Lumen: A Princeton Course Search and Recommendation Engine

COS435: Information Retrieval, Discovery, and Delivery

Christopher Hay Junhan Chen

https://lumen-princeton.herokuapp.com

https://github.com/haychris/class-recs

I pledge my honor that I have not violated the Honor Code in writing this report.

-Christopher Hay, Junhan Chen

1. Introduction

Princeton offers myriad courses each semester across a multitude of departments, and for many students, choosing which courses to enroll in during course enrollment period can often be distressing. Not knowing what courses they want to take, need to take, or may be interested in, many students end up inefficiently browsing the course listings site (currently maintained by the Registrar's Office) as if aimlessly browsing a catalog, while those who have some sense of their interests are constrained to using basic query searches on fields such as course title or department.

Through Lumen, we aim to solve this problem by helping students quickly and easily search for relevant courses they would want to enroll in. Named after the standard measure of light, our product vision was to help students discover interesting and relevant courses they did not know existed in an easy and efficient manner, to inspire them to learn more and try out new fields by taking those courses, and to ultimately *enlighten* them. Lumen features a fast query search option that improves upon the Registrar's Office basic search features, allowing students to search for classes using queries such as "interesting LA" or "engaging lectures" and quickly obtain relevant results. An alternative personalized recmmendation option instead utilizes data on courses the student has previously taken, then returns tailored course recommendations based on how much the student has liked or disliked his or her past courses.

The remainder of this paper is organized as follows: *2.Goals & Measurement* defines Lumen's goals and metrics to measure success, *3.Product* describes Lumen's features, use cases, and workflow, while *4.Implementation* describes the implementation of Lumen. Finally,

5.Results presents and discusses Lumen's ability to achieve its goal as measured via metrics and user studies.

2. Goals & Measurement

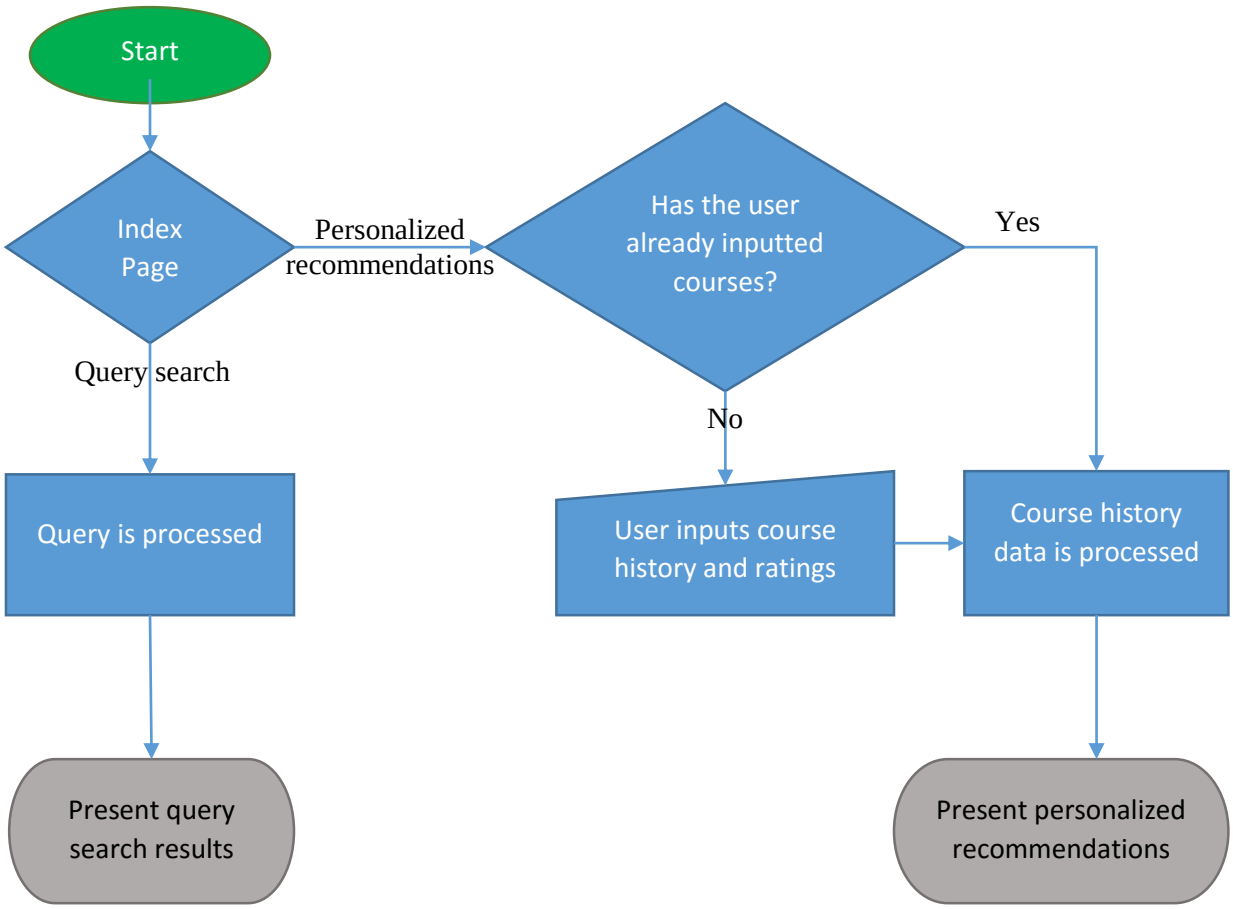
Lumen's main goal is to enlighten student users about courses that interest them by allowing the user to easily perform a search and efficiently receive relevant results. To measure success, we will ask selected users to rank all retrieved results as not relevant (score 0), slightly relevant (score 1) or relevant (score 5), and give one overall rating of the user experience. Using the result scores, we will then calculate narrow precision, broad precision, reciprocal rank, and expected reciprocal rank (ERR) to arrive at a total of four metrics to measure success. User experience will be rated on a 1 to 5 scale. Additionally, narrow precision will be measured by taking the number of relevant results divided by the total number of retrieved results, and broad precision will be measured by taking the number of relevant OR slightly relevant results divided by the total number of retrieved results. Reciprocal rank is defined as 1 divided by the rank of the first relevant result, and ERR will be calculated using the following equation:

$$ERR = \sum_{j=1}^{n} \left(\frac{1}{j} \prod_{k=1}^{j-1} \left(1 - \frac{2^{score_k} - 1}{2^5}\right) * \frac{2^{score_j} - 1}{2^5} \right)$$

3. Product

Our product is hosted at https://lumen-princeton.herokuapp.com, while the code repository can be found at https://github.com/haychris/class-recs.

The index page provides a very brief pitch of Lumen and aims to give the user a first impression that consists of a very clean and consistent chalkboard-themed user experience. From the index page, the user can then either do a fast query search for a course if they already have an idea in mind, or allow us to recommend them courses by clicking on the “Enlighten me” button on the bottom. A workflow diagram is presented below:



Both types of searches present the top twenty results to the user and styled identically to maintain a consistent user experience; results are ranked in order of descending relevance. The left-hand component of each result was styled to be reminiscent of a chalkboard and presents

information about the course's department, number, title, and professor, while the right-hand component presents the overall rating of the course from the most recent semester it was offered, as well as a sample selection of student comments regarding the course. Clicking on the left-hand component of each score reveals a more detailed breakdown of each course's ratings from the most recent offering into Classes, Coursework & Exams, Readings, Feedback, and Overall, with the individual ratings presented in a chalk-colored bar graph format. A link to the course's Registrar's page is also provided for each result, allowing the user to quickly and conveniently view the course's official page if the user wishes to do so.

For recommendations only, all results with a department designation of WRI or FRS (i.e. all writing seminars or freshman seminars) are filtered out and discarded. As course offerings for both type of courses are very volatile and often change annually (e.g. FRS101 from one year is unlikely to have the same content as FRS101 the following year, though the description for every freshmen seminar is identical), we concluded that these courses will rarely be relevant to the user and instead clutter the results page if included. With the benefits of omitting all WRI and FRS courses outweighing the costs, we decided to omit results with either department designation.

3.1 Search

At its core, our search system utilizes tf-idf weighting of documents scraped from the registrar and student comments for each course, and then adds additional boosts specific to the domain of course searches at Princeton (such as major/certificate boosts and a course PageRank boost). The exact details of these boosts will be discussed in greater detail in *4.2 Search Implementation.* Once the user issues a query, we pull the relevant columns for the tf-idf scores and the various boosts, and then score each course as the product of its tf-idf score and its boosts. After sorting the results, we present the user with a list of relevant courses. In addition, we scan

through all comments for the course (from most recent to least recent) and retrieve the two comments that contain the highest number of query terms. Then, we present these comments with bolded query terms in each presented course result.

3.2 Recommendations

If the user instead wants receive personalized recommendations based on their course history instead ("Enlighten me"), then we check for a cookie that indicates the user has already provided their course history. If such a cookie does exist, then the user is taken directly to the results page. Otherwise, if no such cookie exists, then the user is redirected to an input page where they are asked to enter past courses and ratings (scale of 1-5 or "Hate" to "Amazing", rendered using jQueryUI) for each course. By default, this input page starts by showing freshman year with four courses per semester; users can then choose to add additional courses for each semester (max seven courses per semester) or add/remove years (max junior year) as needed. The user can also select their major and certificate if they wish to do so. Once the user finishes and clicks on the "Inspire" button, the inputted course data is then saved to a cookie and the top twenty personalized recommendations are presented to the user. These results are styled identically to the query search results so as to maintain a consistent user experience, but if a course happens to fulfill a major or certificate requirement for the user, then an additional notification will appear on the left-hand side of that result. If the user wishes to edit or update their course history, then they are also able to do so from the results page by clicking on the prompt near the top of the page. Course and ratings data will automatically be read and parsed from the cookie then loaded so as to save the user the hassle inputting everything again.

Major and certificate requirements are currently limited to ECO, COS, and ANT and Applications of Computing, Creative Writing Certificate, and Certificate in Linguistics respectively, as this feature was meant to be a proof of concept at the time of writing.

After inputting their course history, the user is brought to a list of our top 20 recommendations based on their ratings for previously taken courses. The exact methodology for scoring will be discussed in detail in *4.3 Recommendation Implementation.*

4. Implementation

4.1 Data Pipeline

For this system, documents are defined to be the combination of data scraped from the course registrar and from comments and ratings provided by students. Specifically, we heavily modified a web crawler provided by COS 333 in order to automatically download every course page from the registrar since Fall 2013. Then, it automatically scrapes the title, professors, department/catalog number, description, and prerequisite fields from each html file and uses this to generate a CSV file containing all relevant features. In addition, we received three more CSV files from the Registrar's office containing the ratings and comments for all courses since Fall 2013. However, a few hundred of the comments are improperly formatted (and in a myriad of different improper formats). As a result, a small number of comments were ignored. Then, we collect all known information for each course id, and create various data structures for quickly looking up relevant information, preferring to redundantly use memory in order to increase performance since the memory used is still well within a typical laptop's memory constraints. In order to create the documents used for tf-idf processing, we joined together all collected

information (as listed above) for a course id, across all semesters. Then, we utilized the TfidfVectorizer in the Scikit-Learn[1] python module. Here, we use a set of 300 pre-defined English stop words from the Glasgow Information Retrieval Group. Then, we use a customized version of the NLTK[2] word tokenizer and WordNet lemmatizer that is customized to correctly parse the three letter department names and the two letter distribution names. Additionally, the lemmatizer was customized to also convert all three digit numbers to their "root" course number, i.e., a "435" becomes a "400". With this lemmatizer, we are able to group terms by their relevant root words, allowing for improved information processing. In addition, any term that appears in over 90% of all documents is dropped, and every term that only appears once is dropped.

Next, we cluster each document based on the tf-idf matrix, using K-Means clustering with K=50 and by utilizing the best clustering in 20 attempts, using an inertia score to rank clusters. Although we implemented K-Means, Affinity Propagation, Non-Negative Matrix Factorization, Latent Dirichlet Allocation, and DBSCAN (using Scikit-Learn), we found that K-Means qualitatively performed the best. To make this decision, we looked at the most representative features for each cluster produced by each algorithm, and qualitatively decided that the clustering produced by K-Means was the best. From the clusters produced by K-Means, we could easily see a "math cluster", an "architecture cluster", an "independent work cluster", etc., but the same could not be said for each of the other clustering methods.

After finding a clustering, we then needed a similarity function for points in the K-Means vector space. While a simple inverse distance function was found to achieve decent results, we found that adding a slight transformation to the K-Means space improved results significantly.

---

[1] L. Buitinck et al. "API design for machine learngin software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning. 2013, pp. 108-122*

[2] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python.* O'Reilly Media, 2009.

Here, we define the similarity function between two points as $\frac{1}{EuclideanDistance(x_1,x_2)^3+0.5}$. With this, we then find the similarity between every course and every cluster center, then normalize each vector of similarities with an L1 normalization to produce a probability-esque metric for each course.

In addition, we perform an extra step of processing in order to create a graph structure for the courses. In this step, we search every document for the mention of other courses, and then create a dictionary for each course that contains each course mentioned as the key and the number of times it was mentioned as the value. Then, each of these dictionaries is stored as the value in an aggregating dictionary. With this, we then use the Networkx module to build a graph where each course id is a node, and mention is an edge. As a result, we end up with many edges between two pairs of nodes (representing many mentions of one course in the other's document and vice versa). These multi-edges are then aggregated into a single undirected edge with weight equal to the number of edges that were collapsed. With this graph, we were able to visualize how the courses were structured together, and ultimately we were able to calculate the PageRank for each course by using this graph.

Lastly, we manually constructed a CSV file that will contain the course requirements for each major. In this, each row contains the requirements needed to obtain the specified major for that row, and each column contains a group of courses. If a course is taken in each column for a given row (without repeating the same course across columns), then the user has fulfilled all requirements for that major. In order to save time, but to still demonstrate the proof of concept, we only constructed these rows for three majors: Computer Science, Economics, and Anthropology. Additionally, we constructed another CSV file with a similar structure that will contain the course requirements for every certificate. Again, to save time while still

demonstrating a proof of concept, we only constructed the rows for three certificates: Applications of Computing, Creative Writing, and Linguistics. With these CSVs, a "Planner" object was constructed that contains methods for determining if a course satisfies any of the requirements for a specified major. In addition, the object pre-computes a matrix where each major is a row, each course_id is a column, and each entry is a boolean indicating whether that course satisfies any of the requirements for that major. A similar matrix is computed for certificates. As we shall see later, these matrices allow for efficient computation of boost vectors for major/certificate, as ultimately this now amounts to returning a row from each matrix.

4.2 Search Implementation

As mentioned earlier, the implementation for search primarily uses the tf-idf scores to weight each document for a given query. Specifically, we lemmatize the query and then sum the tf-idf columns for each transformed query term. Then, we normalize the scores such that the max score is always 1. Next, we boost these scores by a few more factors. First, we boost courses that satisfy requirements for the user's major. This is done by retrieving a row from the "Planner" object that specifies whether each course_id satisfies any requirement for the user's major. Then, this vector is transformed such that every True entry becomes equal to the desired boost value, and every False entry becomes equal to 1. For search, the boost used for majors is 1.3. In addition, we apply a similar boost for courses that satisfy the user's certificate requirements. For search, the boost used for certificate is 1.2. Lastly, we obtain the pre-computed vector of max-normalized PageRank values for each course, and then multiply this vector by the boost value; for search, the boost value used is 1.1.

Notably, we choose to implement each boost as a multiplicative boost instead of an additive boost. As such, we define the final score for a course as $final\_score = tf -$

$idf\_score * major\_boost * certificate\_boost * pagerank\_boost$. We chose to implement multiplicative boosting in order to guarantee greater consistency across the score distributions. That is, we're more likely to get a smooth distribution of overall scores, instead of needing to carefully re-weight boosts in order to avoid one from constantly dominating the others.

4.3 Recommendation Implementation

In order to compute recommendations in a quick and efficient manner, we rely on doing as much computation in the pre-processing stage as possible. As a result, much of the computation done for recommendations was mentioned in the Data Pipeline section. Specifically, we are going to be using the vectors for each course that contained the probability-esque measure of that course being in each cluster.

With a dictionary to easily lookup these probability vectors, we begin the recommendation process by initializing a vector of "cluster scores". Then, we loop through the courses that the user rated. For each course, we first translate the department/catalog number into a course id, and then grab the probability vector for that course. If the user rated this course X, then we add $(X - mean\_rating) * prob\_vector$ to the vector of cluster scores. After doing this for each course rated by the user, the "cluster scores" vector can be thought of as a weighted sum of ratings for each cluster. As such, it allows us to assess other courses. So, for every course not rated by the user, we grab its probability vector and then compute the dot product between the "cluster scores" vector and the probability vector to get a personalized rating for the course. Once this is done for every course, we max-normalize the scores and then apply the boosts for major, certificate, and PageRank. Each of these boosts is applied in an identical manner as done

in search, with the exception that the boost values are different. For recommendations, the major boost is 1.007, the certificate boost is 1.005, and the PageRank boost is 1.05.

5. Results

5.1 Search Results

User studies were conducted on four users using query search, where users were asked to search using a query of their choice and rank all results as not relevant (0), slightly relevant (1), and relevant (5). The four metrics as calculated from each user's relevancy assessment are presented below:

Table A. Measurement of Results for Query Search

Metric	User 1	User 2	User 3	User 4	**Avg**
Query	"Easy PDF"	"Brian"	"Robotics"	"Fun"	
User Experience	4.5	5	5	4.5	**4.75**
Precision (narrow)	0.7	0.8	0.66	0.45	**0.65**
Precision (broad)	0.8	0.8	1	0.6	**0.8**
Reciprocal Rank	1	1	1	1	**1**
ERR	0.98	0.98	0.97	0.98	**0.98**

Users praised the user experience when using Lumen's query search feature and mentioned how fast and easy it is to use. Lumen's precision as measured if only counting the completely relevant courses averaged to be 0.65, while a broader definition of precision that includes the slightly relevant courses averaged 0.8. The first result returned for all users were

also all completely relevant, giving us a perfect average reciprocal rank of 1 and consequentially a high average ERR of 0.98. Overall, Lumen's query search feature ended up being quite accurate and was well received by the users surveyed.

5.2 Recommendation Results

User studies for personalized recommendations were also conducted, and the results are presented below:

Table B. Measurement of Results for Personalized Recommendations

Metric	User 1	User 2	User 3	User 4	**Avg**
User Experience	4	5	5	4.5	**4.63**
Precision (narrow)	0.5	0.15	0.45	0.55	**0.41**
Precision (broad)	0.65	0.3	0.6	0.75	**0.58**
Reciprocal Rank	1	0.1	1	1	**0.78**
ERR	0.98	0.12	0.98	0.98	**0.77**

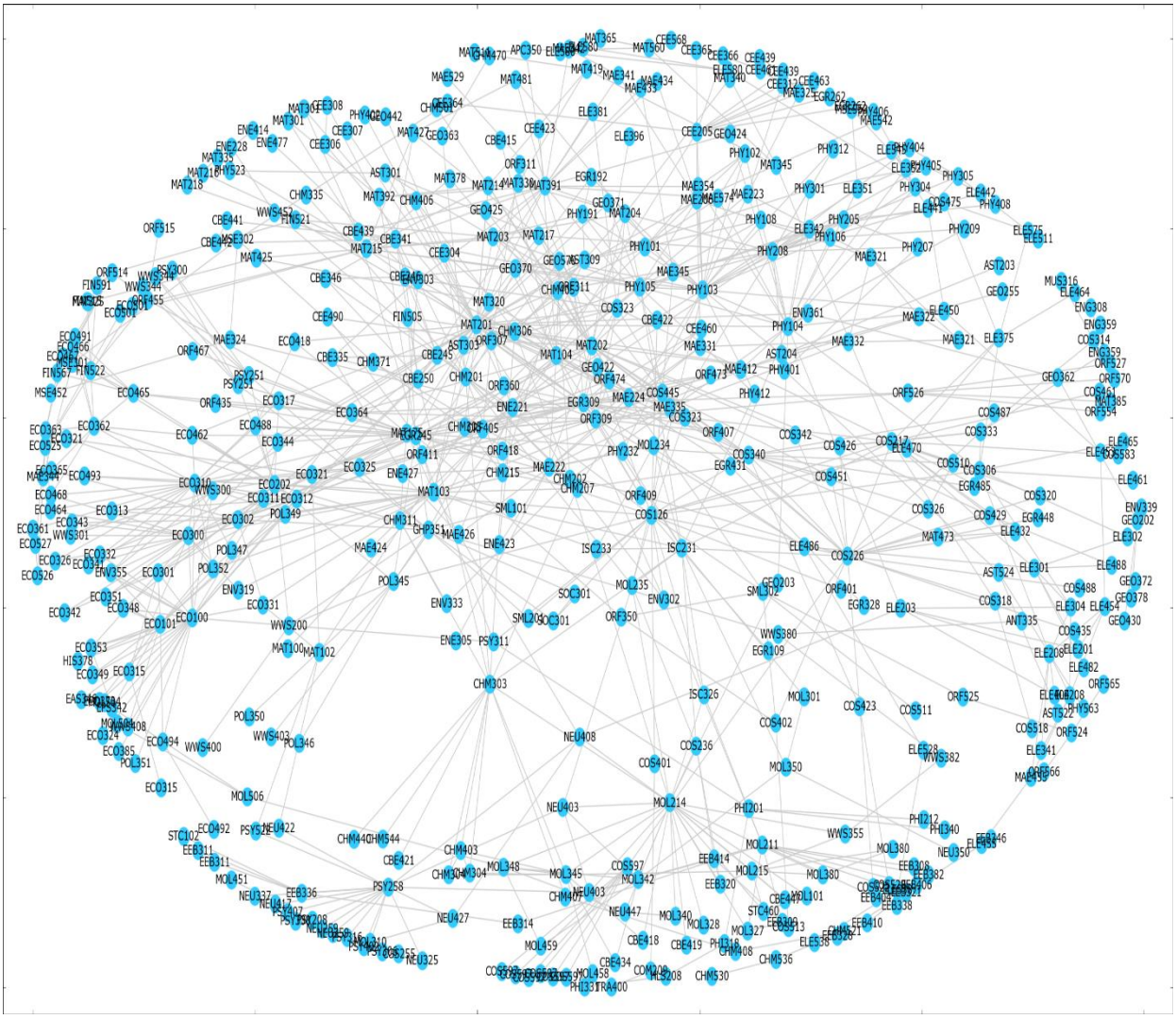
Users also liked the user experience for Lumen's personalized recommendations feature, but some did mention that it would have been nice to somehow streamline the initial course input workflow a bit more. Compared to query search, both the narrow and broad definitions of precision are lower, now averaging 0.41 and 0.58 respectively; users said that most results were relevant, but the irrelevant results were often courses that they had already placed out of or took a more advanced version of instead (e.g. MAT204 over MAT202). A possible future refinement that addresses this is expected to greatly boost precision. Reciprocal rank and ERR were also

quite high for most users; the one outlier (user 2) had low reciprocal rank and ERR due to Lumen recommending placed-out courses as mentioned above.

5.3 PageRank and Course Network Analysis

In computing the PageRank scores for each class as mentioned previously, we came across an interesting finding regarding the structure of the course graph. As the edges occur whenever a course's document mentions another course (primarily in the comments or course prerequisites sections), the graph naturally segments itself into a number of connected components that revolve around a particular department (mostly language departments) or source of material. However, there is also an enormous connected component spanning various departments, with 467 courses as nodes and 1041 edges. If the entire graph is denoted as G, then let the subgraph containing this connected component be denoted as G1.

For G1, we found that the radius is 7, the max degree is 44, the average degree is 4.458, the mean eccentricity is 8.952, and the average shortest path length is 4.595. In addition, the distribution of degrees follows a power law. As a result, G1 satisfies many of the properties of traditional social networks. A visualization of G1 is presented below.



Subgraph G1 of Course Network

6. Conclusion

Lumen aims to enlighten student users about courses that interest them in an efficient and accurate manner, and preliminary user studies have shown that Lumen was able to achieve its goal. Users praised Lumen's user experience and quality of results for query searches, while personalized recommendations were satisfactory but left room for continued refinement.

In the future, ideas for further exploration include testing the effectiveness of peer recommendation methods such as user- or course-based matrix completion (requires a critical mass of user course data), while automatically omitting course results if the user has already placed out or taken more advanced versions would likely increase Lumen's personalized recommendations. Additional features such as automatically retrieving and loading course history based on the user's netID (assuming CAS support is added) would further improve the user experience.