Peggy Renwick

STEPS TO DATA MINING

Setting up for demos

Open a Terminal/shell window
OS X: F4 → type Terminal → Return

cd to your workshop directory, e.g.

```
cd /Users/peggy/workshop
```

Copy in two big files from the BAAP USB drive:

```
cp /Volumes/USB\ DISK/
AudioBNC_sample_for_BAAP/thephonebook.txt .

cp /Volumes/USB\ DISK/
AudioBNC_sample_for_BAAP/wordtriplets.txt .
```

Make the python script executable:

```
chmod +x check_wordpair_BAAP.py
```

If you're using a Mac, follow these commands to prepare your computer to run these demos (and any other analysis you conduct with the Audio BNC sample). All examples use your Workshop directory (e.g. /Users/peggy/workshop) as the working directory unless otherwise noted with a cd command. Additionally, each command should be typed on a single line on your computer and followed by pressing the Enter/Return key; and capitalization of letters in filenames & filepaths should be followed.

Example: Searching an index

Step 1: Find a techno-buddy, if needed

Step 2: Open a Terminal/shell window
OSX: F4 → type Terminal

cd to your workshop directory if needed, e.g.

```
cd /Users/peggy/workshop
```

Type **ls** (& return) to see directory contents

View a file's beginning:

```
head thephonebook.txt
head -20 thephonebook.txt
```

Once forced alignment has been done, we are left with a set of .wav files and corresponding TextGrids. Next we want to learn what is in the TextGrids; how many tokens of interest there are; and where these tokens are in the audio. Depending what you're looking for, you can make this easier by compiling all the TextGrids into a single file, in which each row has the information for a different segment, or word. We call this an index. In our studies we have been interested in *pairs* of words, so we made an index of all pairs of words, of the form A B FILENAME START-A END-B / B C FILENAME START-B END-C, etc., for the whole corpus. We then search the index file for combinations of words that interest us, and have direct access to the audio information specific to those words.

Here is an example of how to search the index (thephonebook.txt).

Commands to know:

cd – change directory

ls – list directory contents

head – view first 10 (or X) lines

Searching for phones

Search for all instances of “M”:

```
grep \"M\" thephonebook.txt
```

Look at only the first few:

```
grep \"M\" thephonebook.txt | head
```

How many are there?

```
grep \"M\" thephonebook.txt | wc -l
```

We can search for all the instances (tokens) of a particular phoneme, in thephonebook.txt. If we do this (grep) and just hit “enter”, the Terminal prints out every line: We see that there are many instances of it, but it’s not useful for our research. Just to see what we’re getting, we can search for e.g. “M” and view only the first 10 examples (head). If we want to count how many M’s there are, we can count the number of lines (wc -l) in thephonebook.txt that have the sequence “M”.

Commands to know:

grep – searches for a regular expression. Here, we’re looking for the string “M” (with double quotes), but we have to *escape* those characters with the slash \, giving the syntax \”M\”.

Pipe (|) – this passes the output of one command into the input of the next one, allowing you to chain actions together.

wc -l: This command COUNTS things. In this case by specifying “-l”, we’re asking it to count the lines it receives as input.

Comparing phone counts

Search for multiple phones simultaneously:

```
grep \"[LMNP]\" thephonebook.txt |
  awk '{ print $1 }' |
  sort |
  uniq -c |
  sort -nr
```

We can also see how many unique items there are of a particular phone, and one good reason to do this is to compare the relative frequencies of different items of interest. So let's compare the relative frequencies of L, M, N, and P.

To do this, we're going to chain together several commands. The commands appear on multiple lines on this slide, but you should type them all on one line at your Terminal prompt. First, search for the phones with grep, and pipe the output. The output of our grep feeds into this awk command. Awk is a wonderful language for data-wrangling, which processes the input one row at a time. Here we are asking awk to print the first field ($1) of each row, where the fields are separated by spaces ($0 refers to the whole line; $1 to the first field, $2 to the second, etc.). That means awk is printing the "phone" field of each row. Now, pipe the output of the awk command to "sort", which puts all the rows in alphabetical order; and then to the uniq –c command, which is going to count the number of unique tokens of each type in our list. Finally, once it's done the counting, we sort the counts in numerical order, and reverse it (-nr), to see the biggest number at the top. Now hit return.

This command may take a minute to compute; the sort command is somewhat labor-intensive. The output of these commands appear on the next slide.

Relative frequencies of phones

```
grep \"[LMNP]\" thephonebook.txt | awk
  '{ print $1 }' | sort | uniq -c | sort -nr
```

Output:

```
256177 "N"
125880 "L"
110016 "M"
62614 "P"
```

Occurrence of homorganic [mp] limited by frequency of [p]

N is four times more frequent than P.

Can we have a balanced data sample?

UNIVERSITY OF OXFORD

Not with spontaneous speech!!

These numbers demonstrate the wide range of relative frequencies across different phones in the Audio BNC. They're sorted from most to least frequent: There are more than a ¼ million N's, half that number of L's, 110K M's, but only 62K tokens of P. In other words, N is roughly 4x more frequent than P. This means that if we wanted to gather data comparing the acoustics of N and P, for some reason, having a balanced data sample is going to be very difficult. In fact, it's nearly impossible to have a balanced data sample from a spontaneous speech corpus like this one. Imagine if we just wanted to look for words with the homorganic cluster [mp]: The number of tokens we can get is already going to be limited by the lower relative frequency of [p] with respect to [m]. Now imagine what happens when we want to find specific words, or pairs of words, with the kinds of phonological characteristics that we typically specify in a linguistic experiment: These natural consequences of Zipf's law almost immediately limit what we can do even with Big Data.

Searching & listening

Generate the input:

```
grep "\"LADIES\" \"AND\" \"GENTLEMEN\""
  wordtriplets.txt > ladies_gentlemen.txt
```

Run the listening script:

```
./check_wordpair_BAAP.py ladies_gentlemen.txt .1
```

Type **y** if you hear “ladies and gentlemen”, **n** if you don’t

Move the results to a subfolder:

```
mkdir LADIES_AND_GENTLEMEN
mv LADIES*.wav LADIES_AND_GENTLEMEN
```

Let’s combine searching for a phrase, with checking whether the results are good. Now we’re going to search for whole words, and in fact we can search for triplets of words, with a special index we’ve given you called wordtriplets.txt. (If you want to see what it looks like, you can use the “head” command to view the beginning of it.)

The first command on this slide generates the input for our listening script: It searches for all tokens of “ladies and gentlemen”, and writes them to a file called “ladies_gentlemen.txt”.

Next we run the listening script, by typing the second command. The last bit, the “.1”, indicates that we want the script to play 100ms of padding both before and after the word pair. If the alignment is off by just a little bit, this will really help us have a higher success rate. After the second command, follow the instructions in your Terminal window, listening to each clip and typing “y” for tokens where you hear “ladies and gentlemen”, and “n” for tokens you want to ignore.

If your .py script refuses to run: Open the .py script using a text editor. Search for the text string “/Volumes/USB\ DISK/AudioBNC_sample_for_BAAP/wavs/”. It is possible that this is not where the audio files are not actually located on your system. If that is the case, change this file path and try running the script again. Otherwise, make sure that the file is executable (see chmod +x command earlier in the slides).

Once you’re done listening, use the command line to move the .wav files to a new directory. First, make a subfolder (mkdir command); then, give the “move” command for the .wav files beginning with “LADIES”. Using the asterisk means you’ll move all the .wav files whose names start with “LADIES”.

Scripted workflows

`awk`: text-based data wrangling

`sox`: command-line audio processing

`wget`: downloads content from a web URL

`Praat`: scriptable software for phonetic analysis

`esps`: command-line phonetic analysis

`R`: complex data transformations, scripted plotting (also statistical analysis)

…and many more!

We've just taken you from "zero" to "dataset" in less than 20 minutes. So if that doesn't convince you of the value of using scripts and command line tools to speed up your work, I don't know what will. So to encourage you even more, let me just mention a few tools that we have used to speed up our processes. We'll use some of these in the remainder of the workshop.

Example: a Praat script

Open Praat
Praat → Open Praat script…

Select `extract-f0-BAAP.praat` → Open

- Reads in your .wav files
- Creates a pitch object for each
- Gets f0 measurements every 10 ms
- Writes the results to a .csv file

Get directory: in Terminal, **`cd`** to the `LADIES_AND_GENTLEMEN` directory.
Type **`pwd`** and paste the result into Praat's Directory window.

We have provided a Praat script that will sample f0 in the tokens of "ladies and gentlemen" that you've just created. Follow the instructions on these slides to run it.

Running the Praat script

Get directory: in Terminal, **`cd`** to the `LADIES_AND_GENTLEMEN` directory.
Type **`pwd`** and copy the result.

In Praat script: Run → Run

location: `/Users/peggy/workshop/LADIES_AND_GENTLEMEN/`
paste in your working directory

output: `/Users/peggy/workshop/ladies_f0.csv`
specify an output .csv file

The output file from this should be ladies_f0.csv (or whatever you name the output file). It should contain two columns: a filename column, and an f0 column of numbers and "—undefined—" values.

Example: analyzing f0 in “FIFTEEN”

The next set of slides demonstrate an analysis of f0 in tokens of “FIFTEEN” in your Audio BNC sample.

Rather than using the .py script to generate tokens, we will use the `wget` program to download them from the Audio BNC servers. You need an internet connection for this. Otherwise, use the `Good.zip` file provided at the workshop.

Analyzing FIFTEEN

Search for tokens & look at output

In your `workshop` directory:

```
grep "^\"FIFTEEN\"" wordtriplets.txt |
head
```

wget

Search for tokens, write to file for wget

```
grep "^\"FIFTEEN\"" wordtriplets.txt |
tr '_' ' ' |
awk '{print "http://bnc.phon.ox.ac.uk/data/"
$4 ".wav?t="$8","$9 }' > wgetlist
```

The grep command finds all instances of “FIFTEEN” in wordtriplets.txt.

The tr command replaces the underscore with a space, allowing awk (in the next command) to write fields $4, $8, and $9 into a set of web addresses that will direct to the Audio BNC server hosted by the Oxford Phonetics Laboratory. These web addresses are written to a text file called wgetlist.

Download Audio BNC clips

Make directory for `wget` output:

```
mkdir FIFTEEN_wavs
```

Run wget:

```
wget -i wgetlist --directory-prefix=FIFTEEN_wavs/
```

Make a directory into which your .wav clips will go. (This is a good idea because there are 172 of them.)

Run the wget program, using the wgetlist as input, and specifying that the files should be downloaded to the FIFTEEN_wavs folder.

Rename files

```
cd FIFTEEN_wavs
```

Make a renaming script:

```
ls | awk '{ print "sox", "FIFTEEN_wavs/"$0,
  "FIFTEEN_clips/FIFTEEN"NR".wav" }' > ../rename_wavs

cd ../

mkdir FIFTEEN_clips
```

Make script executable: `chmod +x rename_wavs`

Run script: `./rename_wavs`

Go into the FIFTEEN_wavs directory and rename the files, because their names are currently awkward and long.

This set of commands takes the list (ls) as input, and takes advantage of the audio manipulation program sox to rename them, placing them into a new subdirectory that you have created, called FIFTEEN_clips. Once you run the rename_wavs script, you can check that it's run correctly by cd'ing to the FIFTEEN_clips directory and looking at the file list (ls).

Extracting f0

Open Praat

Praat → Open Praat script…

Select `extract-f0-BAAP.praat` → Open

In Praat script: Run → Run

location: `/Users/peggy/workshop/FIFTEEN_clips/`

output: `/Users/peggy/workshop/fifteen_f0.csv`

Next, you can extract f0 information from all tokens of FIFTEEN, just like we did for “ladies and gentlemen” in the previous example. The output of the script will be fifteen_f0.csv, if you follow the naming conventions on this slide.

Filter bad data

Run filtering scripts:

```
for i in `awk -F, -v threshold=3 -f data-filter.awk <
  fifteen_f0.csv | awk '{ print $3 }'`;
> do grep $i, fifteen_f0.csv; done |
> awk -F, '{ if ($2!="--undefined--") print $0 }' >
  fifteen_f0_filtered.csv
```

In order to run the R script on the next slide, which requires at least 3 good data points per filename to fit a polynomial, we must remove from consideration all filenames with less than three good data points; and we additionally remove all lines where the f0 is --undefined-- by Praat. This is accomplished via a for loop, whose output is piped through a couple more commands, but ultimately writes to the text file fifteen_f0_filtered.csv.

Functional data analysis in R

`z-score-f0-poly-BAAP.R`

- Performs z-score transformation of f0 data
- Fits 2nd-order polynomial to each file
  - Intercept
  - Linear (slope) term
  - Quadratic term
  - Adjusted R^2 for curve fit

Script output: `fit2.coeff.f0` object & .csv file

This script generates coefficients for each set of f0 data