CS5740, Spring 2017: Assignment 2

Due: March 13, 2017

Make sure to specify at the top of your names, Kaggle team, and GitHub repository URL.

Submit your code in a runnable form. We must be able to run your code from vanilla Python command line interpreter. Make sure to document your code properly.

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.

Starter Repository: https://goo.gl/547Q1P (GitHub Classroom invitation link)
Evaluation Kaggle: https://inclass.kaggle.com/c/cs5740-word-embeddings

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!*

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, along with a copy of WordNet is available to download here:

http://www.cs.cornell.edu/courses/cs5740/2017sp/res/a2-data.zip

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

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

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 Kaggle 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:

[1] More information: http://www.statmt.org/lm-benchmark
[2] For a description of the CONLL format, see: http://universaldependencies.org/format.html
[3] http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

```
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 with different choices for at least three of:

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

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.

Kaggle Instructions. We will use private Kaggle competitions to manage the evaluation and provide a leaderboard. Please work within your group only. As with Assignment 1, you may make one group submission to the competition per day until the deadline.

Use the evaluation scripts as described above to develop and evaluate your embeddings. When you would like to submit to Kaggle, use `similarity.py` along with `similarity/test_x.csv` to generate a submission file that you can upload to Kaggle. The competition 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).

Submission. Your submission on CMS should include a writeup in PDF format. In your write-up, 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). 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. The writeup page limit is four pages. You do not need to submit your GitHub Classroom repository, but make sure to include your final (reduced) embedding in your repository.

Grading. You will be graded to approximate evaluation in real life. The following factors will be considered: your technical solution, your development and learning methodology, the quality of your code, and the performance of your model on our held-out super secret test data.

Submission and Writeup Guidelines.

- It should be made clear what data is used for each result computed.
- 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.
- 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. Specify exactly what the numbers mean (e.g., F1, precisions, etc).
- 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.
- We recommend making figures clear in isolation with informative captions.
- Your submitted code must include a `README.md` file with execution instructions.
- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.