CS7.401: Introduction to NLP | Assignment 3

Instructor: Manish Shrivastava

Deadline: March 23, 2023 | 23:55

General Instructions

1. You should implement the assignment in Python.
2. Ensure that the submitted assignment is your original work. Please do not copy any part from any source, including your friends, seniors, and/or the internet. If any such attempt is caught, serious actions, including an F grade in the course, are possible.
3. A single .zip file needs to be uploaded to the Courses Portal.
4. Your grade will depend on the correctness of answers and output. Due consideration will also be given to the clarity and details of your answers and the legibility and structure of your code.
5. Please start early to meet the deadline. Late submissions won't be evaluated.

1 About

Many natural language processing systems use modern distributional semantic algorithms, also known as word embedding algorithms, to learn meaningful vectors for words. The aim of these algorithms is to create embeddings in such a way that words with similar meanings have similar mathematical representations. Word embeddings can be categorized into two broad categories: frequency-based and prediction-based.

Frequency-based embeddings use different types of vectors, including Count Vector, TF-IDF Vector, and Co-occurrence Matrix with a fixed context window. Prediction-based embeddings, such as Word2vec, use models like Continuous Bag of Words (CBOW) and Skip-Gram (SG). However, the computation of Word2vec's training algorithm can be computationally expensive due to calculating gradients over the entire vocabulary. To address this issue, variants like Hierarchical Softmax output, Negative Sampling, and Subsampling of frequent words were proposed.

This assignment requires implementing one of the frequency-based modelling approaches, such as Singular Value Decomposition (SVD), and comparing it with the embeddings obtained using one of the variants of Word2vec, such as CBOW implementation with Negative Sampling. The analysis will highlight the differences in the quality of embeddings obtained. The terms "word vectors" and "word embeddings" are used interchangeably, but the term "embedding" specifically refers to encoding a word's meaning in a lower-dimensional space.

2 Questions & Grading

Total marks: **100**

2.1 Theory

1. Explain negative sampling. How do we approximate the word2vec training computation using this technique? [**5** marks]
2. Explain the concept of semantic similarity and how it is measured using word embeddings. Describe at least two techniques for measuring semantic similarity using word embeddings. [**5** marks]

2.2 Implementation

1. Implement a word embedding model and train word vectors by first building a Co-occurrence Matrix followed by the application of SVD. [**25** marks]
2. Implement the word2vec model and train word vectors using the CBOW model with Negative Sampling. [**35** marks]

2.3 Analysis

Report these for both models after you're done with the training.

1. Display the top-10 word vectors for five different words (a combination of nouns, verbs, adjectives, etc.) using t-SNE (or such methods) on a 2D plot. [**10** marks]
2. What are the top 10 closest words for the word 'titanic' in the embeddings generated by your program? Compare them against the pre-trained word2vec embeddings that you can download off-the-shelf. [**10** marks]

2.4 Presentation

We would check these points at the time of individual evaluations that would involve code walk-through, explanation of the report analysis, and questions based on the implementation.

- Implementation efficiency & quality
- Report quality
- Inclusion of a readme along with the code
- Crisp & clear explanation of the code during evals

[**10** marks]

3 Training corpus

Please train your model on the corpus linked here:
https://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Movies_and_TV.json.gz

4 Submission format

Zip the following into one file and submit it on the Moodle course portal:

1. Source code (Should include .py files only.)
2. Pre-trained model and the generated embeddings
3. Report answering all the questions
4. Readme file (should contain instructions on how to execute the code, restore the pre-trained model, and any other information necessary for clarity)

Name the zipped file as <roll number>_<assignment3>.zip, e.g.: 2022xxxxxx_assignment3.zip.

Note: If the pre-trained models and embeddings cross the file size limits, upload them to a OneDrive/GDrive and share the read-only accessible links in the readme file.

5 FAQs

1. Do we train the embeddings for all the unique words we see in the corpus?
   You can place a limit: words with a frequency less than five can be ignored. If you have enough compute, you can train for as many words as possible.

2. Do I need to ignore the stop words?
   Usually, stop words occur with very high frequency. It won't help if you blindly ignore the stop words from the corpus. To avoid the effect of the terms that occur very frequently, you can use sub-sampling, which is another technique to decrease the compute cost in word2vec training.

3. Do I need to use lemmatization before feeding the samples to the model? No, for the sake of this assignment, you may not perform lemmatization.

4. I do not have enough compute. Can I reduce the working corpus size?
   Yes, you may reduce the vocabulary size. But make sure you train the model on enough no. of sentences, which should be a minimum of 40,000 sentences.

5. Can I submit my code in Jupyter Notebooks?
   No, the final submission should be a Python script. You may work using a Jupyter Notebook, but make sure to convert it to .py file before submitting.

6. Explain the stuff we have to report under analysis.
   We have asked you to report these two results per model:

   (a) For displaying the top 10 words:
   You can pick the 10 nearest words for a particular word in terms of cosine similarity. After having picked the top 10 words, plot them on a 2D plot. You can use t-SNE (or such) to reduce dimensions before plotting.

   (b) For comparing against pretrained word2vec embeddings from a library:
   Download any pre-trained vectors for English that are available. Get the 10 closest words for the word "camera" in the embeddings generated by you as well as the downloaded model, and compare the two outputs.

   You can use libraries like Scikit-learn and Gensim to read vector files and pick the nearest k words for a query word.

6 Resources

6.1 Papers

1. Efficient Estimation of Word Representations in Vector Space
2. Distributed Representations of Words and Phrases and their Compositionality

6.2 Explanatory supplements

1. Stanford lecture notes - SVD & word2vec
2. A simple, intuitive explanation of word2vec, with Negative Sampling included
3. Skip Gram with Negative Sampling
4. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method 5. On word embeddings in general