Lexicoder 3.0[1]

Mark Daku, Stuart Soroka, and Lori Young

http://www.lexicoder.com/
September 1, 2015

[1]Please send any bugs, queries, or feature requests to: mark.daku@mcgill.ca

Contents

Chapter 1

Introduction

Lexicoder is a simple multi-platform software package for the automated content analysis of text.

1.1 History

The older version of Lexicoder was developed by Lori Young and Stuart Soroka, and programmed by Mark Daku (initially at McGill University, and now at UPenn, Michigan, and McGill respectively). Funding for the development of the original Lexicoder (v2.0) was provided by the McGill Institute for the Study of Canada, the McGill-Max Bell Foundation, the Donner Canadian Foundation, and the Fonds québécois de recherche sur la société et la culture. The previous version of Lexicoder (v2.0) ran as a standalone java application, and required a level of patience to use that was beyond most normal people.

1.2 Current Release (Version 3.0)

The current implementation of Lexicoder (version 3.0) is a new implementation of Lexicoder v2.0. This version was re-written from the ground up, implements a new engine, and is designed to be called from R, Stata, or the Command Line on PC or Mac. The learning curve has been severely reduced, and we hope that this tool will become a part of more academic toolkits.

Lexicoder 3.0 is in beta. It is provided without any warranty or guarantee. Please use at your own risk and always be sure to have a backup of any data you are working with, as some Lexicoder functions alter source files.

As we are continuing to develop the software, if you encounter any difficulties, bugs, or ideas for additional functions, please contact Mark Daku at mark.daku@mcgill.ca.

Chapter 2

Getting Started

2.1 Calling Lexicoder

Place the Lexicoder.jar file and the lex script in a directory of your choice, for example /users/guest/Desktop/Lexicoder

In order to run Lexicoder, you must specify the full path[1]. For example,

```
/users/guest/Desktop/Lexicoder/lex
```

Alternately, if you have placed Lexicoder in your working directory (not a bad idea), then you can call Lexicoder by using the current directory shortcut:

```
./lex
```

In most cases, if you cannot get Lexicoder to run at all, you have probably gotten the path wrong. If you are still unable to run Lexicoder, and you are sure the path is correct, then you can run the jar file directly:

```
java -jar Lexicoder.jar
```

Please note that all of these do exactly the same thing, and that the preference for using the 'lex' script is to make your code easier to read.

This manual assumes that you will insert the proper path before 'lex' in all of the examples provided. textbfFor those running Lexicoder scripts directly from R (or Stata), Section 6 has some information and examples that will make things much clearer. It advisable to look over Section 6 before continuing here.

[1]Note: You can also add the Lexicoder directory to your system path, which will allow you to call it from anywhere just by typing 'lex'. To do so, please consult the documentation for your OS on setting system paths.

2.2 Lexicoder Output

In most cases, Lexicoder returns information to the command line. This means that if you don't do anything, you will just get your output dumped to the screen. Needless to say, it is often desirable to pipe the output into a file. This is done simply by adding *>file_name* to the end of a command. For example:

```
lex wc dat=odyssey > odyssey_wordcounts.tab
```

All output in Lexicoder is tab-delimited. The output file can then be easily imported into any other software package.

2.3 Overview

Lexicoder requires a minimum of two pieces of information. The first is the function to be performed (see Section 4). The second is a directory where the data files exist. Most commands will require or allow additional options. For a simple example, to perform a word count on the files in the /users/homer/odyssey directory, you would execute the following at the command prompt:

```
lex wc dat=/users/homer/odyssey
```

Note also that Lexicoder will fail and exit if the file specified is not a directory. Thus, if you are looking to perform an analysis on a single file, you must put it in it's own directory first.

Lexicoder returns a simple tab-delimited matrix with the filenames as the row names, and the values of each count separated by spaces. An example follows:

```
case animals speed colour testwords
test1.txt 4 0 0 2
test2.txt 0 0 0 2
test3.txt 0 0 0 3
test4.txt 2 6 3 0
test5.txt 1 2 1 0
```

In this instance, there are five texts (test1.txt though text5.txt), and there are counts for four dictionaries – animals, speed, colour, and testwords. And note that, as described in Section 2.2, if you would like to store this matrix in a file, you will to add a third element to the lexicoder command, as follows:

```
lex wc dat=/users/homer/odyssey > odyssey_wordcounts.tab
```

2.4 Dictionary Creation

For some of the modules you will require a dictionary of words (or phrases) that you wish to find in the text. The new Lexicoder dictionary structure is simple and straightforward to create and work with. Categories begin with a +, and all words following that category (until the next one, or until the end of the file) are considered to be a part of the category.

An example dictionary is below:

```
+colour
red
black
brown
white
yellow
green
+speed
fast*
slow
+country
canada
uganda
denmark
peru
south africa
north korea
```

Note that wildcards (*) are allowed at the end of dictionary entries and will return a count for any word in the text that begins with the preceding character string. In this example, 'fast', 'faster', 'fasts', etc. would all be counted. Also note that multi-word dictionary entries are allowed (e.g South Africa, North Korea, etc.).

This new dictionary format is much easier to work with than the dictionaries used in Lexicoder v2.0. If you are migrating from Lexicoder v2.0, please see Section 5.1 for instructions on how to automatically convert your dictionary to the new format.

Note that if you are creating a dictionary file, it should be saved in plain-text UTF-8 format. Also, be sure to remove any blank lines from the file to prevent possible errors.

Chapter 3

Pre-processing (pre)

IMPORTANT: Pre-processing functions alter the files that are in the directory you specify. Please be sure you have the original data stored elsewhere, as once the cleaning has been performed, the files are permanently altered.

In most cases you will need to pre-process your data. Lexicoder assumes that all cases are one block of text, and as such does not care for spaces or paragraphs. In short:

> Tell me, O Muse, of the man of many devices, who wandered full many ways after he had sacked the sacred citadel of Troy. Many were the men whose cities he saw and whose mind he learned, aye, and many the woes he suffered in his heart upon the sea, seeking to win his own life and the return of his comrades.

Is treated essentially the same way as:

> Tell me, O Muse, of the man of many devices, who wandered full many ways after he had sacked the sacred citadel of Troy.
>
> Many were the men whose cities he saw and whose mind he learned,
>
> aye,
>
> and many the woes he suffered in his heart upon the sea, seeking to win his own life and the return of his comrades.

To be clear: for Lexicoder, the unit of analysis is the text file. Line breaks are used to differentiate between sentences for a few specific processors (see descriptions below), but in general, each file is treated as a single block of text. If you wish to analyze paragraphs or chapters (or another chunk of a larger document) the you must save this text into seperate text files. This can be automated relatively easily, through R or elsewhere.

Pre-processing can either be done externally or within Lexicoder. Manual preprocessing would make sense if you need to retain certain words (for example Green as a last name, denoted by the capital letter) or punctuation.

If you are using the *Lexicoder Sentiment Dictionary* (LSD), we recommend that you use the pre-processing scripts distributed with the dictionary. Currently, they are written and implemented in Applescript (.scpt) and run on a Mac using TextWrangler (a free plain-text editor). These scripts are in the process of being updated and will soon be made available for use within the find replace function (see Section 3.1).

If you using a dictionary without specific pre-processing guidelines, we have included a command that performs several generic cleaning functions. You can process your folder of texts using the following:

```
lex pre dat=data
```

Future versions will give you more control over the cleaning process. Currently, the pre-processing command does the following:

- Normalizes the text: Converts accents and other characters to plain characters.
- Converts to lowercase: Converts all text to lowercase. This has implications for your dictionary, which should also be in lowercase if you choose the generic cleaning options.
- Removes punctuation: Removes all punctuation from the text.

After cleaning with this function, your text will look like this:

> tell me o muse of the man of many devices who wandered full many ways after he had sacked the sacred citadel of troy
>
> many were the men whose cities he saw and whose mind he learned aye and many the woes he suffered in his heart upon the sea seeking to win his own life and the return of his comrades

3.1 Find/Replace (fr)

Description

A pre-processing function that replaces text in your data folder with text specified in a dictionary. Please note that this function alters the original data files, so be sure to be working with a backup.

Usage

```
lex fr dat=data md=replace_dictionary
```

Options

None.

Output

Find / Replace directly alters the files in the data directory. Be sure to be working with a backup of your original dataset.

Usage 1: Find / Replace

Find/Replace requires a different dictionary format than the other dictionary formats, though it is quite simple: the word or phrase to replace is followed by a tab and then word or phrase to replace it (a tab-delimited file with two columns). An example dictionary, replacing emojis with text, would look as follows:

```
tw* twitter_word
:) happy
:( sad
;) wink
;( cry
```

If your text originally looks like the following:

```
twitter is a place where people send tweets
some tweets are :)
some tweets are :(
yesterday I sent twenty tweets to two hundred people
```

The final text will look like this:

```
twitter_word is a place where people send twitter_word
some twitter_word are happy
some twitter_word are sad
yesterday I sent twitter_word twitter_word to twitter_word hundred people
```

Usage 2: Stop Words

The find replace function can also be used to remove 'stop words.' Some words (such as 'a', 'an', ''the', etc.) are of no interest to analyses, and removing them before processing can ensure that analyses are focusing only on the words of interest. Stop words may be removed using the fr function, with a slightly different format.

To remove words instead of replacing them, simply set the replace value of the word you are searching for to '.' (without quotes).

For example:

```
a .
an .
the .
on .
then .
```

Additional Notes

Note that the fr function does not make any assumptions about capitalization or other pre-processing, so TW* is not the same as tw*. A good use of this function would be to code certain punctuation (or emojis) that you would like to retain before running the cleaning script.

Chapter 4

Main Functions

For any of the following functions, 'data' must be a directory full of text files. Required and optional switches are described below.

4.1 Word Count (wc)

Description

Simple function that returns the word count per case.

Usage

```
lex wc dat=data
```

Options

None.

Output

```
case wordCount
1.txt 45
2.txt 54
3.txt 98
4.txt 675
...
[filename] [wordcount]
```

4.2 Dictionary Count (dc)

Description

Using a single dictionary file, returns the count of each dictionary category for each case. Requires a valid directory of data, and a valid dictionary file.

Usage

```
lex dc dat=data md=main_dictionary
```

Options

None.

Output

```
case cat1 cat2 ... [cat_n]
1.txt 2 4 ... n
2.txt 4 0 ... n
3.txt 6 0 ... n
4.txt 4 4 ... n
...
[filename] [cat1] [cat2] [cat_n]
```

Additional Notes

Please note that *dc* will only count the wildcard and not any occurrence of a full word. For example, if your dictionary looks like this:

```
+characters
odyssyeu*
penelop*
+maincharacters
odysseus
penelope
+phrases
tell me o muse
```

You will never get any counts for *maincharacters*.

This is because the module works by replacing occurrences of wildcards in the text with the dictionary entry. For example, if your text reads:

```
the quick brown fox out foxed the other foxes
```

And your dictionary reads:

```
+animals
fox*
...
```

Then the processor will convert your text to:

```
the quick brown fox out fox the other fox
```

4.3 Hierarchical Dictionary Count (hdc)

Description

The Hierarchical Dictionary Count will look for co-occurrences of one dictionary within another **at the sentence level**. If your text files are not split into sentences (that is, they do not contain line breaks), then the hdc will look for co-occurrence within the whole body of text. In short, once hdc finds an occurrence of an entry from the first dictionary (md), it then examines that sentence for an occurrence of an entry from a second dictionary (sd). In this way you could, for example, see what issue areas are mentioned alongside particular politicians or parties.

Usage

```
lex hdc dat=data md=main_dictionary sd=secondary_dictionary
        [td=tertiary_dictionary]
```

Options

The tertiary dictionary (td) is optional, and allows for a third dictionary to be included in the conditions. If td is used, counts will only be returned when categories from all three dictionaries overlap.

Output

```
With two dictionaries:

case main_category sub_category cooccurrence
1.txt liberal environment 2
1.txt liberal economy 4
1.txt conservative environment 1
1.txt conservative economy 5

With three dictionaries:
```

```
case main_category sub_category sub_sub_category cooccurrence
1.txt liberal environment positive 2
1.txt liberal environment negative 1
1.txt liberal economy  positive 4
1.txt liberal economy  negative 2
1.txt conservative environment  positive 1
1.txt conservative environment  negative 3
1.txt conservative economy positive 5
1.txt conservative economy negative 1
```

4.4 Keyword in Context (kwic)

Description

The Keyword in Context (*kwic*) module will return all occurrences of a word in its proper context, allowing for a quick examination of particular words in the dataset. Given a keyword, it will return a default of five words before, and five words after that word within a given sentence, allowing for a quick glance of how words are being employed in the corpus.

Usage

```
lex kwic dat=data kw=keyword
```

Options

If desired, you may alter the size of the window that kwic returns by using the *win* option.

```
lex kwic dat=data kw=keyword win=window_size
```

If the optional context window (win) is not set, then the kwic module will return five words on either side of the found keyword. It will not breach the boundaries of sentences.

Output

```
case keyword context
book24.txt ulysses do not go out against ulysses
book24.txt ulysses your own arrangement that ulysses came home and took his
book24.txt ulysses most reasonable arrangement now that ulysses is revenged
book24.txt ulysses the others had done dinner ulysses began by
book24.txt ulysses quite near and said to ulysses
book24.txt ulysses could that is to say ulysses
book24.txt ulysses forth ulysses leading the way
```

Additional Notes

Note: As of Lexicoder (3.0), *kwic* does not support wild cards or phrases.

4.5 First Mentions (fm)

Description

Returns the raw index of the first mention of each category in a given dictionary. For example, if you have a dictionary with different actor names in it, you can see if one actor or another is consistently being mentioned first (or last). Works with phrases and wild cards. Returns a -1 if the dictionary category is not found in the case.

Usage

```
lex fm dat=data md=main_dictionary
```

Options

None.

Output

```
case            fm_ulysses      fm_penelope     fm_calypso      fm_neptune
book01.txt      674     4131    773     1074
book02.txt      769     13470   -1      -1
book03.txt      4097    -1      -1      242
book04.txt      6966    5386    26657   17971
book05.txt      244     9824    308     12818
...
```

4.6 Most Common Words (mcw)

Returns a master word list for the dataset along with total frequencies.

Description

Usage

```
lex mcw dat=data
```

Options

None.

Output

```
word    count
tell    259
me      904
o       18
muse    5
of      3058
that    1116
ingenious       1
hero    11
who     594
travelled       9
...
```

Additional Notes

Note, as of Lexicoder (3.0) the master word list is unsorted.

4.7 Phrase Comparison (pcp)

Description

This function identifies phrases of a specific length, and then checks to see if it occurs within other cases. The dat option specifies the directory the files are located in, but the root case and the comparison case files must be specified in a separate dictionary file. The format of the dictionary file is as follows:

```
+[root_filename1.ext]
[comparison_file1.ext]
[comparison_file2.ext]
[comparison_file3.ext]
...
[comparison_file_n.ext]
+[root_filename2.ext]
[comparison_file1.ext]
[comparison_file2.ext]
[comparison_file3.ext]
...
[comparison_file_n.ext]
```

Usage

```
lex ppc dat=[data] file=[file_list] length=[phrase_length]
```

Options

```
length=[phrase_length]
```

Output

```
CASE COMPARISON_CASE PHRASE COUNT
file_name1 file_name2 phrase1 1
file_name1 file_name2 phrase2 1
...
```

Additional Notes

You will only ever receive a count of '1' if there is a co-occurrence. Future versions of this function will allow for the identification of multiple co-occurrences. Please note that this function is quite processor intensive, and can take quite a long time to complete on a large dataset.

4.8 Contextual Tagging (tag)

Description

The tag function generates a very ugly, but possibly useful HTML file with different dictionary categories tagged in user-defined colours. The output can be read in any web browser, and allows the user to see the text that is being processed, and how the categories break down.

Usage

```
lex tag dat=data md=main_dictionary > [output.html]
```

For this tool to be used, you must alter your main dictionary to include colour codes after the category names.

```
+red#FF0000
crimson
red
rust
+blue#0000FF
navy
aqua
```

The colour takes the standard HTML hex codes for colours #RRGGBB - bright green is #00FF00, medium blue is #000088, etc.

Options

None.

Output

```
This module streams out an HTML file that can be loaded into any web browser.
```

Additional Notes

Depending on the size of your dataset, the resulting HTML file may be very large, and it may take an extremely long time to process. For quick snapshots of what is happening in the data, it is recommended that the user run the tagging function against a smaller subset of the overall dataset.

It is not necessary to tag text as black (#000000), as this is the default. For more information on HTML Hex (RGB) colors, see http://en.wikipedia.org/wiki/Web_colors

As of Lexicoder (3.0), the contextual tagger only works with a single dictionary, and does not work with hierarchical dictionaries.

Chapter 5

Advanced Functions

5.1 Dictionary conversion (lcd)

Description

This function simply converts a dictionary in the Lexicoder 2.0 format into a format that is compatible with the new engine.

Usage

```
lex lcd md=old_dictionary sd=new_dictionary
```

Options

None.

Output

Outputs a new dictionary, as specified in the sd option.

Additional Notes

You must specify a new file to create for the dictionary file using the sd option. Lexicoder will not overwrite a dictionary file, and will not output the new dictionary to the standard output.

5.2 Dataset conversion (dsc)

Version 2.0 of Lexicoder required all text to be in a tab delimited file. In Lexicoder 3.0, each case must be in its own file, all in the same directory.

A tool is provided in Lexicoder (3.0) to convert older Lexicoder datafiles to the new format. To do so, use the 'dsc' function:

```
lex dsc dat=old_datafile out=new_directory
```

The new directory must not exist already, or Lexicoder will exit.

The new dataset will name each file according to its ID and the .txt extension. E.g. 1.txt, 2.txt ... 1234.txt, etc.

Note: If Lexicoder is throwing exceptions, it may be that your original file is not in the proper encoding. Try saving the file with UTF-8 encoding (done easily in TextWrangler, or other text editing packages). This should address the issue.

It is strongly advised that you begin with the raw text instead of converting an old file. Lexicoder v2.0 files did not preserve sentence structure, a useful addition to version 3.0.

Chapter 6

Integration with R and Other Software Packages

Lexicoder (3.0) has been designed to interface easily with R. This chapter makes clear exactly how that integration works. Stata integration is also simple; while we do not go into detail about Stata integration here, that process is very similar to what follows.

First, Chapter 2 makes clear that you will need to place the Lexicoder.jar file and the lex script in a folder of your choice, and then point towards that path when you run Lexicoder. In R, this involves setting the R working directory to the folder in which the Lexicoder files exist. (Often, it is simplest to put the Lexicoder files in the same folder as your text data.) You can do so using the setwd command; if your data were in same folder as described in Section 2.1, your R command would be as follows:

```
setwd("/users/guest/Desktop/Lexicoder")
```

Once this is the R working directory, you should be able to run all the Command Line prompts used in this guide by using the current directory shortcut. On a Mac, you would do so by invoking a system command. For a simple wordcount, where the corpus is stored in an 'odyssey' subfolder in the 'Lexicoder' folder listed above, this could look as follows:

```
system("./lex wc dat=odyssey > odyssey_wordcounts.tab")
```

If you are working in R, then, the adjustments in the line above are required for all Lexicoder commands. To be clear, the more general example given for the dc command in Section 4.2, which is

```
lex dc dat=data md=main_dictionary
```

would, in R, be implemented as follows:

```
system("./lex dc dat=data md=main_dictionary")
```

Lexicoder results are in plain-text tab-delimited files, which are easily imported in Excel, or any other spreadsheet software. These are also easily imported into R using the read.table command, for instance:

```
results <- read.table("odyssey_wordcounts.tab", header=T,sep="\t")
```

A more complete sample workflow, in form of an R script file, will be made available at lexicoder.com.

Chapter 7

Limitations

7.1 Wildcards

Wildcards are supported at the end of a dictionary entry only. Thus, a dictionary entry that reads:

```
politic*
```

will count 'politic', 'politics', 'political', etc.

At the moment, Lexicoder (3.0) does **not** support wildcards at the beginning of an entry. For example, you cannot have a dictionary entry that reads '*political' if you are looking for entries of both 'political' and 'apolitical.' Multiple wildcard support is planned for future versions of Lexicoder.

7.2 Unique dictionary entries

This version of Lexicoder (3.0) does not support having the same word in multiple dictionary categories. If 'Harper' is in the 'Politicians' category, it cannot also be in the 'Magazine' category. Having multiple dictionary entries that are the same **will not cause an error**, however only the first category that the word belongs to will be counted.

7.3 Illegal dictionary characters

You may use any character or word in your dictionary, except that any dictionary that begins with + is treated as a category. Thus, you cannot have a dictionary entry that reads:

```
+1
```

This entry will create a category called '1' and will not be treated as a dictionary word.

Chapter 8

Example: Simple workflow

The following describes a typical way of using Lexicoder from the command line. (A workflow in R will be made avaiable at lexicoder.com.) We have each book of Homer's Odyssey in its own file in the directory 'odyssey.'

```
book1.txt
book2.txt
...
book24.txt
```

8.1 Step 1: Cleaning

First, we want to pre-process the text (this will vary depending on which dictionary is being used). For this task, we will run the generic cleaning package against the text. This will remove special characters, convert everything to lower case, and remove all punctuation.

Note that in the cleaning phase, the original data files will be overwritten. Be sure you are not working with a copy of your original dataset. (Our recommendation is that you keep a raw version of the dataset in a safe, zipped folder.)

Note that we can continue to use the cleaning raw text files for content analysis; we can also generate a new 'dat' file, which may improve performance in subsequent analyses. Line 1 below simply cleans the raw text files. Line 2 cleans those files and save a separate dat file, which can be used for subsequent functions in place of the directory name.

```
[1] lex pre dat=odyssey
[2] lex dat dat=odyssey out=odyssey.dat
```

8.2 Step 2: Analysis

For this example, we are outputting on the command line to tab delimited files. If you wish to integrate directly into R, see section 6. First, we want to know how many words are in each of the cases, which we can use later to standardize counts. Second, we want to see how many times each character is written about in each book, so we collect the dictionary analysis from *characters.lcd*. Third, we are really interested in knowing how 'Penelope' is described, so we pull out all mentions of Penelope to see if this is worth further investigation. Finally, we want to look at the tagged dictionary entries in context, to see if (for example) some books are referring to all of the characters more than others, so we create a tagged HTML file from the dataset and the *characters.lcd* dictionary.

```
[1] lex wc dat=odyssey > odyssey_wc.tab
[2] lex dc dat=odyssey md=characters.lcd > odyssey_characters.tab
[3] lex kwic dat=odyssey kw=penelope > odyssey_penelope.tab
[4] lex tag dat=odyssey md=characters.lcd out=odyssey.html
```

Note that additional examples, practice files, and R scripts will be made available at lexicoder.com.

Chapter 9

Additional Information

9.1 Suggested Citation

If you use this software, please cite it with the following:

Daku, M., Soroka, S., and Young, L. 2015. Lexicoder, version 3.0 (software). Available at: www.lexicoder.com.

9.2 Dictionaries

Several dictionaries have been developed to work with Lexicoder. They are available at: http://lexicoder.com/download.html

9.3 License and Copyright

The text from Homer's odyssey, included here as an example, was taken from the Internet Classics Archive at MIT (http://classics.mit.edu/Homer/odyssey.mb.txt).