CS224N/Ling 237 Homework #1

Due: Wed, Apr 7, 2004

This assignment is an introduction to working with text corpora and standard Unix tools. It's partly to get you looking at text as a data distribution, partly practical stuff, so you can do more with standard Unix tools, and it has a tiny bit of written work thrown in.

Collaboration policy

The wording here is a bit complex, but I think it makes sense. Please ask if you have any questions, and make sure you adhere to it.

- You may talk to anybody you want about the assignments, including working through problems together in groups. Indeed, we encourage you to work in groups, and to work with different people through the quarter.
- Written questions: *However,* (i) you *must* state on your written assignment the people you discussed problems with, and (ii) you are *not allowed to take detailed notes* in any group sessions that will appear verbatim in assignment write-ups. Everybody has to turn in written homework answers that are written solely by himself/herself.
- *Programming parts/projects:* These can be done by oneself or in groups of at most 3, and people may submit a joint submission or identical material, which is assumed to be the joint work of all partners.

What you can do with corpora using just standard Unix command line tools?

There are a lot of specialist pieces of software for dealing with text corpora, and of course you can also write programs, but working with corpora invariably involves various operations of moving, munging, and reformatting text, and it's really useful to have an understanding of how much of that can be done with standard Unix tools without having to write programs. This also makes it a lot easier to do exploratory data analysis.

Unix was initially invented in the context of supporting text processing, and developed many tools that work on text files. Moreover, the original Unix philosophy concentrated on small tools that do one job well, and which can be flexibly combined, see, for instance:

`http://www.gnu.org/software/textutils/manual/textutils/html_chapter/textutils_10.html#SEC52`

There are some limitations to that philosophy: the success of Perl has shown the power of flexible text processing embedded in a complete, if ugly, programming language, and most people who do corpus-based NLP learn Perl at some point. But you can do a lot just by piping together command line tools in Unix, as we'll investigate here.

What you should do: first, you should look through and make sense of (using man pages, etc. as needed) the first 20 pages of Ken Church's Unix for Poets (attached). We put the whole tutorial online, but my personal opinion is that the first 20 pages is enough for the following reason: after that the tutorial moves into using awk (and later still C). Now, while choices among tools of overlapping functionality are a personal matter (though sometimes marked by zealotry), my personal opinion is that awk is an anachronism, and learning it is a waste of time. If you need more power than you can get from the the basic command line tools, then you're probably better off using Perl (or another programming language). Awk sits in an awkward place where it can do half of what you'd hope for from a programming language, but then you always find that it can't do the other half (or certainly not without pain). As a partial alternative to awk, the assignment uses a little more of sed than is described by Church (though again, if you end up trying to write loops in sed, you probably should be using Perl).

We'll assume that you are using the GNU version of these tools (GNU TextUtils, sed, grep, etc.). This is what you get by default on the Leland systems (i.e., providing you have /usr/pubsw/bin first in your PATH). Most of these commands are fairly common to all Unix variants, but in a couple of places (options to sort, regular expressions supported by sed, options to ptx) there can be a bit more variation in what is supported than optimal. You can find documentation for the GNU TextUtils at the URL below (or using the man pages, or info):

```
http://www.gnu.org/software/textutils/manual/textutils/html_chapter/textutils_toc.html
```

Looking at corpora

We give here a quick illustration of using sed to pick out the text part of corpora marked up in a simple form of "stand off" SGML notation (where each line is either text or an SGML element). You need a real SGML/XML parser to properly parse SGML/XML.

The first is material from the Wall Street Journal from the files:

```
/afs/ir/data/linguistic-data/Newswire/wsj/1996/WS96010*
```

which we have copied to:

```
/afs/ir/class/cs224n/data/
```

Look at one of them:

```
cd /afs/ir/class/cs224n/data/
more WS960102
```

Note that the main text of articles is enclosed in `<TEXT> . . . </TEXT>` blocks in SGML/XML style. We'll just ignore the headlines.

You can almost get out the text out with the following sed command:

```
sed -n -e '/<TEXT>/,/<\/TEXT>/p' WS960102 | more
```

By default sed does something to lines of text, and then outputs them, so you can get the contents of a file with the spelling of color changed to (British/Australian) colour by doing:

```
sed -e 's/color/colour/g' wsj7_001 | more
```

The s command does a substitution, and the g modifier says to apply it multiple times if possible. In the previous sed command, we only want text printed when we want it printed, so we use -n to suppress the default printing of lines read in and then manually ask to print with the p command text that is in ranges between a `<TEXT>` and a `</TEXT>`. This is almost what we want, but it leaves those `<TEXT>` markers in. One of several ways to get just the body without the tags is to instead delete with the d command things that are outside TEXT elements:

```
sed -e '1,/<TEXT>/d;/<\/TEXT>/,/<TEXT>/d' wsj7_001 | more
```

This deletes from line 1 through the first `<TEXT>` and then deletes ranges between an end `</TEXT>` and a `<TEXT>` inclusive (and also the tail of the file after the last `</TEXT>` – see the sed man pages. In general a sed command consists of a line number range followed by a single letter command. The semicolon separates multiple commands given to sed. The command is given via the -e flag (actually optional), with filenames then following. A line number range can either be a number (which matches just that line number), or a regular expression, which matches all lines matching the regular expression, or a range with a beginning and end, separated with a comma. The beginning and end can be line numbers or regular expressions, and depending on which is used this will pick out one or potentially many text ranges. If a line number range is omitted (as in the substitution example above), then the command is applied to all lines. The three commands we have seen are s (substitute), d (delete), and p (print). There are some others, which you can find out about in the man page, or from a helpful site such as:

```
http://www.selectorweb.com/sed_tutorial.html
```

However, these 3 commands are all you need for what we will do.

Questions

1. Tokenizing the WSJ files: We need to extract text from the body region, and then tokenize it (that is, cut it into what we'll loosely regard as "words"). In his early slides, Church gets words using the `tr` command, by deleting everything that isn't an alphabetic character. An obvious alternative would be to divide into words on white space (hint: you might do this with `tr`, by changing things to newlines, such as by `tr ' ' '\n'` and then getting rid of blank lines. Note: when Church uses `'\012'`, that's a baroque octal escape sequence for the newline character, more commonly written as `'\n'`.

    (a) Give shell command sequences that will read the WSJ text and tokenize in these two ways and write the words one per line to files.

    (b) Compare the files using diff. Give 2 different cases in which both does the wrong thing or something questionable.

Henceforth, so that we can all get the same answers, suppose we normalize the content of the first 6 files of WSJ text as follows (the stuff below is all one long command line, and assumes that you have gone to the corpus directory by doing cd /afs/ir/class/cs224n/data):

```
sed -e '1,/<TEXT>/d;/<\/TEXT>/,/<TEXT>/d ;/^|/d' WS96010* |
sed -e 's/<p>//g;s/$[0-9.,]*[0-9]/DLLR/g;s/[0-9][0-9.]*%/PRCNT/g;s/[0-9][.,0-9]*/NMBR/g' |
tr -sc 'A-Za-z' '\012' | tr 'A-Z' 'a-z' > ~/wsj.words
```

This fancier version deletes paragraph marks and lines beginning with '|' (mostly tables), maps numeric tokens onto one of PRCNT DLLR NMBR, and then lowercases all words (including these!)

2. Word tokens, types, and frequencies

   (a) Under the definition of 'word' that results from the commands given above, how many word tokens are there in this corpus, and how many word types are there? (See the wc and uniq commands.)

   (b) Make a dictionary of the words used in the corpus (using sort and uniq).

   (c) Now make a dictionary with frequencies of each word. You want to use the `uniq -c` command for this on a sorted list of words.

      i. Before peeking, choose 3 everyday words (like "fork"), 3 less common words (like "persuasion") and 3 rare words (like "oxymoron"). Find out and report how often each occurs in the corpus.

      ii. What are the 15 most frequent words?

      iii. What are the 10 most common word bigrams and 10 most common word trigrams in the corpus (see Church's Bigrams slide)? – give lists. Give the commands you used to generate the WSJ trigrams starting from the wsj.words file above.

3. Zipf's Law in practice: Finally, let's validate Zipf's Law on these texts (see M&S, pp. 23–29 for Zipf's Law). With uniq one can get counts of words. You can keep just the counts with the cut command: "cut -f 1". One can then sort these counts and then one can use uniq again to get count-counts. Finally, one can number them with ranks using the nl (number lines) command. (Note: the `sort -nr` command will do reversed numerical sorting.)

   (a) Give the command(s) that will produce a file with a column of ranks and a column of frequencies for one of these corpora. For instance the output should look something like:

      1 3842
      2 3116
      3 2310
      ...

   (b) For doing the rest of this, one could use a Unix graphing/math package if you know one (Matlab, Splus, GnuPlot, etc.), but I think the easiest thing to do is work with Excel!

      For Excel: Import the rank-frequency file. There are various ways to do that: either with a text file, and importing it with File|Open, or using cut and paste, and then the Data|Text to columns... option, which works really well on this kind of columnar data. You will then use Excel's chart maker (the bar chart toolbar icon, which looks a bit like a bookshelf). Make new columns that are the log of the value of those columns, and then graph the log-log values as an XY scatterplot. Include a plot in your assignment. (Alternatively, to get a log scaled graph, make a normal graph, then right click on an axis, choose Format Axis, and then select the Scale tab and check the Logarithmic scale checkbox. But I don't think that will allow you to do the next question as easily.)

(c) Right click on a data point, and do "Add Trendline". Ask for a linear regression model, and for the equation to be printed. What is the slope for the corpus? Does the data show the kind of curvature suggested by Mandelbrot's law (as discussed in the text)?

4. Zipf's Law more theoretically: Is Zipf's Law a profound discovery about natural language text, or a very general phenomenon that arises in ranked data?

   Let S be an alphabet of N symbols. Let X be a uniform random process that produces each symbol of S, plus a space symbol, with uniform probability $1/(N+1)$. X will thus generate text where words w over S have a certain probability $p(w)$ of occurring. If we rank words according to p, however, there will be huge stretches where the ranking r is arbitrary, since all words of equal length have equal probability. Let $r'(w)$ be the average rank that a word of length $|w|$ would be assigned according to r.

   (a) Show that, in the limit (as word length ℓ increases), "Zipf's law" holds for text randomly generated according to X. In particular, show that in the limit there is a linear relationship between $\log p(w)$ and $\log r'(w)$.

   (b) What is the slope of the line in the limit?