# Statistical NLP - Assignment 5

**due Thursday November 30 - at 11:59pm by email to aparikh@cs.nyu.edu and abhinavg@nyu.edu**

In this assignment, you will explore word embeddings as well as complete 2 problem set style questions. Please note while there is a leaderboard for this assignment, there is no extra credit for achieving the highest score (There is extra credit for the most creative idea).

**Deliverables:**

- Result file in github repository.
- Changed code files and/or references to packages that you used (in the writeup).
- Writeup emailed to instructors and problem set questions. The recommended writeup length for this assignment is 2 pages. To give you some freedom to not worry about space saving, you may submit a report up to 4 pages (including any figures / charts). Reports above 4 pages will be penalized.
- Answers to problem set questions. If you use good handwriting, you are welcome to handwrite them and email us a scanned copy.

**Submission Format:** Please title your submission email **Assignment 5 submission**. You should submit a zip file (firstname_lastname_hw5.zip) containing the report in pdf format (firstname_lastname_hw5.pdf) and the zipped code directory. Write your netID and your collaborators' netID (if any) in the report.

**Part 1 (Word Embeddings)**

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 code (code5.zip) and 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 (20 pts):**

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.

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.

**Leaderboard (20 points):** 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.

For full credit on this part you must achieve a score of $0.62$. (You will be eligible for partial credit for lower scores, but the penalty is likely to be more strict since in previous years most students were able to achieve this score)

**Writeup (20 pts):** Please submit a writeup describing what you did for the above parts. The recommended length is 2 pages, detailing your explorations, results and findings for this assignment. You will be scored on two main criteria:

- The soundness / creativity of the different techniques you tried.

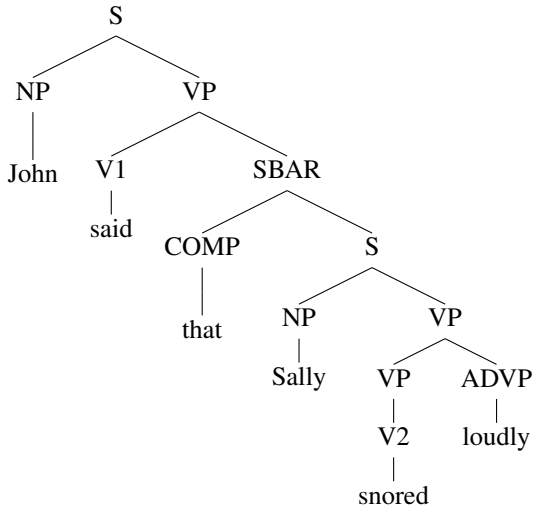
- Presenting the results of your experiments thoroughly (even if not all of them achieved positive results). As always graphs/tables are encouraged.

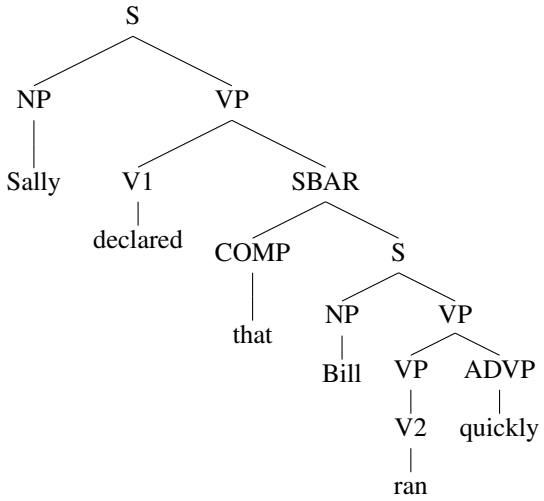
**Extra credit:** There is no extra credit for achieving the highest score on this assignment.

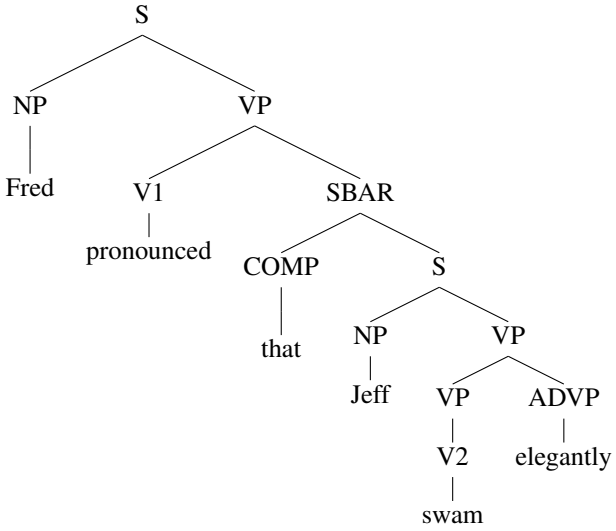
- (5 extra credit points): Have the most creative idea in the class as determined by the instructors. (If no ideas are particularly creative then no one will get extra credit. )

**Part 2: Problem Set Questions**

**Parsing (20 points) (borrowed from Michael Collins' and Regina Barzilay's Advanced Natural Language Processing class)**: Bob decides to build a treebank. He finally produces a corpus which contains the following three parse trees:




Alyssa then purchases the treebank and decides to build a PCFG and a parser using Bob's data.

- Show the PCFG that Alyssa would derive from this treebank (i.e. rules and probabilities)
- Show two parse trees for the string **Jeff pronounced that Fred snored loudly** and calculate their probabilities under the PCFG.
- Alyssa is shocked and dismayed, (see 2(b)), that **Jeff pronounced that Fred snored loudly** has two possible parses, and that one of them that Jeff is doing the pronouncing loudly has relatively high probability, in spite of it having the ADVP loudly modifiying the "higher" verb, pronounced. This type of high attachment is never seen in the corpus, so the PCFG is clearly missing something. Alyssa decides to fix the treebank, by altering some non-terminal labels in the corpus. Show one such transformation that results in a PCFG that gives zero probability to parse trees such "high" ADVP attachments. (Your solution should systematically refine some non-terminals in the treebank, in a way that slightly increases the number of non-terminals in the grammar, but allows the grammar to capture the distinction between high and low ADVP attachment to VPs.)

**Language Modeling ( 20 points)**: In addition to the Mnih and Hinton 2007 paper, this question is based on other papers. The paper authors/titles are not provided to prevent revealing the answer to the question.

Let $n$ be the order of the language model i.e.

$$P(w_i|w_1, ..., w_{i-1}) \approx P(w_i|w_{i-n+1}, ..., w_{i-1}) = P(w_i|w_{i-n+1}^{i-1})$$

For this question, the term *inference time* refers to one call to the language model i.e. computing $P(w_i|w_{i-n+1}^{i-1})$ under the language model for one choice of $w_{i-n+1}^{i-1}, w_i$.

- Characterize the inference time complexity and memory complexity of Kneser Ney language models in big-O notation. You should decide what variables are important to model when computing these two aspects.
- Consider the particular neural language model (Mnih and Hinton 2007) below (where $\mathbf{e}_w \in \mathbb{R}^d$ is a trainable vector for the word $w$, $C_j \in \mathbb{R}^{d \times d}$ is a trainable transformation matrix, and $b_w \in \mathbb{R}$ is a scalar bias.

$$\mathbf{p}_i = \sum_{j=1}^{n-1} \mathbf{e}_{w_{i-j}} C_j$$

$$\nu_i(w) = \mathbf{p}_i^T \mathbf{e}_w + b_w$$

$$P(w_i|w_{i-n+1}^{i-1}) = \frac{\exp(\nu_i(w_i))}{\sum_{v \in \mathcal{V}} \exp(\nu_i(v))}$$

  What is the memory and inference complexity of the above neural language model in big-O notation?
- One common trick for reducing the inference time complexity of neural language models is to assign each word $w$ to a hard class $c$ and model the probability as:

$$P(w_i|w_{i-n+1}^{i-1}) = \sum_c P(w_i|c, w_{i-n+1}^{i-1}) P(c|w_{i-n+1}^{i-1}) \mathbb{I}[w \in W_c]$$

  where $W_c$ denote the set of words that are part of cluster $c$ and $\mathbb{I}[w \in W_c]$ is the indicator function.
  Please parameterize each of the above two probabilities of the right hand-side ( $P(w_i|c, w_{i-n+1}^{i-1})$ and $P(c|w_{i-n+1}^{i-1})$ ) based on the neural language model in the previous section (i.e. specify what model you would build for each). Furthermore, characterize the computational advantage of this approach in big O-notation and describe how the number of classes and size of each set $W_c$ should be chosen to minimize the inference complexity.