Project Progress Report

Author: Michael Dickens

Author: Mihail Eric

Preliminary Query Searching

We maintain inverted indexes for a variety of features, including instructor, location, time, etc. Then we match queries against these indexes to find appropriate courses. In other words, we maintain a database mapping from a course instructor to the courses taught by that instructor. We maintain similar databases for a number of different broad features (location, time, unit count), as this will allow for efficient search of basic input queries.

For certain types of queries, we only need to perform minimal NLP. For example, if the user enters the words 'Jerry Cain' into a query, she almost certainly wants to find courses taught by Jerry Cain. In this case, we simply need to recognize that Jerry Cain is an instructor. We can do this by matching the string 'Jerry Cain' against a set of instructors. This sort of analysis should work for a number of different forms, and here is where our collection of category database-inverted indices would be useful.

Feature Extraction

We can examine a query in terms of its features which can be extracted via a combination of POS-tagging and chunking. We extract the features of a query (instructor, day of week, etc.) and then match these features against a related feature set for each course. We can use a fairly simple search algorithm to find the best-matched courses for a query--assign weights to each feature and find the course with the highest-weighted match. There are some subtleties to this approach, but for the most part it is fairly straightforward. However, small variations in implementation can lead to non-trivial differences in outcome; therefore, we must experiment to determine what search heuristic works best.

The biggest challenge, then, arises in extracting the features of both queries and course descriptions. This portion of the program requires intelligent natural language processing. Evaluating queries perfectly requires much more advanced NLP than we have the time or expertise to implement; so we identify simpler, but still useful, heuristics that we can use.

We maintain a collection of features extracted from all available courses as well as a collection of features for each search query. Then it becomes a matter of figuring out a good heuristic for assigning a score to the matching between a given course and a search query. We can then simply return the N highest scores achieved for matchings between courses and a query, for some previously decided upon n. One way we are considering to represent the feature-score approach is to assign a sparse feature vector for each query and course and then compute a score between a course and feature as a dot product between two feature vectors. Then we find the N courses with

the highest scores and return them to the user.

Parsing

We extract relevant features by developing a syntactic parser. This is for the purpose of extracting useful information from a query/course description that can then be used for effective information retrieval. A syntactic parser provides us with useful information about query/course features.

We begin processing our query through the use of the Part-of-Speech (POS) Tagger included in Python's Natural Language Toolkit. Once a sentence has been properly tagged, we can identify chunks by sequences of tags combined with an appropriate regular expression parser. Chunking is a more feasible method of extracting information from text through smaller sequences of subtokens without having to parse the entirety of an input. We will also implement a named entity recognition system based on a collection of feature extractors, so that we can deduce proper nouns of interest in a query (e.g. professors, building names, etc.).

We create a lists of categories of interest, containing words that belong to that category as follows:

`professors=['Jerry Cain', 'Mehran Sahami', 'Vadim Khayms', etc...]`

`buildings_on_campus=['Cubberley Auditorium', 'Mudd Chemistry Building', 'Green Earth Sciences', etc...]`

Given the query, 'courses taught by Jerry Cain', NLTK POS-tagging will give us the following:

`"[('courses', 'NNS'), ('taught', 'VBD'), ('by', 'IN'), ('Jerry', 'NNP'), ('Cain', 'NNP')]"`

After appropriate chunking using the provided NLTK chunker, we have the following:

`(S courses/NNS taught/VBD by/IN (PERSON Jerry/NNP Cain/NNP))`

Notice that this particular instance of chunking allows us to immediately determine that our query involves a name search, and is therefore most likely related to searching for courses taught by a certain professor. Then our task is to develop a series of good chunking schemas based on the format of course descriptions and the queries that we anticipate, whereby the desired information can be efficiently extracted.

Concept-Related Course Search

We may also support searches for courses related to a particular concept, e.g. 'courses about AI'. This requires our program to understand what concepts are related to AI and find those concepts referenced in a course description. For this, we may use Princeton's WordNet, which provides a service very similar to what we need: a database of connections between words. This is somewhat like a thesaurus, but more general in that it gives words that are related to each other and not just those with isomorphic definitions. The latter would be too constraining--a search for 'courses about AI' should return courses on machine learning and NLP, even though neither of those is

synonymous with AI.

We have also considered building a graph of relations between courses and then using this graph to find courses in the same concept space as a subject. Let us return to the example of 'courses about AI'. We index the course list and determine that CS221 is related to CS229 (perhaps they have similar terms in their course descriptions). The course title of CS221 includes the phrase 'Artificial Intelligence', so we know it is 'about AI'; CS229 does not contain this phrase anywhere in its description, but our program infers that it is 'about AI' because it is related to CS221.

In more abstract terms, the program finds courses directly related to a query and then searches the course graph for the nearest neighbors, where proximity is determined by the conceptual similarity between two courses.

We have not spent much time considering related-course search; it seems considerably more difficult to handle than most sorts of queries, and may be less generally useful. However, we believe that a fully-featured course search ought to allow the user to search for courses related to a particular concept.

Example Queries to Support

`'courses taught [by {NAME PROFESSOR}] [in {QUARTER OF YEAR}] [at {TIME OF DAY}]'`

`'courses taught by {PROFESSOR QUALIFIER}'`

`'courses [that meet] on {DAY OF WEEK}'`

`'courses in {NAME DEPARTMENT} department'`

`'courses on {TOPICS/SUBJECT MATTER OF COURSE}'`

Conclusion

Our work on this project can be reduced to the following sets of subtasks that we need to accomplish:

- Finish implementing database management
- Feature extraction of queries and course descriptions
- Parse texts for information extraction through a combination of POS-tagging, chunking, and named entity recognition
- Implement an algorithm for matching query features to course features

Our next steps involve implementing robust feature extraction and using extracted features to naively match courses. New challenges may arise as we implement our program in more detail, but we feel confident that we can implement a useful course search in the coming weeks.