Toolkit 2.0 Guide

Contents

1 Overview

This guide will provide a general overview of the Toolkit, its various functions, outputs, and common usage. For more details and relevant files/scripts, see the GitHub. If you don't have access to the repository and should, please email me at cjsolo5@umd.edu. Please also email me if you find any typos, errors, or other areas where the guide could be improved.

1.1 Motivation

The English Sublexical Toolkit is a package of R functions and datasets for processing and understanding English words at various levels of analysis. It currently contains supports 22,492 unique English words, and contains functionality for computing all relevant measures for any provided set of valid words and pseudowords (words containing valid sequences of respective graphemes and phonemes)[1].
The Toolkit also has its own conventions for pronunciations; see Appendix 3.1 for reference. This serves to simplify the representation of phonological units in the R code. Scripts exist to translate between the Toolkit convention and the IPA convention; these will be discussed later. From the original paper:

> "This work introduces the English Sublexical Toolkit, a suite of tools that utilizes an experience-dependent learning framework of sublexical knowledge to extract regularities from the English lexicon. The Toolkit quantifies the empirical regularity of sublexical units in both the reading and spelling directions (i.e., grapheme-to-phoneme and phoneme-to-grapheme) and at multiple grain sizes (i.e., phoneme/grapheme and onset/rime unit size). It can extract multiple experience-dependent regularity indices for words or pseudowords, including both frequency indices (e.g., grapheme frequency) and conditional probability indices (e.g., grapheme-to-phoneme probability). These tools provide (1) superior estimates of the regularities that better reflect the complexity of the sublexical system relative to previously published indices and (2) completely novel indices of sublexical units such as phonographeme frequency (i.e., combined units of individual phonemes and graphemes that are independent of processing direction)." [2]

We'll go over each of these units, as well as the methods for obtaining them, for both pre-computed words in the Toolkit as well as pseudowords

2 Getting Started

It will be easier to follow along with the following sections if you have an active R environment with the Toolkit loaded. Navigate to a directory of your choice and either open the R REPL or a new R file, and set your working directory to the folder containing the `Toolkit_v2.0.RData` file:

```
setwd("path/to/your/directory")
```

[1] A "valid" phoneme-grapheme mapping is one that is supported in the Toolkit at the desired level. Novel mappings are added as they are encountered in the corpus, which continues to grow with Toolkit updates; if a provided mapping isn't supported by the Toolkit, it is highly likely that the mapping is invalid in the English language.

[2] The English Sublexical Toolkit: Methods for indexing sound–spelling consistency. `https://doi.org/10.3758/s13428-024-02395-3`

Then simply call

```
load("Toolkit_v2.0.RData")
```

It should take a few moments for the file to load in to your environment; once it completes and a new terminal prompt appears, you are ready to use the Toolkit. I personally use VSCode to work with R, so screenshots of scripts and the environment will be taken from there; using RStudio or any other desired environment is completely fine.
With the RData file loaded, you'll be able to see all datasets and functions; if however you'd like to look at only the code directly, open *PROCESSING_SCRIPTS.R*.

3 Measures

For the various measures computed by the Toolkit, there are two discriminating categories: *weight* and *grain size*. For each combination of these, a word has various *probability* and *frequency* measures. Words also have *parsing probability* (PP) measures.

3.1 Weight

Measures in the Toolkit are either *type-weighted* (where frequency of the word in the corpus is NOT taken into account) or *token-weighted* (where frequency of the word in the corpus is provided by the user and taken into account). In the Toolkit's data, preprocessed datasets involving the suffix "_freq" indicate token-weighted measures; if "_freq" is not present in the dataset name, that indicates type-weighted measures.

Measures can be further weighted based on the position of the mapping in the word; by default, this is how measures are calculated (see the relevant functions in the Functions section). Tables named with the suffix "_noposition" indicate that their measures disregard the position of the mapping in the word when accounting for frequency.

To illustrate this:

- `scored_words_PG` has data for all words mapped at the phoneme-grapheme level, type-weighted and taking into account mapping position.
- `scored_words_PG_freq` has data for phoneme-grapheme-mapped words, token-weighted and taking into account mapping position.
- `scored_words_PG_noposition` has data for phoneme-grapheme-mapped words, type-weighted and disregarding mapping position.
- `scored_words_PG_freq_noposition` has data for phoneme-grapheme-mapped words, token-weighted and disregarding mapping position.

The case is analogous for all other datasets. By convention, in token-weighted position-disregarding datasets the "_freq" will come before the "_noposition" in dataset naming.

3.2 Grain Size

The Toolkit currently supports four different "grain sizes," or units of granularity for understanding phoneme-grapheme mappings in words. These are indicated in table names as the following:

- `PG:` phoneme-grapheme level.

  The phoneme-grapheme grain size is the smallest possible level of analysis for a word. A phoneme is a single phonological unit, or sound, and a grapheme is the collection of one or more letters corresponding to the phoneme in a word.

- `OR:` onset-rime level.

  The *onset* is the initial consonant or consonant cluster of a word (and its corresponding phonological representation), while the rime is the subsequent vowel and remainder of the word.

- `OC:` oncleus-coda level.

  The *oncleus* is the combination of the *onset* (initial consonant cluster) and *nucleus* (internal sound between the initial and final consonant cluster). The OC level considers the onset and nucleus together; hence, oncleus.

- `ONC:` onset-nucleus-coda level.

  Consideration of the onset, nucleus, and coda separately.

3.3 Probabilities and Frequencies

The Toolkit provides five different measures (and each can be weighted by any of the options previously discussed, for any of the grain sizes). These are:

- `PG:` spelling consistency; $p(G|P)$, the probability of a spelling G given phonemes P
- `GP:` reading consistency; $p(P|G)$, the probability of phonemes P given a spelling G
- `PG_freq:` the $\log_{10}$ frequency of the correspondence between pronunciation P and spelling G, identical in both the spelling and reading directions
- `P_freq:` the $\log_{10}$ frequency of the phoneme P
- `G_freq:` the $\log_{10}$ frequency of the grapheme G

3.4 PP Measures

PP, or *parsing probability* measures, describe the likelihood that a sequence of letters will be grouped into a specific grapheme. The Toolkit function to calculate these is currently a work-in-progress. These can also be weighted.

See *data/precalculated/pp_measures.csv* for the set of PP measures that have been pre-computed for the 2.0 Toolkit corpus.

3.5 Measure Naming

Measures are named according to the following scheme:

<grain_size>_<weight>_<measure>.<statistic>

Where

- *grain_size* corresponds to one of the grain sizes mentioned above in Section 3.2,
- *weight* corresponds to a weighting style as discussed in Section 3.1 (note for the default weighting with type-weighting and position considered, this field will be "_default"),
- *measure* corresponds to one of the probabilities or frequencies outlined in 3.3, and
- *statistic* corresponds to one of the following: mean, median, max, min, sd (standard deviation).

To take a few examples:

- PG_default_PG.mean

  The average conditional probability of phoneme-to-grapheme mappings for position-constrained phoneme-grapheme units, not weighted by corpus frequency.

- ONC_noposition_GP.max

  The highest conditional probability of grapheme-to-phoneme mappings for position-unconstrained onset-nucleus-coda units, not weighted by corpus frequency.

- OC_freq_PG_freq.min

  The lowest frequency of phoneme-grapheme units for position-constrained oncleus-coda units, weighted by corpus frequency.

- OR_freq_noposition_P_freq.median

  The median frequency of phoneme units for position-unconstrained onset-rime units, weighted by corpus frequency.

There are currently 4 PP measures available, and these are type and token-weighted mean and min values. These are named:

pp_mean, pp_min, pp_freq_mean, pp_freq_min

4 Datasets

With the Toolkit loaded in your R environment, call `ls()` in your terminal to see the full list of datasets available from the Toolkit.

4.1 General Datasets

`wordlist_v2_0`: Contains all English words in the Toolkit.

`<type>_mappings`: Lists of acceptable phoneme-grapheme mappings for various positions. Replace `<type>` with your desired position to obtain the relevant list. The options are *syllable_initial*, *syllable_medial*, *syllable_final*, *word_initial*, and *word_final*. For instance, calling `head(syllable_initial_mappings)` will show us the first six valid phoneme-grapheme mappings in the syllable-initial position (recall that all phonemes are represented using the Toolkit's conventions):

```
> head(syllable_initial_mappings)
  phoneme grapheme
1       @        a
2       @      a_e
3       @      i_e
4       5        i
5       5      i_e
6       O       ou
```

4.2 Mapping Data

The Toolkit has pre-computed datasets for all word mappings at various grain sizes, as well as all mapping probabilities and frequencies for each combination of grain size and weight. The `all_tables` family of datasets contains all probabilities and frequencies for each word individually at the desired grain size and weight (see the Measures section), while the `all_words` family of datasets contains all phoneme-grapheme mappings for all words at the desired grain size.
This will be expanded upon soon. TODO

4.3 Word Data

The Toolkit has pre-computed datasets for all combinations of weight and grain size, containing all measures for these parameters for each of the 22,492 words. These are the `scored_words` family of datasets. See the Measures section for relevant information on what various options for these datasets means, and what measures are provided within.

5 Functions

With the Toolkit loaded in your R environment, call `ls()` in your terminal to see the full list of functions available from the Toolkit; these will be listed with the datasets discussed above.

The Toolkit has a set of functions allowing one to both 1) reproduce the Toolkit's results and 2) calculate measures for any chosen set of valid phoneme-grapheme mappings at a desired grain size.

5.1 Mapping Functions

`map_PG(spelling, pronunciation, map_progress=FALSE)`: Given a valid *spelling* and *pronunciation* (in the Toolkit format), maps the two at the phoneme-grapheme level, showing the live progress as desired. Accepts arguments as lists (enabling words to be batch-mapped). One could, for instance, reproduce the `all_words_PG` dataset by calling the following:

```
all_words_PG <- map_PG(spelling=wordlist_v2_0$spelling,
                       pronunciation = wordlist_v2_0$pronunciation)
```

Or simply map a single word to a pronunciation:

```
map_PG("toolkit", "tUlk1t")
```

This would yield a result of the form:

```
> map_PG("toolkit", "tUlk1t")
[[1]]
[[1]][[1]]
  X6 X5 X4 X3 X2 X1
1  t  U  l  k  1  t
2  t oo  l  k  i  t
3  1  3  4  2  3  5


[[2]]
     [,1]
[1,] TRUE
```

Notice that there will always be two lists of results returned from this (and all other `map`) function; one with the mapping as a dataframe, and the other indicating the success. For all other `map` functions, usage is the same.

`map_ONC(spelling, pronunciation, map_progress=FALSE)`: Maps a valid *spelling* and *pronunciation* at the onset-nucleus-coda level.

`map_OC(spelling, pronunciation, map_progress=FALSE)`: Maps a valid *spelling* and *pronunciation* at the onset-coda level.

`map_OR(spelling, pronunciation, map_progress=FALSE)`: Maps a valid *spelling* and *pronunciation* at the onset-rime level.

`make_tables(mapped_words, weight=FALSE, positional=TRUE)`:

`mapped_words`: A table produced by one of the `map` functions.

`weight`: If `FALSE`, results will be type-weighted; otherwise, if a list of frequencies from a corpus is provided, results will be token-weighted based on frequency. Frequencies for weight from the corpus used in the Toolkit are obtained from `wordlist_v2_0$freq`.

`positional`: If `FALSE`, mapping position will be disregarded when measures are computed.

The four types of `all_tables` datasets are generated using this function. Unless one is using their own corpus of words, it's unnecessary to use this function; however, the Toolkit's results can be reproduced by the following:

```
all_tables_PG <- make_tables(all_words_PG)
all_tables_PG_noposition <- make_tables(all_words_PG, positional=FALSE)
all_tables_PG_freq <- make_tables(all_words_PG,weight=wordlist_v2_0$freq)
all_tables_PG_freq_noposition <-
  make_tables(all_words_PG,weight=wordlist_v2_0$freq, positional=FALSE)
```

Usage is analogous for all other grain sizes. Results can be indexed as follows (using `all_tables_PG` as an example):

```
> head(all_tables_PG[[1]])
  phoneme grapheme     wi     si     sm     sf     wf
1       @        a 0.9823 0.9556 0.9544 0.9964 0.3333
2       @      a_e 0.0088 0.0333 0.0414      0      0
3       @       aa      0      0      0      0 0.3333
4       @      aha      0      0  6e-04      0      0
5       @       ai      0      0  6e-04  9e-04      0
6       @       au 0.0088      0  0.003 0.0018      0
```

Full results omitted as in the `PG` case, there are 527 valid mappings whose frequencies are produced for every word; the first six are shown here.

`map_value(spelling, pronunciation, level, tables, verbose=TRUE)`:

`spelling`: A string or list of strings of words.
`pronunciation`: A string (if `spelling` is a single string) or a list of the same length as `spelling` of corresponding of strings representing the corresponding pronunciations.
`level`: The desired grain size at which to map; valid string inputs are *PG*, *OR*, *OC*, and *ONC*.
`tables`: A table of values produced by `make_tables` corresponding to the grain size specified in `level`, depending on the desired weight.
`verbose`: Whether or not to print a verbose output of progress and completion.

Outputs a list of matrices, where each matrix contains the mapping, probabilities, and frequencies for its respective word at the level mapped. In the preprocessed Toolkit, outputs are the `scored_words` family of datasets (see the Datasets section). To reproduce these (taking PG as an example), perform:

```
scored_words_PG <-
  map_value(spelling=wordlist_v2_0$spelling,
    pronunciation=wordlist_v2_0$pronunciation,
    level="PG", tables=all_tables_PG)
```

Other datasets, such as `scored_words_PG_noposition`, are generated by passing a relevantly-weighted `all_tables` dataset (for instance, `all_tables_PG_noposition`). We can also map an individual word at a desired level and weight; try:

```
map_value("toolkit", "tUlk1t", "PG", all_tables_PG)
```

Which yields:

```
> map_value("toolkit", "tUlk1t", "PG", all_tables_PG)
================================================================================[[1]]
             toolkit     NA     NA     NA     NA     NA
phoneme            t      U      l      k      1      t
grapheme           t     oo      l      k      i      t
position          wi     sm     sf     si     sm     wf
PG            0.9942 0.5317 0.8682 0.2059 0.7859 0.8302
GP                 1 0.3323      1      1 0.8159      1
PG_freq       3.0111 2.0414 2.6522 2.5527 3.5053 3.2947
P_freq        3.0137 2.3139 2.7135  3.238 3.6099 3.3755
G_freq        3.0111 2.5172 2.6522 2.5527 3.5936 3.2947
spelling      toolkit
pronunciation  tUlk1t
PG_accuracy      TRUE
```

See the Measures section for a description of each of the provided values.

5.2 Functions for Data Processing

`summarize_words(mapped_words, parameter)`:

`mapped_words`: A list of words from `wordlist_v2_0` or an output of `map_value` corresponding to the desired word(s) of interest

`parameter`: The desired measure to be obtained: *PG*, *GP*, *PG_freq*, *P_freq*, or *G_freq*

Outputs a matrix of statistics for the desired measure across the inputs. We can see this for a single word, for instance, the various statistics (mean, median, max, min, and sd) for the word "toolkit"'s spelling consistency at the PG level:

```
summarize_words(
  mapped_words = scored_words_PG[which(wordlist_v2_0$spelling=="toolkit")],
  parameter = "PG")
```

```
> summarize_words(mapped_words = scored_words_PG[which(wordlist_v2_0$spelling=$
  spelling pronunciation      mean  median    max    min        sd
1  toolkit        tUlk1t 0.7026833 0.80805 0.9942 0.2059 0.2869119
```

The `all_words_<direction>_probability` family of datasets is also generated using this function:

```
all_words_PG_probability <-
  summarize_words(
    mapped_words = scored_words_PG,
    parameter = "PG")
head(all_words_PG_probability)
```

```
> head(all_words_PG_probability)
  spelling pronunciation      mean  median    max    min        sd
1        a             8 0.6667000 0.66670 0.6667 0.6667        NA
2        a             e 0.8409000 0.84090 0.8409 0.8409        NA
3      aah             a 0.0041000 0.00410 0.0041 0.0041        NA
4 aardvark       ardvark 0.6839286 0.96780 0.9959 0.0041 0.4037696
5    aaron          Eren 0.5082750 0.52230 0.9861 0.0024 0.4597517
6   abacus        @bekes 0.5613333 0.57895 0.9823 0.0483 0.3702818
```

`word_pattern(mapped_words, phoneme, grapheme, position):`

`mapped_words:` An output dataset from `map_value`. The `scored_words` family of datasets works here.

`phoneme:` Desired phoneme in the Toolkit's format. *any* is a valid string input here if one desires only a grapheme corresponding to any pronunciation at the desired position.

`grapheme:` Desired corresponding grapheme. *any* is a valid string input here if one desires only a phoneme corresponding to any grapheme at the desired position.

`position:` Position of the phoneme-grapheme mapping in the word; valid string inputs to this are *si* (syllable-initial), *sm* (syllable-medial), *sf* (syllable-final), *wi* (word-initial), and *wf* (word-final). *any* is a valid input here if one desires the specified phoneme-grapheme mapping but not in a specific position.

Outputs a list of all words present in the corpus containing the specified phoneme-grapheme mapping at the desired position. For instance, if we'd like to obtain a list of all words beginning with the grapheme "ho" (meaning "ho" appears in the *wi* word-initial position), irrespective of pronunciation, we call:

`word_pattern(scored_words_PG,"any","ho","wi")`

Which outputs:

```
> word_pattern(scored_words_PG,"any","ho","wi")
      [,1]
 [1,] "homage"
 [2,] "honest"
 [3,] "honestly"
 [4,] "honesty"
 [5,] "honor"
 [6,] "honorable"
 [7,] "honorary"
 [8,] "honored"
 [9,] "honoring"
[10,] "honors"
[11,] "honour"
[12,] "honourable"
[13,] "honoured"
[14,] "honours"
[15,] "hors"
[16,] "hour"
[17,] "hourglass"
[18,] "hours"
```

`pw_spell(pronunciation, level, tables, minimum=1, param="pg")`:

`pronunciation`: The desired phoneme sequence as a string in the Toolkit format.
`level`: The desired level (PG, OC, OR, ONC) at which to obtain the word.
`tables`: The `scored_words` table corresponding to the given level.
`minimum`: Defaults to 1; if 1, only provides the most likely spelling for the word at the specified level. If 0, provides a list of all possible spellings.

Outputs either the most likely graphemic representation of the phonemes at the desired level, or a list of all possible spellings, depending on the value of `minimum`. For instance, we can either obtain the most likely spelling for the pronunciation "flok" at the ONC level:
`pw_spell("flok", "ONC", all_tables_ONC, minimum=1)`

```
> pw_spell("flok", "ONC", all_tables_ONC, minimum=1)
[[1]]
[1] "floc"
```

Or we can obtain all possible spellings for "flok" at the ONC level[3]:
`pw_spell("flok", "ONC", all_tables_ONC, minimum=0)`

```
> pw_spell("flok", "ONC", all_tables_ONC, minimum=0)
[[1]]
  [1] "flau_ec"   "flau_ech"  "flau_eck"  "flau_ek"   "flau_elk"  "flau_eq"
  [7] "flau_equ"  "fleouc"    "fleouch"   "fleouck"   "fleouk"    "fleoulk"
 [13] "fleouq"    "fleouqu"   "flewc"     "flewch"    "flewck"    "flewk"
 [19] "flewlk"    "flewq"     "flewqu"    "flewec"    "flewech"   "fleweck"
```

Full details omitted as 196 spellings are possible.

6 Advanced Usage

This section will describe more advanced usage of the Toolkit. Follow along with the files in the repository. Some Python will be used and discussed here as well, primarily in the handling of datasets generated from the Toolkit in R.

6.1 Generating All Toolkit Measures

Here, we discuss how to generate a dataset spanning the entire Toolkit containing every measure for every word, at every grain size and weight combination. The full code for this is provided in *scripts/all-measures/extract_all_measures.R* - follow along there. Executing this script as it is will generate the *all_measures.csv* dataset to the same folder (and the precomputed version is present in the *data/precalculated* folder).

6.1.1 Code Walkthrough

First, four lists are defined:

[3]Due to the way graphemes are handled in the Toolkit, words ending in "-*C*e" where *C* is a consonant are conventionally written as "_*C*e". The first result in the list should be read as "flauce".

- `grain_sizes:` Keys of this list are the grain sizes currently available in the Toolkit, and values are the corresponding default dataset (type-weighted and positional) of the `scored_words` family, which is indexed when measures are obtained.
- `weight_options:` Keys of this list correspond to the various combinations of weight options currently available in the Toolkit. "default" refers to type-weighted positional measures. Values are the corresponding string suffixes to be appended to dataset names in order to index the proper respective dataset of the `scored_words` family. More on this below in the function breakdown.
- `measures:` The five measures available in the Toolkit.
- `statistics:` The five statistics available for each measure in the Toolkit.

Removing an entry from any of these lists will cause respective measures to be ommitted from the dataset that is generated. Adjust these lists as necessary to fit the needs of your project.

Next, we define three functions:

- `extract_summary(grain, weight, measure)`

  Extracts the summary (all of the Toolkit's statistics) for the provided `measure` from the dataset named by the concatenation of `grain` and `weight`.

  For instance, if `grain` is *scored_words_PG*, `weight` is *_noposition* and `measure` is *P_freq*, this function will return all of the statistics for *P_freq* from *scored_words_PG_noposition*.

- `filter_columns(df, statistics)`

  Removes all statistic columns from a dataframe that do not correspond to the statistics desired. Does so by keeping columns corresponding to statistics present in the passet in `statistics` list.

- `get_measures(grain_sizes, weight_options, measures, statistics)`

  Returns a dataframe with all of the desired `statistics` for all of the desired `measures` from all datasets indexed by combinations of `grain_sizes` and `weight_options`.

The final section of the code simply writes the generated dataset to a file. If you want to include the pre-computed PP measures (whose generation will be possible in an upcoming Toolkit version), uncomment the final three lines.

6.1.2 Output

By convention, columns in the output are named

`<grain_size>_<token?>_<position?>_<measure>_<statistic>`

where:

- `grain_size:` PG, OR, OC, or ONC. Delineates the grain size with respect to which the measure was calculated.

- `token?:` If the measure is token-weighted, will be "_freq". If the measure is type-weighted and position is included, will be "_default".[4]
- `position?:` Blank if the measure is weighted by position. If position is disregarded, is "_noposition".
- `measure:` The measure described. PG, GP, PG_freq, P_freq, or G_freq.
- `statistic:` What value of the measure is described. Will be mean, median, max, min, or sd.

With this generated, you can either continue to work with the dataset in R, or use another language of your choice and load the csv into a dataframe; for instance, Python with `pandas`.

6.1.3 Usage

To use this script for other projects (for instance, your own corpus as computed by the `map_value` function which outputs datasets of the `scored_words` family), simply adjust the values of the `grain_sizes` and `weight_options` lists to match the naming of your datasets, so that they are properly concatenated and indexed by the functions above.

6.2 Map-Words Pipeline

Section coming soon. See *scripts/batch-mapping/mapping_pipeline.R* for the relevant code.

6.3 Generating Pseudoword Measures

Here, we discuss how to generate all available measures from the Toolkit for a desired set of pseudowords (and their respective pronunciations). Obtaining these is slightly different from obtaining all measures from the Toolkit's preprocessed words, as measures involving those have been mapped and calculated prior; for pseudowords, each combination is novel and thus the Toolkit must compute the measures for the individual word on-the-fly based on its preprocessed dataset.

Code for this is available in
scripts/pseudowords/get_pseudoword_measures.R. We begin by defining some lists allowing us to iterate through all tables for each combination of weight and grain size, for each of the desired measures. As we iterate over these options for each pseudoword and its pronunciation, we need to *map* the word to its pronunciation at the current level:

```
mapped_pseudoword <- map_value(word, pron, level, table_data)
```

Where `table_data` corresponds to one of the tables (default, token-weighted, position-irrespective, token-weighted position-irrespective) at the current `level`. We then, for each available measure, pass the result into `summarize_words` to get all desired statistics:

```
summary <- summarize_words(mapped_pseudoword, measure)
```

You'll then see, in the next few lines, a few functions to manipulate the names of the obtained

[4] This may be changed soon i.e. to remove the "default" indicator.

data for the purpose of including these in a final dataset similar to the one discussed in the previous section for all of the Toolkit measures.

6.4 Obtaining Number of Units

Here we discuss how to determine the number of units present in the Toolkit; i.e., the number of valid mappings at any given level. We also discuss how to obtain the individual numbers of unique phonemes and graphemes. This is useful as updates to the Toolkit often come with added mappings, which adjusts the total number at every level. Follow along in *scripts/all-units/get_all_units.R*.

The process for this is very straightforward; we first simply combine all datasets detailing valid mappings and take the number of unique phonemes and graphemes in the full set to obtain the total number of unique units for each.

For mappings at each individual level, we can simply take the number of mappings of the first word in the dataset, as measures are calculated for each mapping for each word. As such we simply iterate through the four and print the total number.

6.5 Finding Orthographic and Phonological Neighbors

It can be useful to find both the orthographic and phonological neighbors of both words and pseudowords. Here, we discuss the process for utilizing the Toolkit's datasets to find such neighbors at any of the four grain sizes (as well as the individual letter size in the orthographic case). Code for this is available in *scripts/neighbors/neighbor_finders.R*.

First, we define a function *edit_distance_matrices* for computing the edit distance (Levenshtein distance[5]). Note that this function takes in two matrices of the form given in the all_tables family of datasets, and is used whenever we are finding the neighbors at a level other than the individual letter level.

Next, we can define functions for computing the orthographic and phonological neighbors for words. At the orthographic level specifically, we may also be interested in simply the edit distance at the level of individual letters; the function defaults to finding such neighbors and is as simple as calling the *stringdist* function from the package of the same name. At all other levels, we call the function previously created between the words we are interested in and all of the words in the Toolkit's corpus; these are indexed based on the desired level (grain size). The syntax looks complex but is just indexing the vector in the table containing the orthographic representation of the word:

```
mygraphemes1 <-
  level_func(
    spelling=item_spellings[j],
    pronunciation=item_pronunciations[j])[[1]][[1]][2,]
```

[5]The standard metric for determining the distance between two strings as a function of insertions, deletions, and substitutions; see https://en.wikipedia.org/wiki/Levenshtein_distance

The `[2,]` index marker at the end is particularly important as it indexes the orthographic representation itself; it turns out that finding the phonological neighbors is as simple as using the above edit distance function and indexing by `[1,]` instead. Also note that here, `level_func` corresponds to the relevant function from the Toolkit for mapping the word(s) we are interested in at the desired level; for instance, if we would like to find the ONC-neighbors of a word, we must first map that word at the ONC level.

Once we've obtained the graphemes for the word itself, we use the edit distance function to compute the distances between the word and all corpus words. We then simply return all words satisfying the condition that the edit distance is less than the threshold (which defaults to 1).

At this point, it is useful to discuss what this distance actually refers to when the level is not the individual letter. When we discuss insertion, deletion, and substitution of graphemic/phonemic units in such a case, we refer to the insertion, deletion, and substitution of an entire unit, which may be more than one letter. This is why the mapping of the word itself is important as letters and phonemes can be grouped together in different ways (and thus yield different neighbors at different levels).

As mentioned before, the code for obtaining phonological neighbors is nearly identical, except here the tables are indexed differently in order to obtain the phonological, rather than orthographic, representation. The same edit distance function suffices.

7 Troubleshooting

This section will cover some common problems that occur when using the Toolkit, and how to identify their causes.

7.1 Function Hangs without Error

Sometimes, when processing words with certain Toolkit functions in batch (i.e. passing in lists of items at once rather than individual strings), the Toolkit will "hang" instead of completely erroring and concluding program execution. When this happens, iterating through each possible problem individually with a `tryCatch` block allows one to examine each error.

Example 7.1. Consider a dataset *my_pseudowords* containing two columns for spelling and pronunciation: *spell* and *pron*. We'd like to map all of these spelling-pronunciation pairs at the ONC level and summarize them using the `summarize_words` function. However, if there are one or more mappings that would be invalid in this dataset (e.g. not accepted by the Toolkit), then running the following line which would otherwise work:

```
summarize_words(
    map_value(
        my_pseudowords$spelling,
        my_pseudowords$pron,
        "ONC",
        all_tables_ONC),
```

```
    "PG")
```

will instead hang indefinitely in the R terminal. To find the problematic entries, we make use of a simple `tryCatch` block in a loop that maps and summarizes words in the dataset one-by-one (where `s` and `p` are strings), collecting the errors in a list (erroneous entries causing the program to hang are guaranteed to throw a visible error when examined individually):

```
summary <- tryCatch(
    {
        summarize_words(map_value(s, p, "ONC", all_tables_ONC), "PG")
    },
    error=function(e) {
        error_df <<- rbind(error_df, data.frame(spell = s, pron = p))
    })
```

In the above, `error_df` has a *spell* and *pron* column where erroneous entries are collected. If there is no error for an entry, we instead have the result of the call in the `summary` variable.

The full template code for this is available on the GitHub. See *scripts/troubleshooting/hanging/identifying-problematic-entries.R*.

8 Appendix

8.1 Phoneme Reference

The following is a table illustrating the conversions between the IPA pronunciation format and the Toolkit's conventions.

IPA	Toolkit	Example
ɑ, ɒ	a	father *faD3r*
æ	@	cat *k@t*
ʌ	^	mutt *m^t*
ə	e	about *ebOt*
ɔ	c	force *fcrs*
aʊ	O	wow *wO*
aɪ	5	light *l5t*
ɛ	E	net *nEt*
ɜ, ɚ	3r	third *T3rd*
eɪ	8	weight *w8t*
ɪ	1	fit *f1t*
i	i	neato *nito*
oʊ	o	ghost *gost*
ɔɪ	2	poise *p2z*
ʊ	U	look *luK*
u	u	compute *kempjut*
b	b	baby *b8bi*
tʃ	C	watch *waC*
d	d	dude *dud*
ð	D	brother *br^D3r*

IPA	Toolkit	Example
f	f	fig *f1g*
g	g	igloo *1glu*
h	h	hello *hElo*
ʤ	G	juggle *G^gel*
k	k	keep *kip*
l	l	lazy *l8zi*
m	m	money *m^ni*
n	n	night *n5t*
ŋ	N	wing *w1N*
p	p	please *pliz*
ɹ	r	reach *riC*
s	s	scythe *s5D*
ʃ	S	shell *SEl*
t	t	twin *tw1n*
θ	T	think *T1Nk*
v	v	very *vEri*
w	w	willow *w1lo*
j	j	you *ju*
z	z	maze *m8z*
ʒ	Z	pleasure *plEZ3r*