Statistical NLP - Assignment 5

due November 15 - at 5pm by email to petrov@cs.nyu.edu

This assignment focuses on neural networks and vector representations (or embeddings) of words. The primary goal is to learn vector representations of words that point in similar directions for similar words, and different directions for different words. The secondary goal is to revisit the previous assignments and build neural network models for the same tasks (language modeling, classification and part-of-speech tagging - if you are ambitious, you could try to build a word alignment model as well, but that might be much more difficult).

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 that 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.

Resources: You will want to use a large corpus of text. A good one to start with is this one, which is typically used for language modeling:

```
http://www.statmt.org/lm-benchmark/
    1-billion-word-language-modeling-benchmark-r13output.tar.gz
```

The data release (data5.zip) for this assignment is in the usual location. 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 (you can find a definition of the format online if it is unclear). If you want to use more data, you can download the entire corpus (I have a parsed version of the entire corpus, if you think it will help your model, let me know and I can see how to get it to you). Note that the data files can get quite big. Similarly, the embedding files will get big. Please submit only the embeddings for the words in the eval data (more on this later).

The evaluation dataset is this one (it is also provided in the data release):

```
http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/
```

You are not allowed to optimize the vector representation learning algorithm on this dataset, but are welcome to take a look at the data. It is included for your convenience in the /data/wordsim353/ directory.

Other resources that might be useful are (a copy of Wordnet is in the data release):

```
Wordnet: https://wordnet.princeton.edu/wordnet/download/
PPDB: http://paraphrase.org/
```

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. In the simple model that is included, the context is all of the content words in a sentence. The context can also be chosen to capture syntactic and semantically relevant information.

The simple model that is included maps each word into a raw vocabulary space. 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 want to experiment with different:

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

Since this is the last assignment, I am giving you deliberately less guidance than in past assignments. I expect everybody to investigate at least three of the axes outlined above.

Code:

Please download code5.zip. The starting point for this assignment is:

```
nlp/assignments/WordSimTester.java
```

A very basic vector space model of words is included in the code, but you will want to use something more interesting – word2vec is a good starting point. You can implement things from scratch or download a package and extend it. Either approach is fine, but please describe in your write-up what you did.

The code to generate the embeddings should be submitted alongside the embeddings. You are free to use any available neural network packages or NLP tools, such as POS taggers, parsers, and so on. You do not have to build your code within the example package, and you are free to modify publicly available open source code. If you do this, you should be very clear about which modifications are yours in your submission. There is a thread on Piazza which discusses various packages.

You can run the baseline as follows (make sure your classpath includes the math library needed by the evaluation code):

```
java -classpath ../lib/commons-math3-3.5.jar::nlp/util/.*.java
      nlp/assignments/WordSimTester -embeddings <path_to_vectors>
      -wordsim ../data/wordsim353/combined.csv
      -trainingdata ../data/training-data/training-data.1m
      -trainandeval -wordnetdata ../data/wordnet/core-wordnet.txt
```

Output format of embeddings:

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

```
3 2
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 evaluation 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.

Other Tasks: Once you have familiarized yourself with the neural network package of your choice, it should be fairly straight-forward to build models for the previous assignments: Language Modeling is often discussed as an example task and atomic classification should be trivial. For POS tagging, you can consider a greedy classifier that predicts the POS tag of a word based on the local context, without Viterbi decoding. Can you push the accuracy of such a model beyond that of your trigramm HMM tagger? The answer is yes, but it might take a non-trivial amount of work. Word alignment is difficult to do with neural networks, but feel free to try.

I expect you to build neural network models for at least two of the previous assignments. In your write-up, I will look for a description of the model that you built and a discussion of its merit relative to the model that you had originally built and their performance. The models for these tasks will also learn word embeddings. How well do these perform on the word similarity task? Why is their performance better/worse than that of other models?

Leaderboard: Please submit your output for all assignments. The leaderboard will re-run the evaluations for all assignments and update your scores. I am curious to see whether you will be able to beat your personal score from before and perhaps also beat the best score that your classmates had achieved. Please discuss your results in your write-up.

Please also submit the results for your word similarity model. When you run the baseline or evaluate your word embeddings, the harness will output a file containing vector representations of words to *path_to_vectors*.reduced, as well as running the evaluation. The baseline model gets a score of 0.318 on the evaluation dataset. You should be able to beat this quite easily. For this assignment there is no separate development set, i.e. the leaderboard results will be identical to the ones you get locally.