Stefanie Reed
LING83600
Fall 2020

Lab 1

Part 1

The first step I took was to read in the .tsv file, which is did using padas. Then, in order to get the lemmas for each word, I created a function that used NLTK's wordnet/synset feature and returned the first synset for each word.

Next, I made a function called "get_synset" to return the synset to feed into the word similarity functions I made. For path similarity, Leacock-Chodorow, and Wu-Palmer measures, I made the arguments for those functions the tokens from the word pair columns. For the remaining similarity measures that require information content, I downloaded that from the Brown data.

Once all the functions for measuring similarity were defined, I initialized empty lists for their scores. Then I made a loop that would go through the data frame and locate the i^{th} word from each column, which I then passed into all my similarity functions and appended those to the scores lists. Once those were all computed, I made those into columns that I added to the main data frame.

Once I populated my data frame with all of the scores, I computed the correlation by indexing only the generated scores from my data frame then running a loop over those six columns.

For coverage, I used numpy to sum over the scores and divided that by the length of that number and got 200 for each method.

<u>Issues</u>

When trying to compute the Leacock-Chodorow similarity, I got an error that the parts of speech didn't match for their synsets. I tried to fix this for a couple hours and could not figure it out so I just ended up deleting the two entries that gave me the error. Those were "eat" and "drink", and "stock" and "populate".

Part 2

I first tokenized the news data and put in a white space so that the ppmi.py script would accept it. Then I created a .tsv file with the word pairs from Part 1 and used that for the pairs path when running the ppmi.py script from the command line.

I then put the results in the same order as the output from Part 1 so that I could calculate Spearman's rho. I did this by changing the word order when matching word

pairs from the ws353.tsv file using an if statement. To get the coverage, I fed the results into a pandas data frame, ran the “shape” function to get a count of the items, and then used that to find the coverage.

Part 3

The only difference with completing part 3 is that instead of running ppmi.py, I ran word2vec.py.

Part 4

WordNet Methods

	Path Similarity	Leacock-Chodorow	Wu-Palmer	Resnik	Jiang-Conrath	Lin
Correlation	0.4574	0.4574	0.5449	0.5826	0.4885	0.5643
Coverage(# of words)	200	200	200	200	200	200

Distributional Methods

	PPMI	Word2Vec
Correlation	-0.089	0.6201
Coverage(# of words)	178	193

<u>WordNet Methods</u>

These methods link the words to the WordNet ontology, thus reaping benefits from the ontology’s theoretically motivated conceptual structures made by linguists and lexicographers. The similarity is calculated by counting arc hops throughout the hierarchical structure. The various method have different ways of weighting the mereological jumps between senses.

For example, path similarity determines path length between the word senses of the pair. Then it grabs the distance between branches in the ontology counting the number of arcs between hyper/hyponyms.

The Resnik similarity method, which took second place for the similarity score compared to the human judgements, performs better than path similarity because it weighs probabilities depending on where the nodes of the word pairs are located relative to each other.

Distributional Methods

Alternative approach to ontological method is through attempting to “learn” meanings of words from the large news corpus by representing word meanings as vectors that is constructed based on distributional patterns. The word2vec method performed the best out of all the similarity methods.

This approach assumes that words with similar meanings will occur in similar contexts and builds vectors through a prediction. It takes a corpus then predicts either the context from the target word (skip-gram) or the target word from context (CBOW).

Although this method had less coverage than its ontologically-based counterparts, it performed the best, though it does seem to have its limitations. After looking at the results and scores for word2vec, I noticed it did not perform as well as I thought it might with antonyms such as “smart” and “stupid” in that the model gave them higher similarity scores than you’d expect for words that have the opposite meaning. The reason could be that in the vector-space, the proximity is based off of contextual similarity and since antonyms usually occur in similar contexts, it scored it accordingly. A human would understand that antonyms don’t have the same meaning given they are opposites, but the word2vec model does not seem to capture that as well.