Steven Rud COSI 114b

HW3 Write Up

Part 1:

Compare the word vector for "dogs" before and after PPMI reweighting. Does PPMI do the right thing to the count matrix? Why? Explain in a few sentences how PPMI helps.

Here are my dog vectors before and after:

Dogs vector before PPMI reweighting: [91 1 1 91 1 31 11]

Dogs vector after PPMI reweighting: [0.09011897 0. 0. 1.385104 0. 1.05814132 0.]

Before the PPMI reweighting it is a count-based vector that indicates the number of times the word "dogs" co-occurs with each given context word. After applying the PPMI reweighting, the vector is transformed to a distributional vector which helps understand and visualize the underlying semantic relationships between a word, dogs in this case, and its context words. The values in the vector are much correctly much smaller than before reweighting, and the high counts are reduced to more reasonable values for clustering, classification, and similarity comparisons. PPMI helps make a matrix more beneficial and useful by weighting the co-occurrence counts based on the frequency of each word in the corpus. It makes the more common co-occurrences have a lower weight and less common ones have a higher weight making a word and its context word have a more readable distributional value.

Do the distances you compute above confirm our intuition from distributional semantics (i.e. similar words appear in similar contexts)?

Here are the distances:

Distances on full matrix:

2.577348527621889 2.787888289701278 2.314141389656323 2.8283578447888957 3.2782605194278345 3.1698713452863867

Yes, the distances I computed mostly confirm that similar words appear in similar contexts as words that share being both from animals/humans and/or being nouns/verbs have a lower distance so they are closer together and appear more often in similar contexts.

Compute the Euclidean distances of the human/animal nouns/verbs again but on the reduced PPMI-weighted count matrix. Does the compact/reduced matrix still keep the information we need for each word vector?

Here is regular PPMI:

```
[[0.07524186 0.05523587 0.         0.09011897 0.         0.
  0.        ]
 [0.05523587 1.48328242 0.24055212 0.         0.         0.
  0.24055212]
 [0.         0.24055212 1.91058897 0.         0.83600699 0.
  0.        ]
 [0.09011897 0.         0.         1.385104   0.         1.05814132
  0.        ]
 [0.         0.         0.83600699 0.         2.00676751 0.
  0.        ]
 [0.         0.         0.         1.05814132 0.         2.28477696
  0.        ]
 [0.         0.24055212 0.         0.         0.         0.
  1.91058897]]
```

Here is the reduced PPMI:

```
[[ 5.10820616e-02  7.38535607e-03 -2.33874188e-02]
 [ 2.19038408e-03  3.68329844e-01 -8.22197917e-01]
 [ 1.45973868e-03  1.94412364e+00  3.40198852e-03]
 [ 1.64832244e+00 -9.96913717e-04 -4.35216406e-04]
 [ 1.24676663e-03  2.00385333e+00  2.37018913e-01]
 [ 2.48879644e+00 -1.97883788e-03  1.73121456e-03]
 [ 4.90144680e-04  9.76587807e-02 -1.82829723e+00]]
```

Yes, this still keeps the information we need for each word vector as from this I can and did compute and reduced distance and still get values that follow the intuition that similar words appear in similar contexts.

Distances on reduced matrix:

1.9485579942090345 2.605507097308661 1.876551872590782 2.600875900184496 3.1581895934087862 3.0904045007182805

Part 2:

The Table comparing the synonym test results using Euclidean distance vs. cosine similarity and COMPOSES vs. word2vec. The values are slightly different with each run, I put in my highest gotten accuracy for the test. I found in my result the composes word vector was generally higher than the google word2vec vector and Euclidean and cosine would return no differences. The Euclidean and cosine have no difference in accuracy likely because they both used on a vector of similar lengths and aligned in similar directions. Or possibly due to a small bug I had not picked up on.

Method	Accuracy
COMPOSES (Euclidean)	20.00%
COMPOSES (cosine)	20.00%
Google Word2Vec (Euclidean)	19.30%
Google Word2Vec (cosine)	19.30%

First to create the composes vector I extracted with a tar.gz opening and put the word vector values into a txt file in a new folder called temp_folder. Being in a txt file made it easier to read when doing cosine and Euclidean tests for both the synonyms and analogies in part 2.

To create the word vectors for the analogy task and find the accuracy, I first loaded the file and, in each line, I only kept the first two words to ignore anything unnecessary, Then whenever a line was empty then I knew the next 8 lines were needed to kept track of as they were a question. I ignored the next line as it said where the question was from. The next 7 lines were the given analogy, the choices, and the answer which I stored in an individual question list. Where the list contained a full single question. Then I added that list with a question into another list that would store lists of every question. Then I reset the individual question list and continued the process for the whole text file. After having a full list of lists every question I went through and analyzed it. In each question the first two words where the given analogy I needed to base my answer off of. The next 5 lines were the choices, and the final line was the answer. I correlated the letter answer to a number, where a is 0, b is 1, c is 2, etc. Then I ran a vector difference using the composes word vector from earlier for the given analogy. Then I ran

through each choice and ran the same vector difference for them. I compared the vector difference between the choice and given analogy using the cosine similarity test “np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))”. The choice that had the greatest cosine similarity was stored as the answer and outputted. I checked the outputted answer with the given answer (which like I said earlier was stored as the last line of each question list) and divided the correct amount of answers with total questions and got the final accuracy.

My final accuracy was: 42.32%.