N-GRAM PROCESSOR 0.4
A SHORT USER GUIDE

Andreas Buerki
buerkiA@cardiff.ac.uk

CONTENTS

1 INTRODUCTION

The N-Gram Processor (NGP) is an open-source, free software package which produces lists of word n-grams and their frequencies from input text files. Its users will typically be linguists working with corpus linguistic methods, but others may find its functions useful as well. This brief guide introduces the reader to what the NGP is and how to use it. It assumes basic familiarity with a UNIX-like operating system and a rudimentary understanding of how to run software using a command line interface such as is accessed in Linux or OS X through the application 'Terminal'.[1] This guide is structured as follows: in the next section, the the key features of the NGP are listed and some background to the NGP is given, including a history of its codebase, as well as an introduction to the task of n-gram list creation. Installation information, requirements in terms of soft- and hardware, and a tutorial-style introduction to basic operations are presented in section 4. More advanced features and their operation are introduced in section 5. Section 6 discusses compatibility with the Ngram Statistics Package (Banerjee and Pedersen, 2003) and the guide concludes with some final comments and license information. An appendix lists available options in each of the constituent programmes of the NGP.

2 KEY FEATURES

The key features of the NGP are simple enough:

- creation of word n-gram lists out of input text, with n-gram frequencies (global frequencies as well as number of input documents where the n-gram occurs)
- combination of large numbers of lists (of one *n*) into a single list
- unicode support
- support for processing of reasonably large corpora (depending on hardware)

[1] There are a number of introductory texts on these topics, for example Sobell and Seebach (2006) or Muster (2003).

- support for processing of annotated corpora

The main function is the production of word n-gram lists out of input text, where word n-grams are word sequences of length n, so a 2-gram is a sequence of two words, a 3-gram one of three, etc. Lists of any length n-grams (within reason) can be produced (and combined), but each list will only contain n-grams of one particular n and can only be combined with lists of the same n (For software that will consolidate lists of n-grams of different n, see the open-source SubString package[2], cf. also O'Donnell, 2011). The way NGP produces n-gram lists is illustrated in table 1 where 2-grams 1 to 7 (upper part) and 3-grams 1 to 6 (lower part) are produced from an input text. N-grams are created by moving a window of size n progressively through the text and listing resulting n-grams. Each n-gram is then gathered into a list and their frequencies indicated (in table 1, the frequencies would all be 1 since each n-gram occurs only once).

		This	is	an	example	–	an	idealised	example
2-grams	1	This	is						
	2		is	an					
	3			an	example				
	4				example	–			
	5					–	an		
	6						an	idealised	
	7							idealised	example
3-grams	1	This	is	an					
	2		is	an	example				
	3			an	example	–			
	4				example	–	an		
	5					–	an	idealised	
	6						an	idealised	example

Table 1: Splitting of text into n-grams.

One of the principal uses of n-gram lists produced by the NGP is in the context of the extraction of formulaic sequences, multi-word expressions and the like out of corpus data. Here, n-gram lists serve as raw material to an identification of relevant items.

3 SOME BACKGROUND

I put the NGP together while working on my PhD thesis to help me study diachronic change in common expressions (i.e. formulaic sequences) using corpus data. The NGP is a branch of the well-established Ngram Statistics Package (NSP) and thus shares a significant portion of the code base with that project, specifically with versions 1.09 (by Ted Pedersen, Satanjeev Banerjee and collaborators, cf. Banerjee and Pederson, 2003) and 1.10 (a re-write of the NSP by Bjoern Wilmsmann, cf. Wilmsmann, 2007). If settings are used that precisely emulate the workings of the NSP, identical output can be achieved. The NGP, however, is now a entirely separate package and does not interfere with installations of the NSP. For the benefit of those already familiar with the NSP, the main differences between the two software packages lie in the focus of the NSP on the calculation of statistical measures of

[2] http://buerki.github.com/SubString/

association of listed n-grams (hence the 'statistics' in the name), whereas the NGP focuses on the ability to process much larger amounts of data and on the processing of multilingual data, including corpus material written in non-Roman script. Additionally, it makes it possible to track document frequency (i.e. the number of documents in which an n-gram occurs) in addition to overall frequency which can be of great importance for assessing the distribution of particular n-grams. The NGP does not include a statistics module (though it is possible to use the NSP statistics module on output data if a certain combination of options is used, see section 6).

4 INSTALLATION AND BASIC OPERATION

Requirements and Installation

The NGP has the following software requirements:

- perl version 5.12 or later
- bash version 3.2 or later (needed for auxiliary scripts)

Typically, these will already be installed on machines running a Unix or Linux operating system (including Apple's OS X) but can also be installed on machines running versions of Microsoft Windows through the Cygwin project.[3] In terms of hardware, the processing of large amounts of data is extremely demanding on Random Access Memory (RAM) because, for efficiency reasons, lists are assembled in memory. Working with a corpus of a size of around 10 million words, a minimum of 8GB of RAM are deemed necessary for reasonable performance and for larger corpora, requirements are higher on a roughly similar scale. There are ways to drastically reduce RAM usage and therefore process very large amounts of data, but the trade-off is sharply increased processing time. For details, see section 5, below, especially table 2. The NGP displays and outputs unicode and will therefore work most reliably if the environment is set to UTF-8 (to check, type `locale`) and the terminal also displays text in a UTF-8 encoding (this is typically set in the preferences of the terminal software).

To install the software, the following procedure should be followed: after downloading and decompressing (unzipping) Ngramprocessor, open a terminal window, navigate into the Ngramprocessor directory and enter the standard installation commands at the prompt:

```
perl Makefile.PL
make
make test
make install
```

If the last command brings up an error, type `sudo make install` instead and enter the administrator password for the computer when prompted. After this last command, the installer will display the directories where the software was installed which are the standard Perl installation locations. It may be worth holding on to this information in the rare case that the following check fails. If all went well, entering the command `list.pl -V` should bring up the version number of list.pl. If instead, an error message is returned complaining

[3] `http://cygwin.com/index.html`.

that the command was not found, it is likely that the install location needs to be added to $PATH.[4] The following should have been installed:

- `list.pl` – a programme to produce n-gram lists
- `unify.pl` – a programme to combine multiple n-gram lists
- `Ngramprocessor.pm` – a module whose functions are called by list.pl and unify.pl
- `multi-list.sh` – an auxiliary shell script automating the production of a large number of n-gram lists by calling list.pl repeatedly
- `split-unify.sh` – an auxiliary shell script facilitating the memory-efficient combination of large numbers of n-gram lists by calling unify.pl on sub-parts of the lists to be combined

For any of the programmes installed, help is available by calling the -h option, or, for the Perl programmes only, calling the manual entry. For example, `list.pl -h` will bring up up relevant information including all the options available and how the programme should be called. `man list.pl` will display a manual entry for `list.pl`.

Basic Operation

To illustrate the basic operation of the NGP, the example files found in the test directory inside the Ngramprocessor directory will be used and the following examples assume that the current working directory is the `test` directory.

The programme list.pl

To create an n-gram list, `list.pl` is used. It requires the following pattern

- `list.pl [OPTIONS] OUTFILE IN-DIR/FILE+`

where OUTFILE is the name of the list to be produced, IN-DIR/FILE+ is either one or several input files or a directory with input files in it. The following command will produce an n-gram list called out1.lst out of the input file text1.txt:

```
list.pl out1.lst text1.txt
```

Examining the output list (`out1.lst`), the beginning of which is shown in figure 1, a number of things are noticed. On the first line, the total number of n-gram tokens is shown. This is followed by the list of n-gram types and their frequencies, one per line. By default, 2-gram lists are produced and by default the words in n-grams are separated by an interpunct (or middle dot). The number after the final interpunct is the frequency of the n-gram. Although invisible, a single space follows the frequency on each line. Also by default, n-grams across line breaks (i.e. across the newline character) are excluded. If the -l option is passed, n-grams spanning line breaks are included in the list as well.

[4] To add it permanently, refer to the documentation of your system, but to add it temporarily, the following should work: `export PATH="/path/to/install/directory:$PATH"`

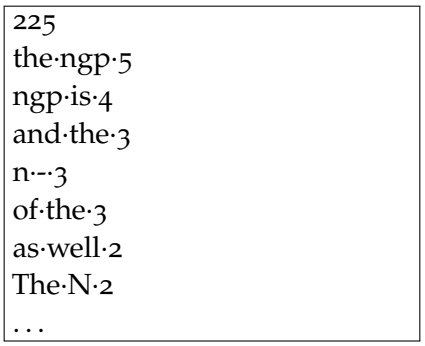

Figure 1: The first few lines of out1.lst

To see a bit more of what is going on, the `-v` option can be used. To specify the *n* of the n-grams, `-n` is used and we can vary the separator using `-p`[5]. The full range of available options to the programmes included in the NGP is listed in the appendix below and is available for each programme by calling the -h (help) option.

The following command produces a 6-grams list out of `text1.txt` using <> as separator:

```
list.pl -v -n 6 -p '<>' out2.lst text1.txt
```

Instead of passing single input files, we can also pass directories with input files in them. The following will produce a single output list of 3-grams (out3.lst) from both text files in the directory `files`.

```
list.pl -v -n 3 out3.lst files
```

If only n-grams of a certain frequency are to be included in the list, the -f option to `list.pl` can be used to specify the minimal frequency an n-gram has to occur with in order to be included in the list. The following command will only include n-grams that occur with a minimal frequency of 2. The effect is merely to exclude the lower frequency n-grams from the list; the total number of n-gram tokens displayed at the top of the list remains the same.

```
list.pl -vf 2 -n 3 out4.lst files
```

The programme unify.pl

So far, only global frequencies were listed for each n-gram, that is the frequencies across *all* input files. If the number of documents in which an n-gram occurs should additionally be listed, n-gram lists first need to be produced for each document, and then combined using `unify.pl`.

- `unify.pl [OPTIONS] IN-DIR/FILE+`

Using the commands below, first two 3-gram lists are created, one each of `text1.txt` and `text2.txt`. Then they are unified using `unify.pl`.

[5] Although the separator can be varied freely in `list.pl`, two things should be kept in mind when choosing a separator symbol: 1) it must be a symbol that is NOT found at all in the input texts (otherwise confusion will result). If necessary, input files might need to be checked and offending characters removed before processing. 2) `unify.pl` automatically recognizes three types of separators: the interpunct (·), the diamond (<>), and the underscore (_) and so it is probably best to stick to one of those

```
list.pl -v -n 3 out5.lst text1.txt
list.pl -v -n 3 out6.lst text2.txt
unify.pl -vd outcombined.lst out5.lst out6.lst
```

If the output file (outcombined.lst) is now examined, the document count appears two spaces after the global frequency of each n-gram. The -d option passed to `unify.pl` was responsible for adding the document count – if it is left out, an output file that is identical to `out3.lst`, above, should be the result:

```
unify.pl -v out3bar.lst out5.lst out6.lst
```

In actuality, n-grams in `out3.lst` and `out3bar.lst` are listed in a slightly different order, but n-grams, frequencies and total n-gram token numbers are identical.

The basic operation of the core programmes in the NGP as illustrated in this section is augmented by two auxiliary scripts and further options discussed in the next section.

5 MORE ADVANCED OPTIONS

multi-list.sh, split-unify.sh and workflows for large corpora

When dealing with large amounts of corpus data from which n-gram lists with global and document frequencies need to be derived, it is not feasible to go about the task by running list.pl an unify.pl in the manner shown above. Depending on how the corpus is structured, the first step in a workflow deriving n-gram lists from the entire corpus using the NGP is to divide the corpus into individual constituent documents. Unless any existing corpus annotations must be preserved in n-grams extracted (see below for what to do in such cases), any markup or annotation (except for line breaks which are relevant to n-gram creation[6]) should be removed.

In a second step, n-gram lists are then prepared for each corpus document. Unless individual corpus documents are extremely large (nearing five million words of text), there should be no issues with RAM limits at this stage. To facilitate this step, the shell script `multi-list.sh` can be used to automatically run list.pl on all corpus documents in a given directory (note that although list.pl also accepts directories with multiple files as input, it creates a total of *one* output list, rather than one output n-gram list for *each* of the corpus documents). `multi-list.sh` is nothing more than a wrapper that calls on `list.pl` to produce n-gram lists for each of the input files. The way `multi-list.sh` is called closely parallels the way `list.pl` itself is called (it also has most of the same options available):

- `multi-list.sh [OPTIONS] OUT-DIRECTORY IN-DIRECTORY`

OUT-DIRECTORY is the directory into which the output is placed, IN-DIRECTORY is a directory with files to be processed (unlike `list.pl`, `multi-list.sh` does not accept a list of files to be processed – only a directory can be specified as input). `multi-list.sh` places a new directory into the OUT-DIRECTORY which contains the output list(s).

The following command places a new directory in the current working directory (represented by the dot) and fills it with one 3-gram list for each of the files present in the directory `files`. The exact name of the output directory depends on options passed

[6] See -n or --newline option to `list.pl` and `multi-list.sh`

to `multi-list.sh`, but it will always contain within it the *n* of the n-gram lists it contains, followed by the word 'comp' for comprehensive (if a single list is contained) or 'per_doc' (if one list per input document is contained within it).

```
multi-list.sh -vdn 3 .  files
```

The -d option makes sure that one output list is produced for *each* input document. If the -d option is not invoked, `multi-list.sh` operates much like `list.pl` in that it produces a single n-gram list from all the input files. As before, -v and -n 3 invoke verbose processing and the production of 3-grams (rather than default 2-grams) respectively.

The next step is to combine all those single lists (depending on corpus size, this may be thousands of lists, though in our test data it is only two lists) into one overall list showing both the global frequency and the document frequency for each n-gram. This could of course be accomplished using `unify.pl` as shown above. However, here the RAM bottleneck is encountered: the combination of thousands of n-gram lists in memory fills a very large amount of RAM space (a multiple of the size of the files to be processed). `split-unify.sh` seeks to address this problem by first splitting lists to be combined, and then combining each section of each list before aggregating the combined sections into a full list.[7] The splitting is done in such a way that n-grams which potentially need combining are found in the same section of all input lists, whereas n-grams that will not need combining are in different sections of the input lists. For example, sections A may contain all n-grams starting with the letter *A* in all the source lists, whereas sections B may contain all the n-grams starting with the letter *B* in all the source lists. No n-gram starting with *A* will ever need combining with one starting with *B* (or rather their frequencies will never need combining), but among those n-grams in A-sections there will be many that are identical across A-sections of the various lists to be combined and will therefore need their frequencies summed while appearing as just one entry (one type) on the combined list. I refer to this type of splitting as alphabet splitting.[8] `split-unify.sh` is called in the following manner:

- `split-unify.sh [OPTIONS] IN-DIRECTORY`

To combine the n-gram lists in the directory that was produced by the command we called earlier, the following command is used:

```
split-unify.sh -vd 3.per_doc
```

After this command has run, the directory `3.per_doc` will be prefixed with `indiv_lists_`, showing that it only contains individual un-unified lists and the unified list (named after the name of the input directory + the extension .lst) will appear in the current working directory. The format of the output list, as shown in figure 2, is slightly different from the output in figure 1: frequencies (global n-gram frequencies and document count) are each preceded by a tab for easier reading (this sort of tidying of the output list can be suppressed by passing the -u option to `split-unify.sh`). The -d option ensures that document counts are

[7] This bottleneck is of course also encountered if we were to produce just one n-gram list from thousands of documents or from one very large document using `list.pl` or `multi-list.sh`. This is why it may be a good idea to have `multi-list.sh` produce lists for each document first and combine those lists into a single list subsequently *even if* no document count is required.

[8] Despite the NGP's aim to be multi-language aware, `split-unify.sh` is currently only operating alphabet-splitting for input using Roman characters. Input documents with text written in other character sets do not currently benefit from alphabet-splitting, although other functions are unaffected.

524		
n·-·grams·	5	2
the·ngp·is·	4	2
of·the·ngp·	4	2
of·the·nsp·	3	1
to·the·ngp·	2	2
of·n·-·	2	2
n·-·gram·	2	2
grams·and·their·	2	2
The·N·-·	2	1
N·-·Gram·	2	1
...	...	...

Figure 2: The first few lines of 3.per_doc.lst

included. If additionally the -i option is invoked, the directory with individual input lists will be deleted and only the unified output list in the current working directory remains (which can save a lot of disk space). By default, `split-unify.sh` employs a 47-way alphabet split. A finer splitting into 84 sections (and therefore a further reduction in memory use) can be achieved by invoking the -b option (for big). Since splitting will take some time and requires a lot of read/write activity, the alphabet splitting should only be invoked where necessary. It can be turned off entirely by passing the -s (for small) option. The -m option (for minimum memory) can be invoked to reduce RAM usage to a minimum. It uses an alternative algorithm which assembles the different lists and consolidates them line-by-line, thus reducing RAM requirements drastically, while taking more processing power and typically significantly more time. Table 2 shows the processing modes available in `split-unify.sh`, their function and use. The choice of the ideal mode of operation depends on the amounts of data to be processed and the hardware (both RAM and processor speed) used. Generally speaking, if 5 million words or fewer are processed, the -s option will usually be the best choice. If, during processing, the available RAM is exhausted and the operating system starts writing page-outs to disk, the programme becomes inoperable and will stall (Most operating systems have a way of monitoring RAM usage and this can be used to get an idea of how much memory is being used for a particular run). If this happens, either the -b option must be used, or, if memory requirements are still too high, the -m option should be invoked.

In this manner, `split-unify.sh` can reduce memory requirements notably and thus facilitate the processing of larger amounts of data if necessary.[9] The auxiliary scripts `multi-list.sh` and `split-unify.sh` therefore aid the processing of large amounts of data and facilitate the handling of a workflow involving a large number of documents.

[9] The smallest amount of memory is used if `split-unify.sh` is operated using the -m option, if texts without annotation are processed (see below) and a one-character n-gram separator is used (both '<>' and '·' take more memory because they are binary symbols).

Option	Processing mode	Use
-s	(small) no split, employing `unify.pl` directly to process data	Suitable for smaller data-sets (up to around 5 to 10 million words, depending on hardware), necessary if statistical measures of association are to be computed subsequently using the Ngram Statistics Package
none	47-way split; combination with `unify.pl`	Suitable if RAM usage needs to be reduced; not suitable for processing data in non-latin scripts
-b	(big) 85-way split; combination with `unify.pl`	Further reduced RAM requirements, slightly increased processing demands; not suitable for non-latin scripts
-m	(minimal memory) alternative algorithm without calling `unify.pl`	Minimal RAM requirements, increased processing demands. This will take significantly longer to process; suitable for data in any script

Table 2: Processing modes available in split-unify.sh.

Token definitions and Stoplists

When producing word n-grams out of text, a decision needs to be taken as to which characters should be part of n-grams, and which characters should not. The former are defined as tokens, the latter as space. When setting up a token definition, the set of characters that should be treated as part of n-grams are defined, the remaining characters, which commonly includes the space character and tabs but can also include punctuation marks and other special characters, are automatically treated as space. The default token definition used by NGP is displayed by calling -t show (or --token show) in either `list.pl` or `multi-list.sh`:

```
list.pl -t show
multi-list.sh -t show
```

Token definitions are presented in the format of Perl regular expressions. To set a different token definition, a text file containing token definitions needs to be prepared and passed to either `list.pl` or `multi-list.sh` using -t FILE, where FILE is (the path to) a text file containing the token definition. The token definition should follow the format of starting with a slash, then listing the tokens, each separated by a pipe (vertical bar symbol) and then ending in a slash, for example /\w+|-|'|&|§|%|\/|\+|ß/. Characters that have special meaning in Perl need escaping to get their literal meaning. \w+ refers to all alphanumeric characters, including underscores and any of the characters following the \w+ symbol, while being treated as tokens, are treated as *separate* from the alphanumeric tokens, such that *bus-stop* is a 3-gram (with *bus*, -, and *stop* representing separate constituents).

The NGP features the option of specifying a stop list of words that have the effect of suppressing the listing of certain n-grams. There are two modes: in ABSOLUTE mode, all n-grams are excluded that contain a word from the stop list. In ADDITIVE mode, only n-grams are excluded that are composed entirely of stop-listed words. A stop list can be supplied to either `list.pl` or `multi-list.sh` by passing the option -o and the stop list (here, the example stop list in the test directory is used):

```
list.pl -o stop_en out7.lst files
multi-list.sh -o stop_en .  files
```

Stop list files need to be formatted such that the mode is indicated in the first line (either as @stop.mode=ABSOLUTE or @stop.mode=ADDITIVE), and stop words then listed, one per line, between `/^` and `$/`. An example is shown in figure 3.

```
@stop.mode=ADDITIVE
/^the$/
/^of$/
/^to$/
/^and$/
/^a$/
/^in$/
...
```

Figure 3: The first few lines of the stop_en stop list

Working with annotated corpora

It may be necessary or desirable to preserve certain corpus annotations (such as part of speech tags or a morphological information) in the n-gram lists produced by the NGP. Generally, since this will increase memory requirements, annotation would ideally be as economic as possible. In order to carry annotation over from the source text to the n-gram lists, it needs to be associated with word tokens. Since only strings of alphanumeric characters are treated as uninterrupted n-gram constituents (see discussion of token definition, above), the underscore must be used to link annotation to the constituent as shown. Input files with annotation attached in this way can be run through `list.pl` or `multi-list.sh` in the manner described above. The file `text1_annot.txt` in the `test` directory is provided as an example of an annotated input text. It contains part of speech tags, attached to word tokens with underscores. Processing in the usual way with `list.pl` yields an n-gram list in which annotation is preserved as shown in figure 4.

```
225
the_DT·ngp_NN·is_VBZ·3
of_IN·the_DT·ngp_NN·3
The_DT·N·-·2
-·Gram_NP·Processor_NP·2
N·-·Gram_NP·2
n·-·grams_NNS·2
the_DT·Ngram_NP·Statistics_NPS·1
methods_NNS·but_CC·others_NNS·1
is_VBZ·structured_VBN·as_RB·1
...
```

Figure 4: The first few lines of a 3-gram list with part-of-speech annotation preserved.

6 COMPATIBILITY WITH THE NGRAM STATISTICS PACKAGE

As mentioned above, the NGP does not include a statistics module and does not therefore produce the type of frequency information that allows the computation of contingency tables such as are required for the calculation of statistical measures of word association

like the MI-score or log likelihood scores. Generally, if those are required, use of the NSP rather than the NGP may be preferable. However, it is possible to pass parameters to `list.pl`, `multi-list.sh`, `unify.pl` and `split-unify.sh` in a manner that allows output n-gram lists to be processed subsequently, using the NSP's `statistic.pl` programme, to calculate statistical association measures. Naturally this requires the installation of the NSP alongside the NGP (since they are separate packages, they can both be installed on a single system). For compatible lists to be created, `list.pl`, `multi-list.sh` must be passed the -a option to calculate the additional frequency numbers and the -s option must be passed to `split-unify.sh`. Document frequencies must *not* be included. In addition, the NSP's default separator symbol '<>' must be specified as the separator (using `list.pl` and `multi-list.sh`'s -p option). It should then be possible to use the lists so produced as input to the NSP's `statistic.pl` programme. The following command, for example, will produce a list that should be processable by the NSP's `statistic.pl`:

```
multi-list.sh -vap '<>' .  files
```

Output files for all the example commands used in this manual are provided in the goldstandard_lists directory inside the test directory. These can be used to verify operation of the NGP-installation. Differences are most likely to arise if different encodings are set in the environment. The gold standard lists were produced using the locale en_GB.UTF-8.

7 COMMENTS, LICENSE AND DISCLAIMER

The NGP is based on the tried and tested code base of the NSP and has been extensively tested taking NSP-output as the gold standard where identical output is expected. Nevertheless, results should always be checked for rough plausibility at the very least. I am conscious that, in its current implementation, the NGP lacks many desirable features and others are implemented in less than elegant ways which is why the NGP is offered as a beta release. Even in this state, however, the NGP has been useful to me and may be so to others who should feel free to adapt it to their needs and expand it. As the software is designated BETA, certain features of the user interface can change, so users are advised to read release notes where any such changes will be discussed.

The NGP is licensed under the GNU General Public license: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The NGP is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with the NGP. If not, see `http://www.gnu.org/licenses/`.

Finally, the NGP repository currently lives at `http://buerki.github.io/ngramprocessor/`, where there is the possibility to get in touch, report issues and sign up to be notified of updates.

A Appendix: synopsis and list of options

This appendix first gives a synopsis of the programmes included in the NGP and then lists and explains the options available in each. A summary of this information is available also by passing the -h option to the respective programme, or, for Perl programmes, by calling up the manual (e.g. by calling `man list.pl` or `perldoc unify.pl`).

Synopsis

- `list.pl [OPTIONS] OUT-FILE {IN-FILE+|IN-DIRECTORY}`
- `multi-list.sh [OPTIONS] [N-SIZE] OUT-DIRECTORY IN-DIRECTORY`
- `unify.pl [OPTIONS] OUT-FILE {IN-FILE+|IN-DIRECTORY}`
- `split-unify.sh [OPTIONS] IN-DIRECTORY`

Note: items in square brackets are optional, in curly brackets either one or the other item can be supplied; + means one or more of these can be supplied.

Options in list.pl and multi-list.sh

-a --all_freq_combos	Produce figures for all possible frequency combinations (required if NSP's statistics.pl is to be used to calculate word association measures).
-h --help	Prints a help message.
-l --newline	Includes n-grams spanning across the new-line character. These are excluded by default.
-n --ngram N	Creates n-grams of N words each. N = 2 by default.
-o --stop FILE	Uses FILE as a stop list and removes n-grams containing at least one stop word (in ABSOLUTE mode) or only stop words (in ADDITIVE mode). Stop words should be declared as Perl Regular expressions in FILE.
-p --separator SEP	Uses SEP as the symbol(s) separating the words of an n-gram. By default, words in n-grams are separated by an interpunct (middle dot), but this can be changed using this option, for example to <> or _. Care needs to be taken that the separator symbol does NOT occur in the input texts, otherwise confusion will result.

-t --token FILE	Uses regular expressions in FILE as token definition. To display the default values, write *show* instead of FILE.
-v --verbose	Prints information as the programme runs.
-V --version	Prints the version number and copyright information.
-w --window N	Sets window size to N. By default, the window size is the same as the *n* of the n-grams; by specifying a larger window size, all combinations of *n* words (in the order in which they occur) inside the window will be included in the n-gram list (i.e. if 2-grams are extracted in a window size of 3, the n-grams produced will be of word 1 and word 2, word 2 and word 3, *as well as* word 1 and word 3).

Note: the long version of the options (starting with two hyphens) can only be used in `list.pl`.

Options exclusive to list.pl

-e --encoding ENC	Handles input and output files with the given character encoding. Default is utf8, but if necessary a different encoding can be forced using this option. A suitable token definition must also be supplied as the standard token definition is in utf8 (use -t option). Non-utf8 is not generally recommended and is here only provided for compatibility with legacy data sets. It is better to convert the data to utf8 as non-utf8 encodings have not been widely used in testing.
-f --frequency N	Suppresses the listing of n-grams that occur less than N times.
-s --set_freq_combo FILE	Uses the frequency combinations in FILE to decide which combinations of tokens to count in a given n-gram. By default, only the frequency of the complete n-gram is kept track of. This option is provided for compatibility with the NSP only; see the NSP's documentation for details.
-d --display_freq_combo	Prints out the frequency combinations used. If frequency combinations have been provided through –set_freq_combo above, these are output; otherwise the default combination is output.

Options exclusive to multi-list.sh

-d	Produces ONE output list per ONE input document.
-e ARG	Causes the output directory to have the ARG prepended to its name.
-f ARG	Causes the output directory to have the ARG appended to its name.
-H	Replaces hyphens (-) with 'HYPH' in output lists.

Options in unify.pl and split-unify.sh

-d --doc_count	Adds a count of how many documents (input files) each n-gram appears in. This figure is appended to the end of each line of output.
-h --help	Displays help.
-V --version	Displays version information.
-v --verbose	Displays processing information as the programme runs.

Options exclusive to unify.pl

-D --display_freq_combo	Shows the current frequency combinations setting.
-e --encoding ENC	Handles input and output files with the given character encoding. Default is utf8, but if necessary a different encoding can be forced using this option. Non-utf8 is not generally recommended and is here only provided for compatibility with legacy data sets. It is better to convert the data to utf8 as non-utf8 encodings have not been widely used in testing.
-s --set_freq_combo FILE	Uses the frequency combinations in FILE to decide which combinations of tokens to count in a given n-gram. If n-gram lists with more frequency combinations than the frequency of the n-gram are needed (such as when statistics of association should be calculated), this option MUST be used, together with the appropriate file. The lists to be combined must contain the necessary frequencies, of course.

Options exclusive to split-unify.sh

-i	Discards directory with individual lists (otherwise this directory is prefixed with `indiv_lists` and retained).
-m	Causes lists to be combined using an algorithm that minimises RAM requirements.
-n	Causes no numbers for calculation of statistical measures of association to be calculated even if present in the input files and no alphabet splitting is in effect.
-s	Uses processing mode 'small' by passing data directly to `unify.pl`.
-b	Runs big version for large amounts of data (84-way alphabet split).
-u	Leaves output lists untidy, that is, suppresses the insertion of tabs between the n-grams, their frequencies and any document counts.

Banerjee, S., & Pedersen, T. (2003). 'The Design, Implementation and Use of the Ngram Statistics Package'. In *Proceedings of the 4th International Conference on Intelligent Text Processing and Computational Linguistics*. Mexico City.

Muster, J. (2003). *Introduction to Unix and Linux*. New York: McGraw-Hill.

Sobell, M. G., & Seebach, P. (2006). *Practical Guide to Unix for MAC OS X Users*. Upper Saddle River, NJ: Prentice Hall.

O'Donnell, M. B. (2011). 'The Adjusted Frequency List: A method to produce cluster-sensitive frequency lists'. *ICAME Journal*, 35 (April). Retrieved from: http://icame.uib.no/ij35/Matthew_Brook_ODonnell.pdf

Wilmsmann, B. (2007). *Re-write of Text-NSP.* (Original work published 2007). Retrieved from https://github.com/BjoernKW/Publications/blob/master/Re-write_of_Text-NSP.pdf