TTIC 31190: Natural Language Processing
Assignment 2: Word Vectors

Kevin Gimpel
Assigned: Jan. 22, 2016
~~**Due: 11:59 pm, Feb. 3, 2016**~~
Due: 8:00 pm, Feb. 5, 2016
Submission: email to `kgimpel@ttic.edu`

Submission Instructions

Package your report and code in a single zip file or tarball, name the file with your last name followed by "_hw2", and email the file to `kgimpel@ttic.edu` by 8:00 pm on Feb. 5, 2016. In your report, please include an estimate of approximately how many hours you spent working on this assignment. This will help us better calibrate future assignments. It will not affect your grade.

Collaboration Policy

You are welcome to discuss assignments with others in the course, but solutions and code must be written individually.

Distributional Word Vectors

You will implement and experiment with ways of creating word vectors. Let's begin by defining a **vocabulary** V and a **context vocabulary** V_C. You can assume for now that $V_C \subset V$. The steps below will ask you to implement different ways of building word vectors for the words in V, using context words from V_C. I will provide you a corpus of sentences from English Wikipedia, along with files that you can use to fill V and V_C.

The evaluation of your word vectors will be **qualitative**. We will not use any quantitative evaluation metrics to evaluate your word vectors. You will print nearest neighbors for certain words and make observations about them in order to compare different methods and choices. Qualitative analysis is a very important part of NLP. Please do not simply print the output after making each change and expect us to notice the differences. You must study the output and write about your observations, supporting them with evidence from the output. Some of the differences may show up for certain words but not for others.

Each time you make a change to your method for computing word vectors, you should evaluate the vectors by performing the following EVAL(M) subroutine, where M is a matrix containing word vectors:

EVAL(M): Using the word vectors defined by M, compute and print the 10 nearest neighbors from V for each word in a set $\mathcal{N}$. $\mathcal{N}$ should contain **16** words, including **6** of your own choosing and the following **10**: {people, flew, transported, quickly, good, python, apple, red, chicago, language}. Use cosine similarity as the similarity metric in the nearest neighbor calculation. So, you

will have to compute $|\mathcal{N}| \times |V|$ cosine similarities. When doing so, omit the original word from the list of nearest neighbors since it will always have cosine similarity of 1. After computing the nearest neighbors, analyze them. What kinds of similarity are being captured by your vectors? Do the nearest neighbors look better for certain words than for others? What properties of those words can help to explain the quality of the nearest neighbor lists? It is often easier to analyze nearest neighbor lists by comparing them across two different sets of word vectors, which you will have many opportunities to do below.

Provided Data

The following data files are provided to you for these experiments:

- `wiki-{0.1,2}percent.txt` - samples of English Wikipedia downloaded May 15, 2015, with punctuation separated from words and all characters converted to lowercase. For any results and analysis reported in your submission, you should use `2percent`. The smaller file is provided for debugging and preliminary testing. It should still produce reasonable nearest neighbors for most common words.
- `vocab-{3k,10k,15k,50k,rare3k}.txt` - vocabulary files containing the {3000, 10000, 15000, 50000} most common words in Wikipedia (one word per line). They are sorted in decreasing order of frequency, so the most frequent words are first. For the required steps, use `vocab-15k.txt` for V and `vocab-10k.txt` for V_C. The `vocab-3k.txt` and `vocab-rare3k.txt` files are provided for some of the optional components.

1 Required:

The following 2 steps are required:

1. Implement distributional counting for word vectors. Build a **word-context** matrix C of counts, with size $|V| \times |V_C|$. Each row of C corresponds to a word in V. Each column of C corresponds to a word in V_C. A particular entry C_{ij} in C should equal the number of times that context word j appears within w words of word i. A context window contains w words to either side of the center word, so it contains $2w$ words in total. Use $w = 4$. For words near the sentence boundaries, pad the sentence with beginning-of-sentence and end-of-sentence characters (<s> and </s>). Use `vocab-15k.txt` to populate V and use `vocab-10k.txt` to populate V_C. EVAL(C).
2. After building C as above, build a new matrix C_{pmi} that contains positive pointwise mutual information (PPMI) values for each entry (i.e., each center word and context word pair). See Section 4.2 of Turney and Pantel (2010) for more details. EVAL(C_{pmi}), compare the results to EVAL(C), and note any differences you observe. Use the same value of w as above (i.e., $w = 4$).

2 Your Choice:

After doing the above, do **two (2)** of the following five components:

1. **effect of hyperparameters**: Perform experiments and analyze the results to determine the effect of certain hyperparameters:

- Compute and compare C_{pmi} for three different choices for V_C: (1) the most frequent 10,000 words (`vocab-10k.txt`, the same as used above), (2) the most frequent 3,000 words (`vocab-3k.txt`), and (3) a set of 3,000 relatively rare words (`vocab-rare3k.txt`). Compare the results of EVAL for each.
- Compute and compare C_{pmi} for three different window sizes, from very small ($w = 1$) to very large ($w = 10$ or 15). Compare the results of EVAL for each.

2. **model variation**: Implement and experiment with the following modeling variations.
   - When computing the counts to fill C, give more weight to context words that are closer to the center word. Define your own weighting scheme based on the distance between the center word and the context word. Compare the results of EVAL to uniform weighting. You may want to use a larger value of w (e.g., $w = 8$) for this comparison in order to see larger differences in the nearest neighbors. (You can use simple counts here if you like instead of PPMI values, but both are reasonable options.)
   - Separate out left and right context words into distinct dimensions of the vectors. That is, each word in V will have V_C dimensions for left context words and V_C dimensions for right context words. Compare the results of EVAL. (Again, you can use either simple counts or PPMI values for this. If you do the latter, you will have to compute different PPMI values for left vs. right context words.)

3. **dimensionality reduction**: Compute the **truncated SVD** of C_{pmi}. That is, compute the singular value decomposition (SVD) of C_{pmi}, then retain only the top k dimensions. See Section 19.5 of *Speech and Language Processing (3rd Edition)* for more details. Experiment with $k \in \{5, 50, 500\}$. EVAL the result for each k. Most programming languages will have free libraries that implement SVD.[1] Note that the expense of this operation will depend on the sizes of V and V_C, so make sure that you use the given vocabulary files to prune these vocabularies. You can further restrict the size of V_C to 3,000 by using `vocab-3k.txt` for this component.

4. **linguistic analysis**: Perform analysis to address the following questions. For each of these, you should add more words to $\mathcal{N}$ and analyze their lists of nearest neighbors. You may also find it helpful to manually inspect instances of words in $\mathcal{N}$ in the original corpus to find typical contexts for each word, and to compare what happens when varying some of the hyperparameters (such as window size, counts vs. PPMI, etc.).
   - What appears to be happening for words with multiple senses (e.g., *bank*, *cell*, *apple*, *apples*, *space*, *frame*, etc.)?
   - How well are rare words being handled?
   - How about proper nouns (which are often rare words)?
   - Are synonyms differentiated from antonyms or are they highly similar?
   - Do nearest neighbors tend to have the same part-of-speech tag as the input word, or do they differ? Does the pattern differ across different part-of-speech tags for the input word? Compare adjectives, adverbs, nouns, and verbs.

5. **visualization**: visualize a sample of your word vectors in two dimensions. Choose a small sample of frequent words to plot (e.g., try 50 words randomly sampled from the most frequent 3000 words in V_C) and use **both** of the following:

[1] In my solution, I used C++ and Eigen (`http://eigen.tuxfamily.org/index.php?title=Main_Page`).

- use principal component analysis (PCA) to reduce the dimensionality of the word vectors down to 2 dimensions, then plot the words
- use t-SNE (implementations are available online, e.g.: `http://lvdmaaten.github.io/tsne/`)

Which visualization method works better for your vectors?
Do the visualizations reveal new insights into the word vectors that were not evident from the nearest neighbors?
For each visualization method, plot the vectors from both C and C_{pmi}. What differences do you notice between C and C_{pmi}? Do the two visualization methods show different kinds of differences between the two?

3 Extra Credit: Your Second Choice:

If you're interested in extra credit, do a third of the five choices from Section 2.

4 Implementation Tips

- The most difficult part of this assignment is probably making your implementation efficient enough to scale up to large corpora.
- You only need to do one pass through the corpus to compute C (or C_{pmi}), so you do not need to store the corpus in memory.
- I would recommend using sparse data structures to store the counts (these may have various names depending on the programming language, e.g., maps, hashes, or dictionaries).
- Using the given vocabulary files can drastically speed up your code and reduce memory usage.
- If you encounter underflow when estimating the small probabilities used for PPMI calculation, you can perform the computation in the log domain (though I did not have to do this for my implementation).

References

Turney, P. D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188. [2]