OF COURSE! A Bayesian Inference Approach to More Dynamic Course Seach

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:

20 points if course desired in first hit, 18 if in second, 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 has same number of units? (+1)

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

5) number of nouns that course descriptions have in common (with expected course)

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

Explore Courses is a search engine used regularly by the Stanford University community for browsing and finding courses. At the present moment, *Explore Courses* can support basis query searches, including somewhat accurate retrieval given a course code and the exact title of a course. 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 from each course.

Bayesian Approach to Course-Relatedness

**NOTE: THIS IS JUST A COPY-PASTE OF MICHAEL's NOTES!!!

First I implemented naiveWeights(), which worked sort of well. Then I implemented bayesWeights(). Finding P(R) and P(F) wasn't too hard, but how to find P(F—R)?

Also, at a certain point I realide I also need to update on a feature not being in the second set as well as a feature being in the second sent–these both provide Bayesian evidence.

Right now for P(F—R) and P(F—R) I just threw in some constants. This seems to work okay.

Then there's the question of how to combine each conditional probability. First I tried using combineProbs(), but that doesn't seem to work too well. It works a lot better to simply take the sum of all the probabilities. Why?

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 "Explore Courses " Java API to acquire information related to the 11,613 courses listed for enrollment for the 2013-2014 academic year. We populated a SQL database, 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 150 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

Metric for Assessment

We developed a simple point-based metric to objectively determined the quality of our course searcher as compared to "Explore Courses."

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