Hector Gonzalez
LING83600: Language Technology, Fall 2020

LAB 1: WORD SIMILARITY

PART 1: WORDNET

Compute the correlation and coverage of WordNet similarity using the six methods supported by NLTK:

Path similarity

```
Spearman rho:   0.4291     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Leacock-Chodorow similarity

```
Spearman rho:   0.4291     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Wu-Palmer similarity

```
Spearman rho:   0.5309     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Resnik similarity

```
Spearman rho:   0.5551     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Jiang-Conrath similarity

```
Spearman rho:   0.4594     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Lin similarity

```
Spearman rho:   0.5439     (P = 0.0000)
Supported pairs: 201   Total pairs: 203
Coverage = 99.0148 %
```

Approach and problems I run into:

I tried to build up the script in small steps from what it was necessary to do first until the Spearman correlation calculation:

1. Made sure I imported `nltk`, `csv`, and `pandas` to read the file.
2. Created some simple scripts to print some lines from the file and played with `wordnet.synsets` to familiarize myself with the file, task, and concepts.
3. Created a loop to go through the file and assign which words I would compare to calculate their similarity. Here is where I encounter my first challenge: the script would break in the pair drink-eat. I realized that drink's first sense was a noun while eat was only a verb. This different part of speech was breaking the loop, so I incorporated `.pos()` to make sure both words in a pair shared that same part of speech. If the POS tag was different, I would skip the word pair.
4. Then, I went ahead to add the Spearman correlation calculation. To do so, I imported `scipy.stats`, and added two lists to store both the similarity calculations and the human similarity judgements.
5. Lastly, I passed both lists through `scipy.stats.spearmanr` and added a few lines of code to calculate `total counts`, `supported counts` and `coverage`.
6. I followed this process to calculate path similarity. For the other similarities supported by NLTK, I copy and pasted the same code and modified the similarity being used. To account for Resnik, Jiang-Conrath, and Lin similarities, I defined `brown_ic` as `wordnet_ic.ic("ic-brown.dat")` and passed it as an argument when needed.

PART 2: POSITIVE POINTWISE MUTUAL INFORMATION

For parts 2 and 3 I used the *news.2007.en.shuffled.deduped* file.

Compute the correlation and coverage of positive pointwise mutual information using the provided ppmi.py script to compute a PPMI table:

```
INFO: 277 words tracked
INFO: 203 pairs tracked
INFO: 80034364 tokens counted
INFO: 152 pairs covered

Spearman rho:   0.0611     (P = 0.5999)
Supported pairs: 76   Total pairs: 203
Coverage = 37.4384 %
```

Approach and problems I run into:

I followed the same approach as in part 1 where I would build small parts of the script and print regularly to make sure the steps I would take were correct:

1. I tokenized the data, where I run into my first problem while attempting this task. I didn't have a visual of how this tokenized file should look like and why. At first, I tried using the method `nltk.word_tokenize(case_folded_line)` to create a long list of words just like I did

for my project in Methods I. It was later when I learned that this didn't make sense and that I had to tokenize each sentence while keeping the sentence boundaries intact. Also, I learned that the expected outcome was a series of tokenized sentences with words separated by just spaces. Once this was clear to me, I was able to finish this part of the code after reviewing some basic coding methods from Methods I.

2. Then, I attempted to run the ppmi.py script. Here I run into my second problem: I had never run scripts from the terminal, this was my first time, and I didn't really know how to do it. Eventually, with some trial and error and some help from classmates I figured it out and I was able to obtain the PPMI results.
3. Then, I created a .tsv file from the Results_PPMI output file from step 2. I took this opportunity to review other Methods I and II concepts such as regular expressions. This time, also, I decided to write the code into a function because I, for some reason, tend to avoid it (see part 1) and because was going to be easier to reuse.
4. Lastly, to calculate the Spearman correlation, I created a function to loop through the human judgement file and then loop through the Results_PPMI.tsv file to find a matching pair. If they matched, then I would append each of the similarity numerical values to a list. From here, the approach was the same as in part 1.
5. For steps 3 and 4, I didn't encounter major problems besides my common mistakes which are the result of not being used to code.

PART 3: WORD2VEC SIMILARITY

Compute the correlation and coverage of word2vec-style word similarity using Gensim and the provided word2vec.py script

```
score = round(w2v.similarity(x, y), 6)

Spearman rho:   0.6053     (P = 0.0000)
Supported pairs: 195   Total pairs: 203
Coverage = 96.0591 %
```

Approach and problems I run into:

I approached this third part in conjunction with part 2, so the steps followed, and issues encountered during the process are the same as the ones described above.

PART 4: SUMMARY

Briefly summarize your results for the three word-similarity techniques.

These three methods seem to have very different results. While the different six word-similarity methods from part 1 resulted in a Spearman correlation between 0.4291 and 0.5551, and the Word2Vec similarity resulted in a Spearman correlation of 0.6053, is interesting to note how PPMI in a correlation of just 0.0611. These results, however, if we consider that PPMI calculates the probability of a word$_i$ occurring with context$_j$ and compares it against that probability of word$_i$ occurring in all other contexts.

By comparing the frequency in which a particular word occurs in other contexts, the probably greatly decreases and this might explain why Spearman correlation for PPMI is lower than in other methods.

Regarding how these results compare to human judgements, all similarity methods seem to fall short when compared to how humans judge the similarity for these word pairs. This to me indicates that humans may rely on other factors in addition to context dependency, information content, or ontology to determine whether two words are similar or not. Additionally, comparing the frequency of a word occurring in all contexts might not be as relevant to calculate similarity. One extreme example is the word pair *tiger-tiger*. Both humans, most of the six similarity methods from part 1, and the Word2Vec similarity method gave this pair the maximum similarity from all pairs. However, PPMI calculated a 12.9 similarity, far from some pairs like *midday-noon* for which it calculated a 21.5 similarity.

The methods in part 1 and part 3 seem, however, to get closer in results to how humans relate two different word senses. Wordnet is based on trees which might mimic more closely to how human cognition stores information. And while Word2Vec does not rely on ontology, it shows that utilizing a large linguistic context from a tokenized text, may provide results closer to human judgements.

The above explanation also relates to the results observed in the coverage for all methods. Word2Vec resulted in a 96% coverage, very close to part 1 where all six methods offered a 99% coverage. Again, how this method is built would explain the wider coverage if compared to PPMI, which only accounted for a 37.4% of the pairs found in the original data. This lower percentage seems logical when considering that the original word pairs may not occur as frequently in the corpus as the original word pairs in the data file.