

CS 101 - Synonym Identification

Spring 2018

Algorithm Due : **April 8th, 2018**

Program Due : **April 15th, 2018**

All work submitted must be your own.

Deliverables : You only need to submit your solution. You must use functions to modularize your work. You should use exception handling where necessary as well. Dictionaries, sets, and lists will come in helpful here.

Synonyms

One type of question encountered in many tests 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
 - a. annoyed
 - b. amused
 - c. frightened
 - d. excited

The correct answer for this question is annoyed. We’re going to build a program that can read novels and find the similarity between words in order to find synonyms.

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 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, \dots, 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 $u = \{u_1, u_2, \dots, u_N\}$ and $v = \{v_1, v_2, \dots, v_N\}$ is defined as:

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

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. See the starter code for how this can be accomplished.

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

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

NOTE : The test data that is used by the graders may be different than the ones you are given. Do not hard code any of the information in this file.

Using the synonyms.py file you can begin to work on the functions that are provided. They are given as they should be implemented with parameters and return values that you should use.

There is a test file provided as well that can help debug and test it. The following are descriptions of what the functions should do. Once you get those working you should find the program to be fairly easy to construct.

get_sentence_lists(text : str) -> list:

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 in the sentence, each one in all-lowercase.

For our purposes, sentences are separated by one of the strings ".", "?", or "!". We ignore the possibility of other punctuation separating sentences. We also assume that every period separates sentences. For example, the string "The St. Bernard is a friendly dog!" is considered to be two sentences: "The St" and "Bernard is a friendly dog".

The words in the list that represents the sentence must be in the order in which they appear in the sentence, and must not begin or end with punctuation.

You should assume that only the following punctuation signs are present in the text file: ",", "-", "--", ":", ";", "!", "?", ".", the single quote, or the double quote. That the single quote is considered punctuation means that, for example, "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']]

get_sentence_lists_from_files(filenamees : list) -> list:

This function takes in a list of strings filenamees, each one the name of a text file, and returns the list of every sentence contained in all the text files in filenamees, 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..

build_semantic_descriptors(sentences : list) -> dict:

This function takes in a list 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 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.

most_similar_word(word : str, choices : list, semantic_descriptors : dict) -> str:

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. 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).

run_similarity_test(filename : str, semantic_descriptors : dict) -> float:

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, we are given a word (all-lowercase), the correct answer, and the choices. The second word is the correct choice. 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”.

Sample Program

```
>>> ===== RESTART =====
>>>
#1 Word : draw, Predicted : paint, Correct : paint
#2 Word : duty, Predicted : task, Correct : task
#3 Word : dour, Predicted : serious, Correct : serious
#4 Word : murder, Predicted : kill, Correct : kill
#5 Word : picture, Predicted : chair, Correct : painting
#6 Word : vexed, Predicted : amused, Correct : annoyed
```

```

#7 Word : watch, Predicted : see, Correct : see
#8 Word : tidy, Predicted : clean, Correct : clean
#9 Word : juvenile, Predicted : young, Correct : young
#10 Word : plausible, Predicted : believable, Correct : believable
#11 Word : strike, Predicted : beat, Correct : beat
#12 Word : tearful, Predicted : crying, Correct : crying
#13 Word : lonely, Predicted : alone, Correct : alone
#14 Word : ardent, Predicted : wise, Correct : keen
#15 Word : burglar, Predicted : robber, Correct : robber
#16 Word : authentic, Predicted : false, Correct : genuine
#17 Word : trip, Predicted : journey, Correct : journey
#18 Word : stroll, Predicted : walk, Correct : walk
#19 Word : speak, Predicted : talk, Correct : talk
#20 Word : begin, Predicted : sit, Correct : start
#21 Word : voyage, Predicted : dog, Correct : journey
#22 Word : stone, Predicted : chair, Correct : rock
#23 Word : revile, Predicted : regard, Correct : hate
#24 Word : genuine, Predicted : interesting, Correct : real
#25 Word : bring, Predicted : fetch, Correct : fetch
#26 Word : shout, Predicted : smell, Correct : yell
#27 Word : ruin, Predicted : destroy, Correct : destroy
#28 Word : sanguine, Predicted : sad, Correct : optimistic
#29 Word : leap, Predicted : sit, Correct : jump
#30 Word : evade, Predicted : avoid, Correct : avoid
#31 Word : infringe, Predicted : walk, Correct : violate
#32 Word : charge, Predicted : admission, Correct : accusation
#33 Word : ruddy, Predicted : wrinkled, Correct : wrinkled
#34 Word : threat, Predicted : danger, Correct : danger
#35 Word : error, Predicted : mistake, Correct : mistake
#36 Word : toil, Predicted : work, Correct : work
#37 Word : serene, Predicted : quiet, Correct : quiet
#38 Word : study, Predicted : examine, Correct : examine
#39 Word : road, Predicted : path, Correct : path
#40 Word : pitiless, Predicted : cruel, Correct : cruel

```

The program predicted 65.000% of the synonyms correctly

Specification

-

Point Breakdown - May be modified as needed

Points	Requirement
5	Header
20	Readability, variable naming, comments, structure
5	Data Structures
5	Proper functions
10	Unit Tests and functionality for get_sentence_lists works correctly

10	Unit Tests and functionality for <code>get_sentence_lists_from_file</code> works correctly
10	Unit Tests and functionality for <code>build_semantic_descriptors</code> works correctly
10	Functionality for <code>most_similar_word</code> works correctly
10	Functionality for <code>run_similarity_test</code> works correctly
15	Program reads all the sample files given, and builds the descriptors and then evaluates the tests

30 points off for programs that crash on expected input.