CourseFinder

Final Report

Wen-Hao Lee
California Institute of Technology
wllee@caltech.edu

Jamie Jackson
California Institute of Technology
jackson@caltech.edu

Isaac Sheff
California Institute of Technology
isheff@caltech.edu

ABSTRACT

Collaborative filtering is a tried and true technique to provide useful recommendations based upon a user's past ratings as compared to those of others. In reality, however, people often rely on their friends' advice, and this can be reflected in their preferences.

At the same time, unified and easy to use course feedback and rating systems are woefully lacking or underutilized at a variety of Universities, including the California Institute of Technology (Caltech).

To this end, we present CourseFinder, a facebook app for rating courses taken at Caltech. Through this, various social network / collaborative filtering recommendation algorithms can be tested. We found that a metric of pure social weight (number of mutual friends over average number of friends on facebook) as a similarity coefficient between pairs of users provided better course recommendations than equal weightings (just, in effect, highest average rating recommendations), but the tried-and-true Pearson Weighting for traditional collaborative filtering remains superior. An attempted blend of the two yielded mediocre results.

Keywords

Collaborative Filtering, Social Network, facebook, Course Review, Caltech

1. MOTIVATION

Many businesses consider it a huge boon to be able to accurately predict which products a customer may want. They have worked so hard at this that they've actually generated hidden networks of users sorted and linked by their mutual interests. Traditionally, such networks are formed using learning algorithms that attempt to establish similarities between the preferences of sets of users.

Consider, however, that the actual likes and dislikes of customers are highly affiliated with that user's social network. A product enjoyed by a user's friends is probably more likely to be enjoyed by that user. Traditionally, people seek out advice in such decisions from those closest to them on a social network. In fact, a study by Rashimi Sinha and Kirsten Swearingen at UC Berkely found that amongst books and movies, recommendations by friends were 30-40% more likely to be "Good" or "Usefull" than traditional network recommendation systems [1].

Enter course recommendations. In a collegiate environment, especially at large institutions with extensive catalogues, students often rely on friends' recommendations to make important, periodic decisions: which courses to take. Students consider themselves more likely to enjoy courses their friends have enjoyed, with professors their friends have found effective.

This provides an excellent testing ground for such a hybrid recommendation system, which would use traditional learning algorithms to suggest courses based on a student's past experience, compared with that of similar students, weighted accordingly for the distance of other students in the social network.

2. PRIOR RELATED WORK

Many traditional recommendation systems, such as Netflix, make use of Collaborative Filtering, in which a database of users' ratings of items is used to predict what a given user will rate an item based upon their past ratings [2]. A traditional, "Memory-Based CF" system calculates a correlation factor between all users, and possibly all items, based on users existing rankings, and predicts rankings of a user's unranked items using the correlation weighted average of other users' rankings. It then recommends the top-ranked ones. There are a wide variety of alterations and improvements upon this simple idea for specific circumstances, computational constraints, and applications.

One such subset is the neighbor-based collaborative filtering model, in which a weighted average of rankings is taken only from a limited number of users most similar to the user in question. This is in some ways analogous to a limited "friend" group on a social network, but the "neighborhood" is made by the algorithm, not the user.

Recommendation networks, however, do not take into account other factors in a user's life that can drive decision-making besides prior experience in the narrow field of whichever item type is being recommended. While many recommendation systems seek to account for such factors, generalization is extremely difficult. However, users seeking recommendations from friends receive an advantage over anonymous systems: trust. A test done at the Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, found that users are more likely to receive better recommendations from other users they trust more, and that to a large extent, friendship on social networks (they used Orkut), mimics trust [3].

The idea of social-network based recommendations is not new. For example, the experimental service FilmTrust attempts to utilize Memory-Based CF calculating the similarity weighting between users as the trust between those users using the FOAF trust model [4]. Synclab Consulting's Hooks App for facebook recommends music found in the libraries of users' friends with many mutual songs in their playlists [5]. A study at the University of Illinois Champaign-Urbana found 95 % of users on an experimental social network-based news recommendation system to be "somewhat to very useful" [6].

As of yet, however, there appears to be a lack of course recommendation services making use of social networks, and a lack overall of social network ranking combined with collaborative filtering weighting users by both similarity and social distance.

3. COURSEFINDER

3.1 Facebook App

CourseFinder a facebook app wherein users can quickly sort through and rate Caltech courses. For each course and instructor users can rate quality as well as provide brief comments. There are also separate fields to recored Grading ratings, grades riecieved, and hours per week spend on the class. Comments are visible only to their friends, but can be made anonymously public. The overall course ratings are used in a CF recommendation system to predict what courses each user would also find to be of high quality, taking into account their past ratings compared with those of other users, as well as their social relations to those other users. This recommendation system is as yet pure experiment, and the algorithmic estimates on the app itself are merely averages.

3.2 Recommendation Algorithm Validation

To evaluate the effectiveness of a particular recommendation algorithm over the course of this project (since there is insufficient time for users to take new classes, and respond with how well prediction matched reality), an assessment was made of the average correlation between predicted ratings and actual ratings (average of "how would this have been predicted to have been rated had I not rated it over how I did in fact rate it"). This procedure is referred to as "leave one out" validation. These correlations can be compared to pure, traditional, Pearson Correlation [2] predicitons, as well as random predictions, and average ratings.

3.3 Data Source

Data was acquired from 168 Caltech students who volunteered to rate courses via CourseFinder after publicity emails were sent out to the Undergraduate and Graduate communities. These emails offended the Academics and Research Council, who felt CourseFinder would detract from traditional TQFR review responses. No further publicity was sought after their opinions were made clear.

4. BACK END

To host a Facebook application, a web server must provide reliable access for all users of that application to a web page. This web page is displayed inside Facebook's frame, but the actual access is directly from the user's browser to the host's web server. The web server itself as well as the javascript in the web page are granted access to Facebook's Graph API, with which they can access user information and send out wall posts, messages, and the like. Google's App Engine cloud service hosts CourseFinder.

4.1 Google App Engine

Google's App Engine service was selected as CourseFinder's host because of its reliability, flexibility, and budget constraints.

App Engine is a cloud service, meaning it is maintained as a distributed system across several large data centers owned and operated by Google. As such, the probability of any "App" it runs becoming unavailable at any one time is extremely small. Additionally, the large number of available servers means that while intense average use may be equally costly, there is little danger of usage spikes overloading the server system, as opposed to a single server just for CourseFinder, which would likely have great difficulty if a large number of users decided to use the app at once. The combination of extremely low downtime and high spike tolerance make App Engine much more reliable than alternative, more traditional hosts, such as Caltech's notoriously poorly maintained UGCS, or a dedicated server.

Several aspects of App Engine make it extremely flexible. For example, App Engine allows services written for the Java Runtime Environment and Python. CourseFinder is written in Python because, as a pure scripting language, and an incredibly flexible one, it allows great ease of manipulation and modularization of services. App Engine's support for Python made it a strong candidate. Additionally, because App Engine uses a noSQL hash table based database structure, data storage of any size is equally fast. In fact, aside from financial concerns over larger server usage, CourseFinder should be able to expand to an arbitrarily large set of institutions and users without speed or algorithmic problems. Furthermore, App Engine allows simultaneous deployment of different versions, allowing multiple tests to be run on the production server without interfering with regular service. This is extremely useful when, for example, different developers are simultaneously testing various new features and need to do so with real or large simulated data.

As a small endeavor, CourseFinder has few resources with which to acquire, set up, run, and maintain a dedicated server facility. In effect, it requires a free service to run. While several are available, including Caltech's own UGCS, App Engine's free service offers more than CourseFinder is likely to require in the near future, and this on top of its other advantages made it the host of choice.

There are some penalties for this. For example, while eventual consistency of data is assured, it is possible that reads from the database that take place shortly after writes to it will not pick up the new data. In the case of CourseFinder, this is not terribly troublesome, as it is not particulary important that data portrayed by "up to the second." A more vexing problem is the noSQL database structure, which makes some forms of data relationships more difficult. Most notably, queries to the database cannot be nearly as complex, with serious problems concerning logical "OR" operations. Again, these are usually minor problems for a system with fairly straightforward data storage and retrieval.

CourseFinder is hosted at
http://courserecommendation.appspot.com.

4.2 Django

Django is a popular web development framework based in Python and built to run atop a variety of server setups. Django-nonrel, built for noSQL databases such as App Engine, was selected as a framework for CourseFinder because of its portability and modularity.

Django is built to be extremely portable. A running instance of Django, called a "project," need only change a few settings variables to switch servers, databases, and even database structures. As a python framework, it is in no way platform specific, and it is entirely open source. Even within a project, the actual services offered, or "apps," can be moved between projects with relative ease. One fortunate side effect is that it is extremely easy to run an instance of Django exactly as it would be run on App Engine on a local machine for testing purposes. It is equally easy to deploy to the production server. App Engine in particular has a number of features built in that are borrowed from Django, such as models and templates (and their associated syntax). Because of these, Django is an especially attractive option when developing on App Engine, but CourseFinder could relatively easily be modified to run on most any other server capable of running Django, should it at any time become a superior option to do so.

Django is also extremely modular, which is useful when developing along separate tracks or as a team. It allows easier division of tasks, and overall more efficient rapid development. For example, rather than a traditional url to file hierarchy structure, Django replies to each HTTP request with the output of a function called a view, which is any function that takes in the HTTP request data and returns a web page. Which view is determined by a regular expression table. This means that it is very easy to add, subtract, and manipulate the "structure" of urls independently from the construction of actual pages. It allows, for instance, the extremely simple "/course/<course number here>/" urls, as opposed to some more complicated PHP scheme with GET url inputs. As another example, django's template system allows a well-designed seperation between content and formatting on the server side by allowing a view to fetch content, and pass it to a template, where the python content is easily inserted into the html template using " " tags, including basic flow control such as loops. This type of system allows, for instance, one developer to focus on content fetching and generation, while another focuses on formatting and presentation.

4.3 Data Structure

The data for Corusefinder is stored in the form of Django Models, for which a similar system is implemented natively for App Engine. Models, rather than the traditional way of thinking of data in tables, are simply objects of specific classes for which certain attributes, called fields, can be saved to the database and searched for in order to retrieve the object again. The types of fields used are CharField (string), DataTimeField, IntegerField, FloatField, URLField, ForeignKey (a link to another object stored in the database), and ListField, a field type specific to App Engine, which stores a list of any other kind of field. Notably, App Engine, as a noSQL database, cannot hand Many-to-Many fields, which store multiple other objects in the database. ListFields of ids, such as the ids of teachers for a course, are used instead.

Django Models also support abstract inheritance. This allows abstract classes controlling common model features to be easily built upon or modified as may be needed. For example, there is an abstract Rating model, which allows the three different kinds of ratings (Teacher, Grading, and Overall), to only specify what differentiates them, and any common aspects of ratings to be changed easily. All models used in CourseFinder inherit the abstract model DateStamped, which ensures they all have a created DateTime storing the creation time of that object, and an updated DateTime, storing the last updated time of that object.

CouseFinder uses the fandjango django app, by Johannes Gorset, to store and update facebook user data.

5. FRONT END

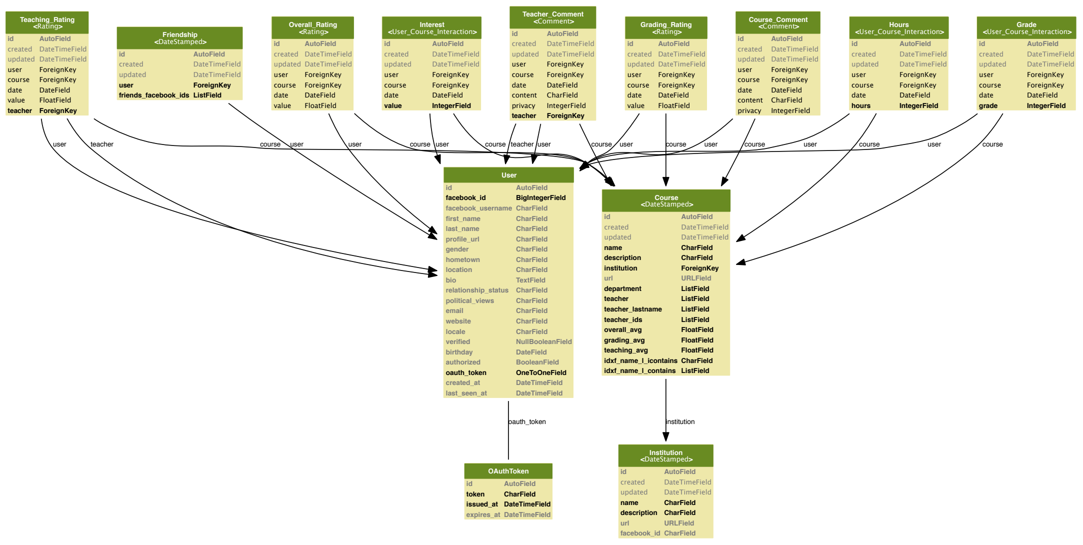
A facebook app such as CourseFinder is a website displayed inside of an iFrame on the facebook app page. In this case, the iframe at `http://apps.facebook.com/coursefinder` displays the CourseFinder web page, located at
`http://courserecommendation.appspot.com/`.

5.1 Main Page

5.1.1 Recommendation List

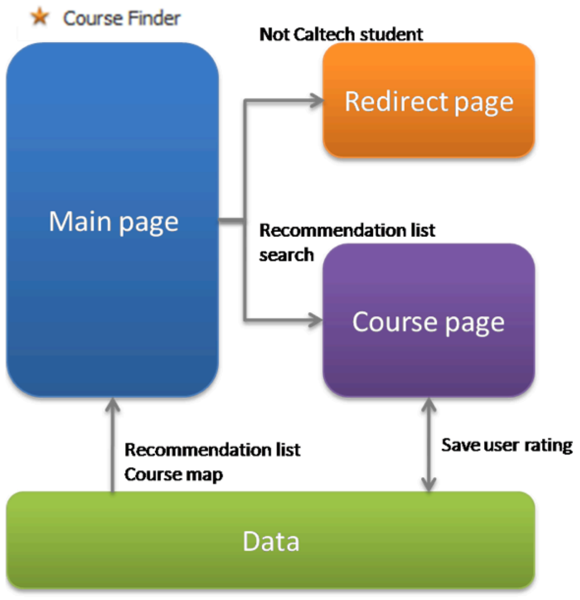
Top recommendation courses are calculated from overall rating averages. Candidates are selected from the same department that user has rated. We use 3D tag cloud by JQuery to present the recommend list. First, setup the mouse event and figure out how far the mouse is away from the center and assign it to a variable controlling the speed of the scrolling list. Then step through each element in the list and give each one a spot on 3D circle. For loop walks through each

Figure 1: Data Structure



Data structure UML for CourseFinder. Note that department is a list of string department abbreviations, and teacher_ids is a list of valid ids of User objects. Also please note that User and OAuthToken are from fandjango, but Johannes Gorset, and that the hideousness of UML is not my fault.

Figure 2: Site Map



one, and assigns an angle to each element. Finally, call the render function repeatedly.

5.1.2 *Search Course*

Search courses from the database by filtering given specific input. Input can be department only, last name of the professor only, part of the course name only or department with course number. Search results will be displayed right after this section. Search results lists include each courseÕs estimated rating and provide rating bars for user to click.

5.1.3 *Course Map*

From courses user has rated, we present userÕs personal course map by Flash chart. Open flash chart project here provides following advantages:

1. Tooltips encourage user interactivity and data exploration
2. We can save the chart as an image
3. Quick and easy to plot radar chart

First, we gather the userÕs rating from the data base. Calculate number of courses in each department. Select top numbered departments and write the data to the json file for each user. Open flash chart reads the json file and generate the radar Flash chart as course map.

5.2 Course Page

`http://courserecommendation.appspot.com/course/<courseid>`

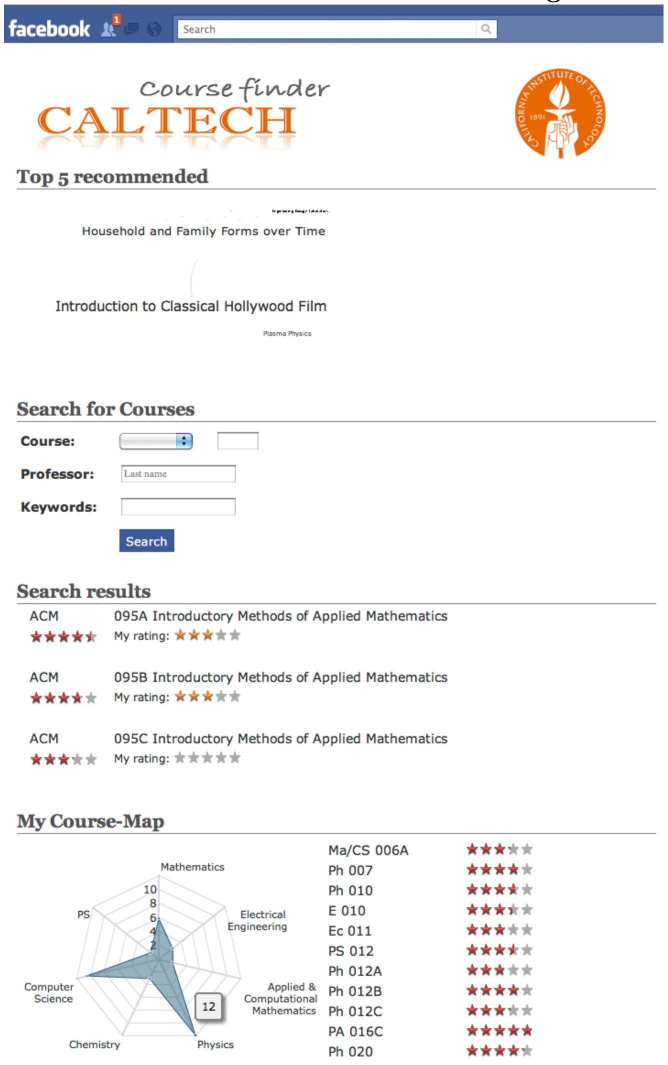
5.2.1 *Algorithmic Estimated Rating*

Currently, we use the averaging rating as predicted rating. The rating values are stored between 0 and 1. Convert the value to percent in total to tell how many golden stars are displayed. The result shows the rating bar by CSS with gray star background images.

5.2.2 *Rating Items*

For user submitting ratings for each item, we use Mootools to deal with it asynchronously. The Javascript generates

Figure 3: Layout



facebook Search

Course finder CALTECH

Top 5 recommended

Household and Family Forms over Time

Introduction to Classical Hollywood Film

Plasma Physics

Search for Courses

Course:	
Professor:	Last name
Keywords:	

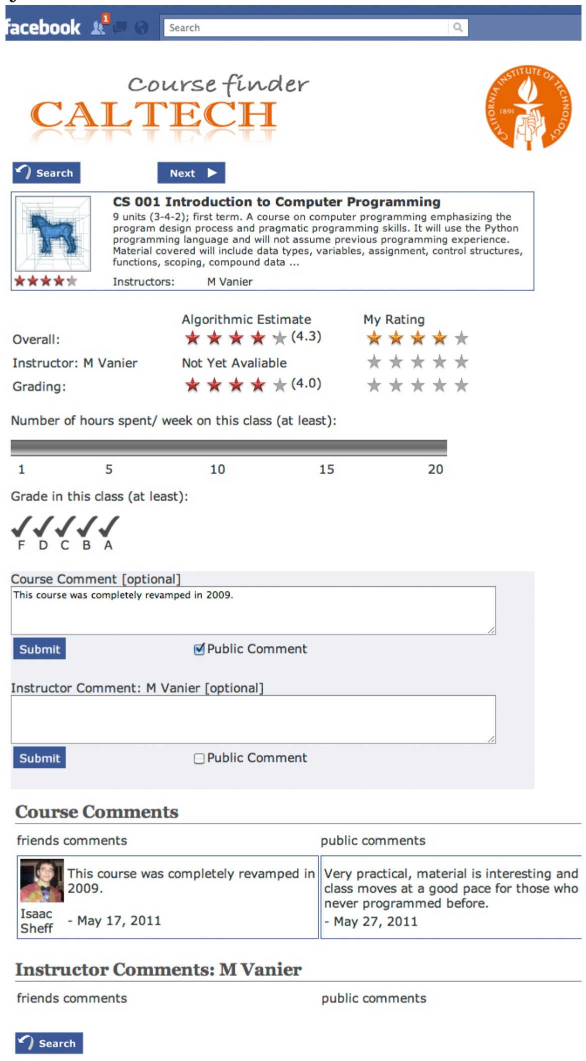
Search

Search results

ACM	095A Introductory Methods of Applied Mathematics
★★★★★	My rating: ★★★☆☆
ACM	095B Introductory Methods of Applied Mathematics
★★★★☆	My rating: ★★★☆☆
ACM	095C Introductory Methods of Applied Mathematics
★★★☆☆	My rating: ☆☆☆☆☆

My Course-Map

Ma/CS 006A	★★★☆☆
Ph 007	★★★★☆
Ph 010	★★★★☆
E 010	★★★★☆
Ec 011	★★★☆☆
PS 012	★★★★☆
Ph 012A	★★★☆☆
Ph 012B	★★★★☆
Ph 012C	★★★☆☆
PA 016C	★★★★★
Ph 020	★★★★☆



facebook Search

Course finder CALTECH

Search Next

CS 001 Introduction to Computer Programming

9 units (3-4-2); first term. A course on computer programming emphasizing the program design process and pragmatic programming skills. It will use the Python programming language and will not assume previous programming experience. Material covered will include data types, variables, assignment, control structures, functions, scoping, compound data ...

★★★★☆ Instructors: M Vanier

	Algorithmic Estimate	My Rating
Overall:	★★★★☆ (4.3)	★★★★☆
Instructor: M Vanier	Not Yet Avaliable	☆☆☆☆☆
Grading:	★★★★☆ (4.0)	☆☆☆☆☆

Number of hours spent/ week on this class (at least):

1 5 10 15 20

Grade in this class (at least):

✓ ✓ ✓ ✓ ✓
F D C B A

Course Comment [optional]

This course was completely revamped in 2009.

Submit [✓] Public Comment

Instructor Comment: M Vanier [optional]

Submit ☐ Public Comment

Course Comments

friends comments	public comments
This course was completely revamped in 2009. Isaac Sheff - May 17, 2011	Very practical, material is interesting and class moves at a good pace for those who never programmed before. - May 27, 2011

Instructor Comments: M Vanier

friends comments public comments

Search

forms on the course page with userÕs previous rating value given. Every time the user clicks on rating items, it sends the value to the server asynchronously.

5.2.3 Search, Next, and Previous Buttons

ItÕs import to have users return to the search page easily. Two access buttons to the main page are on the top and bottom of the course page. Furthermore, Next and Previous buttons have been added to allow users to easily rate several courses in a sequence.

5.2.4 Visual Effects

As a Facebook App, we choose themes that similar to Facebook for consistency. Colors of buttons and characters are the same with Facebook color. We use golden star rating bars for ease of use and universal understanding. For estimated ratings, red star bar can strongly attract attentions. We make pages clean and neat to let users seeking specific information easily. More importantly, it can make users think it is easy to rate courses.

5.3 Data

5.3.1 Course Rating Data

`http://courserecommendation.appspot.com/course/<id>/submit`

For javascript submission only.

5.3.2 User Dourse Map Data

`http://courserecommendation.appspot.com/data/<userid>`

For JSON retrieval only.

6. FACEBOOK INTEGRATION

As a registered facebook app, CourseFinder receives OAuth 2.0 tokens for each user which, while the user is using the app, allow it to made requests to facebook's Graph API.

6.1 Graph API

The Graph API is FacebookÕs framework for accessing and interacting with user information. The social graph is made of objects that have fields and are related to other objects by connections [8]. Information is fetched by HTTPS requests referring to objects, followed by access tokens if needed, in this form:

`https://graph.facebook.com/objectid?access_token=token`

6.2 Objects

The different types of objects are as follows: Album, Application, Checkin, Comment, Domain, Event, FriendList, Group, Insights, Link, Message, Note, Page, Photo, Post, Review, Status message, Subscription, Thread, User, and Video.

Each of these has fields and connections, some of which are accessible to anyone, to anyone on Facebook, or under certain permissions. The primary objects we are interacting with are User objects. The main fields of User objects we are interacting with are name, id, and education. The main permission that we need (to access education) is user education history. The User object connection we are using is picture, that is, the userÕs profile picture.

6.3 OAuth 2.0: Authentication and Authorization

Facebook uses the OAuth 2.0 protocol for authentication and authorization, that is, for making sure entities that request information are who they say they are and have the proper permissions to access the information they request. The main places this comes into play in CourseFinder is when the user logs in, and when CourseFinder requests user information.

Details are available at `http://developers.facebook.com/docs/authentication/`

6.3.1 Access Tokens

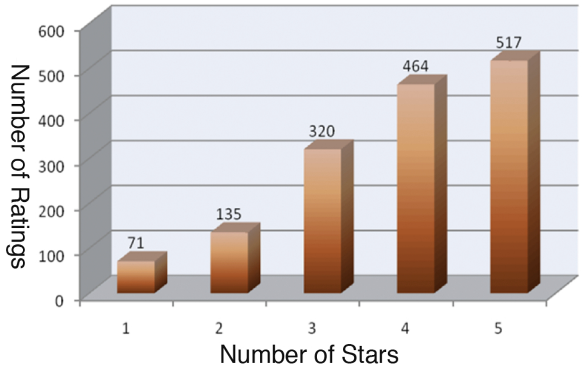
Access tokens are strings of alphanumeric characters that permit the access of particular objects, object fields, or object connections. If a user authorizes the app to access their information, with any particular permissions that app may additionally be requesting, then the app can get access tokens for that information.

7. RESULTS

7.1 Data

Of the 168 students persuaded to visit the CourseFinder app, 94 rated courses. They recorded 1507 Overall Ratings, 291 Teaching Ratings, and 258 Grading Ratings. All 443 courses that received a rating in any category received an Overall Rating.

Figure 4: Stars vs. Number of Overall Ratings



As Figure 4 demonstrates, users had a tendency, at least in Overall Ratings, to rate courses rather highly. Either users simply have low standards, or Caltech Courses are very high quality. We hope for, but cannot verify, the latter.

Figures 5 and 6 demonstrate the decrease in number of users willing to provide an increasing number ratings. This is likely because many users become bored or feel less motivated after a few ratings, and because fewer people have taken larger number of classes, due to, for example, freshmen only having taken about fifteen classes at the time of the study.

7.2 Analysis

Because our objective is to test the validity of social networking as a factor in predicting course preferences, and social networking is a user correlation metric, we elected to

Figure 5: Number of Users With at Least N Ratings

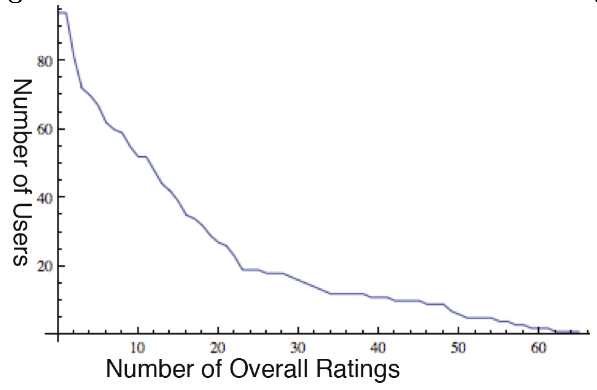
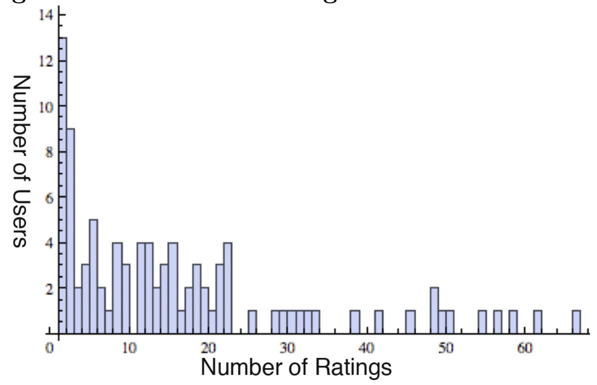


Figure 6: Number of Ratings vs. Number of Users



user a Memory-Based Collaborative Filtering system using User correlations. This means that a correlation factor $w_{u,v}$ was calculated for each pair of users (u, v), and from these predicted ratings for a user concerning an item were calculated based on the mean rating of the user plus the average deviation from the mean of other users's ratings concerning that item, each weighted by the correlation factor with that user. We elected to predict only Overall Ratings, since these are the most generic and most numerous.

$$\begin{aligned}
a &= \text{a user} \\
w_{a,u} &= \text{correlation coefficient between users } u \text{ and } a \\
i &= \text{an item (in this case, a course} \\
r_{u,i} &= u\text{'s rating of } i \text{ (normalized to the 0-1 range)} \\
\bar{r}_u &= u\text{'s average rating} \\
P_{a,i} &= \text{the prediction for } r_{a,i} \\
&= \bar{r}_a + \frac{\sum_{u\in\text{Users}} (r_{u,i} - \bar{r}_u) \times w_{a,u}}{\sum_{u\in\text{Users}} |w_{a,u}|}
\end{aligned}$$

Three different algorithms for the user correlation coefficients were tested: the popular Pearson Correlation Coefficient, a Social Weight Coefficient, and a combination of the two.

7.3 Pearson Coefficient

The Pearson Coefficient is a popular user-user correlation metric, possibly because of its ease of calculation [2] (it can be done by multiplying a matrix, with rows for each users, columns for each item, and entries consisting of the deviation of that user's rating of that item from that user's mean normalized for that users' standard deviation, by its transpose).

One can think of Pearson Correlation like a kind of normalized covariance.

$$w_{u,v} = \frac{\sum_{i\in\text{Items}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i\in\text{Items}} (r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i\in\text{Items}} (r_{v,i} - \bar{r}_v)^2}}$$

7.4 Social Weight

In an attempt to make a socially based weighting system that would neither over-correlate two users nor anti-correlate them (which is nigh impossible to do based on how well people know each other), we elected to make social weight scale with the number of mutual friends. This is normalized over the average number of friends of the two users, in order to ensure that $w_{u,v} = w_{v,u}$, and that the correlation coefficients are normalized to the same approximate range for all pairs of users.

$$w_{u,v} = \frac{\text{Number of Mutual Friends}}{\text{Average Number of Friends of } u \text{ and } v}$$

7.5 Combination

As a simple model that scales both with Social Weight and with Pearson Weight, we used the product of the two. Future studies may want to consider additional, more complex, algorithms.

7.6 Leave One Out Validation

Because there was insufficient time to gather user data, predict next year's ratings, and compare the results, and insufficient data for a proper test set and training set, we elected to use leave-one-out validation, in which the error of a prediction algorithm is approximated as the error in predicting each rating, had that rating not been made. Since we decided to predict using only Overall Ratings (as they are the most generic and most numerous), this is simply the error in the 1507 cases in which each of the overall ratings had not been made. We elected to use mean absolute error (MAE), as in this case, it is most important to get as many ratings as possible close to correct, and there are not increasing penalties for increasing error. This, as opposed to root mean squared error, is a common metric for collaborative filtering evaluation [2].

$$\text{MAE} = \frac{\sum_{\{i,j\}} (P_{i,j} - r_{i,j})}{\text{Number of Ratings Made}}$$

For reference, Monte Carlo tests on random rating predictions yield about .37 mean absolute error.

The Mean Absolute Error of each weighting system is as follows:

$w_{u,v}$	MAE
All Equal Weight (1)	0.2211
Pearson	0.2133
Social Weight	0.2208
Combination	0.2166

7.7 Conclusions

It is clear that while the social weight predictions were marginally (although there is no good calculation for standard deviation of error for a system like this, this is a really small margin) better than all equal weighting, the Pearson Correlation is still the best predictor. The mix of the two was, of course, between Pearson and Social Weight in MAE. Unfortunately, a beneficial hybrid weighting system has not yet been identified.

8. FUTURE WORK

CourseFinder itself, due to the objections of the Academics and Research Council, will likely no longer be popularized as a distinct entity in the future, as it will likely detract form standard TQFR course review responses. Many useful aspects of CourseFinder, such as the ability to quickly rate courses without going into detail, public and private comments, and search functions by teacher, department, and course name, may be implemented in future standard course rating system revisions. This is ultimately up to the ARC. The prediction algorithms, unless further honed, are unlikely to become standard fare in the future, as they are not yet particularly more useful than average ratings.

Using the anonymized CourseFinder data (available soon at the `http://CourseRecommendation.appspot.com`), or other social network / rating data, future studies may want to consider our social weight metric as a known "better than average" metric, as well as probe future possibilities of using both standard and social correlations. Possible avenues of exploration include item based memory Collaborative Filtering, and a more model-based system (in which the algorithm attempts to fit users to a predictive model of some kind) [2].

9. REFERENCES

1. Rashmi R. Sinha and Kirsten Swearingen. Comparing recommendations made by online systems and friends. In *DELOS Workshop: Personalisation and Recom- mender Systems in Digital Libraries*, 2001.

2. Xiaoyuan Su and Taghi M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, vol. 2009, Article ID 421425, 19 pages, 2009.

3. K. Sarda, P. Gupta, D. Mukherjee, S. Padhy, and H. Saran, "A distributed trust-based recommendation system on social networks," in *HotWeb Õ08: Proc. of the 2nd IEEE Workshop on Hot Topics in Web Systems and Technologies*, 2008.

4. Goldbeck, J. and Hendler, J. 2006. FilmTrust: Movie recommendations using trust in Web-based social networks. In *Proceedings of the IEEE Consumer Communications and Networking Conference.* Las Vegas, NV.

5. Synclab Consulting, "Hooks for Facebook." *synclab consulting* 11 Mar. 2011, <http://www.synclab.com/hooks-for-facebook/>.

6. M. Agrawal, M. Karimzadehgan, and C. Zhai. An online news recommender system for social networks. *SIGIR-SSM*, 2009.

7. Gorset, Johannes. "Fandjango." <https://github.com/jgorset/fandjango>.

8. Facebook Developers, "Graph API." <http://developers.facebook.com/docs/reference/api/>.