CS159 - Assignment 5

Part A: Friday, March 10 @ 5pm

Part B: Monday, March 27 @ 1pm

I FOUND THIS ARTICLE ON THE LINGUISTICS OF THE "QUOTATIVE LIKE."
LIKE WHEN YOU'RE LIKE, "SHE WAS LIKE"?
YEAH.
IT FEATURES A QUOTE FROM A LINGUIST, PATRICIA CUKOR-AVILA:
"EVENTUALLY ALL THE PEOPLE WHO HATE THIS KIND OF THING ARE GOING TO BE DEAD, AND THE ONES WHO USE IT ARE GOING TO BE IN CONTROL."
WOW. TURNS OUT LINGUISTS ARE PRETTY HARDCORE.
I *THINK* SHE MEANS DEAD FROM OLD AGE.
I'M GONNA START USING "LIKE" MORE, JUST IN CASE.

`https://xkcd.com/1483/`

In this assignment 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
breed
dogs
inu
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.

Part A: Warm-up

To get started, we're going to do a similarity calculation "manually" to make sure that you understand the concepts and to give you some test data to check your answers against. Submit your answers to the questions below as assignment 5a (well, really just question #2).

1. Read through the rest of this handout and make sure you understand the problem description.

2. Calculate the similarity between the “words” `b` and `d` with `TF-IDF` weighting using `EUCLIDEAN` distance for the data below. Assume each line is a sentence and each letter is a word.

   ```
   a a b c c
   a a d c c
   a a b c c
   e e d g g
   e e f g g
   h h h h h
   ```

   Follow the specifications outlined in Part B below including length normalization, etc. where appropriate. Along with your answer (the similarity), turn in intermediate calculations that will help me verify correctness, in particular:

   (a) the original co-occurrence vectors for `b` and `d` before processing

   (b) the IDFs for each of the “words”

   (c) the length normalized TF-IDF co-occurrence vectors for `b` and `d`

   (d) and the final distance between words `b` and `d` using `TF-IDF` weighting and `EUCLIDEAN` distance.

   You should do this part “manually”, which doesn’t necessarily mean on paper (e.g. you can use Excel, python or some other tool to help with the calculations), but you should not be trying to write the whole program and should instead be doing the individual calculations.

Part B: The program

Input

Like the last assignment, for this assignment you will be creating a script called `wordsim.sh` that outputs the most similar words to a set of input words based on a some of the different weighting schemes and similarity measures we’ve looked at in class. 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>
  ```

(tab separated), where `<weighting>` is one of:

- `TF`: term frequency - use the number of times each word occurs in the word context.
- `TFIDF`: term frequency with inverse document frequency weighting – use the term frequency times the inverse document frequency. Calculate IDF using the `<sentences>` file *treating each line as a separate document.*

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.

Output

For any run of the program, the first thing you should output are some statistics:

1. the number of unique words (after stoplist removal, lowercasing, etc.)
2. the number of word occurrences (again, after preprocessing)
3. the number of sentences/lines

Just print the counts by themselves, one per line, so your output should alway be prepended by three lines of numbers.

Then, for each line in the `<input_file>` you will calculate the 10 most similar words based on the weighting and similarity measure specified (*or less, if there are less than 10 words in the data set–make sure to check for this!*). Output these in order based on the most similar, one per line with the similarity score, tab separated.

For example, if "dog TF COSINE" were in the input file, then you would have an entry in your output:

```
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
```

To make the output explicit, I've included an example at the end of this handout in the Appendix. Please do take a look at this as you develop your program to make sure that you're following the formatting correctly.

Implementation details

- Split words based on whitespace (i.e. don't do any other tokenization).
- 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 the words {`like eat too`}.
- When looking for the 10 most similar words, only consider words that have occurred 3 or more times.
- `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 you 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 when you're thinking about the implementation.

Hints

- You've walked through one example by hand, but I'd encourage you to look at some other ones manually as well. For example, calculate the results for the other measures on the corpus from Part A.
- Be careful about converting between `int`s and `double`s and doing division with `int`s.
- For each of the similarity/distance measures make sure you think about whether smaller or larger means "more similar".
- There is a strong relationship between `EUCLIDEAN` and `COSINE`.

Data

I have provided you with some data to get you started at:

`http://www.cs.pomona.edu/classes/cs159/assignments/assign5-starter.zip`

which includes the following files in the `data` directory:

- `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 lists should be the same.

Writeup

Once you have everything working, play with some different examples to see how well each of the approaches work. Include a short writeup with the output from your system on two or more different words over a range of weightings and similarity measures. Your experiments should highlight strengths and weaknesses in the different approaches and allow you to provide some concrete analysis of the approaches.

In addition to the results, provide a short (i.e. a paragraph or two) analysis of your results. Include all of this as a file called either `writeup.txt` or `writeup.pdf`.

Your writeup will be graded based on the quality of your analysis as well as the quality of your chosen examples.

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.

One additional weighting scheme we talked about in class was PMI. If you implement it, use the following specification:

- `PMI`: pointwise mutual information – use the pointwise mutual information weighting between the word and the feature. Calculate:

$$p(w, f) = \frac{count(w, f)}{count(all\_words)}$$

$$p(w) = \frac{count(w)}{count(all\_words)}$$

$$p(f) = \frac{count(f)}{count(all\_words)}$$

where $count(all\_words)$ is the total number of word occurrences in the corpus, $count(f)$ and $count(w)$ are the total occurrences of f and w, respectively, 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). For the log, use base 10.

When you're done

When you're all done, submit via the normal submission mechanism.

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, tag both people in the submission system, 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
Writeup	10
extra credit	5
total	70 + 5 extra

Appendix

To make the output format concrete, I've included some output here. The formatting is correct, but don't read too much into the actual similarity values since they're from a collection of sentences that you don't have access to (i.e., not the sentences above).

Given an `<inputfile>` as follows:

```
a      TF      L1
a      TFIDF      EUCLIDEAN
a      PMI    COSINE
```

If I called `wordsim.sh` with a stoplist and sentences (containing a very small number of words), then an example output would be:

```
10 unique words
59 word occurrences
13 sentences/lines/documents

SIM: a TF       L1
d       1.000537629821776
e       1.632993161855452
b       2.265448693889128
c       2.632993161855452
f       2.632993161855452
g       2.6329931618554525
some    3.047206724228547
sentence        3.047206724228547
other   3.047206724228547

SIM: a TFIDF    EUCLIDEAN
d       0.5323025555230911
e       0.8454089623179316
b       0.946998582944023
c       1.1841476216541198
f       1.1841476216541198
some    1.4142135623730951
sentence        1.4142135623730951
other   1.4142135623730951
g       1.4142135623730951

SIM: a  PMI     COSINE
d       0.8660254037844387
e       0.6666666666666666
b       0.5773502691896257
```

```
f       0.28867513459481287
c       0.28867513459481287
g       0.0
other   0.0
sentence        0.0
some    0.0
```