Class 1: Unix™ for Poets

[This page is adapted from the wonderful Unix™ for Poets by Ken Church which introduced a generation of researchers to NLP. I have adjusted it for use with languages which may use multibyte encodings such as UTF-8 and as such are not supported by standard utilities like `tr`. To read the original, please click here (https://www.cs.upc.edu/~padro/Unixforpoets.pdf) -FMT]

Text is available like never before. Data collection efforts such as the Association for Computational Linguistics' Data Collection Initiative (ACL/DCI), the Consortium for Lexical Research (CLR), the European Corpus Initiative (ECI), ICAME, the British National Corpus (BNC), the Linguistic Data Consortium (LDC), Electronic Dictionary Research (EDR) and many others have done a wonderful job in acquiring and distributing dictionaries and corpora. In addition, there are vast quantities of so-called Information Super Highway Roadkill: email, bboards, faxes. We now has access to billions and billions of words, and even more pixels.

What can we do with it all? Now that data collection efforts have done such a wonderful service to the community, many researchers have more data than they know what to do with. Electronic bboards are beginning to fill up with requests for word frequency counts, ngram statistics, and so on. Many researchers believe that they don't have sufficient computing resources to do these things for themselves. Over the years, I've spent a fair bit of time designing and coding a set of fancy corpus tools for very large corpora (eg, billions of words), but for a mere million words or so, it really isn't worth the effort. You can almost certainly do it yourself, even on a modest PC. People used to do these kinds of calculations on a PDP-11, which is much more modest in almost every respect than whatever computing resources you are currently using.

I wouldn't bring out the big guns (fancy machines, fancy algorithms, data collection committees, bigtime favours) unless you have a lot of text (e.g., hundreds of million words or more), or you are trying to count really long ngrams (e.g., 50-grams). This chapter will describe a set of simple Unix-based tools that should be more than adequate for counting trigrams on a corpus the size of the Brown Corpus. I'd recommend that you do it yourself for basically the same reason that home repair stores like DIY and Home Depot are as popular as they are. You can always hire a pro to fix your home for you, but a lot of people find that it is better not to, unless they are trying to do something moderately hard. Hamming used to say it is much better to solve the right problem naively than the

wrong problem expertly.

I am very much a believer in teaching by examples. George Miller (personal communication) has observed that dictionary definitions are often not as helpful as example sentences. Definitions make a lot of sense if you already basically know what the word means, but they can be hard going if you have never seen the word before. Following this spirit, this chapter will focus on examples and avoid definitions whenever possible. In some cases, I will deliberately use new options and even new commands without defining them first. The reader is encouraged to try the examples themselves, and when all else fails consult the documentation. (But hopefully, that shouldn't be necessary too often, since we all know how boring the documentation can be.)

We will show how to solve the following exercises using only very simple utilities.

1. Collect a plain text corpus from Wikipedia
2. Count words in a text
3. Sort a list of words in various ways
    - numerical order
    - dictionary order
    - ''rhyming'' order
4. Compute ngram statistics
5. Make a Concordance

The code fragments in this chapter were developed and tested on a Sun computer running Berkeley Unix. The code ought to work on more or less as is in any Unix system, and even in many PC environments, running various compatibility packages such as the MKS toolkit.

0. Collect a plain-text corpus from Wikipedia

You will need the file called `-pages-articles.xml.bz2`. You can find it on the WikiMedia dumps site (http://dumps.wikimedia.org). Go to `Database backup dumps` and choose an XXwiki or YYYwiki, where XX is the 2-letter language code or YYY is the 3-letter language code. You can use `wget` or `curl` to download it. Or alternatively download it through your browser.

To extract the text, you can use WikiExtractor (https://github.com/apertium/WikiExtractor). Then use the segmenters to segment the raw text output into sentences.

1. Count words in a text

The problem is to input a text file, say Wikipedia (a good place to start), and output a list of words in the file along with their frequency counts. The algorithm consists of three steps:

- Tokenise the text into a sequence of words (`sed`),
- Sort the words (`sort -r`), and
- Count duplicates (`uniq -c`).

The algorithm can be implemented in just three lines of Unix code:

```
$ sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt |
sort -r |
uniq -c > wiki.hist
```

NOTE: When the `$` symbol appears at the beginning of a line, it means that this line is a command that you should write into your terminal, do not type the `$` symbol.

WARNING: You should copy/paste both `І` and `І`. If you are not using Avar or another language with palochka, then you should replace `а-яА-ЯІІ` with your alphabetic range, e.g. `a-zA-Z` for scripts based on the Latin alphabet.

This program produces the following output:

```
      5 Яшар
      1 яшавалда
     17 яшав
      1 ячуна
      1 Ячун
      6 ячун
      1 ячула
      1 ячинчІого
      1 ячинчІей
      2 ячине
...
```

WARNING: In some Unix™ environments you should use `gsed` instead of `sed`. If you are getting strange output try this. If you are working in an environment where the only `sed` is a broken `sed`,

then you can try replacing `sed 's/[^а-яА-ЯІІ]\+/\n/g'` with `sed 's/ /\'$'\n/g'`.

The less than symbol "<" indicates that the input should be taken from the file named "wiki.txt", and the greater than symbol ">" indicates that the output should be redirected to a file named "wiki.hist". By default, input is read from `stdin` (standard input) and written to `stdout` (standard output). Standard input is usually the keyboard and standard output is usually the active window.

The pipe symbol "|" is used to connect the output of one program to the input of the next. In this case, `sed` outputs a sequence of tokens (words) which are then piped into `sort`. Sort outputs a sequence of sorted tokens which are then piped into `uniq`, which counts up the runs, producing the desired result.

We can understand this program better by breaking it up into smaller pieces. Lets start by looking at the beginning of the input file. The Unix program `sed` can be used as follows to show the first five lines of the wiki file:

```
$ sed 5q < wiki.txt

География:

Гъизляр:
Гъизляр яги БагІаршагьар () — Туркияб мацІалдаса магІарул мацІалде буссине
гьабуни "Ясшагьар". Цебе заманалда гьениб букІун буго кІудияб лагъзал
ричулеб базар. Гьениса ричун росулел (яги рикъулел) рукІун руго руччабиги,
хІалтІухъабиги.
```

In the same way, we can use the same command, `sed`, to verify that the first few tokens generated by `sed` do indeed correspond to the first few words in the wiki file.

```
$ sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt | sed 5q

География



Гъизляр
```

Similarly, we can verify that the output of the `sort` step produces a sequence of (not necessarily distinct) tokens in lexicographic order.

```
$ sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt |
sort -r |
sed 10q

Яшар
Яшар
Яшар
Яшар
Яшар
яшавалда
яшав
яшав
яшав
яшав
```

Finally, the `uniq` step counts up the repeated tokens.

```
$ sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt |
sort -r |
uniq -c |
sed 5q

      5 Яшар
      1 яшавалда
     17 яшав
      1 ячуна
      1 Ячун
```

2. More counting exercises

This section will show three variants of the counting program above to illustrate just how easy it is to count a wide variety of (possibly) useful things. The details of the examples aren't all that important. The point is to realise that pipelines are so powerful that they can be easily extended to count words (by almost any definition one can imagine), ngrams, and much more.

The examples in this section will discuss various (weird) definitions of what is a "word". Some

students get distracted at this point by these weird definitions and lose sight of the point — that pipelines make it relatively easy to mix and match a small number of simple programs in very powerful ways. But even these students usually come around when they see how these very same techiques can be used to count ngrams, which is really nothing other than yet another weird definition of what is a word/token.

The first of the three examples shows that a straightforward one-line modification to the counting program can be used to merge the counts for upper and lower case. The first line in the new program below collapses the case distinction by translating lower case to upper case:

```
$ uconv -x upper < wiki.txt |
sed 's/[^А-Яа-яI]\+/\n/g' |
sort -r |
uniq -c |
sed 5q
```

Small changes to the tokenising rule are easy to make, and can have a dramatic impact. The second example shows that we can count vowel sequences instead of words by changing the tokenising rule (the `sed` line) to emit sequences of vowels rather than sequences of alphabetic characters.

```
$ uconv -x upper < wiki.txt |
sed 's/[^АЭИОУЯЕЫЁЮ]\+/\n/g' |
sort -r |
uniq -c
```

```
$ uconv -x upper < wiki.txt |
sed 's/[^БВГДЖЗЙКЛМНПРСТФХЦЧШЩЪЬ]\+/\n/g' |
sort -r |
uniq -c
```

These three examples are intended to show how easy it is to change the definition of what counts as a word. Sometimes you want to distinguish between upper and lower case, and sometimes you don't. Sometimes you want to collapse morphological variants (does hostage = hostages). Different languages use different character sets. Sometimes I get email from Sweden, for example, where "y" is a vowel. The tokeniser depends on what you are trying to do. The same basic counting program can be used to count a variety of different things, depending on how you implement the definition of thing (=token).

You can find the documentation for the `sort` command (and many other commands as well) by saying

```
$ man sort
```

If you want to see the document for some other command, simply replace the `sort` with the name of that other command.

The man page for `sed` explains that it inputs a stream and applies various transformations depending on an expression, in

```
sed 's/a/б/g'
```

`sed` changes all instances of 'a' to 'б'. The 's' means match a pattern in the stream and replace it with something else. The pattern and replacement are put between '/' characters, and the final 'g' means do this to all instances of the pattern you see in the stream. The `sed` command,

```
sed 's/[^А-Яа-яІ]\+/\n/g'
```

translates any non-alphabetic character to newline. Non-alphabetic characters include puctuation, white space, control characters, etc. It is easier to refer to non-alphabetic characters by referring to their complement because many of them (like newline) are hard to refer to (without knowing the context).

3. Sort

The sorting step can also be modified in a variety of different ways. The man page for `sort` describes a number of options or flags such as:

Example	Explanation
`sort -d`	dictionary order
`sort -f`	fold case
`sort -n`	numeric order
`sort -r`	reverse order
`sort -nr`	reverse numeric order

`sort -u`	remove duplicates
`sort +1`	start with field 1 (starting from 0)
`sort +0.50`	start with 50th character
`sort +1.5`	start with 5th character of field 1

These options can be used in straightforward ways to solve the following exercises:

1. Sort the words in Wikipedia by frequency `sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt | sort | uniq -c | sort -nr`
2. Sort them by folding case.
3. Sort them by ‘’rhyming” order.

By ‘’rhyming” order, we mean that the words should be sorted from the right rather than the left, as illustrated below:

```
Гьезда
мацIалда
округалда
заманалда
Шагьаралда
севералда
къиблаялда
Федерациялда
бакъбаккуда
Бакъда
МахIачхъала
...
```

‘’севералда” comes before ‘’къиблаялда” because ‘’адларевес” (‘’севералда” spelled backwards) comes before ‘’адляалбиък” (‘’къиблаялда” spelled backwords) in lexicographic order. Rhyming dictionaries are often used to help poets (and linguists who are interested in suffixing morphology).

Hint: There is a Unix command `rev`, which reverses the letters on a line:

```
$ echo "МагІарул мацІ" | rev
Іцам лураІгаМ

$ echo "МагІарул мацІ" | rev  | rev
МагІарул мацІ
```

Thus far, we have seen how Unix commands such as `sort`, `uniq`, `sed` and `rev` can be combined into pipelines with the "|","<," and ">" operators in simple yet powerful ways. All of the examples were based on counting words in a text. The flexibility of the pipeline approach was illustrated by showing how easy it was to modify the counting program to

- tokenise by vowel, merge upper and lower case
- sort by frequency, rhyming order

4. Bigrams

The same basic counting program can be modified to count bigrams (pairs of words), using the algorithm:

1. tokenise by word
2. print $word_i$ and $word_{i+1}$ on the same line
3. count

The second step makes use of two new Unix commands, `tail` and `paste`. Tail is usually used to output the last few lines of a file, but it can be used to output all but the first few lines with the "-n +2" option. The following code uses tail in this way to generate the files genesis.words and genesis.nextwords which correspond to $word_i$ and $word_{i+1}$.

```
sed 's/[^а-яА-ЯІІ]\+/\n/g' < wiki.txt > wiki.words
tail -n +2 wiki.words > wiki.nextwords
```

Paste takes two files and prints them side by side on the same line. Pasting `wiki.words` and `wiki.nextwords` together produces the appropriate input for the counting step.

```
paste wiki.words wiki.nextwords


...
гьениб  букIун
букIун  буго
буго    кIудияб
кIудияб лагъзал
лагъзал ричулеб
ричулеб базар
базар   Гьениса
Гьениса ричун
ричун   росулел
...
```

Combining the pieces, we end up with the following three line program for counting bigrams:

```
sed 's/[^а-яА-ЯІІ]\+/\n/g' | grep -v '^$' > wiki.words
tail -n +2 wiki.words > wiki.nextwords
paste wiki.words wiki.nextwords | sort | uniq -c > wiki.bigrams
```

The ten most frequent bigrams in the Avar Wikipedia are:

```
sort -nr < wiki.bigrams | sed 10q


    236 латиназул   мацIалда
    174 жамагIаталде    гъорлъе
    140 мацIалда    хъизан
    122 хъизан  гIалхул
     94 гъорлъе уна
     90 уна жибго
     89 гъорлъе унеб
     86 лага    черх
     86 гIадамасул  лага
     85 мацIалда    гIадамасул
```

I have presented this material in quite a number of tutorials over the years. The students almost always come up with a big ‘’aha” reaction at this point. Counting words seems easy enough, but for

some reason, students are almost always pleasantly surprised to discover that counting ngrams (bigrams, trigrams, 50-grams) is just as easy as counting words. It is just a matter of how you tokenise the text. We can tokenise the text into words or into ngrams; it makes little difference as far as the counting step is concerned.

5. Shell scripts

Suppose that you found that you were often computing bigrams of different things, and you found it inconvenient to keep typing the same five lines over and over. If you put the following into a file called `bigram.sh`,

```
sed 's/[^a-яA-ЯII]\+/\n/g' | grep -v '^$' > $$words
tail -n +2 $$words > $$nextwords
paste $$words $$nextwords |
sort | uniq -c
# remove the temporary files
rm $$words $$nextwords
```

then you could count bigrams with a single line:

```
$ sh bigram.sh < wiki.txt > wiki.bigrams
```

The shell script introduced several new symbols. The ”#” symbol is used for comments. The ”*"syntaxencodesalongnumber*(*theprocessid*)*intothenamesofthetemporaryfiles.Itisagoodideatousethe"* ” syntax in this way so that two users (or two processes) can run the shell script at the same time, and there won’t be any confusion about which temporary file belongs to which process. (If you don’t know what a process is, don’t worry about it. A process is a job, an instance of a program that is assigned resources by the scheduler, the time sharing system.) Put more simply, `$$hargle` is a temporary file created with a name unique to the currently running process.

6. grep: An Example of a Filter

After completing the bigram exercise, you will have discovered that ‘’латиназул мацIалда” and ‘’жамагIаталде гьорлъе” are the two most frequent bigrams in the Avar Wikipedia. Suppose that you wanted to count bigrams separately for verses that contain just the phrase ‘’латиназул

мацIалда.”

Grep (**g**eneral **r**egular **e**xpression **p**attern matcher) can be used as follows to extract lines containing ‘’латиназул мацIалда”.

```
$ grep 'латиназул мацIалда' wiki.txt | sed 5q

Чараналъул гъветI (латиназул мацIалда "Populus nigra" var. "piramidalis ")
Хъарчигъа (латиназул мацIалда "Accipiter gentilis"), хIинчI.
Маймалак ялъуни Маймун (латиназул мацIалда "Primates ex Homo") — гIалхул
кIудияб ялъуни гьитIинаб хIайван.
Антилопа (латиназул мацIалда "Antilopinae" etc. sufamilies) — хIайван.
TIавус (латиназул мацIалда "Pavo") — гIалхул ва рукъалъул хIинчI.
```

Grep can be combined in straightforward ways with the bigram shell script to count bigrams for lines matching any regular expression including ‘’латиназул мацIалда” and ‘’жамагIаталде гъорлъе.”

```
$ grep 'латиназул мацIалда' wiki.txt |
sh bigram.sh | sort -nr | sed 5q

    236 латиназул   мацIалда
    140 мацIалда    хъизан
    122 хъизан  гIалхул
     85 мацIалда    гIадамасул
     85 лага    черх
```

```
$ grep 'жамагIаталде гъорлъе' wiki.txt |
sh bigram.sh | sort -nr | sed 5q

    166 жамагIаталде    гъорлъе
     85 гъорлъе унеб
     81 уна жибго
     81 гъорлъе уна
     63 росулъ  гIумру
```

The syntax for regular expressions can be fairly elaborate. The simplest case is like the ‘’the land of” example above, where we are looking for lines containing a string. In more elaborate cases, matches can be anchored at the beginning or end of lines by using the ‘’^” and ”$” symbols.

Example	Explanation
`grep gh`	find lines containing ‘’gh”
`grep '^con'`	find lines starting with ‘’con”
`grep 'ing$'`	find lines ending with ‘’ing”

With the `-v` option, instead of printing the lines that match, grep prints lines that do not match. In other words, matching lines are filtered out or deleted from the output.

The `-c` options counts the matches instead of printing them.

Case distinctions are ignored with the `-i` option, so that

```
grep '[аэиоуАЭИОУ]'
```

is equivalent to

```
grep -i '[аэиоу]'
```

Grep also allows ranges of characters:

Example	Explanation
`grep '[А-Я]'`	lines with an uppercase character
`grep '^[А-Я]'`	lines starting with an uppercase character
`grep '[А-Я]$'`	lines ending with an uppercase character
`grep '^[А-Я]\+$'`	lines with all uppercase character
`grep '[аэиоуяеыёюАЭИОУЯЕЫЁЮ]'`	lines with a vowel
`grep '^[аэиоуяеыёюАЭИОУЯЕЫЁЮ]'`	lines starting with a vowel
`grep '[аэиоуяеыёюАЭИОУЯЕЫЁЮ]$'`	lines ending with a vowel

Warning: the ”^” (*circumflex*) can be confusing. Inside square brackets as in ”[^a-яА-Я],” it no longer refers to the beginning of the line, but rather, it complements the set of characters. Thus, ”[^a-яА-Я]” denotes any non-alphabetic character.

Regular expressions can be combined recursively in elaborate ways. For example,

Example	Explanation
`grep -i '[аэиоуяеыёю].*[аэиоуяеыёю]'`	lines with two or more vowels
`grep -i '^[^аэиоуяеыёю]*[аэиоуяеыёю][^аэиоуяеыёю]*$'`	lines with exactly one vowel

A quick summary of the syntax of regular expressions is given in the table below.

Example	Explanation
`ц`	match the letter 'ц'
`[а-я]`	match any lowercase letter
`[А-Я]`	match any uppercase letter
`[0-9]`	match any digit
`[0123456789]`	match any digit
`[аэиоуяеыёюАЭИОУЯЕЫЁЮ]`	match any vowel
`[^аэиоуяеыёюАЭИОУЯЕЫЁЮ]`	match any character that is not a vowel
`.`	match any character
`^`	beginning of line
`$`	end of line
`*pattern**`	match zero or more instances of *pattern*
`*pattern*\+`	match one or more instances of *pattern*

TIP: If you are using GNU `grep` then you can run it with `grep --colour` to get the matched pattern coloured in your terminal. — if you aren't using GNU `grep` you should be!

Exercises with `grep`

- How many uppercase words are there in the Avar Wikipedia? Lowercase? Hint: `wc -l` or `grep -c`
- How many 4-letter words?
- Are there any words with no vowels?

- Find "1-syllable" words (words with exactly one vowel)
- Find "2-syllable" words (words with exactly two vowels)

7. `sed` (stream editor)

We have been using

```
sed 5q < wiki.txt
```

to print the first 5 lines. Actually, this means quit after the fifth line. We could have also quit after the first instance of a regular expression

```
sed '/руго/q' < wiki.txt
```

As we saw before, `sed` also used to substitute regions matching a regular expression with a second string. For example:

```
sed 's/\([рбв]уго\|йиго\)/[COP]/g' < wiki.txt
```

will replace all instances of *руго*, *буго*, *вуго* or *йиго* with '[COP]'. The first argument can be a regular expression. For example, as part of a simple morphology program, we might insert a hash symbol before words ending with "да."

```
sed 's/да[^а-я]/#&/g' < wiki.txt
```

The `&` symbol here refers to the pattern matched, so here it would match да followed by anything that wasn't a lowercase alphabetic character and output a hash symbol followed by the matched pattern.

Exercises with `sed`

- Count word initial consonant sequences: tokenise by word, delete the vowel and the rest of the word, and count
- Count word final consonant sequences