CS5740, Spring 2020: Assignment 2

Due: March 15, 11:59pm

Report page limit: 3 pages

This assignment focuses on building vector representations (or embeddings) of words. Given a large corpus of text, your goal is to build vector representations of each of the words in that text in such a way that the cosine similarity of these vectors accurately represents the semantic similarity of the words they represent. Your representations will be evaluated on the wordSim353 dataset. The evaluation code calculates the similarity of a word pair by taking the cosine similarity of the two words' vectors. The entire dataset is then scored using the Spearman's rank correlation coefficient between these cosine similarities and the human annotations in the dataset.

Forming Groups Please form your groups within 72 hours of the assignment release. First, create your group on CMS, and then form it on Github Classroom. We will not be able to grade assignments where CMS and Github are not in sync. As long as you are not part of a group on CMS, we will consider as not part of a group. **Following the 72 hours deadline, we will do random assignments. We will deduct 5 points from the grade of each group that requires random assignment.** Please use the forum to form groups. If we identify cases of students that do not work together, we will be forced to penalize the assignment heavily.

Please submit your report PDF on both CMS and Gradescope by the deadline.

Submission **Please see the submission guidelines at the end of this document.** In your write-up, at the least, be sure to describe which approach you used for your embedding and a discussion of choices you made in constructing your embedding and deciding on parameters (such as dimension). You do not need to submit the content of your GitHub Classroom repository, but make sure to include your final embedding in your repository for development and test. Please submit only embeddings for the words that appear in the test and development data.

Starter Repository:
https://classroom.github.com/g/YrsyLYhs (GitHub Classroom invitation link)

Leaderboard:
https://github.com/cornell-cs5740-20sp/leaderboards/blob/master/assignment2/leaderboard.csv

Report Template:
https://www.overleaf.com/read/syffkkrttmst

Data and Code Setup Follow the instructions in Assignment 1 for setting up teams on GitHub Classroom and downloading the starter repository. Remember that all team code should be pushed to the repository by the deadline. *Only code present in the master branch of your repository will be graded!* Please implement your solution in Python. You may use DyNet and NumPy, as well as off-the-shelf stemmers and tokenizers.

Training Data You will want to use a large corpus of text. A good one to start with is the billion word language model benchmark dataset.[1] It contains the first 1M sentences in raw text and also analyzed with a part-of-speech tagger and dependency parser. The parsed data is in CONLL format.[2] This dataset is available to download here:

http://www.cs.cornell.edu/courses/cs5740/2020sp/r/a2-data.zip

[1] More information: http://www.statmt.org/lm-benchmark

[2] For a description of the CONLL format, see: http://universaldependencies.org/format.html

If you want to use more data, you can download the entire corpus from here:

http://www.statmt.org/lm-benchmark/

We also recommend using WordNet via the NLTK API or the core 5000-senses file:

http://wordnetcode.princeton.edu/standoff-files/core-wordnet.txt

Evaluation Scripts Also included in the starter repository are three scripts to help with evaluation. In this assignment, you will evaluate your embedding by comparing its predicted similarity between pairs of words to human-scored similarity between pairs in the WordSimilarity-353 dataset.[3] The word similarity pairs for development (`similarity/dev_x.csv` and human scores `similarity/dev_y.csv`) and leaderboard test words (`similarity/test_x.csv`) are included in the starter repository.

Generate word pair similarity scores using your embedding:

```
python similarity.py > prediction.csv
    --embedding your_embedding.txt
    --words similarity/dev_x.csv
```

Evaluate your similarity scores against human ratings:

```
python evaluate.py
     --predicted prediction.csv
     --development similarity/dev_y.csv
```

Your embeddings will likely get quite big. Rather than submitting the entire embedding with this assignment, please submit only the embeddings for the words in the development and test data. You can generate this "reduced embedding" with:

```
python reduce.py > reduced_embedding.txt
     --embedding your_embedding.txt
```

Task There are many papers on the topic of learning word embeddings. You might want to read a selection of these before settling upon an approach. A simple one that uses syntax is here:

http://www.cs.bgu.ac.il/~yoavg/publications/acl2014syntemb.pdf

All models of distributional similarity have a notion of the context in which a word occurs. The context can also be chosen to capture syntactic and semantically relevant information. Embeddings find lower dimensional representations that capture most of the information in the raw feature space with respect to some loss function. The literature describes multiple ways of doing this. You will have to experiment (and report) with different choices for at least three of:

- amounts of training data
- dimensions for the embeddings
- contexts (linear, syntactic)
- objective functions

You are required to have some reasonable handling of unknown words. Simply ignoring them will results in low results on the test data. Some result expectations:

- Development correlations: 0.4+ is OK, 0.5 is good, and above is starting to be interesting.
- Test correlations: 0.3 is OK, but things start to become serious (and interesting) at at 0.35+.

[3] http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Output Format The output embedding file should contain one word vector per line, with the word and each embedding dimension separated by a single whitespace. For example, for three words and 2D embeddings, we can have a file whose contents may look like:

```
cat 0.8 0.0
dog 0.7 0.1
bat 0.5 0.5
```

If the embeddings are not all of the stated dimension, the cosine similarity score will not work! Please submit only embeddings for the words that appear in the test and development data. Otherwise the submission file will get too big. The provided code contains functionality to prune the embeddings to only the ones that are needed.

Leaderboard Instructions We will maintain a leaderboard. To work with the leaderboard, we require that your predictions have the following format: `./results/test_scores.csv` where `.` is the root directory of the repository. Please work within your group only. The leaderboard will be updated once daily.

Use the evaluation scripts as described above to develop and evaluate your embeddings. To create the test predictions, run `similarity.py` along with your learned embeddings `similarity/test_x.csv` to generate a submission file `./results/test_scores.csv`. The leaderboard uses Spearman's rank correlation coefficient to evaluate the word similarity scores generated by your embedding, similar to `evaluate.py`. You should expect a score between zero (embedding captures no relationship between similar words) and one (embedding similarity is perfectly correlated with human-rated word similarity).

Performance Grading The grading of your test performance on the leaderboard will be computed as $\frac{\max(0,c-0.07)}{0.25-0.07} \times 24$, where c is the test correlation from the leaderboard.

Current CS5740 State of the Art The highest performance on this assignment is 36.87 (19sp).

Submission, Grading, and Writeup Guidelines Your submission on CMS is a writeup in PDF format. **The writeup must follow the template provided. Please replace the TODOs with your content. Do not modify, add, or remove section, subsection, and paragraph headers.** The writeup must include at the top of the first page: the names of the students, the NetIDs of both students, the team name, and the URL of the Github repository. The writeup page limit is **3 pages**. We will ignore any page beyond the page limit in your PDF (do not add a cover page). We have access to your repository, and will look at it. Your repository must contain the code in a form that allows to run it form the command line (i.e., Jupyter notebooks are not accepted).

The following factors will be considered: your technical solution, your development and learning methodology, and the quality of your code. If this assignment includes a leaderboard, we will also consider your performance on the leaderboard. Our main focus in grading is the quality of your empirical work and implementation. Not fractional difference on the leaderboard. We value solid empirical work, well written reports, and well documented implementations. Of course, we do consider your performance as well. The assignment details how a portion of your grade is calculated based on your empirical performance.

In your write-up, be sure to describe your approach and choices you made. Back all your analysis and claims with empirical development results, and use the test results only for the final evaluation numbers. It is sometimes useful to mention code choices or even snippets in write-ups — feel free to do so if appropriate, but this is not necessary.

Some general guidelines to consider when writing your report and submission:

- Your code must be in a runnable form. We must be able to run your code from vanilla Python command line interpreter. You may assume the allowed libraries are installed. Make sure to document your code properly.

- Your submitted code must include a `README.md` file with execution instructions. Make sure to document your code properly.
- Please use tables and plots to report results. If you embed plots, make sure they are high resolution so we can zoom in and see the details. However, they must be readable to the naked eye. Specify exactly what the numbers and axes mean (e.g., F1, precisions, etc).
- It should be made clear what data is used for each result computed.
- Please support all your claims with empirical results.
- All the analysis must be performed on the development data. It is OK to use tuning data. Only the final results of your best models should be reported on the test data.
- All features and key decisions must be ablated and analyzed.
- All analysis must be accompanied with examples and error analysis.
- Major parameters (e.g., embedding size, amount of data) analysis must include sensitivity analysis. Plots are a great way to present sensitivity analysis for numerical hyper parameters, but tables sometimes work better. Think of the best way to present your data.
- If you are asked to experiment with multiple models and multiple tasks, you must experiment and report on all combinations. It should be clear what results come from which model and task.
- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.
- Make figures clear in isolation using informative captions.

This assignment was adapted from Slav Petrov.