Due Monday, May 13, 2019

Word Analogy

Part 1 - Word Analogy solver

Write a script called **word_analogy.py**, that solves analogies such as "dog is to cat as puppy is to ___". The script should run with the following command:

./word_analogy.py <vector_file> <input_directory> <output_directory> <eval_file> <should_normalize> <similarity_type>

The vector file is a text file with the format:

word v1 v2 v3 v4 ... v300

v1-300 are the entries of the 300 dimensional word embedding vector

Step 1 will be to read in this vector file into a dictionary where the keys are the words and the values are lists with 300 float entries each.

The input directory will be a path to a directory with several test files. Each test file has an analogy problem on each line. The format is:

A B C D

A-D are words where A is related to B in the same way that C is related to D.

Some examples include:

Athens Greece Beijing China

amazing amazingly apparent apparently

falling fell knowing knew

Step 2 is to read in these analogy problems, ignoring D. Using only A, B, and C, find the best candidate for D. This will be the word with the most similar vector to C_vec + B_vec - A_vec, according to a given similarity metric.

Wether or not the vectors should be normalized before the analogy calculation (C + B - A) is given by should_normalize:

0 - if <should_normalize> is 0, don't normalize, us the vectors as is

1 - if <should_normalize> is 1, normalize **before** the C + B - A calculation. This means if the vector is v1, v2, v3... v300, you should use v1/mag, v2/mag, v3/mag... v300/mag. Where mag is the magnitude of the vector, square root of (v1^2 + v2^2 + v3^2 + ... + v300^2).

The similarity metrics are indicated by <similarity_type>:

0 - if similarity_type is 0, use Euclidean distance (the smaller, the more similar)

1 - if similarity_type is 1, use Manhattan distance (the smaller, the more similar)

2 - if similarity_type is 2, use cosine distance (the larger, the more similar)

You should generate one output file in output_directory for every input file in input_directory. The input_file and output_file should have the same name, but be in different directories. Each output file should be of the format:

A B C D'

A-C are the same as the input file A-C, D' is the word your code generated to solve the analogy. This means that each file in the input_directory should have a corresponding file in the output_directory, with the same number of lines, and the same first three words on each line. Only the last word of each line may be different.

Step 3 is to run your analogy solver on all the input files, save the solved analogies to output files, and calculate the accuracy for each file.

In addition to the output files in the output directory, you should write the accuracy for each file to a separate output file (eval_file). The format of this file should be as shown in sample_eval.txt

Part 2 - Run and Evaluate

You will run the following:

mkdir output00 output01 output02 output10 output11 output12

./word_analogy.sh vectormodel.txt GoogleTestSet output00 output00/eval.txt 0 0

./word_analogy.sh vectormodel.txt GoogleTestSet output01 output01/eval.txt 0 1

./word_analogy.sh vectormodel.txt GoogleTestSet output02 output02/eval.txt 0 2

./word_analogy.sh vectormodel.txt GoogleTestSet output10 output10/eval.txt 1 0

./word_analogy.sh vectormodel.txt GoogleTestSet output11 output11/eval.txt 1 1

./word_analogy.sh vectormodel.txt GoogleTestSet output12 output12/eval.txt 1 2

Write a readme.pdf that includes a thoughtful analysis of the results. (.75-2 pages 1.5 spaced)

Some ideas to consider:

Which similarity metric seemed to work better? Why do you think that is?

What input files did better than others?

Did normalization help? In what cases?

If you used a different vector model (more below) what did you notice about your results?

Part 3 - Using a different vector model (optional/extra credit for all levels)

You may decide to use a different vector model that the one provided. You may make your own either by scratch, using word2vec, or using another tool. Or you can download one. Write about your exploration (no matter how successful) in your readme. What data did you use? How did you pre-process that data (tokenization, removing punctuation, stop words, stemming, POS tagging, ect)?

**Note: if you submit Parts 1 and 2 on time, and do a significant enough Part 3 by the end of term, this could count as your final project