CS115B (Spring 2024) Homework 4
Distributional Semantics Takes the SAT

Due March 19, 2024

Introduction

Distributional semantics is one of the modern methods in natural language processing in solving tasks that require semantic knowledge of a word or relations between words. In this homework, we will learn how to create semantic representations of words from a corpus and how to use them in computational lexical semantic tasks.

In this assignment, you are provided with minimal started code. You should implement the portions of the homework in the functions that correspond to part 1, part 2 and the two tasks of part 2. You are encouraged to create helper functions and classes as needed to complete the homework. Your final submission should clearly print out the information required for each part of the homework (distance metrics, accuracy, etc).

While hard coded paths are not great programming practice (it is best to pass in the paths to files and resources as arguments to your script), it is fine to do so for the purposes of this assignment. Use relative paths when reading from the files. DO NOT use absolute paths that only will work on your computer. (You should open the file as "dist_sim_data.txt" and NOT something like "/home/user/chester/cs115b/homework4/dist_sim_data.txt".

1 Create distributional semantic word vectors

Your task is to create distributional semantic word vectors from a small artificial corpus (see Data and Resources section below). We will need to use Numpy and Scipy for this part of the homework. We break the process down into steps below. For each step, also answer any questions given.

- Compute the co-occurrence matrix (Numpy array) C, such that `C[w, c]` = the number of (w, c) **and** (c, w) bigrams in the corpus. Each line is its own sentence.
- Multiply your entire matrix by 10 (to pretend that we see these sentences 10 times) and then smooth the counts by adding 1 to all cells.
- Compute the positive pointwise mutual information (PPMI) for each word w and context word c:

$$\texttt{PPMI(w, c)} = \max\left(\log\left(\frac{P(w,c)}{P(w)P(c)}\right), 0\right)$$

  Note that these probabilities can be computed directly from your count matrix.
- Now instead of using your (original) count matrix, use the PPMI matrix as a weighted count matrix. Then 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 (short prose will do here; no need to show any math, rather just an intuitive understanding).
- At this point, we have a functional distributional semantic model. Let's try to find similar word pairs. Compute the Euclidean distance between the following pairs (you can use the command `scipy.linalg.norm` to compute the length/norm of a vector):
  - "women" and "men" (human noun vs. human noun)
  - "women" and "dogs" (human noun vs. animal noun)
  - "men" and "dogs" (human noun vs. animal noun)
  - "feed" and "like" (human verb vs. human verb)
  - "feed" and "bite" (human verb vs. animal verb)
  - "like" and "bite" (human verb vs. animal verb)

  Do the distances you compute above confirm our intuition from distributional semantics (i.e. similar words appear in similar contexts)?
- Decompose the matrix using singular-value decomposition (SVD) by using the command `scipy.linalg.svd`, using the commands given below. Then verify that you can recover the original matrix (within a tolerance) by multiplying `U`, `E`, and `Vt` together.

```
import scipy.linalg as scipy_linalg
U, E, Vt = scipy_linalg.svd(PPMI, full_matrices=False)
E = np.diag(E) # compute E
print(np.allclose(PPMI, U.dot(E).dot(Vt)))
```

- Reduce the dimensions to 3 to get word vectors by using the commands below. Now your word vectors are in the abstract semantic space.

```
V = Vt.T # compute V = conjugate transpose of Vt
reduced_PPMI = PPMI.dot(V[:, 0:3])
```

- 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?

2 Computing with distributional semantic word vectors

You are given two types of word vectors.

1. Classic distributional semantic matrix (like in part I). The matrix is trained using a 2-word context window, PPMI weighting, and SVD reduction to 500 dimensions. The co-occurrence statistics are computed from the British National Corpus and the Web-As-Corpus. It is trained by the COMPOSES toolkit. You could in fact make your own using the pipeline from part I, but SVD takes hours in a realistic setting—cubic time ($O(n^3)$) in the number of dimensions.

2. Google's 300-dimensional word vectors trained by deep learning. The version we provide is trained on the Google News corpus and taken from the word2vec toolkit. We will see how their word vectors fare against the more classical method.

We will use these word vectors in semantic tests. Note that we are using an unsupervised learning technique here, so the dataset is not split into train/dev/test.

1. Synonym detection. You are given $\sim$ 900 pairs of synonymous verbs in the data set. We want to use word vectors to detect synonyms. Here is what we need to do:

(a) Create a multiple-choice question test set: For each verb, you will pick 1 of the synonyms and 4 random non-synonyms from the data set. Make 1,000 such synonym multiple-choice questions. The exact format of the synonym test set is up to you, as long as your code can process it.

(b) Your task is to use the word vectors to pick the synonym out of the 5 alternatives for each verb and compute the accuracy. Try using Euclidean distance and cosine similarity. In case you come across any unknown words, any sensible method of handling them is fine; just make a note in your write-up about what you did. Make a table to compare the results from those methods on each set of word vectors (COMPOSES and word2vec).

2. SAT Analogy questions. You are given 374 SAT analogy questions. I am sure most of you did well on the SAT in the past, but apparently they've gotten rid of the analogy portion now that computers can do it.

   (For those of you who did not go to an American college, the SAT is a national standardized test required by most American colleges as part of the application package. One of the sections used to be on lexical analogy. For example,

   MEDICINE : ILLNESS ::

   (a) law : anarchy

   (b) hunger : thirst

   (c) etiquette : discipline

   (d) love : treason

   (e) stimulant : sensitivity

   MEDICINE is used to combat ILLNESS. Law is used to combat anarchy. (Hunger is not used to combat thirst, etiquette is not used to combat discipline, etc.) So (a) is the correct answer.)

   Your task is to use the word vectors to answer these questions correctly. Again, any sensible method of handling unknown words is fine. Submit a write-up describing what you have tried and your results. Unlike the synonym task, you have to be a bit creative with how you use the word vectors.

   Think about where the word vectors in the question and the choices stay in the abstract semantic space. Think about how to obtain a relation vector between the two words (e.g. subtracting the two word vectors? adding? multiplying? concatenating?).

   An average SAT taker got around 57% accuracy on the analogy section (ouch, maybe that's why they got rid of it), and the state-of-the-art system is almost as good. Random guesses get 20% accuracy. You should aim for about 30% accuracy, so significantly better than random guesses.

Data and Resources

You are given the following data sets. A link to the data is on Latte:

Artificial corpus for the first part: `dist_sim_data.txt`

Synonym detection data: `EN_syn_verb.txt`

SAT questions: `SAT-package-V3.txt`

Google's word2vec word vectors (filtered for homework vocab):
`GoogleNews-vectors-negative300-filtered.txt`

COMPOSES' word vectors (filtered for homework vocab):
`EN-wform.w.2.ppmi.svd.500-filtered.txt`

Submission

Submit in a single zip file:

- Any code you write in the course of this assignment.
- Your synonym test set (in any format that your code can read) from section 2.
- A write-up (in PDF format) containing:
  - The answers to the questions in section 1, including the vector distances between the word pairs given, once normally and once using the reduced PPMI-weighted count matrix.
  - A table comparing the synonym test results using Euclidean distance vs. cosine similarity and COMPOSES vs. word2vec.
  - A write-up discussing methods you used for creating word vectors for the analogy task, and your accuracy results.

Grading

Grades will be determined as follows:

- 20% for code submission.
- 20% for section 1 write-up that discusses all points listed above. (14 points on questions and discussion, 6 points on word-pair calculations) .5 points will be deducted for each missing word-pair. Remember, you should have 2 sets of vector distance calculations.
- 10% for synonym test set submission.
- 20% for synonym test results. (10 points for Euclidean distance results, 10 points for cosine similarity results).
- 30% for analogy task discussion and results.
- Within these parameters, partial credit will be assigned where possible. You may notice that the writing and discussion is weighted higher than the actual results. This is not an accident. The results are important but demonstrating your understanding of why things worked (or didn't) is more so (think about it: someone else has already built working distributional semantic algorithms—our job is to prepare you to think in new directions about what you can do with them). For that reason, mediocre results and good discussion will net you more credit than great results without a good discussion. When in doubt, write up the issues you're having and some speculation as to why.

  TL;DR: Please do not skimp on the discussion!