Collaborative Filtering using Weighted BiPartite Graph Projection

A Recommendation System for Yelp

Sumedh Sawant
sumedh@stanford.edu

Team 38
December 10, 2013

Abstract

We implement a personal recommendation system on the Yelp Dataset Challenge dataset using the same novel network-based-inference collaborative filtering algorithm that was proposed by [2] and originally created by [1]. By representing our Yelp dataset using a weighted bipartite graph where edges from user to business are weighted by rating, we pose the recommendation problem as graph projection. More specifically we follow a network based resource allocation process to produce similarity measures (which can both be thought of as weights in a bipartite graph projection) between every pair of users and every pair of businesses, which are then used to produce predicted ratings and recommendations. We evaluate the performance of this system with respect to popular competing ways of creating recommendation systems using different non-network based approaches to collaborative filtering. We then attempt to experiment and improve upon the algorithm by relying on properties of the Yelp dataset and then evaluate the results of these experiments.

Introduction

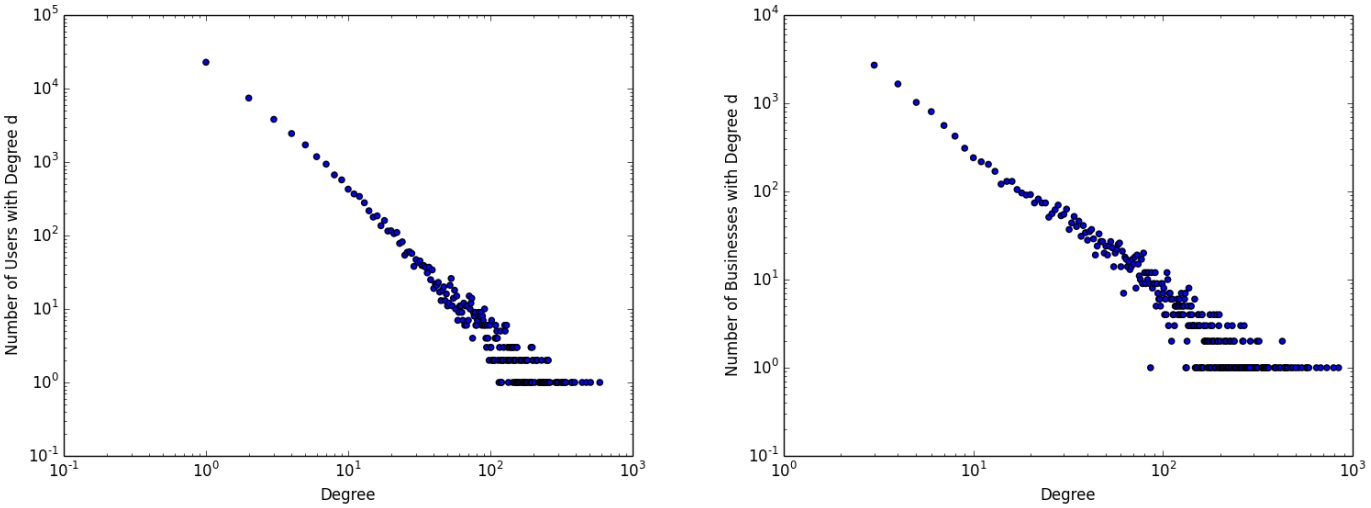
The rise of the popular review site Yelp has led to an influx in data on people's preferences and personalities when it comes to being a modern consumer. Equipped with access to a vast database of reviews, ratings, and general information provided by the community about any business, the consumers in the world today have a myriad of choices even when it comes to picking a specific set of services and/or consumable goods. However, it is often hard for people to make these choices while relying on just the raw data provided by Yelp. In essence, the format in which Yelp presents data about a particular business is not optimal for users to make a quick, efficient choice. Rather, users feel slightly overwhelmed by the pure information provided to them and often desire some means of systematically processing the data to make an informed choice.

Recommendation systems aim to make this problem easier for users by utilizing their personal preferences and those of similar users to suggest potential choices for them. Recommendation systems have historically been created for various machine learning applications to numerous required disciplines. For example, social networking sites such as Facebook utilize recommendation systems to suggest friendships to users. Music and media applications such as iTunes and Spotify utilize similar machine learning and recommendation logic to suggest various songs, videos, movies, etc. to users based off their previous choices and taste. Given this general theme, our project aims to create a recommendation system of Yelp businesses for Yelp users using collaborative filtering. However, whereas traditional collaborative filtering approaches focus only on properties of the dataset to make recommendations, by posing the recommendation problem as graph projection (as in [2]) we plan to implement a collaborative filtering recommendation system that utilizes both the regular properties and the network properties of the Yelp dataset to make more accurate predictions.

Data

Our primary dataset is the Yelp Dataset Challenge data (`http://www.yelp.com/dataset_challenge`) that contains actual business, user, and users' review data from the greater Phoenix, AZ metropolitan area. We represent this dataset as a weighted bipartite graph between users and businesses. Each edge from a user to a business is weighted by the review that the user gave the business. Below we summarize the characteristics and network properties of the dataset:

Number of Businesses	11,534
Number of Users	45,970
Number of Reviews (Edges)	229,879
Total Nodes	57,504
Size of Largest Connected Component (percentage of total nodes)	99.58437673900946%
Edge Set Size of Largest CC (percentage of total edges)	99.91865285650277 %
Average Degree (number of ratings) of User	5.0006308462
Average Degree (number of ratings) of Business	19.9305531472
Average BiPartite Clustering Coefficient	0.15160712331



(a) Degree Distribution of Users

(b) Degree Distribution of Businesses

Figure 1: Distribution of Degrees in Yelp Dataset

Review of Prior Work

Historically (through the review of [4], [5], [6]), collaborative filtering has been a popular way to implement recommendation systems. There are three types of collaborative filtering algorithms in the recommendation system literature:

1. Memory Based Approaches: Memory based collaborative filtering approaches operate over the entire dataset to generate a prediction. Using statistical techniques, they attempt to find a set of users or items that have historically been similar to each other in the past (either a group of users that rated the same products the same way or a group of items that were all rated the same way by the same users). Once this set of similar users or items is found, some algorithm that takes advantage of these similar user/item sets is applied to make a prediction of a particular user's rating for a particular item.

2. Model Based Approaches: Model based collaborative filtering approaches utilize the dataset to learn (using some sort of machine learning algorithm) some sort of probabilistic mathematical model that is later used to make predictions. These approaches think of the collaborative filtering process as calculating the expected value of a user's rating given the rating that user gave other items.

3. Hybrid Approaches: Hybrid based collaborative filtering approaches attempt to combine the memory based and model based approaches in an attempt to use the advantages of both approaches.

In this paper we will consider the memory based approach. As [2] states, the most popular type of memory based collaborative filtering is a neighborhood approach, of which there are two types:

1. User-based collaborative recommendation aims to calculate some similarity metric between all pairs of users and then predict a particular user u's rating for an item i by collecting and processing the ratings of u's "neighborhood" (all the other users with high similarity as compared to u) for i.
2. Item-based collaborative recommendation seeks to calculate some similarity metric between all pairs of items and then predict a particular user u's rating for an item i by collecting and processing the ratings of i's "neighborhood" (all the other items with high similarity as compared to i that u has rated).

Algorithms that adopt either style of approach have three main components: 1.)Some definition of similarity, 2.)Some way of using similarity to construct neighborhoods, 3.)Some way of using a neighborhood of a user or item to make predictions. We can treat all of these components as blackbox abstractions. Namely, any one of these components accomplishes a certain goal but the implementations of the component can be swapped in and out to form various versions of algorithms. There are several popular ways traditionally used in recommendation systems literature to implement each component, involving various similarity functions such as the Pearson correlation coefficient, Cosine similarity metric, etc. However, not until recently has recommendation system literature looked to network properties of data to implement these components. The algorithm from [2] that we implement adopts this idea, using a weighted bipartite graph projection to make accurate predictions.

Methodology

Background

We present some of the theory and formal definitions relevant in this section before defining the algorithm from [2] that this paper is based on in the next section.

Collaborative Filtering Formal Definition

User-based and item-based collaborative filtering are mathematically equivalent by swapping the roles of the user and item. Thus we will just consider the user-based approach in this section. In user-based collaborative filtering, we are trying to calculate the predicted rating $\widehat{r}_{u,i}$ of a user u for a particular item i. We first calculate and store the value of some similarity metric between any two pair of users, and then compute the prediction of the rating of user u towards an item i by computing a weighted average of u's neighbors ratings of i (the weights are the similarity between u and each neighbor).This is formally expressed ([4]) as:

$$\widehat{r}_{u,i} = \overline{r}_u + \kappa \sum_{j=1}^{n} sim(u,j)(r_{j,i} - \overline{r}_j) \qquad \text{where} \quad \overline{r}_u = \frac{1}{I_u} \sum_{j \in I_u} r_{u,j}$$

Here $\overline{r}_u$ is the average rating given by user u (I_u above is the set of items on which user u voted), $sim(u, j)$ is the similarity measure between user u and user j, κ is a normalizing factor so that the absolute values of the similarity metrics sum to 1, and $r_{u,i}$ is the actual rating given by user u to item i (note that this is different from $\widehat{r}_{u,i}$ which is the predicted rating). Various user-based collaborative filtering algorithms differ by the definition of the similarity function $sim(x, y)$ that they use. Popular similarity functions used are the Pearson correlation similarity and Cosine similarity (where I_{xy} is the set of items rated by both users x, y):

Pearson Correlation Similarity

$$sim(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)^2 \sum_{i \in I_{xy}} (r_{y,i} - \bar{r}_y)^2}} \tag{1}$$

Cosine Similarity

$$sim(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i})(r_{y,i})}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}} \tag{2}$$

Using the work of [2], we plan to define a new $sim(x, y)$ function in the algorithm used in this paper based off the network properties of the Yelp dataset represented as a weighted bipartite graph.

Weighted BiPartite Graph Projection

A bipartite graph is a graph of two sets X and Y where edges (assume undirected) are only allowed from one node in X to one node in Y. If we associate a weight w with each edge in the bipartite graph, we get a weighted bipartite graph. In a projection of a bipartite graph only one set of nodes (X or Y) appears with edges from one node to another if they were connected to the same node (from the other set Y or X) in the original graph. We can also add various weights to the edges in the projection. For example one possible weighting for an edge from node x_1 to x_2 in a projection of the set X is the number of nodes in Y that both x_1 and x_2 are connected to. When implementing our algorithm from [2], we will treat the weight of the edges in a projection of a set of a weighted bipartite graph as the similarity measure between the nodes that the edges connect to, thus linking the notions of personal recommendation and weighted bipartite graph projection. As mentioned above, these weights (similarity measures) will be derived from the network properties of our dataset after following a resource allocation process in the network when creating a weighted projection of the bipartite graph.

Algorithm

In this section we discuss the formal definition of the algorithm (all taken from [2]) that we implement on the Yelp dataset in this paper. Again, since the user-based and item-based version of this algorithm are mathematically equivalent by swapping the roles of the user and item, we will just explain the user-based algorithm in this section.

Recommendation Power as Similarity

Our main goal is to find a new $sim(x, y)$ function based off the network properties of the Yelp dataset represented as a weighted bipartite graph. As mentioned above, we will derive the values of our new $sim(x, y)$ by using a network-based process to create a weighted bipartite graph projection where the weights of our edges are the values of our new $sim(x, y)$ where the inputs x, y are the endpoints of the edge. We will differ from the regular definition of a weighted bipartite graph projection in that the weighted bipartite graph projection formed from the result of our network-based process will for any two nodes i, j have two directed edges (one going from i to j with weight w_{ij} and vice versa).

As [1] states, the weight w_{ij} in a weighted bipartite graph projection can be thought of as how important node j is to node i (thus $w_{ij} = w_{ji}$ does not always hold). For example, in our Yelp bipartite network where one set of nodes is users and another set is businesses that the people reviewed, a weighted bipartite projection of the people set has the weight w_{ij} represent how likely person i is to choose person j to recommend a business for him or her. Similarly, in a weighted bipartite projection of the business set, the weight w_{ij} represents how likely business j is to appeal to a particular user provided that he has been to business i. Considering the user set projection example (since the item-based version is equivalent), we can intuitively think of this as each user giving his or her neighbors some amount of "recommendation power" that they in turn can use to recommend businesses. The more "recommendation power" that a user has, the more powerful or influential his or her recommendation is to the original user who gave them that power.

Thus, lets assume that each user has some amount of recommendation power (call this "resource") that he'd like to distribute to all of the other users and $rp(u, v)$ represents the proportion of recommendation power that user u will give to user v. However, users are not directly connected to other users in the bipartite graph, they are connected to businesses who then connect to other users. Thus in order to distribute the resource through the network, a user u must first distribute some amount of resource to all the businesses that he or she rated. The businesses then all give some amount of resource back to v (depending on v's rating of those businesses) in order to form the value $rp(u, v)$.

In order to formalize this notion of distributing the resource, we can consider a 2-step random walk on the Yelp bipartite graph where we walk 2 steps starting from some user node, going to a business node, and coming back to a different user node. Each step that we take has some transition probability of occurring. Intuitively, for the first step in this process we can think of a user as being more likely to give a particular business some of his or her resource if the user rated that business highly. Thus we can think of the probability of the transition from user u to business b as being $\frac{r_{u,b}}{R_u}$ where $r_{u,b}$ is the rating that user u gave business b and R_u is the sum of ratings that user u ever gave. Similarly, we can think of a business b as being more likely to give a user v some of the resource it received if user v rated business b highly. Thus we can think of the probability of the transition from business b to user v as being $\frac{r_{v,b}}{R_b}$ where $r_{v,b}$ is the rating that user v gave business b and R_b is the sum of ratings that business b ever received. Thus the value $\frac{r_{u,b}}{R_u}\frac{r_{v,b}}{R_b}$ is the probability of a transition from user u to a particular business b and back to user v (and thus the amount of recommendation power or resource that user v receives from user u through business b in the network). Since user u distributes resource to all of the businesses and the expression $\frac{r_{u,b}}{R_u}\frac{r_{v,b}}{R_b}$ represents the amount of resource user v receives from user u from one particular business, we can sum over all businesses to get the total amount of resource that user v receives from user u. Thus we have that $rp(u, v) = \sum_{b \in B} \frac{r_{u,b}}{R_u}\frac{r_{v,b}}{R_b}$ as the definition of recommendation power.

Making Predictions using Recommendation Power

We can think of recommendation power as $sim(x, y) = rp(x, y)$. Thus, substituting into our formula from above for collaborative filtering, a prediction $\widehat{r}_{u,b}$ of a user u's rating for a business b can be made as $\widehat{r}_{u,b} = \overline{r}_u + \sum_{v \in U} rp(u, v)(r_{v,b} - \overline{r}_v)$ where $\overline{r}_u = \frac{1}{B_u} \sum_{b \in B_u} r_{u,b}$. In this formula, all definitions follow (besides that U is the set of all users) from the definitions given in the earlier section where this formula was defined. Also notice that $\kappa = 1$ in the formula since $\sum_{v \in U} rp(u, j) = 1$ due to the probabilistic properties of $rp(u, v)$.

Naive Baseline

For our baseline, we adopted a similar but slightly modified approach to [5]. In particular, we predicted the rating $\widehat{r}_{u,b}$ of a particular user u for a business b as $\widehat{r}_{u,b} = \mu + d_u + d_b$ where μ is the average rating of businesses by all users, d_u is the difference of user u's average rating from μ, and d_b is the difference of business b's average rating from μ. Intuitively, this algorithm naively calculates a rating by just taking the average rating of all businesses and incorporating both a user's difference from the average rating and a business' difference from the average rating.

Results

To test our results, we follow a similar approach as in [2] and use cross validation when checking our performance with several evaluation metrics. Specifically, we break the Yelp dataset into two chunks of 90% and 10% each. The first set of 90% is used to train the system and the second set of 10% is used to test the system.

For evaluation metrics, as [3] states popular metrics used to measure the performance of recommendation systems where a user gives an item a rating are **Root Mean Squared Error (RMSE)** and **Mean Absolute Error (MAE)**. Given that our system generates predicted ratings $\widehat{r}_{u,b}$ of a user u for a business b for a test set T of user-business pairs for which the true ratings $r_{u,b}$ are known, RMSE and MAE are defined as:

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,b) \in T} (\widehat{r}_{u,b} - r_{u,b})^2} \qquad (3) \qquad\qquad MAE = \sqrt{\frac{1}{|T|} \sum_{(u,b) \in T} |\widehat{r}_{u,b} - r_{u,b}|} \qquad (4)$$

We thus use RMSE and MAE to evaluate the results of our algorithm. Below we compare the performance of running different collaborative filtering strategies. Namely we look at the performance of the naive baseline algorithm we proposed earlier, the bipartite projection collaborative filtering approach of [2] that this paper is based on, and popular non-network based variations of collaborative filtering using both the Pearson Correlation value and Cosine Similarity as the similarity measures. We run these algorithms for user-based filtering and then report the results as the average of the evaluation metric values of 10 test runs (note that the naive baseline stays the same since there is no notion of specializing it to be user or item centric).

User-Based Approach Results:

	RMSE	MAE
Naive Baseline	2.04676973	1.25894400509
Cosine Similarity	1.72563276664	1.03285481242
Pearson Correlation	1.49575933288	0.95621957806
BiPartite Projection	1.47526194112	0.94222332268

As we can see above, weighted bipartite graph projection performs better in terms of prediction accuracy on the Yelp dataset than any of the other popular collaborative filtering approaches.

Experimentation

In this section, we discuss some of the experiments that we ran by modifying the weighted bipartite graph projection algorithm that this paper is based on. In general, we utilize other network properties of the Yelp dataset to extend the algorithm and attempt to achieve better accuracy in making predictions. We implement each of the following approaches and then evaluate and compare the results to those we listed above.

Clustered Weighted BiPartite Projection

Because our Yelp dataset is sparse, there are not a high number of instances where a pair of users rated the exact same business. This ultimately affects the accuracy of our recommendations. Thus to deal with the sparseness of the dataset, we apply the following experimental approach:

1. Use the k-means clustering algorithm (see appendix) using the recommendation power metric as similarity to partition the users into a set C_u of k_1 user clusters and partition the businesses into a set C_b of k_2 business clusters.

2. Construct a compressed version G' of the original bipartite graph G by creating a set of nodes representing each C_u user cluster and a set of nodes representing each C_b cluster. An edge from a node representing cluster C_u to a node representing cluster C_b exists if any user from C_u ever rated a business in C_b.

3. Run the same weighted bipartite project algorithm from [2] on the compressed graph G' and predict the rating that a user u would give a business b by simply considering the rating that the node C_u would give the node C_b in the compressed graph.

When measuring the results for the above experimental algorithm, we used an approximation to find the most optimal values of k_1, k_2 and ran the approximation approach for the range $0 \leq k_i \leq 1000$ in incremental steps of 25 for both k_1, k_2. Instead of running our algorithm on every single combination of (k_1, k_2) pairs (which would have involved running $40 \times 40 = 1600$ different scenarios), we ran our algorithm on each value (k_1 or k_2) separately while keeping the other value (k_2 or k_1) fixed at 0 (no clustering). The combination of the optimal values obtained from these two approximation runs (which was a total of $40 + 40 = 80$ runs of our algorithm) was our approximated optimal k_1, k_2 pair. We ran the above experimental algorithm and got the following results:

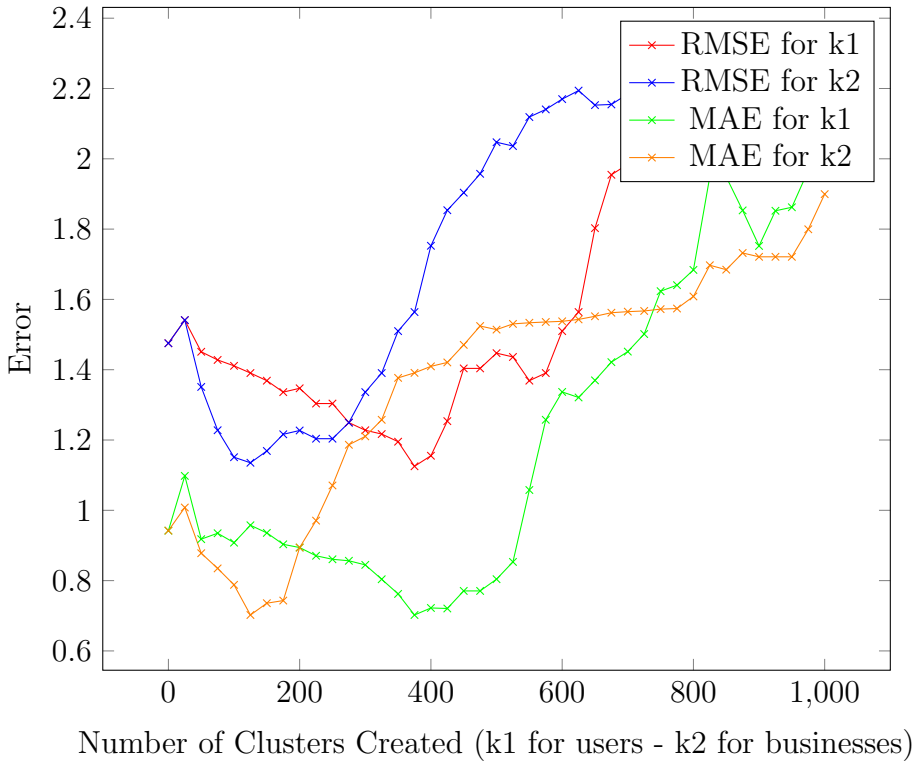


Figure 2: Clustered Weighted BiPartite Projection Evaluation

As mentioned above, each curve for a particular k_i in the graph represents running the clustered version of our algorithm while only varying k_i from 0 to 1000 in increments of 25 and checking the impact on the error metric (RMSE or MAE). Also, we obtained the RMSE and MAE error metrics by using the same approach as the Results section above (90-10 cross validation). As we can see from the graph, $k_1 \approx 375$ and $k_2 \approx 125$ are the optimal values (have the lowest error) for the RMSE and the MAE error to be as low as possible while utilizing clustering. Averaging the minimum error values of the k_1 and k_2 curves at $k_1 \approx 375$ and $k_2 \approx 125$, we have that:

Clustered Optimal Error Metric:

	RMSE	MAE
$k_1 \approx 375$ and $k_2 \approx 125$	1.13525525691294118	0.7022166384729412

Comparing this to our RMSE and MAE values (RMSE was 1.47526194112 and MAE was 0.94222332268) for running the algorithm from [2] without clustering, we can see that we have slightly improved by utilizing the compressed, clustered graph. We can also note the structure of the graph above. Originally when we ran the algorithm from [2] on our graph without clustering, there were not many scenarios where two users had rated the same product, thus making it hard to actually capture the similarity of two users (or vice versa businesses). We can see from the graph that creating clusters of similar users and similar businesses seems to help making better recommendations since we have more scenarios where two user clusters C_{u1}, C_{u2} each have a user that have rated a business from the same business cluster C_b. This enables us to better capture the similarity between two clusters and make better fine-grained recommendations, hence driving down the error rate initially. At the left side of the graph above when we have a really few amount of clusters, we are not capturing enough of the differences between each of the users (or businesses) and thus underfit the data (seen by the sudden spike in error right after 0). As we create more and more clusters (progress to the right on the graph) by increasing k, we capture more and more of the data's specifics and thus are able to make better recommendations (seen by the error dropping in the beginning). Eventually we reach some critical point where we have created the optimal number of clusters to drive down the error. If we create any more clusters than the optimal amount, we approach the scenario where the clusters are so small that we start to capture the really minor differences between the users and treat them as huge differences in making recommendations. Thus by making more than the optimal amount of clusters, we overfit the dataset and start to increase the error metric (as shown from the sudden increase after the decrease to the optimal point on the graph above). The users curve k_1 has a later optimal point (optimal $k_1 >$ optimal k_2) than the businesses curve k_2 because there are more users than businesses in the dataset and more variance in the type of user. Thus we need more clusters to optimally capture the different types of users than we do to optimally capture the different types of businesses.

Multi-Step Random Walks

The algorithm from [2] that we use in this paper was derived from using a 2-step random walk on the bipartite graph to allocate the recommendation power of a user u to another user v. However instead of just using a 2-step random walk, in this experiment we used a summation of the results of multiple random walks (2-step, 4-step, ..., k-step where k is an even number greater than 2) where the starting point of each walk is always a particular node u and the ending point of each walk is always a particular node v (each walk contributes a little bit of recommendation power to v from u). Thus mathematically we had:

$$rp(u,v) = \sum_{b \in B} \frac{r_{u,b}}{R_u} \frac{r_{v,b}}{R_b} + \alpha \Big(\sum_{b_1, b_2 \in B} \frac{r_{u,b_1}}{R_u} \frac{r_{k_1,b_1}}{R_{b_1}} \frac{r_{k_1,b_2}}{R_{k_1}} \frac{r_{v,b_2}}{R_{b_2}} \Big) + \alpha^2 \Big(\sum_{b_1, b_2 \in B} \frac{r_{u,b_1}}{R_u} \frac{r_{k_1,b_1}}{R_{b_1}} \frac{r_{k_1,b_2}}{R_{k_1}} \frac{r_{k_2,b_2}}{R_{b_2}} \frac{r_{k_2,b_3}}{R_{k_2}} \frac{r_{v,b_3}}{R_{b_3}} \Big) +$$
$$\cdots + \alpha^n \Big(\sum_{b_1, \ldots, b_n \in B} \frac{r_{u,b_1}}{R_u} \frac{r_{k_1,b_1}}{R_{b_1}} \cdots \frac{r_{k_{n-1},b_n}}{R_{k_{n-1}}} \frac{r_{v,b_n}}{R_{b_n}} \Big)$$

Note that in the equation above $n = k/2 - 1$. Also $k_1, \ldots, k_n$ represent arbitrary users in our network. Also, α is a decaying factor that is applied to the result of each random walk to ensure that $\sum_{v \in U} rp(u, j) = 1$. Using this new definition of $rp(u, v)$ as our new similarity function, a prediction $\widehat{r}_{u,b}$ of a user u's rating for a business b can be made as $\widehat{r}_{u,b} = \overline{r}_u + \sum_{v \in U} rp(u, v)(r_{v,b} - \overline{r}_v)$ where $\overline{r}_u = \frac{1}{B_u} \sum_{b \in B_u} r_{u,b}$. Thus, we ultimately ran several of these multi-step random walk versions (described above) of the algorithm from [2] on the original graph and graphed as a function of k (the number of steps in the largest random walk on the compressed graph) the error (again obtained through 90-10 cross validation) of the recommendation system:

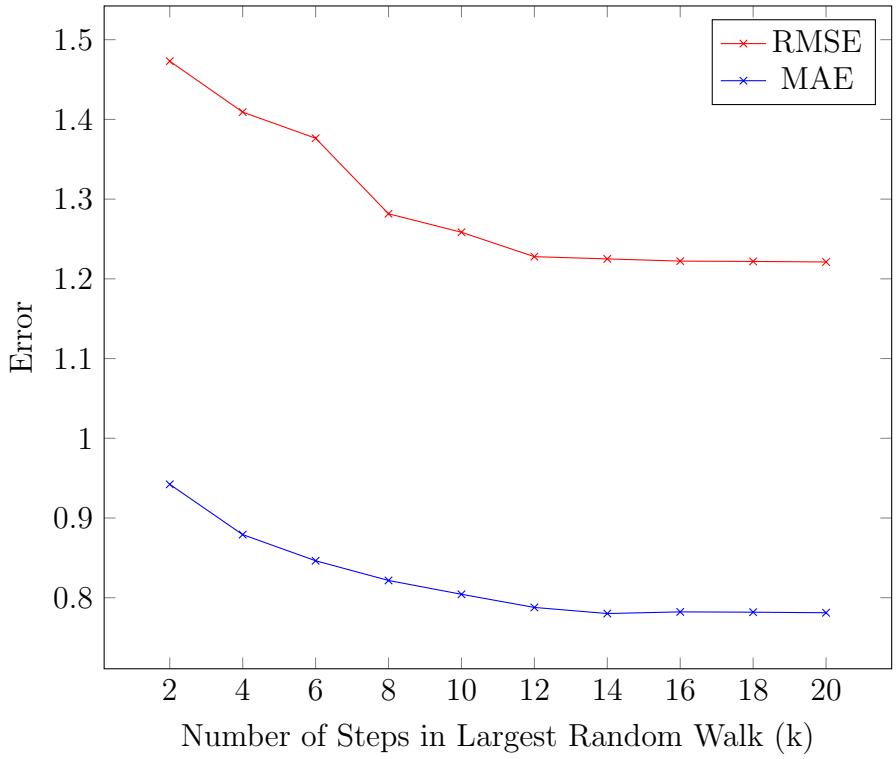


Figure 3: Multi-Step Random Walks Evaluation

Looking at the results, it seems that the error value stabilizes (converges and doesn't really change after this point) around $k = 12$ for a random walk. At that point on the graph, we approximately have that:

Optimal Multi-Step Error:

	RMSE	MAE
$k = 12$ onwards	1.2278148806157374	0.7878149806157374

Comparing this to our RMSE and MAE values (RMSE was 1.47526194112 and MAE was 0.94222332268) for running the algorithm from [2] in its original form, we can see that we have slightly improved by utilizing the multi-step approach. Furthermore, because the multi-step random walk approach involves the summation of the results of multiple random walks, it is extremely inefficient to compute. Looking at the mathematical expression for the recommendation power between two users (or equivalently businesses), we can see that a k-step random walk involves a summation over $k/2 - 1$ different combinations of user (or business) variables, making its time complexity $O(n^{k/2-1})$ since we need to iterate over the entire set of users (or businesses) for every variable in the summation bounds. This can be extremely expensive on large datasets. In essence, we want to avoid having to calculate random

walks for large values of k. Thus we can simply look at the value of k on the graph above where the error starts to converge and use that value of k as our "most accurate" k value. In this case, that value is $k = 12$.

We can also note the structure of the graph above. $k = 2$ all the way at the left of the graph is the same thing as our original algorithm from [2] since that algorithm utilized a 2-step random walk to allocate the recommendation power. As we increase the value of k, we add more and more terms (representing the results of n-step random walks where $2 < n \leq k$) to our expression for recommendation power (which represents the similarity between two users [or equivalently businesses]) making it more and more informative and accurate about the similarity measure that it presents since a user's recommendation power is based on more and more multiple random walk paths that were taken near the node representing that user in the graph. Thus we see a decrease in the error rate as we increase k on our graph above since the similarity measures between two entities become more and more informative. Because we use the decaying factor α (to ensure that $\sum_{v \in U} rp(u, j) = 1$ above), over time the amount of information we gain from doing an extra multi-step random walk decreases. Thus eventually our value starts to converge as we can see on the graph above, and adding the results of additional random walks (higher values of k) to the calculation of similarity (recommendation power) between two entities does not help anymore.

Also, since our graph is sparse, we do not have that many scenarios where two users have rated the exact same business (or vice versa) and thus no matter how informative our similarity statistic is between two users, we do not have enough data in the dataset to optimally calculate similarity. Because of this our improvement is not as drastic as when using experiment one (where RMSE was 1.13525525691294118 and MAE was 0.7022166384729412). The multi-step approach does do a decent job of accounting for this by utilizing the network properties (a random walk would go along the edges connected to a user) to calculate similarity between entities but we'd like to do better.

Hybrid Approach: Multi-Step Random Walks on Clustered Weighted BiPartite Projection

To make experiment two robust to sparseness, we also implemented a hybrid of the first two experiments, trying to take advantage of both types of strategies for making the weighted bipartite graph projection algorithm better. Thus we proceed as follows:

1. Run the clustered weighted bipartite projection algorithm from experiment one to obtain k_1^*, k_2^* which are the optimal number of clusters to be created for users and businesses respectively. Use these optimal k_1^*, k_2^* values to generate the compressed graph.
2. Use the multi-step random walk approach from experiment two on the compressed graph to obtain the optimal number of steps k^* that a random walk must take on the graph to decrease error as much as possible.

Thus to run this experimental approach, we first compressed the original graph into $k_1^* = 375$ user clusters and $k_2^* = 125$ business clusters. Then after running the experiment two approach on this compressed graph, we graphed as a function of k (the number of steps in the largest random walk on the compressed graph) the error of the recommendation system. Note that the error at $k = 2$ starts at the RMSE and MAE error that was found from experiment one since the $k = 2$ case is the same as running the original algorithm from [2] on the compressed graph (which is exactly what we did in experiment one):

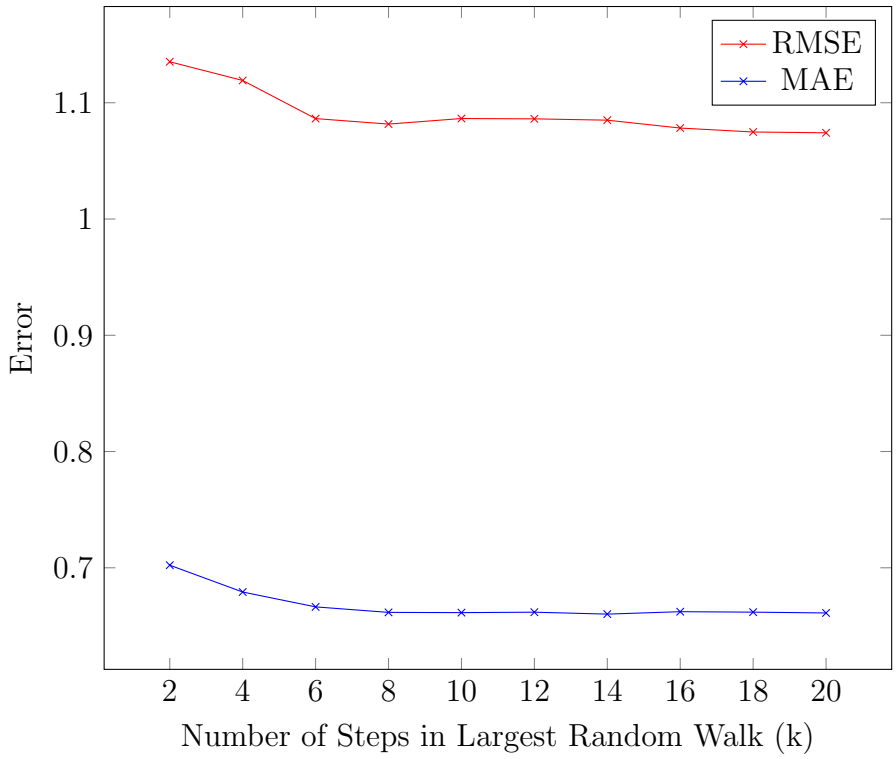


Figure 4: Multi-Step Random Walks on Compressed Graph Evaluation

As we can see above, the value of the error seems to stay at about the same value (converges) starting from $k = 6$. At that point on the graph, we approximately have that:

Optimal Hybrid System Error:

	RMSE	MAE
$k = 6$ onwards	1.08637482099	0.66637482099

Comparing this to our RMSE and MAE values (RMSE was 1.47526194112 and MAE was 0.94222332268) for running the algorithm from [2] in its original form, we can see that we have greatly improved by utilizing this hybrid approach. The shape of the curve can be explained by the same logic as in experiment two except that our initial value (at $k = 2$) is the same as the results of experiment one since we compress the graph before applying the multi-step approach. Here we can see that the error is much lower than in experiment two since we take advantage of the fact that the compressed graph accounts for the sparseness of the class, and thus the similarity measure between two entities on the compressed graph means more than it did with the original graph. Because of this, as we continue to improve the similarity measure to be more and more accurate on the compressed graph, we get recommendations that are even more accurate than before, effectively combining the sparseness-fighting effects of experiment one and the similarity-precision of experiment two in order to collaboratively create a very accurate prediction model.

Furthermore, besides showing an improvement in error, this approach is also much more efficient that the regular multi-step random walk approach from experiment two since we are working with a smaller, compressed graph and the actual error values converge much earlier on ($k = 6$ here instead of $k = 12$ in experiment two) enabling us to avoid doing $O(n^{k/2+1})$ calculations for most large values of k. It seems that we actually get a hybrid of the benefits of experiment one and two.

Conclusion

We first explored the results and effectiveness of the weighted bipartite projection algorithm from [2] as compared to the popular ways of using collaborative filtering to create recommendation systems such as Cosine Similarity, Pearson Correlation, and also compared our results to a naive baseline. We found that utilizing the regular properties and the network properties of our Yelp dataset, we could make much more accurate predictions that we could by just utilizing the direct dataset properties.

While the initial approach from [2] improved our prediction accuracy, we soon realized that our dataset suffered from a sparseness problem. In essence, the bipartite graph representation (where one set of nodes is users, one set is businesses and there are review edges from users to businesses) did not have many scenarios where two users had rated the same business. The algorithm from [2] was not robust to this and ultimately lost some accuracy since the similarity measures it calculated depended on the dataset having many scenarios where two users had rated the same business. In order to improve upon this and the general accuracy of the algorithm from [2], we conducted three different experiments.

Our first experiment used the k-means clustering algorithm to compress our graph into a bipartite graph where one set of nodes represented clusters of similar users, the other set of nodes represented clusters of similar businesses, and there was an edge from a user-cluster-node to a business-cluster-node if any of the users in the user cluster had rated any of the businesses in the business cluster. In this experiment, we found that with the right amount of clustering on the dataset, the intolerance to sparseness by the algorithm in [2] could be combated by essentially transforming our sparse dataset into our own non-sparse definition through some similarity mapping and grouping. Once we had a non-sparse dataset, the algorithm from [2] worked just as it should and gave very accurate results.

Our second experiment modified the definition of the algorithm from [2] to utilize the sum of multiple random walks instead of just a $2 - step$ random walk in its calculation of recommendation power (and hence similarity). While computing higher and higher step valued random walks made the similarity metric more accurate, it was also computationally expensive and could be approximated using the rough value of convergence. Utilizing this, we could get a much more accurate prediction, showing us that recommendation is greatly affected by the accuracy of the similarity function between the entities. However again, our algorithm was not robust to the sparseness problem. No matter how accurate the similarity measure calculation between entities was, it did not matter if those two entities were not closely linked in the dataset graph in any way at all. This led us to our third experiment.

Our third experiment combated the intolerance to robustness in experiment two by utilizing the compression idea from experiment one to map our dataset into a non-sparse compressed bipartite graph, and then running our multi-step random walk approach. Utilizing this hybrid approach we not only managed to get more accurate results (lower error), but we also managed to make our runtime faster since we were working with less data and the error values of our multi-step random walk approach converged much faster. We conclude that this novel approach can be used to efficiently take advantage of the network properties (and regular properties) of a dataset to accurately make predictions and be robust to the sparseness of the dataset.

Appendix

K Means Clustering

From [7], the k means clustering algorithm is an unsupervised machine learning algorithm used to group a set of data into a few clusters where each cluster has similar data. During the algorithm we maintain a set of centroids (representing the "average" point in a particular cluster) and also keep track of which data points are in which cluster/centroid. The algorithm is defined as follows:

1. Initialize cluster **centroids** by randomly picking points from our dataset to be the centroids.
2. Until convergence, we continually repeat the following process:
   (a) loop over all the data points x_i setting the centroid c_i of each data point x_i to be the closest centroid using some similarity metric (cosine similarity, etc.)
   (b) set each centroid to be the average of all the points in it

The algorithm converges when we can no longer move points from one centroid to another during the "centroid setting" phase. In our experimental approach above, we use the recommendation power between two users (or businesses which is mathematically equivalent) as the similarity function in k means clustering. To calculate the recommendation power between a user u and a centroid/cluster (which contains many users) we simply take the values of the recommendation power between the user u and each user in the centroid/cluster and average them.

References

[1] Zhou, T., et al., *Bipartite network projection and personal recommendation*, Physical Review E, 2007. 76(046115)

[2] Shang, M., Fu, Y., Chen, D., *Personal Recommendation Using Weighted BiPartite Graph Projection*, Apperceiving Computing and Intelligence Analysis, 2008. ICACIA 2008. International Conference on , vol., no., pp.198,202, 13-15 Dec. 2008

[3] Gunawardana, A., Shani, G, *Evaluating Recommendation Systems*, <`http://research.microsoft.com/pubs/115396/evaluationmetrics.tr.pdf`>

[4] Breese, J.S., Heckerman, D., Kadie, C., *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*, <`http://research.microsoft.com/pubs/69656/tr-98-12.pdf`>

[5] Koren, Y., *Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model*, <`http://public.research.att.com/~volinsky/netflix/kdd08koren.pdf`>

[6] Su, X., Khoshgoftaar, T. M., *A Survey of Collaborative Filtering Techniques*, Advances in Artificial Intelligence, vol. 2009, Article ID 421425, 19 pages, 2009. doi:10.1155/2009/421425,

[7] Ng, A. *Stanford CS229 Lecture Notes 7a*<`http://cs229.stanford.edu/notes/cs229-notes7a.pdf`>