CPSC 231 – Fall 2017
Assignment 5: Semantic Similarity

Due: *8 Dec 2017, 17:00*

1 Overview

One type of question encountered in the Test of English as a Foreign Language (TOEFL) is the "Synonym Question", where students are asked to pick a synonym of a word out of a list of alternatives. For example:

```
 1. vexed                                          (Answer: (a) annoyed)
(a) annoyed
(b) amused
(c) frightened
(d) excited
```

For this assignment, you will build an intelligent system that can learn to answer questions like this one. In order to do that, the system will approximate the *semantic similarity* of any pair of words. The semantic similarity between two words is the measure of the closeness of their meanings. For example, the semantic similarity between "car" and "vehicle" is high, while that between "car" and "flower" is low.

In order to answer the TOEFL question, you will compute the semantic similarity between the word you are given and all the possible answers, and pick the answer with the highest semantic similarity to the given word. More precisely, given a word w and a list of potential synonyms s_1, s_2, s_3, s_4, we compute the similarities of (w, s_1), (w, s_2), (w, s_3), (w, s_4) and choose the word whose similarity to w is the highest.

We will measure the semantic similarity of pairs of words by first computing a *semantic descriptor vector* of each of the words, and then taking the similarity measure to be the *cosine similarity* between the two vectors.

Given a text with n words denoted by $(w_1, w_2, ..., w_n)$ and a word w, let $desc_w$ be the semantic descriptor vector of w computed using the text. $desc_w$ is an n-dimensional vector. The i-th coordinate of $desc_w$ is the number of sentences in which both w and w_i occur. For efficiency's sake, we will store the semantic descriptor vector as a dictionary, not storing the zeros that correspond to words which don't co-occur with w. For example, suppose we are given the following text (the opening of *Notes from the Underground* by Fyodor Dostoyevsky, translated by Constance Garnett):

> I am a sick man. I am a spiteful man. I am an unattractive man. I believe my liver is diseased. However, I know nothing at all about my disease, and do not know for certain what ails me.

The word "man" only appears in the first three sentences. Its semantic descriptor vector would be:

```
{"i": 3, "am": 3, "a": 2, "sick": 1, "spiteful": 1, "an": 1, "unattractive": 1}
```

The word "liver" only occurs in the second sentence, so its semantic descriptor vector is:

```
{"i": 1, "believe": 1, "my": 1, "is": 1, "diseased": 1}
```

We store all words in all-lowercase, since we don't consider, for example, "Man" and "man" to be different words. We do, however, consider, *e.g.*, "believe" and "believes", or "am" and "is" to be different words. We discard all punctuation.

The cosine similarity between two vectors $u = \{u_1, u_2, \ldots, u_N\}$ and $v = \{v_1, v_2, \ldots, v_N\}$ is defined as:

$$\text{sim}(u, v) = \frac{u \cdot v}{||u|| \cdot ||v||} = \frac{\sum_{i=1}^{N} u_i v_i}{\sqrt{\left(\sum_{i=1}^{N} u_i^2\right)\left(\sum_{i=1}^{N} v_i^2\right)}}$$

We cannot apply the formula directly to our semantic descriptors since we do not store the entries which are equal to zero. However, we can still compute the cosine similarity between vectors by only considering the positive entries.

For example, the cosine similarity of “man” and “liver”, given the semantic descriptors above, is

$$\frac{3 \cdot 1 \text{ (for the word "i")}}{\sqrt{(3^2+3^2+2^2+1^2+1^2+1^2+1^2)(1^2+1^2+1^2+1^2+1^2)}} = 3/\sqrt{130} = 0.2631\ldots$$

2 Tasks

All the code should be in the file `synonyms.py`. Note that the names of the functions are case-sensitive and must not be changed. You are not allowed to change the number of input parameters, nor to add any global variables. Doing so will cause your code to fail when run with our testing programs, so that you will not get any marks for functionality. We provide you with a starter version of `synonyms.py`

2.1 Cosine similarity (1 point)

Implement the function `cosine_similarity(vec1, vec2)` This function returns the cosine similarity between the sparse vectors `vec1` and `vec2`, stored as dictionaries. For example,

```
cosine_similarity({"a": 1, "b": 2, "c": 3}, {"b": 4, "c": 5, "d": 6})
```

should return approximately 0.70.

2.2 Semantic descriptors (2.5 points)

Implement the function `build_semantic_descriptors(sentences)` This function takes in a list `sentences` which contains lists of strings (words) representing sentences, and returns a dictionary `d` such that for every word `w` that appears in at least one of the sentences, `d[w]` is itself a dictionary which represents the semantic descriptor of `w` (note: the variable names here are arbitrary). For example, if `sentences` represents the opening of *Notes from the Underground* above:

```
[['i', 'am', 'a', 'sick', 'man'],
 ['i', 'am', 'a', 'spiteful', 'man'],
 ['i', 'am', 'an', 'unattractive', 'man'],
 ['i', 'believe', 'my', 'liver', 'is', 'diseased'],
 ['however', 'i', 'know', 'nothing', 'at', 'all', 'about', 'my',
 'disease', 'and', 'do', 'not', 'know', 'for', 'certain', 'what', 'ails', 'me']],
```

part of the dictionary returned would be:

```
    { 'man': {'i': 3, 'am': 3, 'a': 2, 'sick': 1, 'spiteful': 1, 'an': 1,
              'unattractive': 1},
      'liver': {'i': 1, 'believe': 1, 'my': 1, 'is': 1, 'diseased': 1},
      ... }
```

with as many keys as there are distinct words in the passage.

2.3 Semantic descriptors from files (2.5 points)

Implement the function `build_semantic_descriptors_from_files(filenames)`. This function takes a list of `filenames` which contains the names of files (the first one can be opened using `open(filenames[0], "r", encoding="utf-8")`), and returns the dictionary of the semantic descriptors of all the words in the files `filenames`, with the files treated as a single text.

You should assume that the following punctuation always separates sentences: `"."`, `"!"`, `"?"`, and that is the only punctuation that separates sentences. You should assume that only the following punctuation is present in the texts:

```
[',', '-', '--', ':', ';', '"', "'"]
```

2.4 Find the most similar word (1 point)

Implement the function `most_similar_word(word, choices, semantic_descriptors, similarity_fn)`. This function takes in a string `word`, a list of strings `choices`, and a dictionary `semantic_descriptors` which is built according to the requirements for `build_semantic_descriptors`, and returns the element of `choices` which has the largest semantic similarity to `word`, with the semantic similarity computed using the data in `semantic_descriptors` and the similarity function `similarity_fn`. The similarity function is a function which takes in two sparse vectors stored as dictionaries and returns a `float`. An example of such a function is `cosine_similarity`. If the semantic similarity between two words cannot be computed, it is considered to be -1. In case of a tie between several elements in `choices`, the one with the smallest index in `choices` should be returned (*e.g.*, if there is a tie between `choices[5]` and `choices[7]`, `choices[5]` is returned).

2.5 Run similarity test (0.5 points)

Implement the function `run_similarity_test(filename, semantic_descriptors, similarity_fn)` This function takes in a string `filename` which is the name of a file in the same format as `test.txt`, and returns the percentage (i.e., `float` between `0.0` and `100.0`) of questions on which `most_similar_word()` guesses the answer correctly using the semantic descriptors stored in `semantic_descriptors`, using the similarity function `similariy_fn`.

The format of `test.txt` is as follows. On each line, we are given a word (all-lowercase), the correct answer, and the choices. For example, the line:

```
feline cat dog cat horse
```

represents the question:

```
feline:
(a) dog
(b) cat
(c) horse
```

and indicates that the correct answer is "`cat`".

Download the novels *Swann's Way* by Marcel Proust, and *War and Peace* by Leo Tolstoy from Project Gutenberg, and use them to build a semantic descriptors dictionary. Report how well the program performs (using the cosine similarity similarity function) on the questions in `test.txt`, using those two novels at the same time. Note: the program may take several seconds to run (or more, if the implementation is inefficient). In your report, include the code used to generate the results you report. The novels are available at the following URLs:

`http://www.gutenberg.org/cache/epub/7178/pg7178.txt`

`http://www.gutenberg.org/cache/epub/2600/pg2600.txt`

If your program takes too long to run, report the results on a shorter text.

If your success rate for the two books above and the file `test.tex` is below 66%, then it is likely that your implementation from the previous sections is incorrect. In particular, you may want to revise how you dealt with punctuation in the source files before you built the semantic descriptors.

2.6 Efficiency (2.5 points)

To receive the points for this part, the function `build_semantic_descriptors_from_files(filenames)` should take less than 20 seconds when run on an a desktop computer from the CPSC labs, when the files are the two novels specified in section 2.5.

3 Hand In

1. Electronic copy of your program using D2L. The TAs will run your program to test that it functions correctly.
2. A `readme.txt` file in which you say what functions you implemented and you report the results from previous sections. You should also give details to help the TAs run your programs and reproduce your results.

4 Marking

You are not allowed to use any specialized modules to solve this assignment. Thus, your source file should have no import statements except (possibly) `import time`.

Additional things you may want to consider.

1. If the program produces run-time errors, your grade will be 0.
2. If the program is long and hard to understand, you may receive a lower score, even if the functionality is OK. If the code is incomprehensible, the TA will not grade it (thus, you get 0).
3. Using global variables decreases your score by 3 points.
4. Including more than 30 lines for any function decreases your score by 2 points. If you need, you are welcome to write your own helper function in addition to the functions specified in the assignment.
5. Writing more than 80 characters on any line of your program decreases your score by 2 points.
6. Comments are mandatory at the beginning of the program, at the beginning of each function (say what the parameters are and what the function is doing), and also before difficult blocks of code. Breaking these rules decreases your score by 2 points.
7. The reason for the deductions above is to tell you that you should avoid those things, so that your program can be read by other people. *Please do not force us to actually deduct those points!*

5 Bonuses

You will receive bonus points only for *substantial* improvements in the running time and/or the success probability. For reference, the implementation of the instructor who prepared the assignment for CPSC 231 runs in 8 seconds on the machines in the CPSC labs and achieves an accuracy of 75%.

Here are some suggestions to explore (feel free though to come up with your own suggestions). If you need, you can submit other python files *in addition to* `synonyms.py` (in this second file you are allowed to import various libraries). However, the core algorithm should be implemented by you. Please state in `readme.txt` if you claim any bonus points and the reasons for that. Also, make sure to include in `readme.txt` detailed instructions about how to run the additional programs you submit.

Plese note that the suggestions below may or may not improve the running time or the success rate of the algorithm.

1. Experiment with removing stop words `https://en.wikipedia.org/wiki/Stop_words` before computing the semantic descriptors.
2. (1 point) Prepare a better test file, possibly corresponding to some other books from the Gutenberg project. To receive the bonus, you need to include a script which will download the books automatically, and submit another file similar to `test.txt` (but containing at least 200 test words for which you want to find the synonyms). All the words appearing in the new test file should have at least 5 appearances in the books - you need to submit code which enforces this. The books should contain at least 1 million words - again, you need to submit code to check this. The program needs to run in under a minute on your books on the machines in the CPSC lab.
3. Instead of using sentences, try to compute the similarity of two words by considering neighbourhoods (windows of let's say 7 words around the reference word).

4. For grammatical reasons, documents are going to use different forms of a word, such as *calm, calmed*, and *calming*. These words would be counted as different words in our algorithm, and this affects the accuracy. To avoid this behaviour, you can try to *lemmatize* first the text. You can use for this the Stanford NLP Toolkit (for example).
5. Experiment with `Word2vect` . Instead of finding the "neighbours" of a word based on propositions, embed any word into an n-dimensional vector space based on the whole input text. Then build the semantic descriptors (as described in the assignment) based on the embedding. Can you improve the success rate?

6 Late work

Since the deadline is right at the end of the semester, no late work can be accepted (according to the regulations of the university).

7 Collaboration

Although it can be helpful to you to discuss you program with other people, and that is a reasonable thing to do and a good way to learn, the work you hand in must ultimately be *your own work*. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is *plagiarism* and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example,

   `# the following code is from http://stackexchange.com`.

   Use the complete URL so that the marker can check the source.
2. Citing sources will avoid accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if your work is not the product of your own efforts.
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you find you are exchanging code by electronic means, writing code while sitting and discussing with a fellow student, typing what you see on another person's console, then you can be sure that your code is not substantially your own, and your sources must then be cited to avoid plagiarism. Making your code available to other students (say by email, discussion forum, github) will likely result in an accusation of plagiarism against you.
4. We **will** be looking for plagiarism in your code, possibly using automated software designed for the task. For example, see *Measures of Software Similarity* (MOSS - `https://theory.stanford.edu/~aiken/moss/`).

Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help, than it is to plagiarize. It is recommended that you get familiar with the official regulations of UofC about plagiarism . If you have any questions or you are not sure if a certain action is allowed, please contact your instructor *before* taking that action.

8 Credit

This assignment was originally developed by Michael Guerzhoy (University of Toronto), Jackie Chi Kit Cheung (McGill University), and François Pitt (University of Toronto). You can find the original description here: `http://nifty.stanford.edu/2017/guerzhoy-SAT-synonyms/`.