Overview

Beginning c. 1997 Princeton University Cognitive Science Laboratory developed an online lexical reference system.[1] While the system was intended for use by linguists and academicians in related fields, I have found it to be an excellent free[2] dictionary. When I began using my Linux machine more frequently[3], I decided it would be desirable to avoid the complicated installation procedures for the windowed browsing verions of WordNet on Linux[4] and create a simple, single makefile and install/run directory solution which could be used by anyone running Linux to quickly produce a quick, clean access to the application via a Linux shell session, i.e., using the command line in a terminal window (I will usually refer to this as “in a terminal session” in this document) as with all the other tools that accompany Linux. I leave the executable program name as simply “**wn**” as that is simply typed, but I will refer to my simplified WordNet command line interface program as “*WNtight*” also, to differentiate between that and the parent application WordNet.

It would be possible to wrap the invocation of WordNet in a shell script, but I personally prefer typing “**wn** *convoluted* **-over**” (I will try to consistently use bold font for commands and options of the shell or WordNet and italic for the search term) in a terminal window to obtain the meaning and available senses of, e.g., the word *convoluted*, which I can then follow up with a number of interesting options available in WordNet to drill down

1 See WordNet - A Lexical Database for English. From their FAQ it appears they began collecting word lists as early as 1985, beginning with the Brown Corpus, transitioning to a polysemy index by Richard Beckwith c. 1988 and various thesauruses and word lists. I began my project by compiling some of the version 1.6 code dating from 1997, successfully isolating the command line interface from the initial Tlc/Tk window interface. The version I am providing here is derived from their 2006 3.0 code (with 3.0 data).

2 The license can be displayed from the application with the option **-l** , the gist of which is you can use, copy, modify and distribute the software and database freely if you acknowledge copyright (2006 Princeton University), make no representations or warranties (software is provided “as is”) on part of Princeton, and despite necessarily having to mention Princeton to comply with these provisions, do so without intent to advertise or otherwise recommend derivative usage.

3 If you are using Microsoft Windows, you can simply download one of the WordNet installer files for versions written for that operating system at the link given above. It provides a small window application with which to access the WordNet functionality.

4 Since Version 2.1 they use GNU Autotools, which creates tortuous multilevel recursive makefiles, and Tcl (Tool Command Language) with the graphical user interface toolkit Tk, creating annoying package update and configuration problems, at least in my Linux system context. The guts of WordNet are written in mercifully clean C and is all we need for a command line interface in a terminal session on Linux.

on the usage of interest to me. For example, “**wn** *convoluted* **-over**” gives me two *senses* for the adjective (possible *parts of speech* or *syntactic category* are noun, verb, adjective, or adverb; a *sense* is the meaning of the word in a particular *synset* or set of words that are interchangeable in some context without changing the truth of the statement communicated, e.g., *convoluted* and *tortuous* are in the same synset).

I can type ↑ (up arrow) on the keyboard in the terminal session to obtain the previous command line and thereby enter the command “**wn** *convoluted*” without option (backspacing to delete it) to obtain a list of all the information WordNet has. For example, it knows *antonyms* of the word in this case, which I can see by entering“**wn** *convoluted* **-antsa**” (obtaining antonym “simple” for sense 2, “uncoiled or straight” for sense 1). Being in a terminal window, it is easy to select and copy portions of the output (which includes commands that apply to the word of interest, e.g., “**antsa**” here, so you don’t have to initially memorize the commands, but rather simply understand the conceptual environment), saving typing when writing in a separate document preparation window or the like (more on this below when we discuss usage in detail; we merely whet your appetite in this Overview).

The WordNet search functions (and other WordNet database access and analysis functions) can be accessed via C-language API and included in your applications (include the library file and the header .h file), if an idea for a new application occurs to you. I am primarily concerned here with providing an easy to install and use free dictionary utility for Linux, with all the power of the WordNet linguistic application available also.

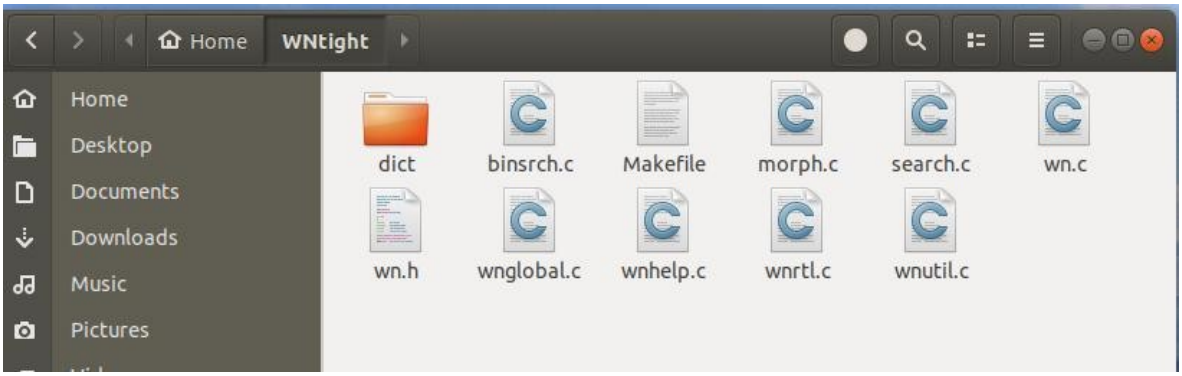
Installation

Create a new directory (aka a folder) in your *home directory*[5] and name it **WNtight**[6]. (I place an end of sentence period outside of an ending quotation mark if there is any chance of confusing it with a system name or command.) Create a sub-directory

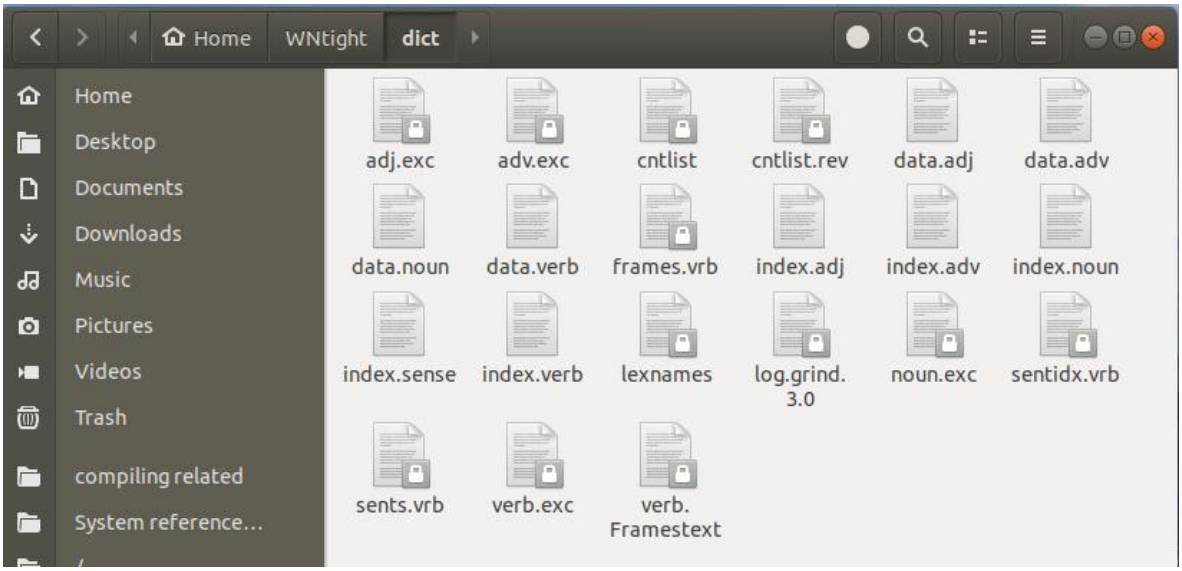
5 Typically the home directory on a Linux machine is of form /home/yourName or ~/yourName with tilde expanding to /home.

6 It seems easier to me to make the executable locally and then copy it to target location so as to avoid permissions problems making in the ultimate system directory /usr/local target.

named **dict** in the new directory (this name and relative location is mandatory). Download the *WNtight* package I provide. Extract the contents and place the eight *.c* files, the single *.h* header/include file, and the Makefile in **~/WNtight**. Place the 21 dictionary files in the sub-directory **dict** (i.e., in **~/WNtight/dict**). You should have a layout something like this (my system is Ubuntu 18.04.4 on a Dell Precision-3541, your windows may appear different depending on your system):



dict subdirectory contents should appear as (it does not matter whether they are read-only or not):



Brief comments regarding C compile environment

I am unable to provide a general tutorial on the subject of how you verify you have the necessary compiler and libraries (and obtain them if not), but there are good online resources you may consult (as did I). See also Appendices A and B of this document for resources for learning Linux and programming (though you should not require much if any programming to install and use *WNtight*).

Perhaps a few comments about my own environment would help you search for information. You should, of course, try compiling *WNtight* first, as I will describe below, i.e., you may get lucky and have everything you need. The code seems to be standard C. In fact, after separating it from the window interface, I compiled the original WordNet C source from 1997 without a problem with my 2019 environment (aside from some inconsequential compiler warnings which I still have even with the 2006 source used in *WNtight*).

You will need a Linux GNU C development environment, i.e., *the GNU gcc compiler, standard libraries and the header files required in order to compile and link programs using the standard C library.* Our single include file, *wn.h*, includes (that is, there is an include statement for this in the file) the <*stdio.h*> standard C library header, i.e., the .*c* source files use the standard C libraries and therefore require the header files. If you are using Ubuntu you may have to install these (I had to). There are forum pages and sporadic official Ubuntu pages online that will guide the motivated seeker towards the correct package(s) to download (the official pages and the apt-get application typically point out dependencies you need to resolve).

An online Ubuntu guide told me that “by default, Ubuntu does not come with the tools required[7] [to build packages, i.e., compile and link to an executable]. You need to install the package *build-essential* for making the package.”

7 This Ubuntu article also tells me basically that if a package requires compiling, then I don’t really need it. Coupled with the increasing tendency to insist on automatic updates to all packages (the Snappy system in Ubuntu; there is also Flatpack/xdg-app for Linux generally and Launchpad), I am getting that Microsoft Windows “*déjà vu* all over again,” paraphrasing Yogi Berra.

build-essential is a list of the “minimal set of packages that are always needed to compile, link and put in a Debian[8] package a standard "Hello World!" program written in C or C++” (quoted text obtained by executing **cat /usr/share/doc/build-essential/list** in my terminal session). This quote makes it appear that it is clear what is required to compile and build applications from C source. I will note that the same description from *build-essential* also adds (perversely, but honestly) that “this file lists the non-essential packages that are build-essential...[but] the list is not closed under the ‘depends on’ relations, so one will usually need to install more packages than given here... actually, there is no authoritative list at all.” In any case, that list contains *libc6-dev* (provides ISO C standard library), *gcc*, *make*, and a couple more of less interest here.

When I ran **sudo apt-get install build-essential** at the command line, **apt-get** reported that new packages *build-essential, g++, g++-7, libc-dev-bin, libc6-dev, libstdc++-7-dev, linux-libc-dev* , and *manpages-dev* would be installed.

g++ and *g++-7* are C++ compilers and *libstdc++-7-dev* contains headers and static library files necessary for compiling in that environment, which we are not concerned with since we are compiling standard C. Package *libc-dev-bin* contains “utility programs related to the GNU C Library development package”[9] and package *libc6-dev* “contains the symlinks, headers, and object files needed to compile and link programs which use the standard C library...” Both of these were originally maintained by GNU Libc Debian and now by Ubuntu Developers (I believe ubuntu.com and launchpad.net offered these).

The command **gcc** on my system is a link to **gcc-7** which in turn is a link to the executable **x86_64-linux-gnu-gcc-7** (the technically proficient will notice the so-called architecture-operating system-libc *triplet* in this name). My compiler reports (with terminal command **gcc --version**) its version as *gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0.*

One warning: if you are prompted to install a dependency or dependencies and it mentions “removing the x-windows system” or the like, it would probably be a bad idea to

8 Debian is one of the most popular Linux systems offering x86 optimized kernels. Ubuntu traces its lineage from Debian.

9 Execute **apt-cache show libc-dev-bin** in a terminal session to obtain this information.

do that[10] (unless you enjoy booting into a non-graphical interface, i.e., black screen with only command line instructions available, and repairing your system piecemeal or simply starting over by restoring your original system image and possibly losing data).

Resume WNtight install

We left off earlier above with the necessary .c source files, .h header file and Makefile placed in the **WNtight**[11] directory in your HOME directory (and **dict** sub-directory populated with the dictionary data files, necessary to actually run the application, but not for compiling). Open a terminal session in the **WNtight** directory. If you check the directory with an **ls** (“ls” as in “list,” not number 1) command, you should obtain terminal output:

```
~/WNtight$ ls
binsrch.c  Makefile  search.c  wnglobal.c  wnhelp.c  wnutil.c  dict     morph.c  wn.c
wn.h       wnrtl.c
```

Type **which gcc** (will search your PATH and indicate where it finds gcc) or **gcc –version** (will invoke the compiler and obtain the version if the compiler can be found). If either of those commands indicated you have the compiler and it is in your PATH variable (if either command works the compiler is in the PATH because it was found), just type **make** (make is a separate program from gcc, but it is generally present in a Linux installation) in your terminal (and enter/return) to compile the WordNet interface source code (*wn.c*) and library source (the other seven .c files) in the **WNtight** directory (your current working directory). You should see output on your terminal screen like:

```
gcc -I. -O -DUNIX -c wn.c
gcc -I. -O -DUNIX -c -o binsrch.o binsrch.c
gcc -I. -O -DUNIX -c -o morph.o morph.c
gcc -I. -O -DUNIX -c -o search.o search.c
gcc -I. -O -DUNIX -c -o wnhelp.o wnhelp.c
```

10 I prefer not to discuss how I came to know this.
11 /home/yourName/WNtight to be specific, with dictionary sub-directory /home/yourName/WNtight/dict

```
gcc -I. -O -DUNIX -c -o wnrtl.o wnrtl.c

gcc -I. -O -DUNIX -c -o wnutil.o wnutil.c

ar rcv libwn.a binsrch.o  morph.o  search.o  wnglobal.o  wnhelp.o  wnrtl.o  wnutil.o;
ranlib libwn.a

a - binsrch.o

a - morph.o

a - search.o

a - wnglobal.o

a - wnhelp.o

a - wnrtl.o

a - wnutil.o

gcc -o wn wn.o -lwn -static -L
```

I deleted the warnings (which appeared interspersed in the compile stdout[12] reports above), of which there are not a few. If you are a purist you are welcome to chase down the cause of the warnings and attempt to remedy, but I am concerned only about *errors* for the time being. If you have no errors in the make/compile/link you probably have a functioning result. If everything went well your **~/WNtight** directory should now have an object file (*.o* extension) for each source (we eschew automatically deleting those following a successful make), a library archive file (*libwn.a*), and an executable program, simply *wn* with no extension.

We did not include a **-k** option (“keep going”) for make, so the make process should have halted on any error. If you enter another make command, it should immediately report “nothing to be done for ‘all’”, verifying that all tasks were accomplished. If you did not get

12 Rather late date to mention, but Linux commands are read from the standard input, a keyboard in our case, and displayed on the standard output device, originally terminal screens, now your computer screen. Abbreviated stdin (standard in) and stdout (standard out). Didn’t find Thomson and Ritchie using those abbreviations yet in their 1974 Unix Programmer’s Manual (perhaps letters were still relatively inexpensive back then).

this far, you can consult the GNU Make Manual[13] and make use of the debug options available to give you an idea what *make* is thinking when it tries to execute the *Makefile*. The error report in the terminal may be fairly obvious as well, e.g., something like “library not found” or the like. However, as we said earlier, the WordNet source files we retained to implement the command line interface, writing a new makefile to permit all of this to occur in a single directory with single makefile, should not make many demands of your system other than to ask for a C compiler and standard C libraries and headers (as we discussed above). It is, of course, necessary that you have GNU **make**, but that is almost certainly true on any Linux system, as we suggested earlier. The **ar** and **ranlib** tools (used to place the library object files into a library file with .a extension and create and index within the file with which programs may locate and use the library functions) are standard Linux library tools (you can find out more about those by typing **man ar** and **man ranlib** in your terminal session[14]). Let us assume you have successfully made the *wn* executable and are ready to make an installation directory and copy the executable over there.

Copy executable and support files to run directory

I have the header file *wn.h* hardwired[15] to require the operating[16] directory in absolute path: **/usr/local/WNtight/** [17]. You will need superuser privileges to do the following. On an Ubuntu system you use **sudo**. On other Linux systems you may use **su**. Open a new shell (terminal session) and navigate to **/usr/local** . Copy the path of the **/home/yourName/WNtight** directory where we just made the executables above (you could copy that in the original terminal window where you just made the executables) . It

13 Obtain at GNU Make Manual.

14 John R Sheets includes an excellent discussion of libraries (“Dealing with Libraries”) in his free book “Writing GNOME Applications,” available at Writing GNOME Applications.

15 Some might prefer to use new environment variables instead. There does seem to be traces of such a variable in one of the library sources, but my aim was to get this utility up and running quickly with a minimum of trouble (for myself and for those who might want to use it).

16 In other words, the directory where the executables and dictionary files should be located to make the program available for use.

17 To be tiresome, you can navigate to root “/” (the directory hierarchy organization can be thought of as an upended tree, branching as you move down from root to subdirectories) where **ls** would show subdirectories including “**bin etc lib usr**” (and more). **cd** to **user** (which contains, among other things, “**bin include lib local**”, then **cd** to **local**. **sudo mkdir WNtight** while in **/usr/local** to produce **/usr/local/WNtight** new directory.

should be (you can see this at your prompt or type **pwd** to get the full path) **~/WNtight** or **/home/dalton/WNtight** (replace “**dalton**” by whatever name used for your home directory on your Linux system). When used like this as the first character of a path, the tilde “**~**” expands into the full path for your home directory to save typing if you choose to use it.

Use that copied path name to make a *copy command* **cp** (syntax is **cp [option] source destination**) which will simultaneously (1) make a directory (defaulting to the source directory name **WNtight**) in destination **/usr/local** and (2) copy the files and subdirectory over from the original location in your home directory:

(base) dalton@dalton-Precision-3541:/usr/local$ **sudo cp -r ~/WNtight .**

Please be careful here! *That command will copy recursively every file and sub-directory in the first path named* (in this case the *source* directory is **~/WNtight** before the period *destination* at the end of the command line). I have used font color to replicate the color coding on my own terminal and draw your attention to the fact that the current working directory location of your terminal session is usually shown as the prompt (the blue /usr/local in this case[18]). *Please be sure you are executing the command from a terminal window open in the* **/usr/local** directory **.** The period “**.**” at the end of the command line means “the current working directory,” i.e., the location of that terminal session. That is where the *from* directory **~/WNtight** and all its contents will be copied *to*.

If you check the contents of your current working directory **/usr/local** now with **ls** you should see **WNtight** appear as one of the subdirectories, e.g., bin etc games include lib man sbin share src texlive WNtight . Enter the new directory **WNtight** (command **cd WNtight**) and you should see the source, object, executable files and the **dict** subdirectory if you have successfully copied them over.[19]

We could do an initial test while in the new directory by command **./wn -l** . The “./” tells the shell to find the **wn** executable in your present location (rather than look

18 The terminal window back in the make directory would be ~/WNtight **.**

19 If the copy command executed without error and you don’t see them, you may be in for an entertaining search of your computer to find out where you sent them. See the resources listed in Appendix A for instruction in using Linux.

elsewhere on the computer, following the paths stored in the environment variable PATH), which should be **/usr/local/WNtight**.[20] The terminal should respond with the WordNet license: WordNet Release 3.0 This software and database is being provided to you, the LICENSEE, by Princeton University under the following license...**.**[etc.]

If you obtained this response, then the executable for *WNtight, i.e.,* **wn** , functions. To check if it can find its dictionary files in the **/usr/local/WNtight/dict** subdirectory, type **./wn dog -over** (this asks for an overview of the word “dog”). You should get many lines of information, i.e., seven senses of the noun “dog,” e.g., #7 being “andiron, firedog,...”

If instead of information about “dog” you receive several lines of errors, e.g., WordNet library error: Can't open datafile(/usr/local/WNtight/dict/data.noun)..., then you should check your local **dict** sub-directory to assure the dictionary files were copied in above. If they are there, then it is likely you have not correctly named the operating directory dictionary location, that being hardwired into **wn.h** as DEFAULTPATH = /usr/local/WNtight/dict .[21] See Appendix A: Linux resources for resources to bring you up to speed for troubleshooting if that is not the solution.

Assuming those two tests worked ok from the operating directory **/usr/local/WNtight** , you just need to add that path to your PATH environment variable so you can execute **wn** from anywhere on your computer. Different Linux installations use different profile files to get user paths into the environment. On my system the **.bashrc** file (is a text file) in my home directory ~ allows me to get the job done by inserting the following lines (I always include a comment line with date and motivation for later reference) at the end of the file using a simple text editor (like **gedit**[22]):

```
# 04/21/20 add path for wn
```

20 If you have not gotten confused. I recall a saying from some religious text to the effect that “in much speaking there is evil.” I now have a better appreciation why Linux application developers try to script automate almost all of the installation process.

21 If you are sufficiently familiar with C compiling you could of course edit the DEFAULTPATH in the wn.h file to some other path. I would try to get things working first.

22 **gedit** is a graphical interface text editor included with the GNOME graphical desktop environment. You might have **kedit** if using the KDE desktop. Linux terminal interface editors (using command line environment) usually include **vi** or **vim**.

```
export PATH="$PATH:/usr/local/WNtight"
```

Close your terminal window and when you open it again, it should add this location to the end of the list of paths to automatically search when you invoke an executable script or program. Check with **echo $PATH** in a new terminal session and get something like the following on your screen:

```
/usr/local/texlive/2019/bin/x86_64-linux:/home/dalton/anaconda3/bin:/home/dalton/
anaconda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/usr/local/games:/snap/bin:/usr/local/WNtight
```

I made the final path bold in the above terminal response to make it easily visible. If your system is differently configured and won’t accept the PATH modification and file I used, see the article at Unix & Linux Stack Exchange - How add path to path for a number of different ways to accomplish this task (and my Appendix A: Linux resources cites resources that put those procedures into context if unfamiliar to you). After the path is successfully added you should be able to invoke **wn** from your home directory or elsewhere.

General usage of WNtight

Now for the fun part.[23] As I write this manual (in *LibreOffice Writer*) I might want to check on my usage of the word “usage” in the heading (perhaps I wonder whether “use” might be preferable). I bring up a terminal window[24] and enter **wn usage** . On the terminal window I see response:

23 I include in the WNtight package the original WordNet 3.0 html documentation files. You may want to bring those up in your browser to consult in parallel with my “learn by doing” tutorial. They define all of the terms you see in my tutorial.

24 I keep a shortcut to the terminal emulator for the GNOME desktop, GNOME terminal, on my desktop side bar (sometimes called “the dash”). I can just move focus there (i.e., move the mouse pointer with my touchpad on my laptop) and click on the terminal icon to bring up a new terminal window. Most systems have a keyboard sequence to bring up a terminal session also. On my system it is somewhat clumsy, having to press “fn”-key + “Alt”-key then (while holding those keys down) “F2”. That brings up a small window where I can type in the application name, *gnome-terminal*, and hit “enter”-key to bring up the terminal. A third way available on my Ubuntu 18.04.4 system is to hit the “Super”-key (has a Microsoft Windows logo on the key, or on Apple pc the Command key) and type “terminal” then select one of the many available.

Information available for noun usage

-hypen Hypernyms

-hypon, -treen Hyponyms & Hyponym Tree

-synsn Synonyms (ordered by estimated frequency)

-derin Derived Forms

-famln Familiarity & Polysemy Count

-coorn Coordinate Terms (sisters)

-grepn List of Compound Words

-over Overview of Senses

No information available for verb usage

No information available for adj usage

No information available for adv usage

That tells me there is more information for the noun part of speech, so I enter **wn usage -over** (one of the command options listed in the response) in my terminal window and obtain (for those who want to simply check the definition of a word, **wn yourWord -over** may be sufficient):

Overview of noun usage

The noun usage has 3 senses (first 2 from tagged texts)

1. (9) use, usage, utilization, utilisation, employment, exercise -- (the act of using; "he warned against the use of narcotic drugs"; "skilled in the utilization of computers")

2. (1) custom, usage, usance -- (accepted or habitual practice)

3. usage -- (the customary manner in which a language (or a form of a language) is spoken or written; "English usage"; "a usage borrowed from French")

I see that sense “1” has (9) representations in the *semantic tagging data*, i.e., “usage” appears nine times in their semantically tagged input files[25] (if you combine a

25 A collection or corpus of English (or your language of choice or birth) has samples of sentences, e.g., the Brown corpus, with a million words of running text of English prose printed during 1961. The Brown

textual corpus like the Brown Corpus with a *lexicon* like WordNet so that every substantive word in the text is linked to its appropriate sense in the lexicon, you will have *semantically tagged* the corpus, paraphrasing from the WordNet documentation). Sense “2” appears once in the semantically tagged input corpus. The third sense has no tag representation number so it has been arbitrarily assigned. Either sense “1” or “2” seem appropriate to my context. We could look at a list of synonyms listed in order of their frequency of appearance using the options suggested, i.e., **wn usage -synsn** (you may notice a pattern here of a base option with the part of speech appended, e.g., -syns for synonym and appended “n” for the noun sense; I simply let *WNtight* tell me the applicable options):

```
Synonyms/Hypernyms (Ordered by Estimated Frequency) of noun usage

 3 senses of usage

 Sense 1

use, usage, utilization, utilisation, employment, exercise

     => activity

 Sense 2

custom, usage, usance

     => practice, pattern

     => survival
```

---

corpus is *syntactically tagged*, i.e., the parts of speech (nouns, verbs, etc.) are assigned to its words. To tag those words *semantically* is a much more difficult task. In that case you have to know what the sentences *are about*, rather than simply their grammar, their placement of the parts of speech. Because current computers are simply advanced abacus devices rather than conscious entities, it is necessary to somehow assign meaning (a string of characters) by manipulating input symbols (letters of which words are composed, propositions that include certain syntactic elements in certain orders, etc. in large databases of prose). The latest technique is machine learning, which is a euphemism for trial and error. For example, if you have computer algorithms to impute a meaning string to a sentence (e.g., it says the “city New York has a lot of covid-19 cases”, that string emitted from the computer, representing the thought by a human) you can train the algorithm by correcting it when it is wrong and reinforcing the algorithm when it is correct (by changing weights of analytical terms etc.). In our case, someone semantically tagged the use of the word “usage” found in the corpus files on which the tagging was based. So it appeared a certain number of times in for example the sense of a noun meaning “act of using.” If you want to explore Natural Language processing, a good start is Natural Language Processing with Python Natural Language Processing with Python. The open source NLTK Toolkit software is available at NLTK site (the Book Collection includes WordNet and enhanced analysis at command line).

Sense 3

usage

=> language, linguistic communication

That output of the **-synsn** option tells me that “usage” is a kind of activity in sense “1” (see the arrow pointing to the hypernym), a kind of practice or pattern in sense “2”, etc. We could look directly at the hypernym with **wn usage -hypen**

Synonyms/Hypernyms (Ordered by Estimated Frequency) of noun usage

3 senses of usage

Sense 1

use, usage, utilization, utilisation, employment, exercise

=> activity

=> act, deed, human action, human activity

=> event

=> psychological feature

=> abstraction, abstract entity

=> entity

(I didn’t show all three senses output this time.) This shows us sense “1” has a hierarchy of containing categories, forming statements of the form, “usage is a kind of activity,” “a kind of human activity”, etc. (a hypernym contains a hyponym, X is a kind of Y with Y the hypernym). **wn usage -derin** gives us “derivational morphology links between noun and verb forms” (show the words that appear related because of their actual spelling or because of the meaning):

Derived Forms of noun usage

2 of 3 senses of usage

Sense 1

use, usage, utilization, utilisation, employment, exercise

RELATED TO->(verb) use#1

=> use, utilize, utilise, apply, employ

Sense 2

custom, usage, usance

RELATED TO->(verb) use#6

=> use

This output tells us that sense “1” of “usage” is related to the verb “use” in its sense “#1.” If you take a look at “use” with command **wn use -over** (very lengthy output of senses of “use” as noun and as verb), its verb sense “1” is apt, “put into service; make work or employ for a particular purpose...”[26]

Well, I believe I have given you a sense of the usage of Wntight/WordNet.

Appendix A: Linux resources

If you are new to Linux, you could begin with a Unix tutorial, downloadable at Unix Tutorial - UK Surrey (or view online). Follow up with Machtelt Garrels’ “Introduction to Linux – A Hands on Guide” (available free online also, somewhat dated but still highly recommended). “The Linux Command Line,” by William Shotts is a resource you will find yourself consulting often (as with Garrels’ Intro). It is also available online without charge. If you want to understand the components of a Linux system (or any modern day computer), Mohamed Zahran at New York University has available most of his

26 The amount of output on the word “use” is unsurprising considering our descent from *homo habilis*, the “handy man” able to fashion and *use* many tools to survive.

computer science course (“Computer Systems Organization,” CSCI-UA.0201-003[27]) discussing computer architecture, operating systems, and compiler interaction from a programming viewpoint (nice presentataion in slide format with plenty of figures and graphics).

Any of the *Stack Exchange* or *Stack Overflow* sites dedicated to Unix, computer science or specific flavors of Linux (e.g., Ubuntu) are highly recommended, not only for detailed answers to specific questions but for references they often provide. *Wikipedia*, as usual, is not a bad point of departure for many subjects (it is always good practice to follow their references and search[28] on the topics elsewhere also). The *man pages* on any Unix/Linux system contain a lot of information on specific programs and tools (e.g., type **man gcc** in a terminal session for information on the compiler). Official manuals from the GNU (a recursive acronym “GNU’s Not Unix”, though it really *is* Unix unless you are talking to someone in the legal profession) Project are usually available at their site[i], “Using the GNU Compiler Collection,” “GNU Make,” “GNU C Library Reference Manual,” “GNU C Reference Manual” (the last two geared towards progamming in C rather than accomplishing the compile and link operations, addressed by the first two documents listed).

27 Sometimes links to such things change over time, I think I got this at CSCI-UA.0201-003 Computer Systems Course NYU. The professor also gives a personal site link Mohamed Zahran personal site. If the content is still online, you should be able to find it with this amount of information.

28 Be careful out there on the internet when searching. I always follow a three-step protocol before clicking on a serp (search engine results page) link: (1) do a *whois* on the url (several good whois sources available, e.g., Whois - Domain Tools) and inspect date created (older is preferable), country of origin (you know who are more often domains of hacker/crackers), and institution or name (2) do a Virus Total check on the url, looking at their details tab also (often better than *whois*). Norton and IBM also have some free reputation checks available (though IBM makes you work to get to the check). (3) conduct a Google search on the site url using the “url -site:url” search option, i.e., get a listing of what other sites say about the site url target.

Appendix B: Programming/computer science

If you have no programming background and want to learn (you don’t have to be a programmer to install WNtight though), MIT makes their classic entry-level course text (required for science and engineering majors), “The Structure and Interpretation of Computer Programs,” by Abelson, Sussman, and Sussman, available online[29]. A less complicated introduction to programming may be had in a Python-oriented text “How to Think Like a Computer Scientist - Learning with Python” by Downey, Elkner, and Meyers (available online, try How to Think Like a Computer Scientist, with the usual caveat that links may not always remain usable over time, but the reference usually will be available elsewhere).

If you want to get a handle on programming in the C language, Brian Kernighan’s “Programming in C – A Tutorial,” is still available online free. (think I found it at a Bell Labs archive). The soft-cover book I used to learn C in 1984 was the “white book,” “The C Programming Language,” by Kernighan and Ritchie (Dennis Ritchie transformed an earlier “B” language into “C” 1971-1973 and in 1978 Brian Kernighan and Ritchie wrote the “white book” that was the language definition until ANSI standardized C 1983 on) . I couldn’t find a free copy online but it is undoubtedly still available from vendors.

The Stack Exchange and Stack Overflow sites devoted to computer science and programming in various languages are a high-quality source of answers to specific questions and references.

29 If the link changes over time, look for the text at the general link MIT Press.

i You may have to persist to find a manual rather than a sequence of online html pages (can always print those to pdf, but is time-consuming, could site scrape but is tricky). Even if the manual is older than the html online version, it is still useful generally.