Massachvsetts Institvte of Technology
Department of Electrical Engineering and Computer Science
6.863J/9.611J, Natural Language, Fall 2012

Assignment 1 – Reading & Response: Walking the walk, talking the talk.

Handed out: Wednesday, September 5
Due: Sunday, September 9th 6pm for in-class discussion – final version due Monday, September 10th after class.

Assignment Goals:
(1) How to be a good consultant and learn to argue both sides of the same story;
(2) Getting acquainted with the NLTK system and Python; installing these on your own computer as needed or else learning how to run NLTK on Athena;
(3) Learning about *n*-grams and the statistical/grammatical 'dialectic.'

Readings: (note, for future reference, BRT = "Berwick reading time")
(1) Jurafsky & Martin, extract from chapter on *n*-grams, ch. 4, pp. 1–13, on website (below) *or* from the text, ch. 13, pp. 83-94; pp. 114-116. [BRT: 30 min]
http://web.mit.edu/6.863/www/readings/ngrampages.pdf
(2) S. Abney, extract from "Statistics and Linguistics," in R. Bods, *Statistical Linguistics,* section 4.2. [BRT: 10 min]
http://web.mit.edu/6.863/www/readings/abney96pages.pdf
(3) N. Chomsky, extract on grammaticality from *The Logical Structure of Linguistic Theory*, 1955. [BRT: 10 min]
http://web.mit.edu/6.863/www/readings/chomsky55b.pdf

What you must do for this assignment:
1. **Read** the assigned material above. **YOU MUST Limit** your written response to two pages, singled-sided, using 11 pt. type or larger, with reasonable interline spacing and margins. **Please use plaintext or pdf and email your response to me over the weekend, by Sunday 6pm EDT**. **Responses received after this time lose half-credit (but you can still redeem yourself in the follow-up discussion). (This is a general rule.) Email** to: 6.863-graders@mit.edu **and include in the** Subject: **line of your email the text:** 6.863 Reading and Response 1
2. **Bring a hardcopy** of your written response to class on Monday so that you can refer to it, scribble changes on it, and be prepared to defend it. You can/should revise your response in light of our discussion and re-submit it by the end of the day on Monday (i.e., midnight, EDT). Any late responses will not count.
3. **The first part of your write-up** should be a **brief** answer to the **four** NLTK/python warm-up questions below, on page 4. This really is intended to make sure you can successfully run the NLTK on Athena or on your own computer. The amount of time this should take, apart from reading, is minimal. If you have any qualms or issues about NLTK, then the TA will be able to help you on Thursday.
4. **The final two pages (no more than 2!)** should be your 'consultant's report' as described below. There's a reason for the strict page limitation: We want you to learn to be concise. Before you start writing, carefully read the Style Guide as described on the course web page. We are very picky about the points listed, so you should review every paper you write to be sure that you have adhered to the commandments. Otherwise, you will drive us crazy with rage.

Now, on to the assignment itself.

You have just been hired as a consultant at Google, to assist with their new "Google Speech" project. Much like their Google Books project, Google Speech aims to collect many, many millions of examples of spoken sentences, initially just in English, transcribed into written text.[1]

You overhear Google employee #25 talking to Google employee #200 about their plans for this data:

"Look," she says, "plainly, the number of sentences one person will *ever* hear or speak in a lifetime is finite – and so therefore is the collection of all the sentences we'll ever put into Google Speech, even if it's trillions and trillions of examples. This collection, a 'corpus,' constitutes the set of 'observables' for natural language. It is this corpus that we have to model. It's just like when we observe some other natural phenomenon, like the motion of the planets. We can use lots and lots of astronomical observations, and then, once we know the position of, say, Saturn at many points in time, we can predict where it will be in the next moment, just by using our collected data. So for example, the probability of where Saturn is at time *t* is just contingent on where it was at some finite measured number of instances in the past. With sentences, we can do the same thing via the method called *n-grams*, as described by Jurafsky & Martin or in Manning & Schütze (see your reading list). An *n*-gram is just a way of predicting what the n^{th} word in a sentence will be, given the *n*–1 preceding words. And that's what we have lots of data about. We can use the probabilities of such sequences to capture what we need to know. For example, like JM says, if we see the sequence "I'd like to make a collect…" then a very likely next word is *call*, or *phone*, or *international*, but not *the.* It should be a snap. We can do fancier statistics if we need to – I know that sometimes specific, very long sequences won't ever show up in our corpus, so they'll have a frequency of zero, but we now have sophisticated ways of estimating this kind of missing data."

Employee #200 replies, "Wait a minute. Are you sure that's the right thing to study? Isn't the set of sentences that even one person can potentially produce countably infinite? How do you determine what goes on your list, and what does not? And I'm a bit troubled by your physics analogy. I don't think Newton would have appreciated it. Sure, Copernicus and Kepler collected lots and lots of data sequences, but what underlies them, *F=ma*, isn't merely a statistical approximation – it's an absolute principle. A *theory.* What you want to model – the true 'observables' – isn't what's in the 'outside world', the sequences of words or sentences, but rather the principles of the 'inside world' – the 'cognitive machinery' that produces or perceives this or that collection of sentences." (For the philosophically inclined, and even if you are not so inclined, I would urge you to read an excellent discussion of the often subtle notion of idealization in science and the experimental method by Prof. Nancy Cartwright, on the class website. Even if you just read Section 2, about 4 pages in.)

Keep these arguments in mind. You're going to write a brief report on them in a bit, but first you get some experience of your own with *n*-grams.

---

[1] Google Speech is fictional… or used to be. But when I first wrote this assignment some years back, I thought of the idea of Google having an *n*-gram corpus with so much data as a kind of joke. Little did I realize that Google would take me seriously – they have made an n-gram corpus of written text, with *n* up to 5, available to the public, using an *n*-gram viewer written by a former TA of this class while he interned at Google: http://books.google.com/ngrams/

Part 0: Setting up NLTK

NLTK is a Python toolbox for natural language processing that we'll be using all term. The first thing you should do is set up NLTK. The latest version is 2.0, and is hosted here: http://nltk.org/. **Note, however**, that we will be using version 0.9.7 on Athena.

On Athena

If you will be using Athena to do 6.863 assignments, NLTK is mostly set up for you, but you **must** run it on Athena machine running Linux (i.e., Debian currently). You can connect to athena.dialup.mit.edu with the following ssh command:

```
my_machine> ssh mylogin@athena.dialup.mit.edu
Password: <your Athena passwd here>
```

Or you can connect this way:

```
my_machine> ssh mylogin@linux.mit.edu
Password: <your Athena passwd here>
```

Alternatively, you can login from any web browser by visiting https://linerva.mit.edu/

Make sure you are using the **bash** shell to run the assignment correctly. To check which shell you are running run:

```
athena% echo $0
```

If the result for this command is not **"bash"** you should run the bash shell as follows:

```
athena% bash
```

Important: If this is your first time running bash you need to also run:

```
athena% cp /usr/prototype_user/.bash* ~
```

You can also change your default shell on Athena to bash permanently by following the instructions on http://kb.mit.edu/confluence/pages/viewpage.action?pageId=3907089 (this change will not take effect until at least four hours after you follow these instructions).

We can't install Python packages such as NLTK into the standard Python 2.6 installation on Athena. To get around this, we've created a new installation of Python in /mit/6.863/python2.5. You need to run the following commands to setup your environment paths correctly to use the new Python installation so that your 6.863 code runs correctly. You should add the following lines to your **~/.bashrc.mine** (create the file if it doesn't exist):

```
add 6.863
source /mit/6.863/fall2012/bash_environment
```

These commands will ensure that you are running the correct python and have access to the /mit/6.863 locker when you log into athena and run bash. After doing so, the next time you start bash, you should get a message welcoming you to 6.863 Fall 2012. You can verify that your python path is correctly set by running:

```
athena% which python
```

and see, **`/mit/6.863/python2.5/bin/python`** as output. If you see something instead like **`/usr/bin/python`**, 'exit' the shell, verify you have added the 2 lines to ~/.bashrc.mine and run 'bash' again. Alternatively, run the **`add 6.863; source /mit/6.863/fall2012/bash_environment`** commands after starting bash.

If, after following these steps, you continue to get the wrong output when running **`which python`** *please* email the TA immediately about it we can make sure you can run programs correctly on Athena.

To test that NLTK is working, run the following and make sure you see no errors:

```
athena% python
>>> import nltk
```

Part 1: Programming *N*-grams

We want you to answer one very simple question after running an *n*-gram program in NLTK. This programming task should take no more than a few minutes, but feel free to play with the system all you want.

We've written a Python file with helper functions that read in a training corpus, create an *n*-gram model from it, and generate sentences from that model. You can download/copy the file **`generator.py`** from http://web.mit.edu/6.863/fall2012/code/assignment1/. Once you have done that, you can load all functions in the script with this Python command (as long as **`generator.py`** is in the same directory from which you start Python):

```
>>> from generator import *
```

The main function, **`ngram_generator`**, takes in three arguments: (1) the *n* for your *n*-grams (1=unigram, 2=bigram a.k.a. Markov order 1, 3=trigram, etc.); (2) the number of sentences to generate; and (3) the training corpus to use.

The sentences are generated by calculating the probability of a word given the *n*–1 words before it, normalizing the probabilities of the possible words. The program selects a random word by producing a random number between 0 and 1 and then picking the appropriate word. It stops when it produces some end-of-sentence symbol (a period, question mark, etc.).

You can now load an NLTK corpus, or sentence database, corresponding to a 10% sample of the 49,208 sentence University of Pennsylvania Treebank. (You can find the text files this sample is loaded from in the **`corpora/treebank`** sub-directory underneath wherever the **`nltk-data`** package was installed. On Athena it's **`/mit/6.863/share/data/corpora/treebank`**. You can find an incredible assortment of other NLP corpora in its sister directories.)

There are many functions for accessing this corpus, including **`treebank.parsed_sents()`**, **`treebank.tagged_sents()`**, **`treebank.sents()`**, **`treebank.tagged_words()`**, and **`treebank.words()`**. Typing **`help(treebank)`** at a Python prompt will tell you what you can do with it (as it does for many Python objects). For now, you'll want **`treebank.words()`**, a function that outputs the contents of Treebank word by word.

An example putting this all together:

```
>>> import nltk
>>> from generator import *
>>> from nltk.corpus import treebank
>>> ngram_generator(2,3,treebank.words())
```

The program should now pause for a bit as it builds up its table of *n*-gram frequencies, and then spit out 3 sentences constructed from bigram statistics, similar to this:

```
Sentence 1:  But by bribe , some win -- copy them to make any damages in
`` bulk '' Mr. Chase , Ms. West Germany-based metals and Maytag , the
almost all the national over-the-counter trading floor debate .

Sentence 2:  That sounded here -- An index-arbitrage activity considerably
.

Sentence 3:  .
```

Of course we can invoke the function any number of times and generate different sequences each time:

```
>>> ngram_generator(2,3,treebank.words())
Sentence 1:  Quant -- is a lower offer a premium .

Sentence 2:  `` page , director at least initially drafted by Austrian
firms and 257 million .

Sentence 3:  The only a power supplies , not politics as the first half
of them installed base , 1988 .
```

You should now try this yourself and generate a few sentences of your own.

OK, what happens if we boost the *n*-gram 'window' to 3? (This is called a *trigram* model.) We are then making use of the 2 preceding words to predict the next – increasing the context we make use of. Intuitively, this should make the approximation to English 'better.' Here's an example:

```
>>> ngram_generator(3,3,treebank.words())
Sentence 1:  '' She adds that legislation curbing it would impede safety
or a researcher and professor at Harvard University 's Graduate School of
Medicine .

Sentence 2:  Beginning in 1980 , courts in several states in its
shareholder rights plan .

Sentence 3:  Despite federal disaster relief , the period .
```

This certainly seems like more natural English.

Assignment Part 1 – *N*-gram Python warm-up questions:

1. Try generating a few 4-gram and 5-gram sentences. Can you describe what happens?
2. Why is it happening?
3. How could you remedy this situation?
4. How could you ever get the system to use new words? (Please include your computer output for the 4-gram and 5-gram sentences along with your answer to this question.)

Side note: If you want to play with the generator, you can use other corpora too. Below is an example where we import the public domain 'Gutenberg' project, several pieces of literature, and select Walt Whitman's long poem, "Leaves of Grass." Try some on your own.

```
>>> from nltk.corpus import gutenberg
>>> print gutenberg.files()
('austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt', 'bible-
kjv.txt', 'blake-poems.txt', 'blake-songs.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'milton-paradise.txt',
'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-
macbeth.txt', 'whitman-leaves.txt')
>>> ngram_generator(3, 3, gutenberg.words('whitman-leaves.txt'))

Sentence 1:  But aside from the eyes for nothing , From it falls distill
' d as this day surrounded by the past struggles succeeded ?

Sentence 2:  This face of an inch nor a particle of an old Broadway stage
- driver , the scented bay - tree wide spreading its branches , Without
the farthest conceivable one coming a bit tamed , I take my place .

Sentence 3:  } The Singer in the depths the storm and buffeting gusts of
wind and falling , dying .
>>> quit()
```

Assignment Part 2: Writing your 2-page maximum consultant's report

Now consider again the arguments by the Google employees.

Your assignment: take the first employee's side and argue for her viewpoint. Then, switch sides to the second employee and argue **against** the first employee's viewpoint. **Note:** you will be called on in class to argue both sides, so be prepared!

A good consultant should be able to do a good job arguing both the positive and negative sides of a thesis. And it's quite common for one or both of the positions to be poorly formulated –people are only human. A good consultant should aim at a sympathetic interpretation no matter what the position and attempt to sharpen the proposals, since the goal isn't to score debating points with cheap, shallow, and easy shots. We suggest using 1 page for each side of the argument.

Before you write your report, you do a bit of research on your own, though, about *n*-grams and the like, in order to focus what you say, by going through the three readings given at the beginning of the assignment.

1. You read pp. 1–13 of the Jurafsky & Martin chapter 4 mentioned above, in order to appreciate the definition of an *n*-gram model and learn that an *n*-gram model is the same as a 'Markov model of degree or order (of approximation) *n*–1.'

2. You read section 4.2 from Abney's paper, and focus on the second page (the number at the bottom of the page says '20') where he attempts to define 'grammaticality' in terms of probability and Markov order of approximation as follows. For any sentence *s*, let $P_n(s)$ be the probability of *s* according to the best n^{th} order approximation (i.e., *n*+1 gram model) to English. Then a sentence is grammatical iff the limit as *n* goes to infinity of $P_n(s)$ is greater than 0, i.e.,

$$\text{grammatical}(s) \leftrightarrow \lim_{n\to\infty} P_n(s) > 0$$

In your answer, you attempt to address what Abney's statement means. Is it coherent? Does it make sense? What does it really say about sentences? Does it 'work'? Is it too broad? Too narrow?

3. You read the extract starting at section 36.1 from Chomsky's 1954-55 Ph.D. thesis on one putative difference between 'probable' and 'grammatical' and how statistics and linguistics might connect. This is the first appearance in print of the famous sentence 'Colorless green ideas sleep furiously,' but unfortunately it's not often read in context, in full. This will give you the chance to read the original. (Compare what Chomsky says in this full extract, esp. example (13) and fn. 19, to the lead-in quote to the Jurafsky and Martin that heads their chapter.) In addition, **pay attention** to how Chomsky determines, using empirical evidence! – that 'colorless green ideas sleep furiously' is actually just as grammatical as 'revolutionary new ideas appear infrequently,' even though their strict probability of occurrence might differ wildly. In fact, Chomsky *also* proposes a way to deal with the kind of issues that you discovered above, when you tried to generate 4- and 5-grams. (As it happens, this approach was 're-discovered' 50 years later as an "exciting new development" in natural language processing – without the realization that it was a re-discovery. In this course we shall see that this seems to happen over and over again, so one of the aims is to demonstrate the value of history and scholarship.) Chomsky also mentions Markov models when he uses the (then conventional) term 'n^{th} order of approximation.' In your answer, please compare Chomsky's explanation to Abney's and the *n*-gram approach. (Please note: as with anything written by Chomsky, while you might not agree with it, it is almost always true that it will be far deeper than one might at first suspect. That is true about this particular excerpt. It is not widely read, but the 'popular version' of it that is better known, in his later *Syntactic Structures* (1957), as far as I can tell has been misinterpreted over and over again, in many papers and lectures to this day, usually written by computer science people. So take heed. Read it again. It is not long, but it is deep. End of sermonette.)

With all that behind you, you should now write up your report on Google Speech.

Bonus thought to add to the discussion: consider the following quote attributed to the late Fred Jelinek (then of the IBM speech group, 1988): "*Anytime a linguist leaves the group the recognition rate goes up*." Now consider and comment on the following analogical aphorism: "*Anytime a particle moving near the speed of light leaves the group, our classical mechanics prediction rate goes up*."

(Again, for thinking about this last point more deeply, we refer you to the paper by Nancy Cartwright posted on the course web site.)