Harvard Course Recommender System

Justin Kaashoek, Nicolas Lepore, Natalie Margulies

CS136, November to December 2019

1 Introduction

Recommender systems are prevalent throughout our daily lives, from YouTube recommending what video to watch next, to Spotify recommending what new song to listen to. Still, many areas remain that could make use of recommendation systems, one of which is a Course Recommendation system. The idea of course recommendation systems is not a novel idea, but it has yet to be applied to Harvard.

In this paper, we propose different approaches to create a course recommendation system that is specific to Harvard College. We believe this would be a valuable resource to Harvard undergraduates. There are many courses available to undergraduates, and it can often be overwhelming trying to select classes every semester. Students must also seek to fulfill requirements every semester, which adds another layer of difficulty in selecting courses. Especially considering the transition to different General Education requirements, students are often unsure what requirements they have yet to fulfill. With a course recommendation system, we seek to alleviate some of the stress of choosing classes. We hope to be able to eventually create a system that recommends classes to students that a student would find interesting while fulfilling the necessary requirements to graduate.

To accomplish this goal, we first discuss a theoretical implementation of a course recommendation system at Harvard. We discuss a recommendation system that is purely content-based and recommends classes based on a student's previous courses. We then discuss a system that is purely collaborative-based, where classes are recommended to a student based on how other students have rated similar classes. We conclude the section by discussing the theory behind a hybrid implementation of the two models as well as expectations and potential issues of an actual implementation. We then seek to implement a recommendation system based on self-generated data. We implement these purely content-based and a purely collaborative-based system, leaving a hybrid system for future work. Finally, we evaluate the results of each.

2 Theoretical

In this section, we discuss the theoretical implementation of a course recommendation system for Harvard.

2.1 Content-Based using Previous Courses

In a content-based recommender, user i's rating of class j is estimated based on finding classes that are similar to classes that user i likes (or rates highly) and dissimilar to classes that user i dislikes (or rates poorly). Therefore, we can use a student's previous courses, along with their ratings for such courses, to determine an effective recommendation of future courses.

In this content-based approach, we can define each class to have a content profile $w_j = (w_{j1}, w_{j2}, ..., w_{jK}) \in [0,1]^K$, for K different attributes, and each attribute k has a weight w_{jk} in the range [0,1], indicating the degree to which attribute k is associated with class j. For example, the attributes could represent the subject of a course, where the weights are 1 if the subjects match.

We chose the following attributes for our content profile: the subject (STEM or humanities), department the course falls under, whether there's a lab component, lectures per week, time slot, difficulty level, class size, whether it has a final exam or final project, number of midterms, and number of homework assignments, as well as a few other attributes.

2.1.1 Similarity Heuristic

Each user i can be modeled to have their own content profile $w_i = (w_{i1}, w_{i2}, ..., w_{iK}) \in [0,1]^K$, and each attribute k has a weight w_{ik} in the range [0,1], indicating the degree to which user i likes classes with attribute k.

To build our content user profile, we could survey a student about which subjects they enjoy, or infer based on their concentration. We could also use their previous ratings to infer their content profile by looking at their Q scores for each class that fits a particular genre and counting the number of likes vs. dislikes to construct a 0-1 vector to describe a user by threshold entries. High like-to-dislike would map to a 1 and low like-to-dislike would map to a 0.

We then estimate the rating $\hat{r}_{ij}$ on an class j for user i based on the distance between the user and class content profiles, w_i and w_j. We can use the cosine similarity measure:

$$\hat{r}_{ij} = \cos(w_i, w_j) = \frac{w_i \cdot w_j}{||w_i||_2 ||w_j||_2}$$

where the numerator is the dot product and the denominator is the product of each Euclidean distance.

Our ultimate goal is to display the best courses for a student given their content profile. Therefore, we can use the cosine similarity scores to order courses, and provide these in the course search tool of my.harvard.

2.1.2 Naive Bayes Classifier

Instead of creating user content profiles, we can create a probabilistic model for attributes of classes that a user likes or dislikes. Then, we can predict the probability that a user will like the class based on the class's content profile.

In the naive Bayes classifier, assume our attributes are binary, 0 for dislike and 1 for like. We parameterize the naive Bayes model with the probability $p_i \in [0, 1]$ that a user likes a random class and for each rating $s \in \{0, 1\}$, for each attribute k, $P(weight_k = 1|rating_i = s) = \theta_{iks}$ that an class with rating s has attribute k.

The probabilistic model for a class's content profile and the user rating works in the following way:

1. For a random class, user i's rating is 1 with probability p_i, and 0 otherwise.

2. If the user's rating is 1 then each attribute k has weight 1 with probability θ_{ik1} and weight 0 otherwise; if the user's rating is 0 then each attribute k has weight 1 with probability θ_{ik0} and weight 0 otherwise;

This is naive because each attribute associated with an item is sampled independently.

We can then train the model separated for each user based on the content profiles of classes they like or dislike. We predict rating of user i for class j based only on the content profile of the class. We decided not to implement this version of content-based recommender, because we would be fitting to randomly generated data, and it wouldn't be too useful to learn.

2.2 Collaborative Filtering using the Q Guide

Collaborative filtering involves collecting ratings of classes by other students, and using similar students (user-user) or similar classes (item-item) to predict how our specific student would rate a certain class. This is useful to know since a class that a student may rate highly is one we would like to recommend to the student.

2.2.1 User-User

For user-user collaborative filtering, we can estimate a user i's rating on item j in the following way:

1. Finding a neighborhood N_i of users, consisting of users who are similar to user i and have rated item j.

2. Estimate user i's rating for item j as the similarity-weighted average rating on item j by users in neighborhood N_i, by

$$\hat{r}_{ij} = \frac{\sum_{i' \in N_i} sim(i, i') r_{i',j}}{\sum_{i' \in N_i} |sim(i, i')|}$$

where $sim(i, i')$ is the similarity measure between users i and i'.

In this way, we can create a neighborhood of like-minded students, and we can test different neighborhood sizes to see which is the best. We modeled this problem for a small number of students, but it is also interesting to explore how this scales to a few thousand students. We can use optimization techniques to decide the best neighborhood size, and from there, create accurate ratings.

2.2.2 Item-Item

For item-item collaborative filtering, we estimate user i's rating on class j in the following way:

1. Find the neighborhood N_j of classes, which consists of classes that are similar to class j, and have been rated by user i.

2. Estimate user i's rating for class j as the similarity-weighted average rating by user i on the classes in neighborhood N_j,

$$\hat{r}_{ij} = \frac{\sum_{j' \in N_j} sim(j, j') r_{i,j'}}{\sum_{j' \in N_j} |sim(j, j')|}$$

where $sim(j, j')$ is the similarity measure between classes j and j'.

Here, we can create a neighborhood of similar courses, and we can test different neighborhood sizes to see which is the best. We modeled this problem for a small number of courses, but it is also interesting to explore how this scales to a few hundred courses. We can use optimization techniques to decide the best neighborhood size, and from there, create accurate ratings.

An issue here is that courses are can be more uniquely different than students. Each course at Harvard is meant to cover a different topic, so it is sometimes hard to draw similarities between them, even if they are in the same department. On the other hand, many students can have very similar courseloads, and this helps us figure out the kind of neighborhood they fall into.

2.3 Hybrid Approach

There are four typical approaches in combining content-based and collaborative filtering in recommendation systems.

1. Ensemble approach: We take some combination of recommendations from the different systems. We could also use different systems in different situations. For class recommendations, this could look like recommending the class that has the highest average recommendation out of the following: a content-based recommendation using a cosine similarity heuristic, a content-based recommendation system using a Naive Bayes Classifier, and a user-user collaborative filtering recommendation.

2. Calculate a recommendation based on student-student collaborative filtering where student profiles are content-based, or do the opposite, and calculate course-course collaborative filtering where course profiles are content-based.

3. Hard code domain knowledge into the recommendation system. For example, that stem majors may not like humanities classes as much.

4. Combine course attributes and student profiles in a single framework and predict how much a student will like a class using a deep learning approach.

It is unclear which of these approaches would be best in practice.

2.4 Incorporating Requirements

Any of the systems previously discussed can easily be adapted to take into account requirements by removing classes from the recommendations that fulfill a requirement that has already been fulfilled. A drawback to this approach is that we may remove classes that a student would find very interesting. We could solve this problem by allowing a user to input a preference into the system that toggles whether we factor requirements into the recommendation. If the student wants to see only interesting classes regardless of requirements, we run the system as normal. If they want to take required classes that semester, we remove any classes that do not fulfill a requirement that the student has not already fulfilled.

Another option to incorporate requirements is as follows. Firstly, we can eliminate all classes that a student has taken before, similar to how Netflix does not recommend movies that have already been viewed. Then, in the content-based approach, we can give their content profile a value of 0 for attributes that represent that a class fulfills a certain GenEd. For concentration requirement classes that are typical to take in freshman or sophomore year, we can add an extra weighting in favor of these classes for underclassmen. In addition, we must factor in pre-requirements for other classes, and only show classes in which a student satisfies the pre-reqs. We can even show the classes that then become available once a student fulfills pre-requirements.

2.5 Expectations

Mainly due to the lack of available data, which is discussed in greater detail in the following section, we keep our expectations for an implementation of the previous theoretical discussion relatively low. We believe that reasonable expectations include building a pure content-based and a pure collaborative filtering system, acknowledging that we will likely have to randomly generate our data. Our results, then, will not have much meaning. We believe we can implement both a user-user and item-item collaborative system. We do not expect to be able to implement a hybrid approach because we would need to make a number of assumptions on which of the four hybrid approaches we would implement and do not know which one would work best for a course recommendation system.

2.6 Potential Issues

1. Data Collection

   We decided to mainly create our own data for this project, for a variety of reasons:

   - Harvard is not allowed to release registrar course data for privacy issues, and obtaining the information directly from students would pose a participatory concern, perhaps introducing survey bias into the system as well.
   - For the Q Guide, recommender systems are not designed to work with aggregate data. The systems require the individual ratings from each user to determine a recommendation because each recommendation is tailored specifically to the user. However, the Q Guide data that is shown is aggregated and anonymized, and upon requesting any form of tabulated data from Harvard, they denied us access.
   - Acquiring individual Q scores from students is rather infeasible, considering the lack of incentives that we can provide, other than monetary incentives. However, one useful incentive is giving students access to this recommender system at the beginning of each semester, since this would save them time, thus being valuable. The school incentivizes participation in the Q Guide by allowing students to view their grades earlier if they provide the feedback.

2. Lack of Class Exploration

   Many students find their favorite classes rather serendipitously, and a recommender system may take that phenomenon away from students. If students don't have to pore over the course search tool, they may all end up taking very similar classes, with higher Q scores and the best professors, even if every recommendation is tailored individually. This is similar to the "Harry Potter"

effect that Amazon observed in its recommender system. This may not be advantageous for Harvard's business model, and could hurt academia.

However, the Netflix recommender system accounts for this by randomly recommending rather serendipitous titles, and our recommender system could do this as well.

3 Implementation

3.1 Algorithm

We implemented this theoretical framework as described below.

3.1.1 Content-Based Implementation

We created class and student data for our content-based approach. The class data was randomly assigned the attributes stem/humanities, department, level (0, 100s, 200s), class size, time slot (just categorical values of the order of time slot), lab component (whether it has a lab component), times per week, final exam (whether it has a final exam), midterms (number of midterms), final project (whether it has a final project), and homeworks (number of homeworks).

We then put these values in the range of [0,1] by using dummy variables if the attribute was categorical, and a range between [0,1] if it was quantitative. The student data was generated based on sample Q Guide data that we randomly generated. We initially assign random weights for all attributes that classes are rated on, showing with what intensity they value a certain attribute. We then update the random weights based on two factors:

1. Major: If the student is declared as a STEM major, we assign a higher weight for STEM classes. If they are a humanities or social science major, we assign a lower weight.

2. GPA: If the student has a low GPA, we assign a higher weight for classes below the 200 level. We reason that students who are doing poorly in their classes will not want to take difficult upper-level classes. We assume that students with a high GPA could value classes at all levels equally.

There are a number of other factors that we could have considered to manipulate weights, but we believe these two to be the most important.

Lastly we used cosine similarity to see the best matches between students and classes, and we showed which classes we would recommend to students in descending order.

3.1.2 Collaborative Filtering Implementation

We randomly created a ratings matrix, and replaced 0's with NaNs to signify that the user had not rated that item.

For user-user (student-student) and item-item (course-course), we centered and normalized our ratings to obtain our baseline rating, then we compared these baseline ratings to other students for user-user and other courses for item-item.

After finding the neighborhood in order of similarity, we took the closest few neighbors to calculate the ratings, which was the sum of the similarities between our user/item and the neighbor user/item multiplied by their rating. We then divided this product by the sum of absolute value similarity between our user/item and the neighbor user/item, following the formula given in Section 2.2.1 and 2.2.2. This gave us our rating for a specific user and item.

3.2 Results: Analysis of Algorithm Accuracy

We analyzed the efficacy of our each recommender systems by testing the theoretical recomender values against the randomly generated actual data.

3.2.1 Accuracy of Content-Based Filtering

Unfortunately, analyzing the efficacy of content-based filtering with given weights $w_j = (w_{j,1}, w_{j,2}, ...)$ is largely impossible without verified data because we don't know the weights w_j that each student assigns each criterion. We found that modifying weights based on concentration and GPA had no impact on recommendations. This indicates a shortcoming of our content-based recommendations and is likely due to the fact that there are so many class attributes. Changing only a few weights, then, has limited impact on the final outcome. Given limitations in Q Guide data collection and bias in survey data, we are unable to produce a verifiable actual ranking of classes for each student in the randomly generated data set. Verification of content-based filtering would require a deeper understanding of the weights each student ascribes to each descriptor.

3.2.2 Accuracy of Collaborative Filtering

Although we didn't have verified historic data to assess the accuracy of collaborative filtering rankings, we decided to compare the original rankings with the projected outputs of both collaborative filtering models. We first converted each of the rank profiles into a weak preference ordering and then compared the accuracy of two sets of rankings by taking the square root of the average of the squared deviations of the ranking positions from their final value. We compare the efficacy of the recommendation in the collaborative filtering recommendation system across niterations = 15

on a randomly generated sample size of n = 10 students. *Figure 1* shows a sample recommendation output from one bootstrap trial and a sample calculation of the difference in ranking. Across all bootstrap iterations, we find that the average standard deviation between actual and predicted output to be relatively small (approximately 0.76).

Actual Preference Ordering:

Student	Preference Ordering
0	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C1 \succ_{s_0} C0 \succ_{s_0} C6 \succ_{s_0} C4$
1	$C0 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C2 \succ_{s_0} C3$
2	$C3 \succ_{s_0} C0 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
3	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C4 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
4	$C6 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C0$
5	$C0 \succ_{s_0} C1 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C7 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
6	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3 \succ_{s_0} C5$
7	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C0 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3 \succ_{s_0} C6$
8	$C3 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C2 \succ_{s_0} C4$
9	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C0 \succ_{s_0} C1 \succ_{s_0} C2 \succ_{s_0} C3 \succ_{s_0} C5$

User-User Prediction Preference Ordering:

Student	Preference Ordering
0	$C2 \succ_{s_0} C5 \succ_{s_0} C7 \succ_{s_0} C3 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C0 \succ_{s_0} C6$
1	$C0 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C2 \succ_{s_0} C3$
2	$C3 \succ_{s_0} C0 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
3	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C4 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
4	$C6 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C0$
5	$C0 \succ_{s_0} C1 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C7 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3$
6	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3 \succ_{s_0} C5$
7	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C0 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C3 \succ_{s_0} C6$
8	$C3 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C5 \succ_{s_0} C6 \succ_{s_0} C1 \succ_{s_0} C2 \succ_{s_0} C4$
9	$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C0 \succ_{s_0} C1 \succ_{s_0} C2 \succ_{s_0} C3 \succ_{s_0} C5$

Figure 1: Example Preference Ordering Conversion from a Bootstrap Iteration

Actual preference ordering S0:

$$C2 \succ_{s_0} C7 \succ_{s_0} C5 \succ_{s_0} C3 \succ_{s_0} C1 \succ_{s_0} C0 \succ_{s_0} C6 \succ_{s_0} C4$$

Theoretical preference ordering S0:

$$C2 \succ_{s_0} C5 \succ_{s_0} C7 \succ_{s_0} C3 \succ_{s_0} C1 \succ_{s_0} C4 \succ_{s_0} C0 \succ_{s_0} C6$$

$$\sigma_{S0} = \sqrt{\frac{0+1+1+0+0+(2)^2+1+1}{8}} = 1$$

Equation 1: Calculation of Standard Deviation for S0

3.2.3 Comparison of User-User and Item-Item Collaborative Filtering

We compare the efficacy of in recommendation between user-user and item-item collaborative filtering recommendation system across niterations = 15 on a randomly generated sample size of n = 10 students in order to test which algorithm gives more consistent results with the actual ranking. Since rating were randomly generated in range (1, 5) with equal probability, we would expect user-user and item-item collaborative filtering to yield similar mean rankings and mean differences across multiple bootstrap trials by the Law of Large Numbers. *Figure 2* shows that the mean ratings across both collaborative filtering systems are consistent across multiple trials. The mean difference between predicted and actual rankings straddles 0, with an average difference of 0.112 across all iterations.

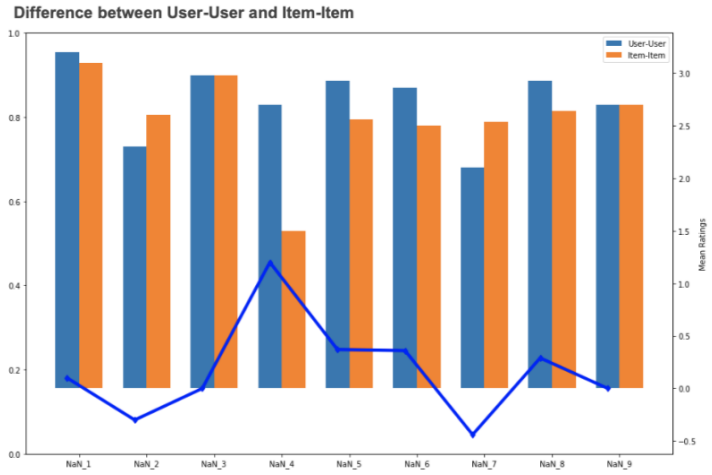
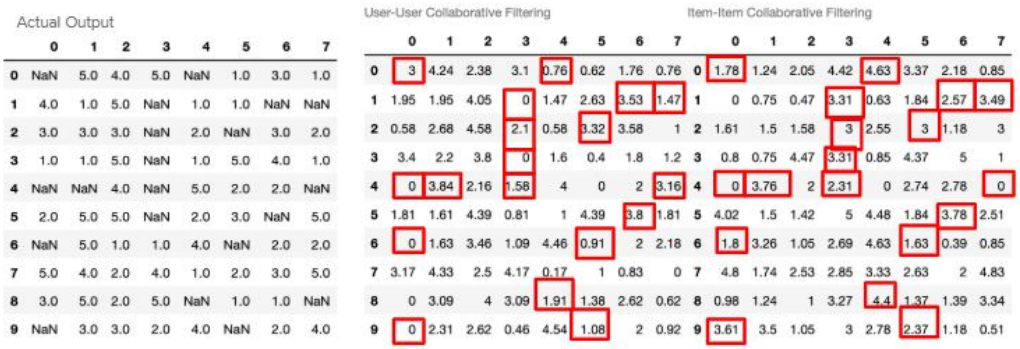


Figure 2: Comparison of Mean Difference and Mean Ratings across User-User and Item-Item Based Collaborative Filtering

3.3 Results: Theoretical Implementation

In 3.2, we used all theoretical rankings produced by collaborative filtering in order to verify the accuracy of the algorithm. In this section, we walk through a theoretical implementation of how the algorithm would work. Once we generate predictions of all classes, we would use only the NaN predictions to generate a preference ordering for classes the students has not taken as shown below in *Figure 3* through red highlighting of untaken classes.



Actual Output

	0	1	2	3	4	5	6	7
0	NaN	5.0	4.0	5.0	NaN	1.0	3.0	1.0
1	4.0	1.0	5.0	NaN	1.0	1.0	NaN	NaN
2	3.0	3.0	3.0	NaN	2.0	NaN	3.0	2.0
3	1.0	1.0	5.0	NaN	1.0	5.0	4.0	1.0
4	NaN	NaN	4.0	NaN	5.0	2.0	2.0	NaN
5	2.0	5.0	5.0	NaN	2.0	3.0	NaN	5.0
6	NaN	5.0	1.0	1.0	4.0	NaN	2.0	2.0
7	5.0	4.0	2.0	4.0	1.0	2.0	3.0	5.0
8	3.0	5.0	2.0	5.0	NaN	1.0	1.0	NaN
9	NaN	3.0	3.0	2.0	4.0	NaN	2.0	4.0

User-User Collaborative Filtering

	0	1	2	3	4	5	6	7
0	3	4.24	2.38	3.1	0.76	0.62	1.76	0.76
1	1.95	1.95	4.05	0	1.47	2.63	3.53	1.47
2	0.58	2.68	4.58	2.1	0.58	3.32	3.58	1
3	3.4	2.2	3.8	0	1.6	0.4	1.8	1.2
4	0	3.84	2.16	1.58	4	0	2	3.16
5	1.81	1.61	4.39	0.81	1	4.39	3.8	1.81
6	0	1.63	3.46	1.09	4.46	0.91	2	2.18
7	3.17	4.33	2.5	4.17	0.17	1	0.83	0
8	0	3.09	4	3.09	1.91	1.38	2.62	0.62
9	0	2.31	2.62	0.46	4.54	1.08	2	0.92

Item-Item Collaborative Filtering

	0	1	2	3	4	5	6	7
0	1.78	1.24	2.05	4.42	4.63	3.37	2.18	0.85
1	0	0.75	0.47	3.31	0.63	1.84	2.57	3.49
2	1.61	1.5	1.58	3	2.55	3	1.18	3
3	0.8	0.75	4.47	3.31	0.85	4.37	5	1
4	0	3.76	2	2.31	0	2.74	2.78	0
5	4.02	1.5	1.42	5	4.48	1.84	3.78	2.51
6	1.8	3.26	1.05	2.69	4.63	1.63	0.39	0.85
7	4.8	1.74	2.53	2.85	3.33	2.63	2	4.83
8	0.98	1.24	1	3.27	4.4	1.37	1.39	3.34
9	3.61	3.5	1.05	3	2.78	2.37	1.18	0.51

Figure 3: Conversions of NaNs to Rankings

We see that user-user and item-item recommendations generally agree with each other, which is as expected given our results in section 3.2.3.

4 Conclusion

We are successfully able to provide course recommendations for content-based and collaborative filtering systems. Although it is difficult to draw any meaningful conclusions based on these recommendations, the fact that we were able to generate this system is a success in itself. The system is a long way from being used in practice, however. It is possible that the separate content-based and collaborative filtering systems would work well independently, but in practice, we would want to use a hybrid approach.

One of the biggest issues that we ran into was the lack of data. We spent a lot of time generating our own data, hoping to make it somewhat meaningful, whereas if we had available data from the Q Guide or our own survey of students, we could have made major improvements to the system. Before a system such as this is used in practice, we would have to test it on such data. Despite these shortcomings, we believe that we have laid both the theoretical and practical groundwork for a system that could actually be used in the future.