OF COURSE! Using Bayesian Inference to Build a More Dynamic Course Search

CS221 Final Project by Michael Dickens and Mihail Eric

December 12, 2013

TODOs

Metric to employ:

If no classes returned, search result of 0.

Only top 5 course searches considered in evaluation.

Things to consider:

50 points if course desired in first hit, then 30, 25, 20, etc.

1) if course in same sequence as searched for course (+10,+8,...)

2) if course in same department (+5,+4,+3,+2,+1)

3) if course is same course in a different department (+10)

4) if course has same instructor (+10,+8,+6,+4,+2)

5) if course descriptions are sufficiently similar, judged subjectively (+5,+4,...)

If specific course being searched, then we know that the user has a particular course in mind for their search.

Data (25 searches across different 5 different domains (5 searches per domain))

1) specific course code

2) instructor name

3) course title searchs (including segments of course titles)

4) department code

5) more complex queries ('courses taught by', "cs109 and cs154")

Introduction

ExploreCourses is a search engine used regularly by the Stanford University community for browsing and finding courses. At the present moment, ExploreCourses can support basic query searches, including somewhat accurate retrieval given a course code and the exact title of a course. Almost any query search that does not fall into one of these two classes of searches will either return unrelated class results or more often no results at all. For this project, we sought to develop an improved course searching program that would be more robust in that it could support more diverse user input queries and would also be more dynamic in that the courses that were returned were more related given a universal metric that we define for assessing relatedness. The metric will be explained in a later section.

To achieve improved robustness, we implemented some basic natural language processing schemes for extracting useful and relevant information from a user input. The information that we were specifically looking for included course titles, course codes, department codes, and instructor names. To find more related courses, we extracted a variety of features that we considered relevant from all the data we could attain about a course and then we created a course-relatedness "graph" that assigns a relatedness score to each pair of classes, given their extracted features. To compute the relatedness score, we utilized a Bayesian inference scheme whereby we calculated the probabilities that two courses are related given that they have a pair of features in common. The Bayesian approach and the features used will be explained in later sections.

Feature Extraction

In order to determine an accurate label for relatedness between two courses, we had to extract a useful collection of features from the data for each course. This required some experimentation in order to find a good balance: too many features and the algorithm runs slowly; too few, and we cannot perform useful inference.

We used the following initial set of features:

- words in the title
- words in the description
- course code name
- course code ones digit
- course code tens digit
- instructors
- minimum units
- maximum units

Later, we took each of the course code features and combined them with each title, instructor, and description feature to create a set of binary features. This roughly quadruples the number of features in the set. We also tried using word bigrams in the title and description, but this did not add substantial benefit.

A Bayesian Approach to Course-Relatedness

To determine course relatedness, we began simply by taking the total number of matching features between two courses. This approach proved too coarse: it matched courses with many common features that weren't actually all that related to each other.

A sudden insight came when we realized we could do much better by taking a Bayesian appraoch. Instead of simply counting the number of features in common, interpret each feature in common as Bayesian evidence that the two courses are related and perform a Bayesian probability update. Similarly, if a feature does not occur in common between two courses, consider this evidence that they are not related.

Then, instead of considering each feature as equally strong evidence, weigh a feature against the prior probability of that feature occurring. We figure out the prior probability of a feature by counting how frequently it occurs in the database.

Thus, to find the probability that two courses are related, we update a prior probability estimate with the evidence given by each feature found in the two courses.

Query Parsing

In order to satisfy a user's input queries, we need to extract useful information from a given query. Using the Python Natural Language Processing Toolkit, we employed the following natural language processing scheme: tokenize query, tag with parts of speech, chunk appropriately using regular expression grammars. Once a query is chunked, useful information can be derived through analysis of the corresponding parse tree. Using this scheme, we are able to support user query searches consisting of more complex phrases

We search for instructors using a simple grammar that searches a string for sequences of proper nouns. We support course code and department code searches by tokenizing a query into unigram and bigram tokens and checking a set of course/department codes for matches. We also support title searches by searching for the input string in a set of all course titles. Talk about use of NLTK POS-Tagger, chunker, regex grammar.

Data

We used the ExploreCourses Java API to acquire information related to the 11,613 courses listed for enrollment for the 20132014 academic year. We populated a SQL datebase, using the sqlite3 Python library, with the following information for each course: course title, course code, instructors teaching the course, minimum units of credit received for taking the class, maximum units units of credit received for taking the class, and course description.

However, for the purposes of the assessment, we used a database of a reduced subset of approximately 170 random classes taken from the CS and MATH departments. We had to utilize a reduced database because it was too time-consuming to create a comprehensive relatedness graph for all 11,613 courses.

Metric for Assessment

We developed a simple point-based metric to objectively determine the quality of our course searcher as compared to ExploreCourses. To evaluate the performance of our searcher, we generated a random subset of 25 courses taken from our reduced database. We subdivided these 25 courses into 5 domains of query searches with 5 courses per domain: searching a specific course code such as ‘CS109’, searching for an instructor name such as ‘Mehran Sahami’, searching for a course title or subset of such as

TODO: Make this an itemized list

3) course title searchs (including segments of course titles)

4) department code

5) more complex queries (’courses taught by’, “cs109 and cs154”) The robustness of our searcher as compared to ExploreCourses using our metric

Results

Further Work

**Support searches based on user’s history using more standard supervised machine learning classifiers.

**More complex query searches

**Spell-correction (edit distance)

**Play around with other schemas for determining relatedness coefficient

**Incorporate into the actual site.

References