Final Project

http://cmpt456.csil.sfu.ca:8020

Branden Crawford - bsc7@sfu.ca
Ryan Wong - yfw1@sfu.ca
Jared Daley - jsd12@sfu.ca
CMPT 456 - Spring 2013

I. Why?

Teachr is a vertical search engine for cs.sfu.ca that focuses on searching for teachers in the Computing Science department. It takes in any text query from a user and tries to match it with information we have gathered for each teacher. The information available to us consists of the classes a teacher has taught, the news articles they are included in, their dossier and contact information, and as an added bonus, their results on RateMyProfessor.com.

While we have made the site gauged more towards students it is useful for anyone looking up information on any teacher in the department. Students can use it to get a feel for how well a teacher might teach a particular iteration of a class. At the moment, one might go read a few RateMyProfessor reviews and sign up. However, a RateMyProfessor review is extremely objective and doesn't really tell you much about whether a teacher will be good or not for a particular class. This is especially true when a teacher has taught one iteration of the class in question (which went poorly), but also teaches a much more successful class often. These reviews will overwhelm the poor reviews that may give a skewed result in this particular instance.

However, with a teachers Teachr dossier in front of you, you can read about a teachers educational background, teaching and research interests, publications, previously taught classes, related news articles, and see their rating on RateMyProfessor. This allows you to gauge whether or not a teacher is particularly interested in the subject they're teaching, whether they're experienced in the field, or whether they've had several semesters to hone their curriculum. These are much better ways of estimating a teacher's success in teaching a course. If you get an predominately AI-focused professor teaching a databases course, you may choose to stay away.

In addition to students, anyone trying to do basic research on a teacher would benefit from Teachr as well. A lot of information is packed in to their dossier, as has been stated. If the sufficient information isn't given on the engine itself, we've tried to return the other points of interest that a researcher may be looking for, such as publications by the teacher, their webpages, or news articles related to them.

Also, there are sessionals that may have only taught one or two classes. They may not be actual professors and thus don't have any information available on the cs.sfu.ca page that directly links to them. Teachr aggregates the information from course outlines in order to give them as much of a profile as possible.

The information on cs.sfu.ca is perfectly sufficient for our purposes, save for the RateMyProfessor information. This is why we picked teachers to do a vertical search on: the abundance of information. The news articles were available to parse, as were the teacher profiles and the course outlines. Then it was just a matter of arranging them into a presentable and useful manner. In fact, we have even more information available to us in the form of teacher webpages, but unfortunately, the limit on time made parsing these pages impractical.

II. What?

Teachr searches over three major sets of data: the names of teachers, their interests (research and teaching) and publications, and the courses that they've previously taught. Names and courses are well formed, which make separating the three sets relatively easy. Queries are strings consisting of words from one, two, or all three sets. Usually, users will choose one set, but Teachr can handle queries from all three sets at once. Results are organized by relevance.

You can choose to search by first or last name: "Richard" or "Vaughan" or "Richard Vaughan" or "Vaughan Richard". This will return the set, as expected, of people named 'Richard', 'Vaughan', 'Richard

Vaughan', or in the last case, people with 'Vaughan' in their name union-ed with people with 'Richard' in their name with Richard at the top.

You can search by class number: "CMPT 456" or "456" or "413" or "CMPT 371". In actuality, the word 'CMPT' (as well as 'MACM') is a stop word so only the number is taken into account. If a number is entered into a query, it is treated as a class number. Numerical searches in the other two categories make no sense. Even teachers with "the third" as part of their name (of which there are currently none) would have "III" instead of the number "3". "371" would return the list of teachers that had previously taught CMPT 371, as would "CMPT 371".

You can choose to search by interests: "Bioinformatics" or "Robotics" or "Robots" or "Medical" or "Medicine". Interests are words that aren't stop words, aren't numbers, and aren't in the engine's set of names. Queries of this type are stemmed (by a Porter stemmer) and cross checked on an index of stemmed words and their non-stemmed equivalents. Therefore, words of the same stem should return the same results. "Robotics" and "Robots" will return the set of teachers with the stemmed-word 'robot' in their stemmed interests and publications.

Finally, you can search for a query that consists of one, two, or all three types of queries. As has been stated, results are organized by relevance. So a search for "Vaughan Richard 300 Robot" will return all the 'Vaughan's, all the 'Richard's, all teachers that have taught CMPT 300, and all teachers interested in robots. The results will have Richard Vaughan at the top since he is in all four categories, while another Richard that taught CMPT 300 would be lower down since they'd only be in 2 or 3 categories.

Also, there is a spelling corrector based on the edit distance algorithm that will ask you if you meant query X in your current query returns less results than query X. Sometimes this comes up with a good estimation of what you were trying to say. For example, "richord vughan" asks if you meant to search "Richard Vaughan". However, it is based on unigram parsing, and so sometimes the engine will return odd results for correction. For example, "juan pie" will ask if you meant to search for "Jan Pei" which isn't a name. However, it will return "Jian Pei" when searched thanks to the fact that "Pei" returns "Jian Pei" and each word is parsed separately.

Feel free to try as many queries as you can think of. Good things to try are, as has been stated, names, class numbers and topics in the field of computer science. Be aware that there are stop words, including the word 'computer' and 'science', which will return nothing as they returned nearly the entire set of teachers.

<u>III. How?</u>

i. Getting the Data - Parsing and Crawling

To begin, we needed the dataset to be parsed. The first part of this was to gather the teacher and course information. We used separate web crawlers for parsing the teacher and course information and for parsing the news and events and RateMyProfessor information. This enabled us to use methods for parsing these websites which best fit the information contained in them.

The web crawler for parsing the teacher and courses pages used the python urllib library to retrieve web pages. For the teachers pages we used the listing of faculty and emeritus professors on the cs homepage to get a list of pages to parse. Each one of these pages follows the same general format, so information could be extracted from these pages by searching for specific headers and html tags and then storing the information in them. For items like publications and education, we use the html list tags and page breaks to correctly break up and store these items as lists. Once all the information was gathered, we directly connected the python to the SQL database and filled it with information.

For the course information, we pulled data from the list of course outlines on the cs homepage. We used the Python HTMLParser library, which is the standard python library for parsing tag marked text, to parse this information. By using this library, we were able to easily extract objects from the course outline pages. Each piece of course data is then added to the courses table in the database as a separate entry. Each piece of data is then cross-referenced with the teacher who taught the course for easy recall.

The web crawler for news and events was implemented using Scrapy, a python based web crawling framework. We need all news from news and events within http://www.cs.sfu.ca to show all related news to each professor. In order to make all this data available, the web crawler needs to crawl all the webpages which are located in the news and events path within the http://www.cs.sfu.ca domain. When each page is crawled, all the html code will be parsed if it is a news page. Each news page is parsed to see if it contains one or more professors' names. Other pages such as the outline of the news page will not be parsed. Instead all links to actual news pages are followed using our user-defined rules. The data that we need is the title of the news, the hyperlink and the year, These can all be retrieved by searching the specific html tags in the pages. The results will then be passed into user-defined functions with SQL commands to create and update the relationship between professors and news by linking the teachers and news tables.

Finally, we needed the data from RateMyProfessor. I took the names that had been gotten from the profile pages and course outlines. I then used Python and BeautifulSoup to crawl over the Simon Fraser University section of RateMyProfessor.com for links to profile pages of teachers whose names matched the ones on the site. I then parsed over these pages to get the aggregated data given in the results. The Python script outputted a list of SQL commands that linked a teacher_id with a set of results. When a teacher page or teacher result comes up, this table is joined in order to get the result.

ii. The Website - User Interface and Backend

The website is built on an Ubuntu Server and a basic LAMP server. The graphics and user interface were done by hand using a graphics framework called Twitter Bootstrap, with a few custom images, such as the one for a missing picture in the results and the ones for the how-it-works section, done by the group. We used jQuery to make dealing with the Javascript easier.

The main website structure has been done in PHP using a back-end framework called CodeIgniter. CodeIgniter makes dealing with the MySQL database trivial using an interesting version of the MVC paradigm. The query handling is done using Python scripts called from the PHP backend, as Python is better at handling text thanks to its easy-to-use libraries. Python libraries that were used include the following: Pickle, to store and open the indices and lists of words from file; NLTK, for the Porter Stemmer and tokenizer; BeautifulSoup, for handling web pages; MySQL-Python, for dealing with MySQL through Python.

iii. Data

Teacher, course, news and RateMyProfessor data is stored in the MySQL database, having been parsed from cs.sfu.ca. We do this in order to not have to parse each page each time we make a query. We have scripts that can be run to update the data on the page, though these are not active due to limited system resources and the fact that we don't want to bog down the system. Turning them on would just be a matter of a well-placed CRON script. The tables are linked together using linking tables, i.e. teachers and courses are linked together using a table of teacher_ids and course_ids called teachers_courses. This is valid SQL practice and makes sure that everything is pretty well normalized. Other data, such as the search indices, stop word list and list of unstemmed search terms, are saved as pkl files which are to be read in by Python as particular data structures when needed.

iv. How Autocomplete Works

The autocomplete part of the search engine is actually a library within Twitter Bootstrap called 'typeahead'. Whenever a page with a search box is loaded, the names of all the teachers are stored in a hidden div at the bottom of the page. The wonder of dealing with a finite set of data is that we can do this without sacrificing any efficiency or speed. When the switch at the top corner of the page is turned on, the data source is set to that div and auto-complete works. When it is off, the data source is set to null and the auto-complete ceases to function. The on/off state is stored in a cookie called "tac" that is available to the whole site.

v. How a Query Works in the System

When you submit a query, it is passed to a function in the PHP backend. The function determines whether you've entered a query via POST (through a text box with a submit button), or via argument (in the address of the page, like when you click on a 'Did You Mean' suggestion). Either way, the queries are handled the same.

First, if the query is empty, nothing is done. The page just returns. If there is a query, it is passed into the Python script to handle the different types of data. This search engine uses word-based unigrams to do everything, as we believed that using bigrams or trigrams would be difficult and not provide much more accuracy due to a relatively small search set. Therefore, the query is split on space characters and each word is parsed separately.

First, the data is sanitized. Everything is made to be lowercase, to match the indices, and stop words are removed completely. This is why words like "computer" and "science" return nothing at all. Basically, you're searching for nothing. The remaining data is then split up into names, numbers and interests. Numbers are separated using a try-catch statement that tries to cast each string as a number. If it is successful, it is added to the number set. If not, it is added to a remaining words set. Names and interests are separated by checking words against the index of valid names. If it is a valid first or last name, it is in the name set. If not, it is in the interests set.

a. Names

Names are handled by basic SQL query. Each name is already deemed a valid name since it was checked against an index. We get results from the database where the first name or last name matches each name it is given. It then returns the teacher IDs of these people. Positive name results are given a higher weight in the result than any other search type because it is the strongest way to identify a person using the types of queries we allow.

b. Class Numbers

Class numbers are also handled by SQL query. We just check whether there is a class name like %x% where class names are of the form "CMPT 316" and x is of the form "316". If so, it returns all the teacher ids that have taught that class. Teachers that have taught it more times are given greater weight in the final query result.

c. Interests

Interests are any other remaining words after removing names and class numbers. These are then stemmed using NLTK's Porter Stemmer, and then checked against an index (a Python dict, in this case) of stemmed words that match words with teacher ids. We then return the list of teacher IDs for each interest word. Each positive result is given a weight of 1 since these are the most vague types of results.

We get an ordered list based on the weight of the results for each of the teacher IDs that fully or partially match our query. Partial matches get a lower weight and full matches get a higher weight. The IDs are then retrieved from the SQL database, and the data is then displayed on the results page.

d. Spelling Mistakes

Every query is run through our edit-distance-based corrector script. This returns the most likely correct spelling of every word in the query. If everything is spelled right, nothing changes and the same query is returned. If something is spelled wrong, it looks through a list of non-stemmed interest words and names, tries to correct it using edit distance, and then queries on the resultant 'fixed query'. If the 'fixed query' returns more results than the actual query, it suggests, "Did you mean to search for X?" to you. If not, it doesn't present you with any such prompt. Unfortunately, this is where the unigram-ness of the system is most felt, since it sometimes returns names that do not go together, such as “Jan Pei”. This is not an actual name, but it is a valid result since there is someone named “Jan” and someone with the last name “Pei” in the system.