Semantic similarity

COMP0015 Term 2 Coursework 1 – 40% of the module

This document explains the arrangements for the second and final coursework. You will work in pairs to create an application for calculating the similarity scores for words according to the specification set out in this document.

Deadline

Midday Thursday 26th March 2020.

Overview

Students wishing to study at an English-speaking university are often asked to take a Test of English as a Foreign Language (TOEFL) test. One part of these TOEFL tests involves students picking a synonym for a particular word from a list of options. 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, your program will find the 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 a TOEFL question, it's possible to compute the semantic similarity between a given word and all the possible answers, and to pick the answer with the highest semantic similarity to the given word. Specifically, for a word w and a list of potential synonyms s_1 s_2, s_3, s_4 you can 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.

In this coursework, you will measure the semantic similarity of pairs of words by first computing a semantic descriptor vector for each of the words, and then calculating a similarity measure, known as the cosine similarity, between the two vectors.

Given a text with n words denoted by $(w_1, w_2, \ldots, w_n)$ and a word w, let $desc_w$ be the semantic descriptor vector of w computed using some 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. The semantic descriptor vector for the word "man" is:

```
{"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}
```

The cosine similarity between two vectors $u = \{u_1, u_2, \dots, u_n\}$ and $v = \{v_1, v_2, \dots, v_n\}$ is defined as:

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

It’s not possible to apply the formula directly to the semantic descriptors described above since we do not store the entries which are equal to zero. However, the cosine similarity between vectors can be computed by only using the positive entries. You have been given the code for this calculation, see function named `cosine_similarity` in the starter code.

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

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

For the purposes of this coursework, you will make several assumptions which simplify the amount of programming required, including:

1. All words are stored in all-lowercase, “Man” and “man” are considered to be the same words.
2. The words “believe” and “believes” and the words “am” and “is” are considered to be different words[1].
3. All punctuation should be removed from the text.
4. Your program will only detect the semantic similarity for a word w with potential synonyms s_1 s_2

Other simplifications are described in the relevant sections below.

Your Assignment

You are provided with some starter code in the Python program `synonyms.py` which you must download and save before starting the assignment. You must answer all 3 questions described in this section.

Question 1

Implement the following functions in `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 to the program. Doing so will cause your code to fail when the automated tests used for marking are run.

Part (a) `get_sentence_lists(text)`

This function takes in a string text, and returns a list which contains lists of strings, one list for each sentence (as defined below) in the string text. A list representing a sentence is a list of the individual words, each one in lower case.

[1] In a more sophisticated implementation, the programmer would pre-process the text to account for plurals and different verb conjunctions.

Sentences: sentences are separated by one of the strings "`.`", "`?`" or "`!`". Again, this coursework is a simplistic implementation leading to the situation where the string:

```
"The St. Bernard is a friendly dog!"
```

Will be considered to be two sentences: "`The St`" and "`Bernard is a friendly dog`"

Sentences can span multiple lines of the file so it won't work to process the file one line at a time.

Words: words in the list that represent the sentence must appear in the order in which they appear in the sentence and must not begin or end with punctuation. You can assume that only the following punctuation is present in the text file:

"`,`", "`-`", "`--`", "`:`", "`;`", "`!`", "`?`", "`.`", the single quote and the double quote.

This means, for example, that: "`don't`" is considered to be two words, "`don`" and "`t`". The possessive form "`school's`" is also considered to be two words, "`school`" and "`s`".

For example, if the text file contains the following (and nothing else):

```
Hello, Jack. How is it going? Not bad; pretty good, actually... Very very good,
in fact.
```

then the function should return the following list:

```
[['hello', 'jack'],

['how', 'is', 'it', 'going'],

['not', 'bad', 'pretty', 'good', 'actually'],

['very', 'very', 'good', 'in', 'fact']]
```

Part (b) `get_sentence_lists_from_files(filenames)`

This function takes in a list of strings named `filenames`, each one the name of a text file, and returns the list of every sentence contained in all the text files in `filenames`, in order. The list is in the same format as in Part (a), but with the files rather than a string serving as the source of the text.

Part (c) `build_semantic_descriptors(sentences)`

This function takes in a list named `sentences` which contains lists of strings 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 the list `sentences` represents the opening of *Notes from the Underground* as above, 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.

Part (d) `most_similar_word(word, choices, semantic_descriptors)`

This function has the following parameters: a string `word`, a list of strings `choices`, and a dictionary `semantic_descriptors` which has been built using function `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`. 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).

Part (e) `run_similarity_test(filename, semantic_descriptors)`

This function takes in a string `filename` which is a file in the same format as `test.txt`, and returns the percentage of questions on which `most_similar_word()` guesses the answer correctly using the semantic descriptors stored in `semantic_descriptors`.

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

```
feline cat dog cat horse
```

represents the question:

```
feline:

(a) cat

(b) dog

(c) horse
```

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

Question 2

Add code to test each of the functions in 1a-1d. You are also encouraged to test any helper functions you have written. A helper function performs some sub-task and makes your program easier to understand. You should submit all the files needed to test your functions so that the marking team can run the tests.

Question 3

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. Write a short report describing how well the program performs on the questions in `test.txt`, using those two novels at the same time. Note: the program may take several minutes or more to run. 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

Submitting your assignment

At the submission link on moodle:

1. Make sure your student numbers (not your names) are included in comments at the top of your program.
2. Upload your program.
3. Upload your report.
4. If you have included any additional functionality not specified in the brief, please submit a short document describing what you have done. Keep this short, it is not graded. Upload all additional materials.

Assessment

You are expected to show that you can code competently using the programming concepts covered so far in the course.

Marking criteria will include:

- Correctness – your code should perform as specified
- Programming style – your variable names should be meaningful and your code as simple and clear as possible. See section Style Guide further on for more detail.
- Whether you used and wrote more helper functions if needed - do not needlessly repeat code and if your function is more than about 20 lines long, consider decomposing it even if the new function will only be called once.
- Your assignment will be marked using the rubric at the end of this document. This is the standard rubric used in the Department of Computer Science. Both members of the team will receive the same mark unless the lecturer has been notified of issues and determines otherwise. Marks for your project work will be awarded for the capabilities (i.e. functional requirements) your system achieves, and the quality of the code.

Additional Challenges

- Additional marks may also be gained by taking on extra challenges but you should only attempt an additional challenge if you have satisfied all requirements for the coursework.
- You could write some automated tests for your functions using pytest or, more simply by using assert statements.

Plagiarism

Plagiarism will not be tolerated. Your code will be checked using a plagiarism detection tool.

Style Guide

You must adhere to the style guidelines in this section.

Formatting Style

1. Use Python style conventions for your function and variable names (pothole case: lowercase letters with words separated by underscores (_) to improve readability).

2. Choose good names for your functions and variables. For example, `num_bright_spots` is more helpful and readable than `nbs`.
3. Use a tab width of 4 or 8. The best way to make sure your program will be formatted correctly is never to mix spaces and tabs -- use only tabs, or only spaces.
4. Put a blank space before and after every operator. For example, the first line below is good but the second line is not:

   ```
   b = 3 > x and 4 - 5 < 32
   b= 3>x and 4-5<32
   ```

5. If you add functions, write a docstring comment for each function. (See below for guidelines on the content of your docstrings.) Put a blank line after every docstring comment.
6. Each line must be less than **80 characters** long *including tabs and spaces*. You should break up long lines using \.

Docstrings

If you add your own functions you should comment them using docstrings. Take a look at the code you've been given for some examples. Your comments should:

1. Describe precisely *what* the function does.
2. Do not reveal *how* the function does it.
3. Make the purpose of every parameter clear.
4. Refer to every parameter by name.
5. Be clear about whether the function returns a value, and if so, what.
6. Explain any conditions that the function assumes are true. Examples: "n is an int", "n != 0", "the height and width of p are both even."
7. Be concise and grammatically correct.
8. Write the docstring as a command (e.g., "Return the first ...") rather than a statement (e.g., "Returns the first ...")

UCL Computer Science: Marking Criteria and Grade Descriptors

UCL

	Fail		Pass (2:2)	Merit (2:1)	Distinction (1st)		
1-19: Misunderstanding of assignment or similar 20-29: 5 inadequate 30-34: 4 inadequate 34-39: 3 inadequate	Inadequate	Weak	Satisfactory	Good	Excellent	Outstanding	Exceptional
	Below 40: BSc: Fail MEng: Fail	40-49: BSc: 3rd MEng: Fail	50-54: Low pass 55-59: High pass	60-64: Low merit 65-69: High merit	70-79	80-89	90+
1 **Quality of the response to the task set: answer, structure and conclusions**	Either no argument or argument presented is inappropriate and irrelevant. Conclusions absent or irrelevant.	An indirect response to the task set, towards a relevant argument and conclusions.	A reasonable response with a limited sense of argument and partial conclusions.	A sound response with a reasonable argument and straightforward conclusions, logical conclusions.	A distinctive response that develops a clear argument and sensible conclusions, with evidence of nuance.	↔	Exceptional response with a convincing, sophisticated argument with precise conclusions.
2 **Understanding of relevant issues**	Misunderstanding of the issues under discussion.	Rudimentary, intermittent grasp of issues with confusions.	Reasonable grasp of the issues and their broader implications.	Sound understanding of issues, with insights into broader implications.	Thorough grasp of issues; some sophisticated insights.	↔	Exceptional grasp of complexities and significance of issues.
3 **Engagement with related work, literature and earlier solutions**	Very limited or irrelevant reading.	Significant omissions in reading with weak understanding of literature consulted.	Evidence of relevant reading and some understanding of literature consulted.	Evidence of plentiful relevant reading and sound understanding of literature consulted.	Extensive reading and thorough understanding of literature consulted. Excellent critical analysis of literature.	↔	Expert-level review and innovative synthesis (to a standard of academic publications).
4 **Analysis: reflection, discussion, limitations**	Erroneous analysis. Misunderstanding of the basic core of the taught materials. No conceptual material.	Analysis relying on the partial reproduction of ideas from taught materials. Some concepts absent or wrongly used.	Reasonable reproduction of ideas from taught materials. Rudimentary definition and use of concepts.	Evidence of student's own analysis. Concepts defined and used systematically/ effectively.	Evidence of innovative analysis. Concepts deftly defined and used with some sense of theoretical context.	↔	Exceptional thought and awareness of relevant issues. Sophisticated sense of conceptual framework in context.
5 **Algorithms and/or technical solution**	No solution to the given problem, completely incorrect code for the given task.	Rudimentary algorithmic/technical solution, but mostly incomplete.	Reasonable solution, using basic required concepts, several flaws in implementation.	Good solution, skilled use of concepts, mostly correct and only minor faults.	Excellent algorithmic solution, novel and creative approach.	↔	Exceptional solution and advanced algorithm/technical design.
6 **Testing of solution (e.g., correctness, performance, evaluation)**	No testing or evaluation done.	Few test cases and/or evaluation, but weak execution.	Basic testing done, but important test cases or parts of evaluation missing or incomplete.	Solid testing or evaluation of solution, well done evaluation with good summary of findings.	Very well done test cases, excellent evaluation and very high quality summary of findings.	↔	Exceptionally comprehensive testing, extremely thorough approach to testing and/or evaluation.
7 **Oral presentation or demonstration of solution**	Poorly done presentation or demonstration, very low quality.	Ineffective oral presentation or demo of the solution.	Able to communicate, present and/or demonstrate solution and summarise work in appropriate format.	Overall good presentation or demo, persuasive and compelling.	Very high quality of delivery. Use of presentation medium with professional style.	↔	Flawless and polished presentation, exceptional quality of demonstration.
8 **Writing, communication and documentation**	Style and word choice seriously interfere with comprehension.	Style and word choice seriously detract from conveying of ideas.	Style and word choice sometimes detract from conveying of ideas.	Style and word choice work well to convey most important ideas. Well documented.	Style and word choice show fluency with ideas and excellent communication skills.	↔	Reads as if professionally copy edited. Exceptional high quality of writing.
9 **Formatting aspects, visuals, clarity, references**	Poorly formatted, inappropriate visuals, and incorrect reference formatting.	Formatting, visuals and referencing seriously distract from argument.	Formatting, visuals and referencing sometimes distract from argument.	Formatting well-done and consistent, good visuals and consistent referencing.	Formatting, visuals and referencing are impeccable.	↔	Exceptional presentation, impeccable formatting of the document and references.