CS159 - Assignment 5
Due: Friday, April 1 at 6pm

For our last assignment (yay!) we will be exploring different approaches for calculating word similarities based on distributional similarity. For example, given the word `dog` we might come up with the following similar words:

```
terrier
inu
breed
dogs
spitz
```

You may work with a partner on this assignment (I encourage you to try and work with a different partner) and you may use whatever programming language you like.

1 Input

Like the last assignment, for this assignment you will be creating a script called `wordsim.sh` that runs the word similarity calculator. This script should take three arguments as parameters:

```
wordsim.sh <stoplist> <sentences> <inputfile>
```

- `<stoplist>`: a list of stop words, one per line, that should be ignored from the input.
- `<sentences>`: a list of sentences/text fragments, one per line, to be used for training the distributional similarity method.
- `<inputfile>`: a file consisting of lines of the following form:

  ```
  <word>    <weighting>    <sim_measure>
  ```

  where `<weighting>` is one of:

  - `TF`: term frequency - use the number of times each word occurs in the word context.

- `TFIDF`: inverse document frequency - use the term frequency times the inverse document frequency. When calculating the IDF, treat each line as a separate document.
- `PMI`: pointwise mutual information - use the pointwise mutual information weighting between the word and the feature. Calculate $p(w, f) = count(w, f)/count(all\_words)$, $p(w) = count(w)/count(all\_words)$ and $p(f) = count(f)/count(all\_words)$, where $count(all\_words)$ is the total number of word occurrences in the corpus, $count(f)$ and $count(w)$ are both over the whole corpus (i.e. not only in word contexts) and $count(w, f)$ is the number of times word f was seen in the context of word w, counting duplicates (i.e. if f occurred twice in one context, you'd count it twice).

and `<sim_measure>` is one of:

- `L1`: L1 distance, normalized by the L2 (Euclidean) length of the vectors.
- `EUCLIDEAN`: Euclidean distance, normalized by the L2 (Euclidean) length of the vectors.
- `COSINE`: Cosine distance, normalized by the L2 (Euclidean) length of the vectors.

2 Output

For any run of the program, the first thing you should output are some statistics: the number of unique words (after stoplist, etc. removal), the number of word occurrences (again, after preprocessing) and the number of sentences/lines.

For each line in the `<input_file>` you will calculate the 10 most similar words based on the weighting and similarity measure specified. Output these in order, one per line with the similarity score, tab separated. For example:

```
SIM: dog TF COSINE
terrier   0.6757599580387642
now   0.6732906004244482
kind  0.6564829111760163
called    0.6561324054538721
considered   0.6526006142705167
today 0.6505379660187838
type  0.6481430531459309
possible  0.6470210473696648
used  0.646499660989564
also  0.6403788033560914
```

3 Implementation details

- Split words based on whitespace.
- Lowercase all words

- Before calculating any statistics, contexts, etc. remove all stop words.
- Before calculating any statistics remove all words that are not exclusively letters (i.e. no numbers, punctuation, etc.).
- We will use a context of 2 words on each side of a word. If a word occurs multiple times in a context, count it multiple times. If you are at a line boundary (beginning or end) only count to the boundary. For example, if we had the input sentence:

  ```
  I like to eat bananas too .
  ```

  and `to` was in our stoplist, then we would preprocess and get:

  ```
  i like eat bananas too
  ```

  and if we then wanted the context for `bananas` we would get {`like eat too`}.
- For `PMI` the log should be base 10.
- When looking for the 10 most similar words, only consider words that have occurred 3 times or more.
- Make sure to follow the input and output specifications exactly. Look at the test input and output examples in the data directory and make sure yours match formatting, etc.
- My version takes less than a minute to load the data and calculate the test example. As with all of the assignments, do think about efficient data structures, etc. when you're thinking about the implementation.

4 Data

In `/common/cs/cs159/assignments/assignment5/data/` I have provided you with some data to get you started:

- `stoplist`: a stoplist
- `sentences`: the data
- `test`: an example input file
- `test.out`: the output for the test input file

Because of rounding errors, etc. your numbers may be just slightly different, but your formatting and list should be the same.

5 Writeup

Once you have everything working, I want you to play with some different examples to see how well each of the approaches work. Include in your writeup the output from your system on two different words over a range of weightings and similarity measures (ideally the same between the two examples). Provide a short (i.e. a paragraph or two) analyzing your results and explaining them. Include this as a file called either `writeup.txt` or `writeup.pdf`.

6 Extra credit

For extra credit you may implement additional weighting schemes and/or additional similarity measures. If you do this, include examination of these in your writeup and also include the extra parameters in the README. Points will be awarded based on the difficulty of the approach as well as its effectiveness.

7 When you're done

When you're all done, follow the directions on the course web page for submitting your work in the dropbox. Make sure that your code compiles, that your files are named as specified. If you get an error when submitting, try changing the name of the folder to include a version number and resubmit.

Include all of your code and your writeup. Your script `wordsim.sh` should be in the base directory and should work from there.

If you worked with a partner, put both people's last names on the submitted directory, but only submit one copy.

If the naming of the files isn't obvious about where various things are being done, please include a `README` file explaining your organization.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file you modify.
- Each class and method should have appropriate JavaDoc comments (doc strings if you do it in Python).
- If anything is complicated, put a short note in there to help me out if there are any issues.

This is a non-trivial assignment and it can get complicated, which makes code style and comments very important so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.

Grading

Part	points
General	20
TF	5
TFIDF	5
PMI	5
L1	5
Euclidean	5
Cosine	5
Efficiency	5
style/commenting	5
extra credit	5
total	60 + 5 extra