A Collaborative Filtering Prediction Algorithm for ClassRank Subject Recommendations

by

William G. Tetler

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2008

Author: _____
Department of Electrical Engineering and Computer Science
December 19, 2007

Certified by: _____
Edward C. Barrett
Senior Lecturer in Writing
Thesis Supervisor

Accepted by: _____
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

A Collaborative Filtering Prediction Algorithm for ClassRank Subject Recommendations

by

William Tetler

Submitted to the Department of Electrical Engineering and Computer Science

December 19, 2007

in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Undergraduate students at M.I.T. typically utilize three resources when selecting subjects: course specific evaluations, faculty advisors, and peers. While these resources have distinct advantages, they are all limited in scope. The ClassRank web application has been developed to bridge the gap between these resources by providing a simple institute-wide system for undergraduate students to evaluate and rate subjects. The application also provides a solid platform to build new tools utilizing subject evaluation data. To extend the initial core functionality of the ClassRank system, a rating-based subject recommendation algorithm was added to offer students an unbiased perspective on potential subjects of interest. Developed as a Ruby on Rails plugin and then integrated into ClassRank, the recommendation algorithm analyzes subject ratings and provides personalized suggestions to students about subjects that would likely fit their interests and educational goals. The ClassRank web application and recommendation algorithm will provide the M.I.T. undergraduate student body with a unique and invaluable resource for subject selection.

Thesis Supervisor: Edward Barrett
Title: Senior Lecturer, Program in Writing and Humanistic Studies

Acknowledgements

I would like to thank Professor Ed Barrett for supervising my work on this project and guiding me throughout the implementation and writing process. Also I would like to thank Dan Malconian for help designing and conceptualizing the ClassRank application. Finally, I would like to thank Erin Wagner for her assistance editing and revising my writing.

Contents

List of Figures

1 Introduction

Currently it is very difficult for M.I.T. undergraduate students to accurately judge the subjects in which they are considering enrollment. As a result, students rarely have the luxury of selecting subjects by teaching style or difficulty level. Selecting appropriate classes is especially important for M.I.T. students. Requirements for many degree programs are strict and lengthy; therefore, students rarely have the scheduling flexibility to recover from more than one or two inappropriate subject choices.

Limited resources and guidance result in most students selecting subjects based on compatibility with their schedules. This is especially true for classes satisfying broad General Institute Requirements (HASS-D or HASS-CI) outside of a student's degree program. Even for those subjects specifically required by a degree program, comparing different teaching methods provides a student the ability to optimize their learning experience.

Traditionally, students utilize three resources to choose subjects.

1. Underground Guides – Providing historical course specific subject evaluations.
2. Faculty Advisors – Providing personal recommendations from past experience.
3. Peer Network – Providing personal recommendations and historical evaluations.

While all of these resources provide tremendous value to students, information provided by each is limited in scope:

1. Underground guides offer historical peer evaluations; however, current systems lack ability to form evaluations into personal recommendations and are limited to specific courses.
2. Faculty advisors, while able to offer personalized recommendations, typically have only knowledge of subjects in their specific field of interest and often lack the "undergraduate experience" necessary to provide accurate assessments of work load, teaching style, and other comparative measures.
3. A student's peer network can offer both evaluations and recommendations for subjects from a wide verity of academic disciplines; still, a student's network of fellow undergrads is usually limited in size and, therefore, limited in use.

The ClassRank web application aims to bridge the gap between these resources by providing a simple web-based system for M.I.T. undergraduate students to evaluate and rate subjects. The application will aggregate a large historical database of subject evaluations to aid current students in the discovery of subjects which fit their style of learning, personal interests, and educational goals.

ClassRank's database of subject ratings contains valuable information derived from correlations between individual evaluations. Through analyzing and generalizing the preferences of students, a recommendation algorithm can easily deduce subjects of interest to a particular individual student. As preference algorithms and recommendation systems have become very widespread in application, there are many possible methods of

constructing such predictions. A number of very intuitive suggestion algorithms have been conceived as research into the area has spread, providing accurate predictions at a very low computational overhead. Utilizing a preference algorithm to generate personalized subject recommendations for students greatly enhances ClassRank's utility and, hopefully, will serve as a catalyst for the system's adoption.

1.1 Research Goals

The goal of this research is to:

1. Implement ClassRank, a web application for M.I.T. undergraduate students to evaluate subjects through a simple set of ratings;
2. Create a reusable Ruby on Rails recommendation plugin; and,
3. Integrate the plugin into ClassRank in order to provide students with personalized subject recommendations.

Through accomplishing these goals, ClassRank will demonstrate the power of new web technologies, even in an unconventional application, by complimenting the traditional resources used by students with unique and unbiased subject recommendations based on a large historical dataset.

1.2 Outline

The following section provides a brief discussion of the ClassRank web application, including its motivation, architecture, supporting technology, and design. Section 3 goes into detail about integrating a recommendation algorithm into ClassRank, including and the key factors behind choosing an algorithm and building it into a Ruby on Rails

recommendation plugin. This includes a brief overview of common collaborative filtering techniques and an extensive discussion of slope one predicators. In addition, the section documents the implementation of the recommendation plugin and its integration into ClassRank. Section 4 goes on to discuss the results and final product of the research. To conclude, Sections 5 and 6 summarize the overall contribution of the research and suggests a number of future enhancements to both ClassRank and the recommendation algorithm.

2 The ClassRank Web Application

ClassRank is a web application for M.I.T. undergraduate students, first conceived as an advanced undergraduate project in the spring of 2007. The initial ClassRank infrastructure allows students to browse and post subject evaluation data.

The design of ClassRank and data collected through subject evaluations serve as a solid platform to enhance and expand the functionality of the system. Initially, ClassRank will integrate a recommendation algorithm that compares subject evaluations and provides personal recommendations to students, discussed in Section 3.

2.1 Motivation

The ClassRank web application aims to educate M.I.T. students about subjects that directly align with their interests and personal preferences. Currently, it is very difficult for students to accurately assess the various aspects of subjects in which they are considering enrollment. In addition, no institute-wide subject evaluation tools exist, making it very difficult to aggregate necessary data. To aid students with the subject selection process, ClassRank provides tools to browse the aggregate subject evaluations from a large historical peer network and enables students to utilize this data in order to draw conclusions about subjects of interest.

2.2 MVC Architecture

A natural architecture for database-backed web applications compartmentalizes functionality into three discrete components: models to represent data, views to construct the user interface, and controllers to coordinate underlying logic. This pattern, known as

Model-View-Controller (MVC), naturally decouples an application's functionality. As a result, changes to a single component of a MVC application typically require little or no modification to the others. The MVC architecture can greatly simplify integration of large user interface or data storage changes into existing applications, as the underlying implementation of individual components can change drastically with little effect on the rest of a system. Untangling the model, view, and controller components of an application greatly helps to control the complexity of large-scale applications.

2.2.1 Models

The state of a MVC application is maintained through models. Models represent data records, both temporary and persistent, and ensure the data being represented is valid. Models also provide logic to translate raw data into more a meaningful form for the application to display. In addition, models are responsible for the specific mechanisms used to store and retrieve data. Typically, models will utilize a database as the underlying storage mechanism to record persistent data.

2.2.2 Views

Views generate a user interface for MVC applications, presenting the data from models in a specific format and layout. As MVC decouples the views and models, implementing multiple interfaces is trivial. Typically, views also capture user interaction and notify the application of events.

2.2.3 Controllers

Controllers respond to events and trigger specific actions that interact with models and present resulting views. Within the MVC architecture, controllers are responsible for

handling the exchange of information between the model and view components and ultimately define the functionality of the application.

2.2.4 Application Flow

Applications with MVC architectures cycle through a simple pattern of use.

1. Initially an event, most likely requested by a user through the application's interface, triggers an action within a controller.
2. Then the controller's action interacts with models, generating information necessary to satisfy the request.
3. Next, the controller's action invokes a view, passing along the information it has gathered.
4. Finally, the updated view, in the context of the resulting information, is presented to the user and the application waits to receive the next request.

This application flow matches the general behavior of common web services: receive a request for a page and respond to the browser with HTML. For web applications, which generate information dynamically before presentation, an MVC framework provides a well designed architecture to isolate complexity.

Although ClassRank will initially be a relatively small system, an MVC design will ensure the application can serve as a solid platform for future expansion. In addition, new web application frameworks have abstracted MVC principles so successfully that even small systems can be built with greater efficiency.

2.3 Supporting Technology

The ClassRank web application is implemented on top of the open-source Ruby on Rails web framework. Rails provides a robust platform for ClassRank ensuring the application is well designed, easily extendable, and efficiently developed.

2.3.1 Benefits

Rails is written on top of Ruby, a very dynamic object-oriented scripting language, and aims to increase the speed and ease of web application development. In large part, Rails achieves this goal through organizing applications into a strict MVC structure. Additionally, Rails provides:

- A built-in development web server and build system;
- Integration of robust Asynchronous JavaScript and XML (Ajax) libraries;
- Integrated testing;
- Powerful database abstraction; and,
- Support for plugins that allow developers to easily isolate and distribute extensions to the framework's functionality.

The structure and design of Ruby on Rails ensures that web applications are developed as efficiently as possible and comply with agile development principles. As a result, Rails significantly decreases both the amount of time and amount of code necessary to build web applications with rich user interfaces and powerful functionality.

2.3.2 Fundamental Design Principles

While Ruby on Rails takes many measures to accelerate development of database-backed web applications, the entire framework is designed to comply with two key concepts: *Don't Repeat Yourself* (DRY) and *Convention over Configuration.*

- DRY: Rails allows developers to isolate and abstract functionality very efficiently, preventing duplication. Duplicated source code can be inconsistent, hard to change, and difficult to test. Localizing functionality also decreases the amount of source code in the application.
- Convention over Configuration: Rather than relying on developers to manually specify every configuration detail, Rails requires action only for behavior which differs from the standardized conventions. Through these assumptions, Rails seamlessly connects the application's models, views, and controllers without unnecessary setup, greatly simplifying the development process. As an application develops and advanced configuration is needed, the default behavior can be easily changed from the default value.

Ruby on Rails successfully translates the concepts of DRY and *Convention over Configuration* into a lightweight and efficient web application framework.

2.3.3 Plugin Support

Ruby on Rails provides a plugin system to extend the core functionality of the framework. Plugins allow code for specific functionality to be implemented within a self-contained architecture. As a result, plugins allow developers to distribute and reuse pre-built code, keeping the Rails framework as light as possible while providing support

for cutting-edge functionality. Further, plugins enhance the reliability of applications by isolating potential problems and protecting code from unintended changes.

The plugin capabilities of Rails provide an optimal platform for the implementation of ClassRank's recommendation functionality.

2.4 Design Overview

2.4.1 Data Models

ClassRank is supported by a simple set of underlying data models representing semesters, courses, subjects, students, and subject evaluations, shown in **Figure 1**. Dependencies between models are represented by arrows and attribute names and types are contained within the box of each model.

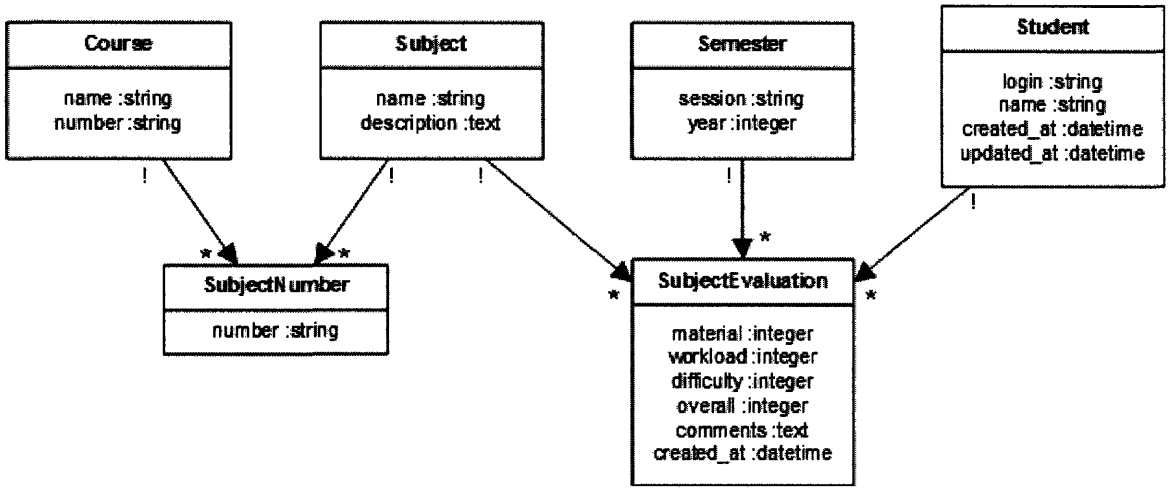

Figure 1: Overview of ClassRank models and attributes.

Read-Only Models

The Course, Semester, Subject Number, and Subject models are responsible for information that students are not allowed to directly modify.

The Course model contains two fields, a name (i.e. Electrical Engineering and Computer Science) and a number (i.e. 6) of an M.I.T. course. Both fields are stored in the ClassRank database as strings to accommodate course numbers includng letters (i.e. 21W, CMS). Course data is pre-populated into the ClassRank system through a Rails database migration that creates models by parsing a list of courses formatted as comma separated values.

The Semester model stores two pieces of information about each semester: the session, a string, and the year, an integer. Before a new record is created, the Semester model validates that the session be either spring or fall. New semesters are created automatically if the Semester model determines from the current month and year that the current semester has changed.

The Subject Number model joins courses and subjects, as one subject can have many subject numbers under different M.I.T. courses. The Subject Number model contains a string for the number of a subject for a particular course. For example, the number stored in the Subject Number model for 6.001 would be 001, as the course's number ("6" in this case) can be derived through the model's associated Course model.

Finally, the Subject model stores the name and description of a subject. Additionally, the model is associated with at least one Subject Number model. Similar to semesters, subjects are populated through ClassRank's database migrations, which load a file of subjects (represented as comma separated values) and create new Subject models.

Student Model

The Student model represents information about the students utilizing ClassRank. The model contains the login and name of a student, as well as the date the student first accessed ClassRank.

Subject Evaluation Model

The Subject Evaluation model stores evaluations submitted by students. Evaluations contain four subjective ratings corresponding to a student's opinion of the subject as a whole, as well as the subject's material, workload, and difficulty. In addition, the Subject Evaluation model allows students to store general comments about a subject. To keep track of the subject, the semester it was taken, and the student providing the evaluation, the model also associates itself to a Subject model, Semester model, and Student model. The Subject Evaluation model also validates each evaluation to ensure that the ratings fall within a specific range, within 1 and 200, and that comments are not too long.

2.4.2 Controllers and Actions

The core functionality of ClassRank is defined by three controllers: the Students controller, the Subject Evaluations controller, and the Subjects controller, shown in **Figure 2**. All of ClassRank's controllers inherit functionality from the Application controller, which implements core actions to authenticate and authorize students before executing an action in any of ClassRank's controllers. The specific actions available to students are listed within the box of each controller.

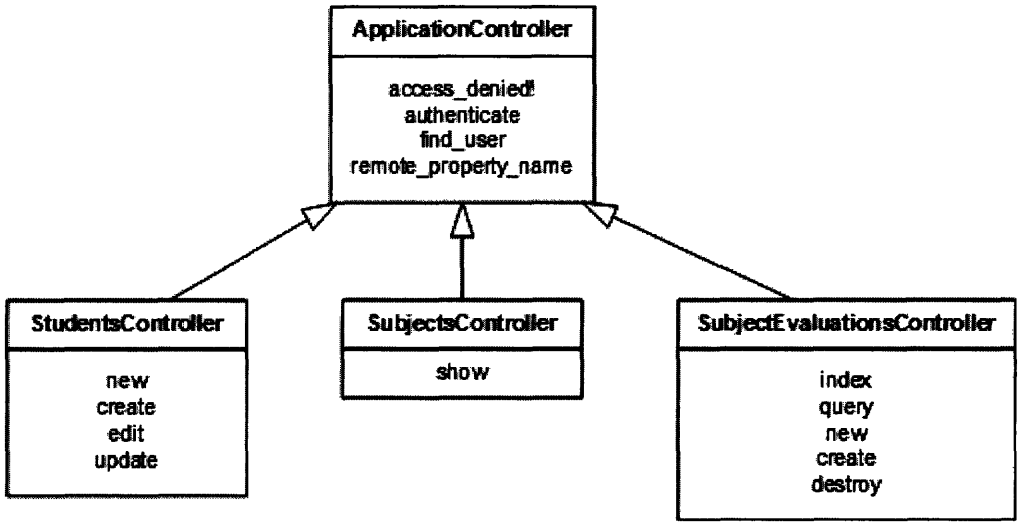

Figure 2: Overview of ClassRank controllers and actions.

Students Controller

The Students controller is responsible for creating records for new users of the system. If a student is accessing ClassRank for the first time, the Students controller will create a new Student model to represent them, and forward the student along to the subject evaluations controller.

Subject Evaluations Controller

The Subject Evaluations controller is responsible for adding, viewing, and deleting subject evaluations. Additionally, the Subject Evaluations controller provides an action to search for subjects and evaluations by subject number.

Subjects Controller

The Subjects controller implements one action to display information about a subject, including a list of all its evaluations.

2.4.3 User Interface

ClassRank was designed to have a simple and intuitive user interface. The user interface provides a number of conveniences for students.

- Ajax pagination of long lists of data – ClassRank provides pagination for lengthy data sets without requiring page reloads, only updating the portion of the page containing the listed information.
- Auto-complete textboxes for subject number queries – Students can type subject numbers and instantly view a list of matching subjects, avoiding the potential frustration of not exactly matching the number of a result without sacrificing any accuracy.

Screenshots of the ClassRank interface are included in **Appendix A**.

2.5 Deployment Details

ClassRank is designed to be deployed on the Web Script Service maintained by the M.I.T. Student Information Processing Board (SIPB). ClassRank authenticates and retrieves login information from students by utilizing M.I.T. personal certificates. This eliminates the requirement for a self-contained user authentication system and ensures that only current students can access the application.

Populating ClassRank with subject records is accomplished through the use of database migrations built into Ruby on Rails. Information about each subject, recorded as comma separated values and contained in an input file, is loaded into the database after it is

created. Ideally, future revisions of ClassRank will automate the collection of subject data and update the database accordingly.

3 Incorporating Subject Recommendations

Unlike current M.I.T. subject evaluation systems, ClassRank provides a platform to implement tools which utilize the subject evaluation data collected by the system. As subject evaluations attempt to quantize student's opinions, they are very similar to the product ratings provided by users on many e-commerce websites. Just as these websites utilize reviews to suggest products to customers, ClassRank could utilize evaluations and ratings to suggest subjects to students. Therefore, as a natural first enhancement to ClassRank, functionality to provide students with subject recommendations was integrated into the system.

3.1 Design and Development Procedure

In order to provide subject recommendations to students, a recommendation algorithm was implemented by:

1. Researching and choosing a collaborative filtering recommendation algorithm;
2. Creating a Ruby on Rails plugin to provide recommendations for a generalized usage scenario;
3. Integrating the plugin into the existing ClassRank application; and,
4. Updating the ClassRank user interface for use of the new functionality.

Rather than conducting significant changes to the ClassRank infrastructure, utilizing the plugin capabilities of Ruby on Rails allows for recommendation functionality to be isolated and encapsulated in a separate package. In addition, developing a plugin allows the recommendation functionality to easily be reused in other applications. Plugin

development also ensures that the algorithm is created in as general a scope as possible, abstracting the underlying environment-specific implementation problems from the design of the prediction engine.

3.2 Recommendation Algorithms

Recommendation and preference algorithms have become a commonplace on the internet. Heavily utilized on e-commerce websites, such as Amazon.com and Netflix, recommendation algorithms serve millions of internet users with product recommendations and suggestions daily (Linden, Smith, & York, 2003). Additionally, many heavily trafficked online social networks and tools revolve around delivering useful and interesting personalized content to users through preference algorithms (Kautz, Selman, & Shah, 1997). A variety of collaborative filtering algorithms provide the basis behind most modern recommendation systems in practice.

Collaborative filtering algorithms come in many different flavors, largely customized for each application, but all revolving around a fundamental principle: those who have agreed in the past will agree in the future. Without this basic assumption, aggregated data both actively volunteered by users and passively collected through browsing patterns would be useless in the prediction of unknown preferences. In collaborative filtering systems, users collaborate through offering their own data to the greater network. In exchange, users receive more accurate personalized recommendations and improve the overall quality and utility of the recommendation system.

In the past decade, collaborative filtering algorithms have been the topic of much research. Efficiency has been one area of focus, leading to the formalization of constant

time collaborative filtering algorithms (Goldberg, Roeder, Gupta, & Perkins, 2001). Additionally hybridized recommendation algorithms have been formed, combining techniques in collaborative filtering, webpage ranking methodology, and traditional neural networks to dramatically improve accuracy of results and performance (Teow & Katabi, 2006). Modifications to traditional collaborative filtering methods are very widespread and many more varieties are publicly available. The appropriate and most accurate algorithm depends entirely on the application and its requirements.

3.2.1 Algorithm Requirements

Several criteria were used to choose an appropriate collaborative filtering algorithm.

- Ease of implementation and maintenance – The algorithm should be intuitive, enabling the average engineer to fully understand its inner workings.
- Quickness and efficiency – The algorithm should be able to produce recommendations for a user almost instantaneously.
- Initial prediction requirements – The algorithm should be able to generate useful recommendations for users with few ratings.
- Accuracy of recommendations – The algorithm should produce relatively accurate recommendations, but not at the sacrifice of simplicity or speed.

While numerous collaborative filtering algorithms exist, few actually satisfy all four of the above requirements. Many algorithms are very quick and accurate, but too complex to be intuitively understood. Additionally, many utilize the basic fundamentals of linear algebra, but are slow or require large amounts of data to form accurate recommendations.

However, there is a family of extremely powerful and accurate prediction algorithms grounded through intuition and computationally inexpensive.

3.2.2 Slope One Collaborative Filtering Algorithms

Slope one prediction algorithms are interesting candidates for real-world prediction systems, operating on an intuitive assumption of *popularity differentials* between pairs of items. Slope one schemes pre-compute the difference between the ratings of two items to quantify how much one item is preferred to another.

Slope one predictors take a general linear form: $f(x) = x + b$. The family of algorithms assumes the predictor's slope is one, estimating only the popularity differential (the predictor's intercept b) to predict the user's rating of a new item. The slope one schemes estimate the predictor's intercept as the average popularity differential of two items. Essentially, the schemes take the average difference between the ratings of the item Y, whose rating we would like to predict, and the ratings of another item X, whose rating we know. Therefore, to predict a user's rating for an item Y, the popularity differential between item Y and an item X, previously rated by the user, is added to the user's rating for item X.

To provide a quantitative example of the intuition behind slope one algorithms, consider two users, A and B, and two items, X and Y. Suppose user A has supplied ratings of 1 and 2.5 for items X and Y respectively, as shown in **Figure 3**. User B, however, has only supplied a single rating of 2 for item X. It is clear from this situation that item X is rated more than item Y by $2 - 1 = 1.5$ points. From this information and the rating of item X

from user B, it is natural to predict that the unknown rating of user B for item Y will be $2 + 1.5 = 3.5$ points.

	Item X	Item Y
User A	1	2.5
User B	2	?

Figure 3: User ratings for prediction example.

Slope one algorithms differ in how they form an overall prediction of an item's rating from the aggregate predictions formed using the average popularity differentials of the item and all other items that the user has rated. The Basic Slope One algorithm simply forms a single prediction by averaging the predicted ratings that it computes for each of the user's rated items. A more accurate approach, the Weighted Slope One algorithm takes the weighted average of the predicted ratings, ensuring that individual predictions with more robust popularity differentials have a greater influence over the final value.

As slope one algorithms generate a prediction of a user's rating for an item, forming recommendations requires that predictions be generated for all of a user's unrated items. Items with the highest predicted ratings would form the recommendation for the user.

A performance assessment of slope one algorithms demonstrates the power of the schemes in comparison to other common collaborative filtering algorithms (Lemire & Maclachlan, 2005). The research tested a set of standardized benchmarks using two test datasets from EachMovie and Movielens. EachMovie, a dataset maintained by Compaq Research, and Movielens, a similar dataset from the Grouplens Research Group at the University of Minnesota, present large volumes of historical data collected from movie rating websites. Benchmarks were conducted on a training set of 50,000 ratings and a

test set of over 100,000 ratings. The team computed the All But One Mean Average Error (MAE) on predicted ratings from three slope one algorithms, as well as two simple collaborative filtering schemes (Per-User Average and Bias from Mean) and two more advanced techniques (Adjusted Cosine Item-Based and Pearson). The analysis indicates that slope one algorithms are much more accurate than simple techniques and are very competitive with much more complex collaborative filtering schemes (**Figure 4**).

Scheme	EachMovie	Movielens
Bi-Polar Slope One	0.194	0.188
Weighted Slope One	0.198	0.188
Slope One	0.200	0.188
Bias From Mean	0.203	0.191
Per User Average	0.231	0.208
Adjusted Cosine Item-Based	0.209	0.198
Pearson	0.194	0.190

Figure 4: MAE performance comparison of collaborative filtering schemes.

The intuitive nature, quick performance, and accuracy of slope one algorithms satisfy the project's criteria perfectly. In addition, slope one schemes require only a small subset of ratings from initial users in order to generate useful predictions. While the simplicity of slope one predictors could potentially lead to less accurate results, it also allows for developers to easily understand the system and comfortably integrate future changes to enhance accuracy.

3.3 Building a Recommendation Plugin

A Weighted Slope One collaborative filtering algorithm was implemented within a Ruby on Rails plugin. Designed to extend the functionality of a model which handles item ratings, the recommendation plugin defines five class variables and implements five class methods in order to provide suggestions. Of the class variables, three are used simply to

store configuration information for the plugin while the other two serve as persistent data structures necessary for generating predictions. Four class methods of the recommendation plugin are responsible for updating these core data structures, while a single method utilizes this information to generate recommendations for a given user.

3.3.1 Design and Underlying Assumptions

The recommendation plugin is designed to extend the functionality of models which store rating information. The plugin operates under the assumption that it extends the functionality of a model, which stores a rating for an item provided by a user. The rating is assumed to be a field within the model and the item and user are assumed to be associated with the model through standard Ruby on Rails conventions (**Figure 5**).

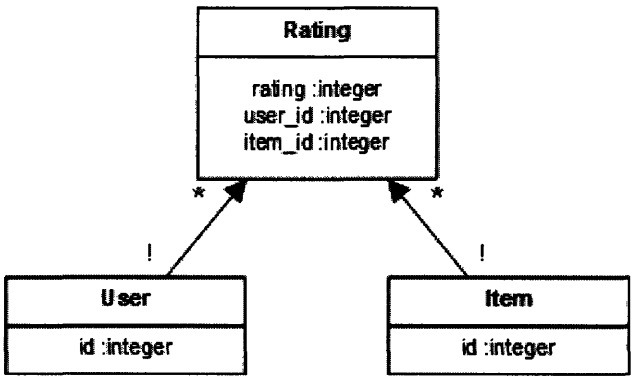

Figure 5: Supporting model structure assumed by the recommendation plugin.

The plugin contains a set of default assumptions for how to access the ratings field and information about the associated models, but allows custom values to be declared for the names of the rating, user, and item models as well as the name of the integer field storing item ratings.

To ease integration into complex systems, the recommendation plugin does not require any modification to the data storage mechanism. While the plugin does require persistent information to generate recommendations quickly, this data is computed when the web application is started, stored in memory, and updated only by necessity when ratings are created, updated, or destroyed. This capability is the direct result of utilizing a slope one scheme.

3.3.2 Class Variables

The recommendation plugin uses five class variables: three to store configuration information and two to store necessary prediction data.

The three configuration variables store:

1. The rating attribute of the model to predict;
2. The name of the associated model for the items being rated; and,
3. The name of the associated model for the users rating the items.

These three attributes provide enough configuration information for the recommendation algorithm to have an understanding of the environment that it is operating in, generally described in **Figure 5**.

The remaining two class variables store the information necessary to generate predictions. Both variables take the same form: matrices with both a row and column for every item in the system. Therefore, each cell of the matrices is correlated with two items: one associated with the row of the cell and the other associated with the column of the cell.

The Differential Ratings Matrix

The first of these item-to-item matrices, the differential ratings matrix, stores the total difference of all ratings between two items. In other words, for each user that has rated a set of two items, the difference between the ratings for these two items is summed and stored in the cell associated with the pair of items.

As there are actually two cells in the differential matrix associated with any pair of items, the upper right triangle of the matrix is symmetrical to the negative of the bottom left triangle. In addition, diagonal of the matrix represents the rating differential between an item and itself, which is always zero.

The Frequency Matrix

The second item-to-item matrix used by the recommendation plugin is the frequency matrix. The frequency matrix stores the number of differentials computed for a pair of items. Stated simply, the frequency matrix can be thought of as the number of users who have provided ratings for two items, represented by the row and column of the matrix.

Similarly to the differential ratings matrix, there are two cells of the frequency matrix for any pair of unique items. Thus, the frequency matrix is naturally symmetric. Further, the values on the diagonal of the frequency matrix, cells which reference two instances of the same item, represent the total number of users who have rated the item.

3.3.3 Class Methods

The recommendation plugin uses five class methods: four to update the differential ratings and frequency matrices and one to generate predictions for a given user.

Update Methods

Four class methods update the differential ratings matrix and the frequency matrix. The four methods handle updating the matrices under specific circumstances, including:

1. Initial construction of the differential ratings and frequencies;
2. Updating the differential ratings and frequencies when a user adds a rating for a new item;
3. Updating the differential ratings when a user modifies the existing rating for an item to a new value; and,
4. Updating the differential ratings and frequencies when a user destroys the rating for an item.

At the application's startup, the recommendation plugin automatically runs the method to create the initial matrices. To ensure that the matrices are kept up to date with the current state of the ratings database, the recommendation algorithm observes when ratings are created, modified, or destroyed and executes the appropriate action in order to update the matrices to the current state of the database.

Recommendation Method

A single method utilizes the differential ratings and frequency matrices to generate predictions for a given user, applying the Weighted Slope One algorithm. Initially, the method generates an array of predicted ratings for every unrated items utilizing the equation in **Figure 6**. The equation states that the prediction for an unknown item u_j is found by summing the scaled predictions from every item u_i in the set of rated items I and dividing by the total frequency of u_j and each u_i.

$$p(u_j) = \frac{\sum_{u_i \in I} diff(u_i, u_j) + freq(u_i, u_j) \times rating(u_i)}{\sum_{u_i \in I} freq(u_i, u_j)}$$

Figure 6: Prediction equation used by the recommendation function.

The recommendation function then sorts the unrated items from highest predicted rating to lowest and returns the resulting list of items and predicted ratings. The ordered list of items corresponds to the recommendations for the user, sorted from best to worst.

3.4 Integration into ClassRank

In order to implement the recommendation plugin into ClassRank, the Subject Evaluation model was updated to include the functionality. After the model could generate subject recommendations utilizing the plugin, ClassRank's interface was updated to allow qualified students to utilize the functionality.

3.4.1 Updating the Subject Evaluation model

Integrating the recommendation plugin into the Subject Evaluation model is accomplished through a single declaration. The declaration initializes the plugin and passes along all necessary configuration information. The Subject Evaluation model serves as the Rating model for the plugin, while the Subject model represents the associated items and the Student model corresponds to associated users, shown in **Figure 5**. As the recommendation plugin supports predictions for only a single rating, the overall rating provided in the subject evaluation forms the basis behind ClassRank's recommendations, representing the rating field for the plugin.

3.4.2 Providing a Subject Recommendation Interface

A new controller was added to ClassRank in order to construct an interface for student's to access subject recommendations. The Subject Recommendations controller has only a single responsibility: display the list of subject recommendations for the student currently logged into ClassRank. If a student has evaluated at least three subjects, a link is presented to generate and view recommended subjects. No option is presented to students who have evaluated less than three subjects, as recommendations are likely to be inaccurate and inappropriate with such a limited training set of data.

A screenshot of ClassRank's recommendation interface is included in **Appendix A**.

4 Results and Discussion

Throughout the course of this research three major milestones were accomplished:

1. The ClassRank web application was conceptualized and fully developed, providing a simple web-based system for M.I.T. undergraduate students to evaluate subjects through a basic set of ratings and browse the past evaluations of subjects;
2. A reusable Ruby on Rails plugin was created to extend the functionality of a model to form personalized item recommendations to users; and,
3. The recommendation plugin was successfully integrated into the ClassRank system with the addition of an interface for students to browse subject recommendations.

Through accomplishing these goals, ClassRank demonstrates the power of new web frameworks, intuitive collaborative filtering algorithms, and agile development practices. Ideally, ClassRank will compliment the traditional resources used by students to select subjects with unique and unbiased subject recommendations based on a large historical dataset.

4.1 ClassRank

ClassRank was designed to serve as a solid platform for future enhancements and upgrades. Unlike the current web-based subject evaluation tools available to M.I.T. students, ClassRank stores just enough data to ensure utility to students and extend the platform with unique value-added tools. ClassRank's design adheres to the MVC

framework and utilizes Ruby on Rails to force agile development practices. As a result, developers unfamiliar with ClassRank will easily be able to gain a full understanding of the application and easily continue to enhance the system's core functionality.

4.2 Recommendation Plugin

In a sea of potential candidates, the Weighted Slope One collaborative filtering algorithm perfectly satisfied the following four criteria:

- The algorithm was easy to implement and grounded on intuition, enabling developers to easily understand the functionality and inner workings of the recommendation plugin;
- The algorithm is able to produce recommendations for a user very quickly, updating the data used to generate predictions only when needed and forming real-time predictions without heavy computation;
- The algorithm is able to generate useful and accurate recommendations for users with few ratings; and,
- The algorithm is known to produce very accurate recommendations, competitive with much more complex and intricate predictions schemes.

As a result of choosing the Weighted Slope One algorithm, encapsulating recommendation functionality within a Ruby on Rails plugin was straight forward. The plugin fully encapsulates and isolates the recommendation functionality from the underlying application, requires little configuration, has no external dependencies, and assumes it is being deployed in a very general environment to ensure compatibility with a wide variety of systems.

4.3 Prediction Accuracy

Unfortunately, it is extremely difficult to judge the accuracy of recommendations without an extensive amount of real usage data and feedback. Therefore, the accuracy of subject recommendations will be unknown until ClassRank has received a significant amount of use and user feedback. Still, the intuition behind the Slope One algorithm suggests that recommendations will be innately accurate to some degree.

Integrating a generalized test and benchmarking suite into the recommendation plugin would be a very valuable enhancement and is detailed further in Section 6.

4.4 Integration of Subject Recommendation into ClassRank

The design of the recommendation plugin allowed subject recommendation to be easily integrated into the core ClassRank infrastructure. Therefore, only a single line of code was necessary in order to generate subject recommendations. To provide an interface for users to view these recommendations an additional controller was added to ClassRank. The ease of the integration process demonstrates the power and efficiency of the Ruby on Rails web framework and its underlying plugin capabilities.

4.5 Future Deployment

While ClassRank is fully functional, additional work still needs to be completed before the tool is released to the student body. Before a production version of the application is deployed, a number of enhancements are necessary to ensure the system can function with little or no maintenance for extended periods of time.

Additionally, after the tool is deployed, it must be marketed to the student body and attract a significant number of subject evaluations before accurate recommendations can be provided to students. This task will require significant planning and flawless execution. As existing subject evaluation tools become web-based, it will become harder and harder to convince students to complete additional evaluations.

As a short term solution, importing subject evaluation data from existing tools into the ClassRank system would provide some incentive for students to utilize the application, as it would aggregate this data from multiple courses into a central location. While recommendations could not be constructed from this existing data, the functionality could be included at a later time as students begin to utilize ClassRank when researching subjects and, hopefully, begin to evaluate subjects on the platform as well.

Future work to the ClassRank web application is discussed in detail in Section 6.

5 Conclusion

In conclusion, ClassRank demonstrates a powerful new way for students to receive input on potential subjects of interest. Through combining subject evaluations over a large historical timeframe, the tool will supplement the traditional resources that students utilize to select subjects: course specific evaluations, faculty advisors, and peers.

In large part, ClassRank's utility will be derived from its subject recommendation capabilities, provided through a Ruby on Rails plugin. As the design of the recommendation plugin is very modular, enhancing the plugin to provide increased accuracy or respond to user feedback is straightforward and guaranteed not to negatively affect the underlying ClassRank application.

After some additional work to the application's deployment infrastructure and a campaign to market the tool to the student body, ClassRank will provide students with a unique and invaluable resource to utilize when selecting subjects.

6 Future Work

As ClassRank was developed as an initial prototype, before the application can be deployed to the student body, additional enhancements and fixes are required. The recommendation plugin also presents many interesting opportunities for future expansion and growth.

6.1 Deploying ClassRank

Before the ClassRank system can be deployed to the student body, work needs to be completed in four areas: marketing, data population, long-term deployment issues, and testing.

6.1.1 Marketing to Student Body

In order to attract students to ClassRank, the tool must be marketed effectively to undergraduate students. Initially, this may be difficult, as other course-specific subject evaluations have moved onto web platforms and students may be reluctant or unmotivated to evaluate subjects multiple times. In addition, subject recommendations will be of little utility with a small initial dataset and unlikely to drive students towards the application.

Work needs to be completed to determine the best way to attract students to ClassRank, either by pre-populating subject evaluation data in some way or introducing new functionality that will attract users without requiring more than a small dataset. In many ways, this problem will be much more challenging than the technical hurdles presented throughout ClassRank's development.

6.1.2 Populating Subject Information

Currently, subject information is loaded into ClassRank through parsing a manually created CSV (Comma Separated Value) file. If ClassRank is to dynamically exist within the M.I.T. environment, it must have a system to update the subjects within ClassRank in a completely automated fashion. One solution might be to scrape the current subject listings on M.I.T.'s webpage at the beginning of each term. Ideally, ClassRank should automatically update subject listings, requiring as little maintenence as possible.

6.1.3 Long Term Deployment

Significant work is needed to prepare ClassRank for long-term deployment. Potential issues could arise as past and present students could have the same Kerberos identification, which would result in the current student receiving a list of evaluations by a past student. In addition, ClassRank should verify the current user is, in fact, an undergraduate student, as currently it authorizes any member of the M.I.T. community with a valid certificate.

6.1.4 Testing

Finally, ClassRank could utilize the built in test capabilities of Ruby on Rails much more extensively to ensure that the platform remains stable during future upgrades. Extensive testing is necessary to ensure the production release of ClassRank remains at a very high quality.

6.2 Recommendation Plugin

There are a number of ways in which to enhance the functionality of the recommendation plugin. In general, most areas would adapt the current algorithm to be much more dynamic.

6.2.1 Receiving and Adopting to User Feedback

Implementing a feedback system into the recommendation plugin would be extremely valuable. This could be done in a number of ways, the easiest allowing users to exclude some unrated items from their recommendations or to disregard some rated items from factoring into their recommendations.

Much more complex systems could also be designed into the plugin, perhaps taking feedback on the effectiveness of the algorithms recommendations. The plugin could then use this user feedback to optimize the prediction algorithm on an individual basis.

6.2.2 Abstracting Recommendation Functionality

Many variants of slope one prediction schemes exist, all relying on the same underlying data structures implemented in the plugin. Slope One algorithms differ only in how they form the predicted rating of an item for a user, therefore allowing the recommendation plugin to implement alternative schemes by rewriting the single function of the plugin that returns recommendations.

These enhancements could be taken a step further and provide an abstraction for the recommendation method altogether. Generalizing this portion of the recommendation functionality would allow new algorithms to be easily added to and utilized by the plugin.

Developers could then swap between multiple recommendation engines by simply passing a configuration option to the plugin

6.2.3 Testing Accuracy

Another area of possible work would be designing a test suite for the recommendation plugin. One possible implementation might have the plugin automatically divide the data into training and test sets and provide an interface to view the accuracy of the predictions. This would allow systems utilizing the recommendation plugin to view the accuracy of the algorithm on real-data from their systems.

References

Billsus, D., & Pazzani, M. J. (1998). *Learning Collaborative Information Filters.* University of California, Department of Information and Computer Science, Irvine, CA.

Goldberg, D., Nichols, D., Ok, B. M., & D. T. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM , 35* (12), 61-70.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval , 4* (2), 133-151.

Kautz, H., Selman, B., & Shah, M. (1997). Referral Web: Combining Social Networks and Collaborative Filtering. *Communications of the ACM , 40* (3), 63-65.

Lemire, D., & Maclachlan, A. (2005, February 7). Slope One Predictors for Online Rating-Based Collaborative Filtering.

Linden, G., Smith, B., & York, J. (2003, January). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* , 76-80.

Pazzani, M. (1999). A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review* , 393-408.

Teow, L. N., & Katabi, D. (2006). *Iterative Collaborative Ranking of Customers and Providers.* Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Labratory, Cambrdge, MA.

Appendix A: ClassRank Screenshots

When first accessing ClassRank, students are presented with a list of their past evaluations (**Figure 7**).

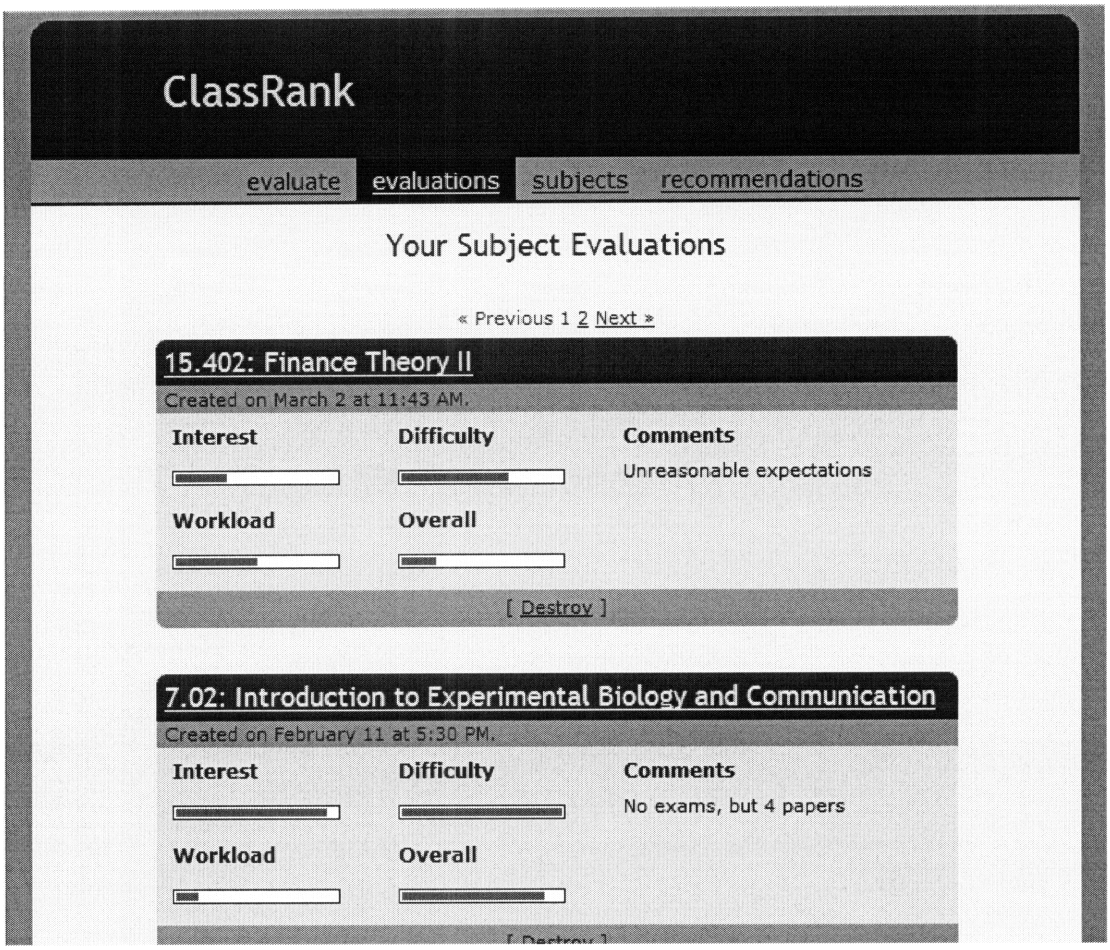

Figure 7: ClassRank screenshot of student evaluations.

To create a new evaluation, students submit a simple form with four ratings and a short comment for completed subject (**Figure 8**).

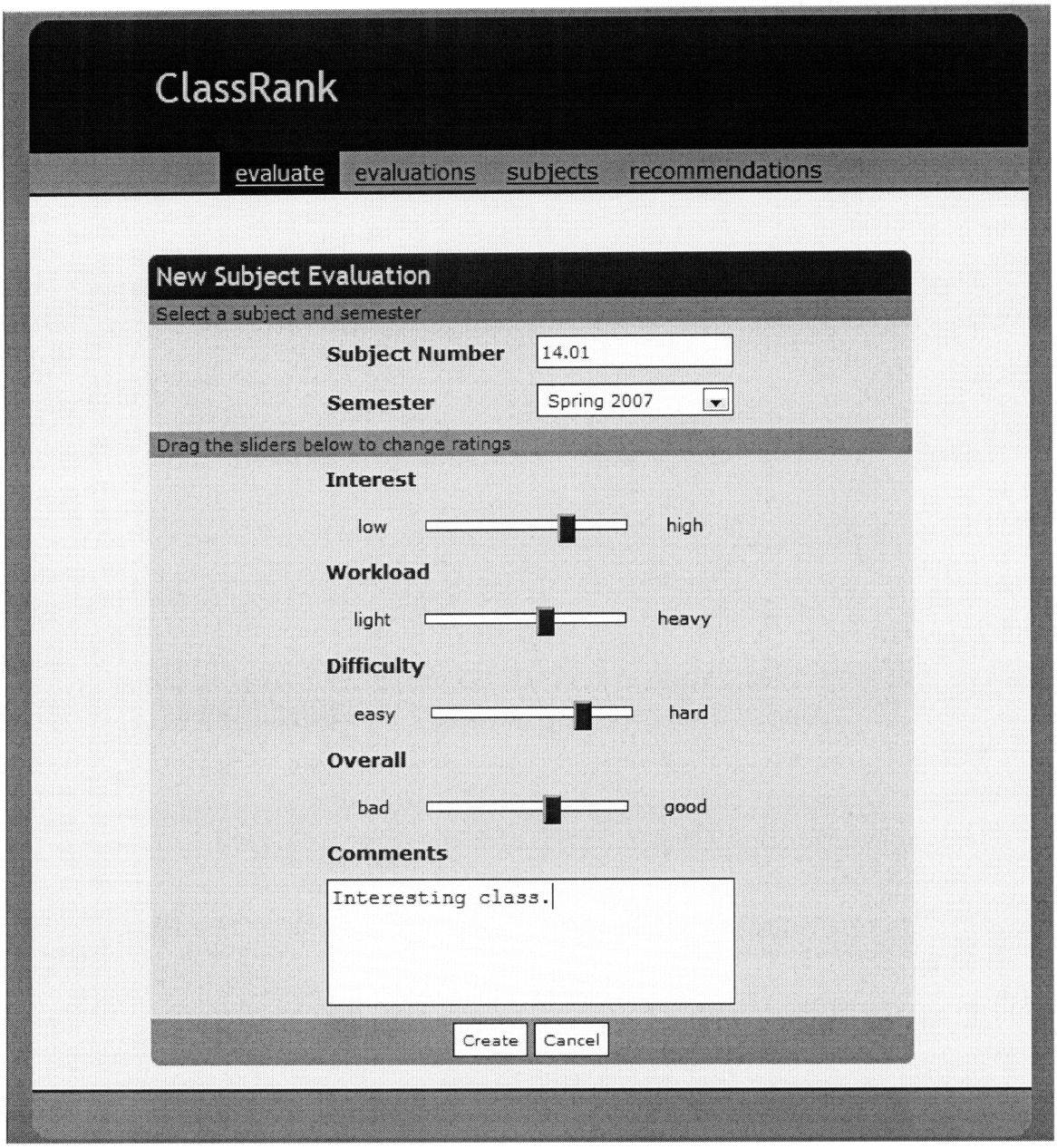

Figure 8: ClassRank screenshot of new evaluation.

Recommendations are presented to students as a list of subjects. ClassRank also displays the predicted rating of the student for each subject (**Figure 9**).

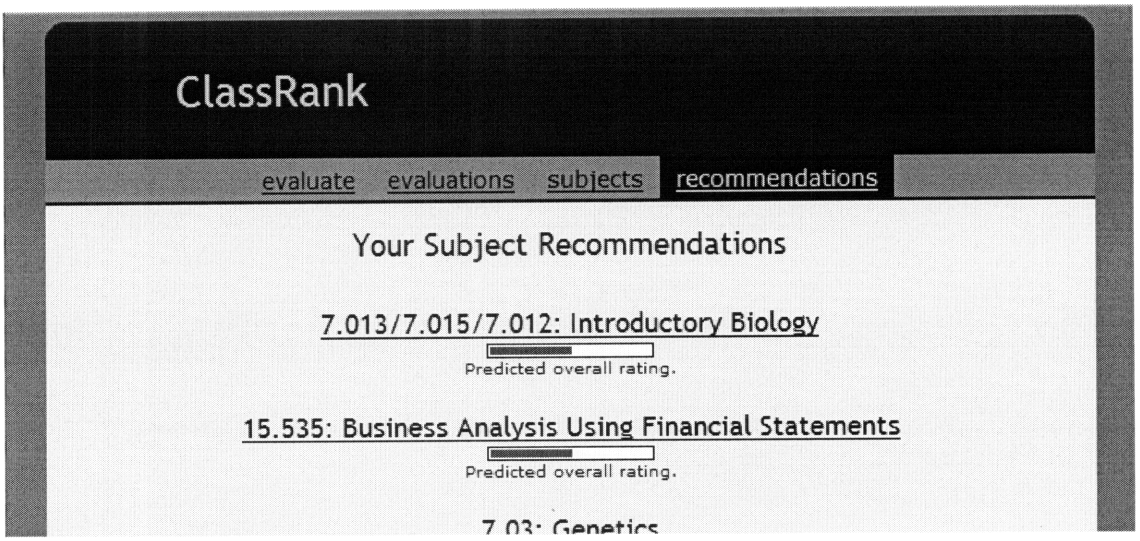

Figure 9: ClassRank screenshot of subject recommendations.