Homework 6:
Word Similarity

Dr. Benjamin Roth
Computerlinguistische Anwendungen

Due: Friday, May 25, 2018, 16:00

Important:
These exercises are based on your work in the previous homework.
You can either use your own code from the previous homework in all places marked with `"# TODO insert code here"` in the file `cooocurrence.py` or wait until the deadline of homework 5 is over (May 18, 16:00) and use this code from the website.

In this homework, you will complete an interactive word similarity query program by using the PPMI weighted co-occurrence matrix to find similar words of a query word based on cosine similarity. In the end, you can compare the results to a singular value decomposition (SVD) version.

You can check your progress using the `unittest`:

```
python3 -m unittest -v hw06_word_similarity/test_word_similarity.py
```

Exercise 1: Saving the most frequent words into Vocabulary [4 points]

Complete the function `vocabulary_from_wordlist(word_list, vocab_size)` in the file `cooccurrence.py`. Given a list of words (*word_ list*) and a number (*vocab_ size*), a **set** that contains only the *vocab_ size* most frequent words from *word_ list* should be returned.

You can check your progress using the `doctest` or `unittest`:

```
python3 -m doctest -v hw06_word_similarity/coooccurrence.py
python3 -m unittest -v hw06_word_similarity/test_word_similarity.py
```

Tipp: Watch out for unused imports, which could be part of a possible solution.

Exercise 2: Completing the class PpmiWeightedSparseMatrix

Familiarize yourself with the class `DenseSimilarityMatrix` in the file `word_similarity.py`. We will now complete a similar class for sparse matrices, and later compute SVD on a sparse matrix.

Exercise 2.1: Completing the constructor [4 points]

The constructor of the class `PpmiWeightedSparseMatrix` takes three arguments:

- *word_ list* (A list of words representing a text),
- *vocab_ size* (Used to define the size of the vocabulary),
- *window_ size* (Used to define co-occurence window size)

and comprises the following steps:

- Use the arguments *word_ list* and *vocab_ size* to create a vocabulary
- Use *word_ list*, *window_ size* and the *vocabulary* to create the co-ooccurrences
- Use the *co-ooccurrences* and the *vocabulary* to get the sparse matrix and the word-to-column mapping, also derive the column-to-word mapping
- Apply PPMI weighting to the created matrix

Tipps: Have a look at the methods of the class `PpmiWeightedSparseMatrix`. Stick to the naming of the class attributes (signalized by the *self* keyword) you will find there and use the functions from the file `cooccurrence.py`

Exercise 2.2: Using Singular Value Decomposition [4 points]

Complete the method `toSvdSimilarityMatrix(n_components)` and return a `DenseSimilarityMatrix` that contains the truncated $U\Sigma$ matrix (the result of transforming with `sklearn.decomposition.TruncatedSVD`)

Exercise 2.3: Efficient Cosine Similarity Computation [4 points]

Complete the method `PpmiWeightedSparseMatrix.most_similar_words(word,n)` to return the most `n` similiar words for a given query word.
Complete the method `PpmiWeightedSparseMatrix.similarities_of_word(word)` to compute cosine similarities for all words. In contrast to the equivalent method in `DenseSimilarityMatrix`, you need to deal with sparse matrices here.

Note:

- A vector in Scipy is always stored as matrix ($1 \times d$ or $d \times 1$).

- For a $d \times d$ matrix `m`, and a $d \times 1$ column vector `v`, the result of `m.dot(v)` is again $d \times 1$.
- Element-wise multiplication in Scipy: `m1.multiply(m2)`
- Use sparse matrix multiplication for all multiplications involving the `word_matrix`. (You can transform afterwards to a Numpy vector using `.todense().A1`)
- Summing a matrix along an axis: `v = m.sum(axis=1)`; Note that the result is a dense matrix `v` of size $d \times 1$, which you can transform into a numpy vector using `v.A1`
- The sparse dot product `dAB=vA.dot(vB)` between a row vector `vA` and a column vector `vB` is a 1×1 matrix. Use `dAB[0,0]` to get the corresponding float value.

Exercise 3: Running the interactive application

If all unittests passed you can run the code on the nltk Brown corpus by calling:

```
python3 -m hw06_word_similarity.interactive_similarity
```

Wait for the Brown corpus to be loaded and then enter a word of your choice. You can compare the output of the similiarty computation with and without SVD.