CSCI 544: Applied Natural Language Processing

Assignment 5: Vector Similarity

Out: **12. October 2018**
Due: **26. October 2018**

Edit of 2018-10-11 00:23:59Z

In this assignment you will implement comparison functions for word representations and use the functions and representations to solve semantic similarity tasks.

Cosine Similarity

Recall that, for vectors of context counts x and y of length n, where i indexes over the context types and x_i is the count of context i, cosine similarity is defined as follows:

$$\mathrm{cossim}(x, y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

A good way to understand the cosine similarity function is that the numerator cares about whether the x and y vectors are correlated. If x and y tend to have high values for the same contexts, the numerator tends to be big. The denominator can be thought of as a normalization factor: if all the values of x are really large, for example, dividing by the square root of their sum-of-squares prevents the whole thing from getting arbitrarily large. In fact, dividing by both these things (aka their norms) means the whole thing can't go higher than 1.

Question 1 (10 pts)

See the file `nytcounts.university_cat_dog`, which contains context count vectors for three words: dog, cat, and university. These are immediate left and right contexts from a New York Times corpus. You can open the file in a text editor since it's quite small. Please complete `cossim_sparse(v1,v2)` in `distsim.py` to compute and display the cosine similarities between each pair of these words. You may not use any functions from packages (e.g. scikit) that contain implementations of cosine calculation. Briefly comment on whether the relative simlarities make sense. Note that we've provided `testq1.py`, some very simple code that tests the context count vectors from the data file. When you upload your completed `distsim.py` to Vocareum (after Q6) we will run that code on vocareum to ensure your implementation is correct.

Question 2 (15 points)

Implement `show_nearest()`. Given a dictionary of word-context vectors, the context vector of a particular query word w, the words you want to exclude in the responses (It should be the query word w in this question), and the similarity metric you want to use (it should be the `cossim_sparse` function you just implemented), `show_nearest()` finds the 10 words most similar to w. Display the similar words, and their similarity scores against the query word w. To make sure your function is working, feel free to use the small `nytcounts.university_cat_dog` database via the provided `testq2.py` program.

Question 3 (20 points)

Explore similarities in `nytcounts.4k`, which contains context counts for about 4000 words in a sample of New York Times. The news data was lowercased and URLs were removed. The context counts are for the 2000 most common words in twitter, as well as the most common 2000 words in the New York Times. (But all context counts are from New York Times.) The context counts only contain contexts that appeared for more than one word. The file `vocab` contains the list of all terms in this data, along with their total frequency. Choose at least six words. For each, show the output of `show_nearest()` and comment on whether the output makes sense. Comment on whether this approach to distributional similarity makes more or less sense for certain terms. Four of your words should be:

- a name (for example: person, organization, or location)
- a common noun
- an adjective
- a verb

You may also want to try exploring further words that are returned from a most-similar list from one of these. You can think of this as traversing the similarity graph among words. Complete the code used to generate answers for this question in `q3.py` and upload it to Vocareum for documentation purposes; provide written responses on CrowdMark below.

Note: You don't need this, but for reference, the preprocessing scripts that were used to create the context data are in the `preproc/` directory (the data itself is not available).

Question 4 (10 points)

Now let's examine similarities in trained word embeddings, instead of raw context counts. See the file `nyt_word2vec.university_cat_dog`, which contains word embedding vectors pretrained by word2vec[1] for three words: "dog","cat", and "university". You can open the file in a text editor since it's quite small.

Please complete `cossim_dense(v1,v2)` in `distsim.py` to compute and display the cosine similarities between each pair of these words. You may not use any functions from packages (e.g. scikit) that contain implementations of cosine calculation. You can test that your implementation is correct with `testq4.py`. We will also test this on Vocareum.

Implementation note: Notice that the inputs of `cossim_dense(v1,v2)` are numpy arrays. If you are not very familiar with basic operations in numpy, you can find some examples in the basic operation section here: `https://docs.scipy.org/doc/numpy-dev/user/quickstart.html`

If you know how to use Matlab but haven't tried numpy before, the following link should be helpful: `https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html`

Question 5 (20 points)

Repeat the investigation you conducted in question 3, but now use dense vectors from word2vec. Comment on whether the outputs make sense. Compare the outputs of using `show_nearest()` on word2vec and the outputs on the sparse context vector (we suggest you use the same words you used in question 3). Which method works better on the query words you choose? Please briefly explain why one method works better than other in each case.

Note that we use the default parameters of word2vec in gensim to get word2vec word embeddings.

Complete the code used to generate answers for this question in `q5.py` and upload it to Vocareum for documentation purposes; provide written responses on CrowdMark below.

[1] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013.

Question 6 (15 points)

An interesting thing you can do with word vectors is perform analogical reasoning tasks. In the following example, we provide the code (in `testq6.py`) which can find the closest words to the vector $v_{\text{king}} - v_{\text{man}} + v_{\text{woman}}$ (other than those words) to obtain a solution for this analogy task:

```
king : man :: __ : woman
```

(Note: the above is a shorthand for the natural language question "King is to man as what is to woman?")

Notice that word2vec is trained in an unsupervised manner. Impressively, it does appear to be able to "reason." (For a contrary opinion, see (Linzen 2016)).

If your implementation so far has been correct, you shouldn't need to modify any code. Simply run `testq6.py` to confirm your answer is correct (no written component is necessary). Inspect the code, though, as you'll need to understand how it is run to complete the next question.

Question 7 (25 points)

Now use the file `word-test.v3.txt` to try your word2vec vectors out on eight different analogy tasks. The groups of relations are delimited by lines starting with a colon (:) and you should see these groups: capital, currency, city-in-state, family, adjective-to-adverb, comparative, superlative, and nationality-adjective. Write a program called `q7.py` that does the following:

For each of the eight relation groups, print the 1-best, 5-best, and 10-best accuracy of your vectors on the group. The n-best accuracy is the percentage of items for which the correct answer was in the top n vectors returned. Use the approach from the sample code in Question 6 to complete the task. That is, given w_1, w_2, w_3, w_4, calculate $w_1 - w_2 + w_4$, obtain the closest words (other than w_1, w_2, and w_4, and compare them to w_3.

On CrowdMark, in the space below, display the table generated by your `q7.py` program. Additionally, for each relation group, show an example of an incorrectly predicted analogy item, along with the correct answer. Are there certain kinds of relations that seem to be predicted more accurately or less accurately by this method? Discuss your opinions and give a rationale for them.

Here's what the table could look like (anything equivalent and readable will do): This shows that 1-best, 5-best, and 10-best accuracy for the task 'capital' is 8.3%, 58%, and 75%, respectively, and accuracy for 1-best, 5-best, and 10-best accuracy for 'currency' is 10%.

capital:	0.083	0.58	0.75
currency:	0.1	0.1	0.1
city-in-state:	...	...	...
family:	...	...	...
adjective-to-adverb:	...	...	...
comparative:	...	...	...
superlative:	...	...	...
nationality-adjective:	...	...	...

Upload your `q7.py` and any other necessary code to Vocareum for documentation purposes.

Question 8 (20 points)

Design two new relation groups and come up with three questions for each group (six total). All words for all questions should appear in your word2vec vocabulary. On CrowdMark, below, report how well the two sets of embeddings perform on your test questions. You're encouraged to be adversarial so that the embeddings for your relation groups might get an accuracy of zero! Discuss any interesting observations you have made in the process.

Extra Credit (up to 20 points)

Try to improve your results on the analogy tasks in `word-test.v3.txt` and on the tasks you created in Q8. You can do this by training your own vectors, downloading different pretrained vectors, using a different similarity metric, or using an entirely different approach to the task (e.g. use WordNet). If you want to use pretrained vectors, some good families are (the following are hyperlinked):

- Word2vec
- GloVe
- Dependency-based embeddings

Report on what you tried and what the effects were. Points will be awarded for creativity and insight.

Notes:

- Be precise with what you report. Embeddings vary not only in the method of construction from data, but also in dimensionality, amount and type of text used, and more.
- Some embeddings may require you to use other software in order to determine similarity. You can use external code for **this part of the assignment only** but you must document your methods and cite your sources.
- If you use a large data set you may see slower performance than in the rest of the assignment. You should not attempt to use very large embeddings on Vocareum.

On CrowdMark, in the space below, describe what you tried and produce a table that clearly shows the impact (whether helpful or not) on the tasks.

What to turn in

- **On CrowdMark:** Your answers to questions 1–3, 5, 7–8, and the extra credit (optional).
- **On Vocareum:** Your completed `distsim.py`, containing correct implementations of the two cosine similarity functions and `show_nearest`. Your completed `q3.py`, `q5.py`, and `q7.py`. Please do not upload any large embeddings you may have downloaded.