Justin Chase
Final Project Proposal
NLP – Ted Pedersen

Problem Description:

Which of these problems are you trying to solve?

I plan to use the large amount of data that has been gathered by Google to develop sets of related words. This project will take a small set of words as user input and will then take advantage of the Google API and the organizational methods of Google's search technology to discover word relationships, which can then be used to create a set of related words.

Describe the problem in general terms and what practical applications the techniques you are developing could have.

In this project the problems will be mainly two-fold: finding potential related words and determining if the relationship of the potential words are strong enough to be considered set worthy. There will be many words found but presumably only certain words will be connected to the given set words in a strong enough relationship to serve our purposes.

In addition to the two main problems there will be a problem to overcome related to the amount and quality of data Google will give us. The quantity of data will be vast and reasonable methods for filtering out unwanted data will be necessary. Many bodies of text found on the web will be very full of words that are too common to accurately be associated with the general subject of the page. Therefore good filtering methods will be needed. Some methods such as removing words found in a list of common words may be implemented. Additionally a part of speech filter may be employed to help select only words that posses a similar part of speech as the given set words.

Overview of Solution/Approach:

What is the general approach you plan on taking?

I will use Google to gather links to web pages relating to the given set of words. Which means that a standard Google query will be performed on each given word and word combination. A weight will then be applied to certain links based on rank and its appearance in multiple queries. Therefore, if a link appears first in the list it will have a heavier weight than links receiving a lower rank. Additionally if a link appears in two separate queries then the weight will be even heavier. After compiling a reasonable amount of links I will search through each of the linked pages and find additional words. This actually means I will connect to the various servers hosting the web pages gathered from the google search queries and download their web page. I will then go through the body of text and create a table of new words then I will filter out common words and words that are the wrong part of speech. Each of the words found would be assigned a weight, which will be based on the weight of the link it was found in and the number of separate links it appeared in.

The main premise is that web pages possessing certain words in their bodies will also posses a certain related subject. and it will use other words to explain this subject further. Thus web pages using similar words may contain some sort of relationship with the subject matter.

Gathering these words and finding that they all appear on multiple pages that seem to relate to a similar subject matter may show a clear relationship between the words.

Algorithms:

*LinkWeight(L) = sum(results – rank[x], 0, results) / (sum(x, 0, results)*number of queries)*

Example: If you were to search for 10 results per query, using 3 keywords you'd have 3 queries (Larry + Moe + Curly & Moe + Curly + Larry & Curly + Larry + Moe) and <= 30 resulting links. So suppose a given link appears in the three queries with the ranks {5, 8, 0} then the algorithm would look like this: (note: a zero indicates it did not appear at all)

((10-5)+(10-8)+(0)) / ((1+2+3+4+5+6+7+8+9+10) * 3) = 7/165 ~= .04242

Words found in the body of this link would be given a weight of .04242. This number represents the overall rank of this link. The higher overall rank of a link the more relevant the words in its body seem to be, therefore the words found in this link are assigned a greater worth.

Note: *sum(weight[x], 0, number of links) = 1*

*TokenValue(W) = sum((frequency[x]/tokens[x])*LinkWeight(x), 0, number of links)*

Example: Given that a word W was found in two links, x1 and x2, which yield 600 and 400 tokens, each having a weight of .18 and .1 respectively. If word W were found 100 times in x1 and 25 times in x2, then the Token Value for word W would be:

((100/600)*.18) + ((25/400)*.1) = 3/100 + 1/160 = .03625

The now weighted words will be filtered then selected from the set based on the highest token values. These selected words will be the set of related words specified by the objective of this project.

Evaluation Plan

How will you show that your solution is valid?

Optimally, the final output of the program will display a set that is intuitively linked by a common element. Additionally I plan on implementing an optional logging feature, which will give verbose information for every step of the process to insure that each step is viable and accurate.

Will you need to find or create "gold standard" data to use as a point of comparison?

I will compare results output from my program with those output from google sets, using identical initial words. The more similar the set words the more successful I will find it to be.

References:

Sergey Brin, Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN systems, 30(1-7): 107-117, 1998.
http://www-db.stanford.edu/pub/papers/google.pdf

Jurafsky, Martin. Speech and Language Processing. Boulder, Colorado: University of Colorado, 2000.

Pedersen, Ted. "Natural Language Processing" Heller Hall 306, U of Minnesota, Duluth. 20 January 2004 - 28 April 2004.