain community detection through greedy modularity maximization, K-clique percolation, and K-way spectral clustering in an attempt to find communities within this contracted movie graph.

The Louvain algorithm used is exactly the same as the one discussed in class: Each node is first assigned its own community, then we try moving each node into a neighboring community and choose the community that maximizes the modularity of the resulting graph. The change in modularity that needs to be maximized is defined as follows:

$$\Delta Q = [\frac{\sum_{in}+k_{i,in}}{2m} - (\frac{\sum_{tot}+k_i}{2m})^2] -$$
$$[\frac{\sum_{in}}{2m} - (\frac{\sum_{tot}}{2m})^2 - (\frac{k_i}{2m})^2]$$

Where $\sum_{in}$ is the sum of weighted edges inside the community, $\sum_{tot}$ is the sum of weighted edges going into the community, k_i is the weighted degree of i, $k_{i,in}$ is the sum of weighted edges between i and the community, and m is the sum of the weights of all edges in the network [8].

The K-clique percolation method is another method of community detection. First, all cliques in the graph of size k (which is given by the user) are found, where cliques are defined as a subgraph where every node is connected to every other node. Next, adjacent cliques are unioned into the same community, where adjacency is defined as two cliques having k-1 of the same nodes. All adjacent cliques are unioned into super-cliques, and the resulting super-cliques are presented as communities.

The last algorithm we used was K-way spectral clustering, which is a generalization of the spectral partitioning algorithm shown in class and done in Homework 3, and more explicitly described by Shi and Malik[9]. To start, we calculate the adjacency matrix A and sparse diagonal matrix D of our graph, with the degree of node i being at D[i][i]. Then, we subtract the adjacency matrix from the diagonal matrix to get the Laplacian matrix L. We proceed to normalize the laplacian and get

$$\widetilde{L} = D^{-1/2}LD^{-1/2}$$

Now, we calculate the top n eigenvectors of $\widetilde{L}$ to receive an M by n matrix, where M is the number of nodes in the graph. We can now view this resulting M by n matrix as M feature vectors of size n, meaning each row is a feature vector for a certain node in the graph. We used n=10 to represent feature vectors of size 10 for this problem. Finally, we used the standard k-means clustering algorithm to partition these feature vectors into clusters and returned those clusters as communities.

4.4 Weighted Bipartite Graph Projection

Following the algorithms described in [7], we implemented a recommendation system based on weighted bipartite graph projection. By doing a weighted projection in the following manner, the weights of nodes will capture more general information about the structure of the graph, where higher weight indicates a stronger correlation with another movie. In the paper, the recommendation system they prescribe is as follows: perform a bipartite projection onto the objects. Then, for a given user u_i we are trying to recommend to, assign initial resources on the objects collected by u_i. In our case, we took the user we were trying to predict ratings, and split up the movies they had seen into a train/test set with a 90/10 split. This allows us to compare the predicted results to the actual results. In the paper, they assigned a unit resource based on whether or not the user had collected that object. In our case, we set the initial weight equal to the rating of the movie, as we wanted to give higher rated movies a higher score. Then, the weight is distributed by flowing equally to the neighbors, and then for each of the user nodes, distributed equally among that user node's neighbors. Mathematically, if our initial resource if $f(o_j) = r_{ij}$ where r_{ij} is the rating our user i gave to movie j, then the

final resource is

$$\sum_{l=1}^{n} w_{jl} f(o_l), w_{ij} = \frac{1}{k(u_j)} \sum_{l=1}^{m} \frac{a_{il} a_{jl}}{k(o_t)}$$

where k is the degree of a node, and a represents if there is an edge between those two nodes.

Finally, once we have the distributed weights, we can take the weight on the movie we are interested in, and scale it based on the min/max weight of the movies the user already rated. Then, based on this scale, we assign it a score of 1-5.

4.5 A Different Approach to Graph Projection

When doing our original projection to detect communities, we created an edge between two movies whenever there was a user that had rated both of the two movies. We didn't take into account how many users had rated both movies, nor did we take into account if users tended to rate both movies the same way. We wanted to incorporate more information in our projection, similar to what Zhou, et al, did in their paper, but captures more about how users rated these two movies.

In order to do this, we wanted to create an edge in the graph if ratings of two movie were correlated. To test for correlation, we used the pearson correlation coefficient, defined as follows. The Pearson correlation coefficient, r when given paired data X, Y, and $\{(x_1, y_1 ...\}$:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where n is the sample size, x_i, y_i are the individual indexed sample points, and $\bar{x}.\bar{y}$ are the sample means.

In our approach, we added an edge between two movies if the Pearson correlation coefficient of the ratings of both movies were positively correlated with each other. We considered ratings for only users that rated both movies. Then, we added an edge between movies if their Pearson coefficient was greater than .75. We wanted there to be significant correlation, as that would imply that these two movies are generally rated similarly by both users. In addition, we created a second graph, in which we created an edge of weight 1 if the coefficient was greater than .75, and an edge of weight -1 if they were strongly negatively correlated, i.e. a coefficient of less than $-.75$.

4.5.1 Predicting Ratings

We then used this projected graph to predict ratings in the following fashion. For a user,movie pair that we were interested in, we look at all the neighbors of the movie in the projected graph. Because of how we did the projection, the ratings for these neighbor movies should be strongly correlated with the rating for our query movie. As a result, we can look at the neighbor movies that our chosen user rated, and average those ratings to estimate what the user would rate that movie. If we are using the weighted projected graph, we look at if the edge is 1 or -1. If 1, we do as previously, but if the edge weight is -1, then what we can do is add (6-rating) instead, as it is negatively correlated. We tried both of these approaches separately to see if there would be a significant difference.

4.5.2 Community Detection Part 2

Furthermore, we used this projected graph with our community detection algorithms. We simply replaced the original projected graph (with an edge between two movies if a user watched both) with this new projected graph containing correlated edges, and ran the Louvain, K-clique, and K-way spectral clustering algorithms.

4.6 Node2Vec

Now that we had a better movie projection graph, we were interested in seeing if we could apply the Node2Vec algorithm to this Pearson projection and learn embeddings for every movie in the projection in order to make better item-item recommendations. Since Node2Vec learns the role of a node in a network based on the surrounding network structure, we decided to run node2vec while varying the outparameter q in order to not only measure the performance of node2vec as a method for item-item recommendation, but also to see if varying the exploration parameter has a measured effect on the performance of the algorithm.

Once we had the embeddings, then for each user-movie pair in our test set, we would get the node embeddings for all the other movies the user had rated, order them by cosine similarity to the embedding for the movie of interest, and then give an average of the most similar other n movies the user had rated, where we also varied n to see if that had an effect on the accuracy of our algorithm.

5 Results and Analysis

5.1 Community Detection

When running the **Louvain algorithm** on the original projected movie graph, where an edge exists if

two users rated the same movie, the algorithm combined the entire movie graph into just two communities, with each community having around half of the movies in our test dataset. Furthermore, when running Louvain on the Pearson correlation graph from section 4.5, it still only discovered 3 optimal communities. This implies that Louvain community detection is not a good algorithm for community detection on our dataset. This could be due to the fact that there exist many edges in both versions of our graphs, which adds noise to modularity calculations.

After running the **K-clique percolation** algorithm with various parameters of K ranging from 5 to 20 on the original projected movie graph, we were only able to ever get a single community containing all nodes from the algorithm. In addition, after running the K-clique percolation algorithm on the Pearson correlation graph from section 4.5, we were still only able to get a maximum of 2 separate communities. This suggests to us that users commonly watch many movies that are dissimilar from each other, even ones that they may not necessarily be interested in. Furthermore, we calculated the clustering coefficient of the contracted movie graph and got values higher than .99, which further highlights our initial belief. Thus, given our dataset and its overall structure, we determined community detection through modularity optimization and K-clique percolation to be insufficient.

With **K-way spectral clustering**, things definitely looked better. When running the algorithm on the original projected movie graph with k = 50, we were able to find 50 communities with somewhat reasonable groupings. However, since it's overwhelming for a user to see 50 groups with hundreds of movies each, we wanted to increase the number of communities. With k = 100, on the original graph, unfortunately, the algorithm again only gave us one community containing every single movie, which was a problem. Finally, we turned to using K-way spectral clustering on the Pearson correlation graph from section 4.5, which succeeded and gave us reasonable results with any number of clusters, including 100. To validate our results, we calculated the Pearson correlation of user ratings between every pair of nodes in a cluster and averaged this over every pair to give us the average Pearson correlation for a cluster. We then averaged this correlation value over every cluster to get the within-cluster Pearson correlation value of 0.6261. To get a baseline Pearson correlation for the entire graph, we treated the graph as a single cluster and calculated the Pearson correlation, which gave us a value of 0.4597. This shows that our K-way spectral clustering algorithm gave us a significantly improved cluster set than no clustering at all.

Here is an example cluster of movies given by the algorithm: [**'Pink Floyd: Inside Pink Floyd: A Critical Review 1975-1996', 'Automotive Series: Porsche', 'Backstreet Boys: Backstreet Stories', 'Opinion', 'Suze Orman: The Courage to be Rich', 'Meat Loaf: VH1 Storytellers', 'Animal Attraction 3', 'Howard Hughes: The Real Aviator', "Tina Turner: Rio '88", 'Butthole Surfers: Blind Eye See All: Live 1985', 'The Drifter', 'Dogs and More Dogs', 'The Jazz Channel: Chaka Khan', 'Mr. Ice Cream Man', 'Todd Rundgren: Live in San Francisco', 'The Witness Files', 'MacArthur', 'Diane Schuur the Count Basie Orchestra', 'Red Shoe Diaries: Hotline', '2000 Years of Christianity', 'Fighter Jets and Attack Aircraft', 'Pink Floyd: Inside Pink Floyd: A Critical Review 1967-1974', 'The Mafia: An Expose - Coming to America/Al Capone', 'Sopranos Unauthorized: Shooting Sites Uncovered', 'Trial', 'Pop and Me', 'Emerso', 'Alanis Morissette: Jagged Little Pil', 'Combat Vietnam: To Hell and Beyond'**]. This cluster seems to contain movies related to music and musicians, with movies about Pink Floyd, the Backstreet Boys, the Count Basie Orchestra, The Jazz Channel, Todd Rundgren, and Alanis Morissette. There are definitely a couple of outliers, such as Combat Vietnam, but the overall clustering seems solid.

5.2 Rating prediction

5.2.1 Evaluation Metric

For our rating prediction, we tested to see how close our prediction got to the true value. In order to quantitatively measure this, we used mean-squared error. Ideally, we want to minimize the mean squared error, as that represents how close we are to the ground truth. For rating predictions $\hat{y}$, ground truth values y, and n samples, we have

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y} - y)^2$$

5.2.2 Results

We compare the mean-square error of all our prediction methods with a random predictor that predicts a random integer between 1-5 for each user. This random predictor has a mean squared error on the user ratings of approximately 3.63, so any error we get

Approach	MSE
User-user	0.803
Item-item	0.872
Weighted bipartite projection	1.73
Pearson Correlation similarity(4.5.1)	**0.543**
Node2Vec	1.08

Table 1: All Methods Mean Squared Error

that is significantly less than this number represents an improvement.

For user-user filtering, we picked 30 of our chosen user's most similar users, and for all the users movies, we used the similar users' adjusted ratings for that movie to predict what the current user's rating would be. This was done using a baseline algorithm by simply averaging the users' ratings for that movie together, then adding our chosen user's average movie rating to that result. Using this approach, we were able to predict the chosen user's movie ratings with an average mean-squared-error of ≈ 0.803, which is quite good for a baseline algorithm. We made sure to remove bias by not considering the user's movie score we were estimating in any of the calculations outlined above (we basically temporarily pretended they had never rated that movie, and found similar users based on that assumption, etc.).

To test item-item filtering, we proceeded the same as in user-user filtering, except we compared the similarity between the test movie and the other movies the user had rated previously. Our pearson similarity performed slightly better than cosine similarity, with pearson having a mean squared error of .872, and cosine similarity having a mean-squared error of 1.473. Both performed worse than user-user, largely due to the issues of the cold start problem, as there is not sufficient information about the movies.

Evaluating the weighted bipartite graph projection on our dataset by performing the algorithm from 4.4, we found that this unfortunately had a mean-squared-error of 1.73, worse than our collaborative filtering efforts. It seems like without additional features, this is not a good approach for rating prediction. This is likely due to the fact that the model in the paper assumed an unweighted graph, whereas our original graph is unweighted. Furthermore, they had the weight flow equally, and although we tried to account for this by having weight flow proportionally to how a movie was rated, because many movies were rated by many users, and many users rated a lot of movies, the overall weight across movies ended up being relatively even, which led to many of our predictions being between 2.5 and 3.5, leading to the mean squared error as stated above.

For the method suggested in 4.5.1, we selected 1000 different users that had rated more than 50 movies. For each user, we split up their movie set into train/test with a 90/10 split. We then tested on the 10% of movies in our test set. We compared the predicted value from both the unweighted projected graph, as well as the weighted projected graph, and we found that on average, the mean squared error for the unweighted graph was .54495, and the mean squared error for the weighted graph is .5427. These results make sense, as both graphs were created with the assumption that edges between movies exist if their ratings are strongly correlated. Thus, the user's rating on the similar movies is strongly correlated with their rating on the test movie. We could improve the values by increasing the bound we require to create an edge, but we found that higher values led to too little data. Additionally, it makes sense that the weighted graph would have higher performance, as it is simply just more information. Since we flip the rating, it is essentially another data point that is strongly correlated. As we can see, although there is a difference, it is very minimal.

For Node2Vec, we decided to vary the exploration parameter q between 0.3 and 2.0, to see if the different values would have an impact on the accuracy of the recommendation algorithm. We also varied the total number of similar movies' ratings to average. Here are the results for a small subset of 100 movies of the entire graph:

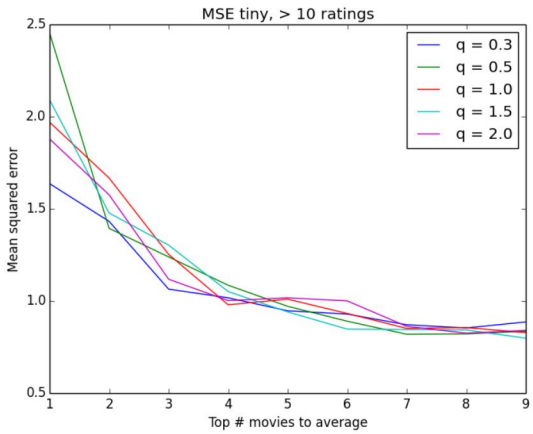

Figure 6: Node2Vec rating performance on 100 movie dataset

As we can see, there does not appear to be a significant difference depending on the value q we choose. This could be either because our values were not extreme enough, or because other exploration parameters in the algorithm have a bigger impact, or because

our data set is too small. So we tried calculating the prediction error on a larger data set of 1000 movies:

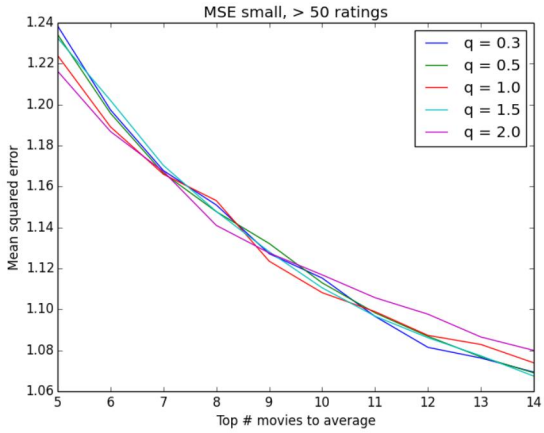

Figure 7: Node2Vec rating performance on 1000 movie dataset

As we can see here, there does not appear to be any effect on the prediction accuracy if we vary the value for q even when we have more data for Node2Vec to generate better embeddings. In the end, the best mean squared error for the Node2Vec algorithm on the Pearson projection graph coupled with cosine similarity between movie embeddings was 1.08 for the larger dataset. Perhaps we need to run Node2Vec on a different projection in order to better predict ratings for movies, since perhaps by using the Pearson projected graph, too much of the underlying information in the graph structure that the random walks would have otherwise found was either lost or flattened so that the predictions became sub-optimal. This would be useful to look into in future research.

In summary, of all the methods we tested (listed in Table 1), Pearson Correlation similarity had the highest performance.

6 Difficulties and Challenges

One thing that was very difficult for us was choosing the correct python library to handle our network analysis. Our first thought was to use Snap, since it was what we had been using in class to analyze the given networks. However, given the bipartite nature of our graph, the large size of our data, and the need to contract the graph efficiently, we ended up switching to another network library. We had experience with the NetworkX library from a different class, so we decided to give that a shot to see if we would fare any better in our attempts to make sense of this huge network. We looked over the documentation [5] and discovered many extremely useful features for our project, including automatic graph contraction and community detection.

This decision still led to difficulties. The graph structure is so large that to load it in memory would require about 40-50GB of RAM, which restricted the size of the dataset we would work with while testing and gathering initial data. As a result, we gathered data on smaller subsets of the entire graph, using between 100-2500 movies instead of all 17770 for most of our tests. On a positive note, even with this smaller dataset, we still had ratings from all the 480000 users, so it provided a good testbed for making sure our algorithms were working well.

Our last problem was that the graph is relatively sparse, as the density is only around 1.1%. This made certain methods, like item-item collaborative filtering, more difficult.

7 Conclusion

In the end, our approach to create a graph projection based on the Pearson Correlation coefficient and use a relatively simple algorithm to average a user's ratings on similar movies as defined as neighbors in this graph projection performed quite well. We managed to incorporate rating information in our projection and it resulted in a fairly good predictor. As an improvement, we could perhaps try other techniques on this projected graph, such as trying cosine similarity or other measures to try and improve the predictions.

While all the techniques we tried to apply to the original as well as the Pearson projection to detect communities and perform clustering eventually bore fruit and we were able to find clusters that both made sense and had higher average correlation than the graph as a whole, we did not get to use this information to inform any of our rating prediction algorithms. This, however, would be an excellent avenue for future research, as one could potentially preferentially weight movie recommendations that are in the same clusters as movies the user has already seen.

Surprisingly, Node2Vec did not turn out to perform as well for this problem as we thought it would. Our hypothesis for why this happened is that we applied the Node2Vec algorithm to a graph that had already been extensively processed, that is, perhaps in creating the Pearson Correlation coefficient movie projection, we got rid of a lot of the underlying structure that could have improved the embeddings the Node2Vec algorithm learned. Something that supports this hypothesis is that both changing the dataset size by an order of magnitude as well as varying the

exploration parameters seemed to have little or no effect on the performance of this predictor. It would be interesting for future research to compare the performance of Node2Vec on different types of graph projections of bipartite graphs for the purpose of item recommendation. It is certainly something we would have wanted to explore ourselves, given more time.

8 Github

Here is a link to our github repository, containing all the code we wrote for this assignment:
`https://github.com/jason2249/cs224w`

9 Individual Contributions

Aleksander did data set analysis, converted the movie adjacency list to a user adjacency list for use in the node2vec algorithms and performance of random selector, and created section 3 in this report. He researched the Node2Vec recommender systems paper in section 2. He also implemented user-user filtering and the recommendation system based on Node2Vec(4.6), and measured their results, as well as results for baseline random algorithms.

Jason researched the Netflix paper in section 2 and guided the exploration of various algorithms used later in the paper. He did all community detection related work, including Louvain, K-clique, and K-way spectral clustering on both the original movie graph and the Pearson correlation graph. He calculated empirical results to prove that the K-Way spectral clustering algorithm worked well.

Nolan researched the bipartite graph paper, implementing it along with item-item filtering. He also created the movie graph projection based on the Pearson coefficient, and implemented the recommendation system discussed in 4.5.1. Nolan also helped in generating various sized sample graphs for easy testing, and worked on maintaining the code base.

All three group members collaborated on writing this report.

References

[1] `https://journals.aps.org/pre/abstract/10.1103/PhysRevE.76.046115`

[2] `https://2018.eswc-conferences.org/files/posters-demos/paper_265.pdf`

[3] `https://arxiv.org/pdf/1607.00653.pdf`

[4] `https://archive.org/details/nf_prize_dataset.tar`

[5] `https://networkx.github.io/documentation/networkx-1.10/reference/algorithms.html`

[6] `http://www.ra.ethz.ch/cdstore/www10/papers/pdf/p519.pdf`

[7] `https://arxiv.org/pdf/0707.0540.pdf`

[8] `https://en.wikipedia.org/wiki/Louvain_Modularity`

[9] `https://people.eecs.berkeley.edu/~malik/papers/SM-ncut.pdf`