1   A practical primer on processing semantic property norm data

2   Erin M. Buchanan[1], Simon De Deyne[2], & Maria Montefinese[3]

3   [1] Harrisburg University of Science and Technology
4   [2] University of Melbourne
5   [3] University of Padua

6   Author Note

Abstract

Semantic property listing tasks require participants to generate short propositions (e.g., <barks>, <has fur>) for a specific concept (e.g., dog). This task is the cornerstone of the creation of semantic property norms which are essential for modelling, stimuli creation, and understanding similarity between concepts. However, despite the wide applicability of semantic property norms for a large variety of concepts across different groups of people, the methodological aspects of the property listing task have received less attention, even though the procedure and processing of the data can substantially affect the nature and quality of the measures derived from them. The goal of this paper is to provide a practical primer on how to collect and process semantic property norms. We will discuss the key methods to elicit semantic properties and compare different methods to derive meaningful representations from them. This will cover the role of instructions and test context, property pre-processing (e.g., lemmatization), property weighting, and relationship encoding using ontologies. With these choices in mind, we propose and demonstrate a processing pipeline that transparently documents these steps resulting in improved comparability across different studies. The impact of these choices will be demonstrated using intrinsic (e.g. reliability, number of properties) and extrinsic measures (e.g., categorization, semantic similarity, lexical processing). Example data and the impact of choice decisions will be provided. This practical primer will offer potential solutions to several longstanding problems and allow researchers to develop new property listing norms overcoming the constraints of previous studies.

Keywords: semantic, property norm task, tutorial

A practical primer on processing semantic property norm data

1. Available feature norms and their format

- Property listing task original work: Toglia and Battig (1978); Toglia (2009); Rosch and Mervis (1975); Ashcraft (1978)

- English: McRae, Cree, Seidenberg, and McNorgan (2005), Vinson and Vigliocco (2008), Buchanan, Holmes, Teasley, and Hutchison (2013), Devereux, Tyler, Geertzen, and Randall (2014), Buchanan, Valentine, and Maxwell (2019)

- Italian: Montefinese, Ambrosini, Fairfield, and Mammarella (2013); Reverberi, Capitani, and Laiacona (2004), Kremer and Baroni (2011)

- German: Kremer and Baroni (2011)

- Portuguese: Stein and de Azevedo Gomes (2009)

- Spanish: Vivas, Vivas, Comesaña, Coni, and Vorano (2017)

- Dutch: Ruts et al. (2004)

- Blind participants: Lenci, Baroni, Cazzolli, and Marotta (2013)

I'm sure there are more, here's what we cited recently.

Define concept, feature for clarity throughout - make sure you use these two terms consistently.

2. Pointers about how to collect the data

a. instructions, generation, verification, importance

I really like the way the CSLB did it: https://cslb.psychol.cam.ac.uk/propnorms

They showed the concept, then had a drop down menu for is/has/does, and then the participant typed in a final window. That type of system would solve about half the problems I am going to describe below about using multi-word sequences. Might be some

54 other suggestions, but for that type of processing, you could do combinations and have more

55 consistent data easily.

56   3. Typical operations performed on features

57   In the next several sections, we provide a tutorial using R on how data from the

58 semantic property norm task might be processed from raw input to finalized output. Figure

59 1 portrays the proposed set of steps including spell checking, lemmatization, exclusion of

60 stop words, and final processing in a multi-word sequence approach or a bag of words

61 approach. After detailing these steps, the final data form will compared to previous norms to

62 determine the usefulness of this approach.

63 **Materials and Data Format**

64   The data for this tutorial includes 16544 unique concept-feature responses for 226

65 concepts from Buchanan et al. (2019) that were included in McRae et al. (2005), Vinson and

66 Vigliocco (2008), and Bruni, Tran, and Baroni (2014). The data should be structured in tidy

67 format wherein each concept-feature observation is a row and each column is a variable

68 (Wickham, 2014). Therefore, the data includes a `word` column with the normed concept and

69 an `answer` column with the participant answer, as shown in Table 1.

70   This data was collected using the instructions provided by McRae et al. (2005),

71 however, in contrast to the suggestions for consistency detailed above (Devereux et al., 2014),

72 each participant was simply given a large text box to include their answer. Each answer

73 includes multiple embedded features, and the tutorial proceeds to demonstrate potential

74 processing addressing the data in this nature. With structured data entry for participants,

75 the suggested processing steps are reduced.

76 **Spelling**

77   Spell checking can be automated with the `hunspell` package in *R* (Ooms, 2018), which

78 is the spell checking library used in popular programs such as FireFox, Chrome, RStudio, and

79 OpenOffice. Each `answer` can be checked for misspellings across an entire column of answers,

80 which is located in the `master` dataset. The default dictionary is American English, and the

81 `hunspell` vignettes provide details on how to import your own dictionary for non-English

82 languages. The choice of dictionary should also normalize between multiple varieties of the

83 same language, for example, the `"en_GB"` would convert to British English spellings.

```r
## Lower case to normalize
master$answer <- tolower(master$answer)
## Install the hunspell package if necessary
#install.packages("hunspell")
library(hunspell)
## Check the participant answers
## The output is a list of spelling errors for each line
spelling_errors <- hunspell(master$answer, dict = dictionary("en_US"))
```

84   The result from the `hunspell()` function is a list object of spelling errors for each row

85 of data. For example, when responding to *apple*, a participant wrote *fruit grocery store*

86 *orchard red green yelloe good with peanut butter good with caramell*, and the spelling errors

87 were denoted as *yelloe caramell*. After checking for errors, the `hunspell_suggest()`

88 function was used to determine the most likely replacement for each error.

```r
## Check for suggestions
spelling_suggest <- lapply(spelling_errors, hunspell_suggest)
```

89   For *yelloe*, both *yellow yell* were suggested, and *caramel caramels caramel l camellia*

90 *camel* were suggested for *caramell*. The suggestions are presented in most probable order,

91 and using a few loops with the substitute (`gsub()`) function, we can replace all errors with

92 the most likely replacement in a new dataset `spell_checked`. A specialized dictionary with

93 pre-coded error responses and corrections could be implemented at this stage. Other paid

94 alternatives, such as Bing Spell Check, can be a useful avenue for datasets that may contain

95 brand names (i.e, *apple* versus *Apple*) or slang terms.

```r
## Replace with most likely suggestion
spell_checked <- master
### Loop over the dataframe
for (i in 1:nrow(spell_checked)){
  ### See if there are spelling errors
  if (length(spelling_errors[[i]]) > 0) {
    ### Loop over all errors
    for (q in 1:length(spelling_errors[[i]])){
      ### Replace with the first answer
      spell_checked$answer[i] <- gsub(spelling_errors[[i]][q],
                                      spelling_suggest[[i]][[q]][1],
                                      spell_checked$answer[i])
    }
  }
}
```

Lemmatization

97 The next step approaches the clustering of word forms into their lemma or head word

98 from a dictionary. The process of lemmatizing words involves using a lexeme set (i.e., all

99 words forms that have the same meaning, *am, are, is*) to convert into a common lemma (i.e.,

100 *be*) from a trained dictionary. In contrast, stemming involves processing words using

101 heuristics to remove affixes or inflections, such as *ing* or *s*. The stem or root word may not

102 reflect an actual word in the language, as simply removing an affix does not necessarily

103 produce the lemma. For example, in response to *airplane*, *flying* can be easily converted to

104 *fly* by removing the *ing* inflection. However, this same heuristic converts the feature *wings*

105 into *w* after removing both the *s* for a plural marker and the *ing* participle marker. Several

106 packages for R include customizable stemmers, notably the `hunspell`, `corpus` (Perry, 2017),

107 and `tm` (Feinerer, Hornik, & Artifex Software, 2018) packages.

108 Lemmatization is the likely choice for processing property norms, and this process can

109 be achieved by installing `TreeTagger` (Schmid, 1994) and the `koRpus` package in R

110 (Michalke, 2018). TreeTagger is a trained tagger designed to annotate part of speech and

111 lemma information in text, and parameter files are available for multiple languages. The

112 koRpus package includes functionality to use TreeTagger in R. After installing the package

113 and TreeTagger, we will create a unique set of tokenized words to lemmatize to speed

114 computation.

```
lemmas <- spell_checked
## Install the koRpus package
#install.packages("koRpus")
#install.packages("koRpus.lang.en")
## You must load both packages separately
library(koRpus)
library(koRpus.lang.en)
## Install TreeTagger
#https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
## Find all types for faster lookup
all_answers <- tokenize(lemmas$answer, format = "obj", tag = F)
all_answers <- unique(all_answers)
```

115 The `treetag()` function calls the installation of TreeTagger to provide part of speech

116 tags and lemmas for each token. Importantly, the `path` option should be the directory of the

117 TreeTagger installation.

```
## This example has both suppressWarnings & suppressMessages
## You should first view these to ensure proper processing
temp_tag <- suppressWarnings(
  suppressMessages(
    ## Note: the NULL option is to control for the <unknown> that appears
    ## to occur with the last word in each text
    treetag(c(all_answers, "NULL"),
            ## Control the parameters of treetagger
            treetagger="manual", format="obj",
            TT.tknz=FALSE, lang="en",
            TT.options=list(path="~/TreeTagger", preset="en"))))
```

118 This function returns a tagged corpus object, which can be converted into a dataframe

of the token-lemma information. The goal would be to replace inflected words with their lemmas, and therefore, unknown values, number tags, and equivalent values are ignored by subsetting out these from the dataset. Table 2 portrays the results from TreeTagger.

```r
## Remove all tags not using
replacement_lemmas <- temp_tag@TT.res
replacement_lemmas <- subset(replacement_lemmas,
                             #ignore punctuation
                             wclass != "punctuation" &
                             #unknown values
                             lemma != "<unknown>" &
                             #numbers
                             lemma!= "@card@" &
                             #token should change more than case
                             tolower(token) != tolower(lemma))
```

From this dataset, you can use the `stringi` package (Gagolewski & Tartanus, 2019) to replace all of the original tokens with their lemmas. This package allows for replacement look-up across a large set of substitutions. The `stri_replace_all_regex()` function includes the column of data to examine, the patterns to find (using `\\b` regular expressions to ensure word boundaries and no partial word replacements), what to replace those patterns with, and other options to ensure the original dataframe with replacement is returned. Table 3 shows the processed data at this stage.

```r
## Install the stringi package
#install.packages("stringi")
library(stringi)
## Replace all the original tokens with new lemmas using \\b for word boundaries
lemmas$answer <- stri_replace_all_regex(str = lemmas$answer,
                       pattern = paste("\\b", replacement_lemmas$token, "\\b", sep = ""),
                       replacement = replacement_lemmas$lemma,
                       vectorize_all = F, list(case_insensitive = TRUE))
```

Word Sequences

130   Multi-word sequences are often coded to mimic a Collins and Quillian (1969) style

131 model, with "is-a" and "has-a" type markers. If data were collected to include these markers,

132 this step would be pre-encoded into the output data, rendering the following code

133 unnecessary. A potential solution for processing messy data could be to search for specific

134 part of speech sequences that mimic the "is-a" and "has-a" strings. An examination of the

135 coding in McRae et al. (2005) and Devereux et al. (2014) indicates that the feature tags are

136 often verb-noun or verb-adjective-noun sequences. Using TreeTagger on each concept's

137 answer set, we can obtain the parts of speech in context for each lemma. With `dplyr`

138 (Wickham, Francios, Henry, Muller, & Rstudio, 2019), new columns are added to tagged

139 data to show all bigram and trigram sequences. All verb-noun and verb-adjective-noun

140 combinations are selected, and any words not part of these multi-word sequences are treated

141 as unigrams. Finally, the `table()` function is used to tabulate the final count of n-grams

142 and their frequency.

```r
## Create an empty dataframe
multi_words <- data.frame(Word=character(),
                          Feature=character(),
                          Frequency=numeric(),
                          stringsAsFactors=FALSE)
## Create unique word list to loop over
unique_concepts <- unique(lemmas$word)
## Install dplyr
#install.packages("dplyr")
library(dplyr)
## Loop over each word
for (i in 1:length(unique_concepts)){
  ## Create parts of speech for clustering together
  temp_tag <- suppressWarnings(
    suppressMessages(
      treetag(c(lemmas$answer[lemmas$word == unique_concepts[i]], "NULL"),
          ## Control the parameters of treetagger
          treetagger="manual", format="obj",
          TT.tknz=FALSE, lang="en",
```

```r
                TT.options=list(path="~/TreeTagger", preset="en"))))
## Save only the dataframe, remove NULL
temp_tag <- temp_tag@TT.res[-nrow(temp_tag@TT.res) , ]
## Subset out information you don't need
temp_tag <- subset(temp_tag,
                   wclass != "comma" & wclass != "determiner" &
                     wclass != "preposition" & wclass != "modal" &
                     wclass != "predeterminer" & wclass != "particle" &
                     wclass != "to" & wclass != "punctuation" &
                     wclass != "fullstop" & wclass != "conjunction" &
                     wclass != "pronoun")
## Create a temporary tibble
temp_tag_tibble <- as_tibble(temp_tag)
## Create part of speech and features combined
temp_tag_tibble <- mutate(temp_tag_tibble,
                          two_words = paste(token,
                                            lead(token), sep = "_"))
temp_tag_tibble <- mutate(temp_tag_tibble,
                          three_words = paste(token,
                                              lead(token), lead(token, n = 2L),
                                              sep = "_"))
temp_tag_tibble <- mutate(temp_tag_tibble,
                          two_words_pos = paste(wclass,
                                                lead(wclass), sep = "_"))
temp_tag_tibble <- mutate(temp_tag_tibble,
                          three_words_pos = paste(wclass,
                                                  lead(wclass), lead(wclass, n = 2L),
                                                  sep = "_"))
## Find verb noun or verb adjective nouns to cluster on
verb_nouns <- grep("\\bverb_noun", temp_tag_tibble$two_words_pos)
verb_adj_nouns <- grep("\\bverb_adjective_noun", temp_tag_tibble$three_words_pos)
## Use combined and left over features
features_for_table <- c(temp_tag_tibble$two_words[verb_nouns],
                        temp_tag_tibble$three_words[verb_adj_nouns],
                        temp_tag_tibble$token[-c(verb_nouns, verb_nouns+1,
                                                 verb_adj_nouns, verb_adj_nouns+1,
                                                 verb_adj_nouns+2)])
## Create a table of frequencies
word_table <- as.data.frame(table(features_for_table))
## Clean up the table
word_table$Word <- unique_concepts[i]
```

```
  colnames(word_table) = c("Feature", "Frequency", "Word")
  multi_words <- rbind(multi_words, word_table[ , c(3, 1, 2)])
}
```

[143] This procedure produces mostly positive output, such as *fingers-have_fingernails* and
[144] *couches-have_cushions.* One obvious limitation is the potential necessity to match this
[145] coding system to previous codes, which were predominately hand processed. Further, many
[146] similar phrases, such as the ones for *zebra* shown below may require fuzzy logic matching to
[147] ensure that the different codings for *is-a-horse* are all combined together, as shown in Table
[148] 4.

Bag of Words

[150] The bag of words approach simply treats each token as a separate feature to be
[151] tabulated for analysis. After stemming and lemmatization, the data can be processed as
[152] single word tokens into a table of frequencies for each cue word. The resulting dataframe is
[153] each cue-feature combination with a total for each feature.

```
## Create an empty dataframe
bag_words <- data.frame(Word=character(),
                        Feature=character(),
                        Frequency=numeric(),
                        stringsAsFactors=FALSE)
## Loop over each word
for (i in 1:length(unique_concepts)){
  ## Create a table of frequencies
  word_table <- as.data.frame(table(
    ## Tokenize the words
    tokenize(
      ## Put all answers together in one character string
      paste0(lemmas$answer[lemmas$word == unique_concepts[i]], collapse = " "),
      format = "obj", tag = F)))

  ## Clean up the table
  word_table$Word <- unique_concepts[i]
```

```r
  colnames(word_table) = c("Feature", "Frequency", "Word")


  bag_words <- rbind(bag_words, word_table[ , c(3, 1, 2)])
}
## Remove punctuation
bag_words <- bag_words[-c(grep('^[[:punct:]]',bag_words$Feature)), ]
```

154 Tab 5 shows the top ten most frequent responses to *zebra* given the bag of words

155 approach. The top ten features in zebra indicate a match to the multi-word sequence

156 approach but the inclusion of words such as *be, in, a* indicate the need to remove irrelevant

157 words listed with features.

158 **Stopwords**

159 As shown in Figure 1, the next stage of processing would be to exclude stopwords, such

160 as *the, of, but*, for either the multi-word sequence or bag of word style processing. The

161 `stopwords` package (Benoit, Muhr, & Watanabe, 2017) includes a list of stopwords for more

162 than 50 languages. For multi-word sequence processing, these values can be removed by

163 subsetting the data to exclude stopwords as unigrams.

```r
## Install the stopwords package or use tm
#install.packages("stopwords")
library(stopwords)
## Remove stop words from either processing approach
multi_words <- subset(multi_words,
                     !(Feature %in% stopwords(language = "en",
                                             source = "snowball")))
bag_words <- subset(bag_words,
                   !(Feature %in% stopwords(language = "en",
                                           source = "snowball")))
```

Descriptive Statistics

¹⁶⁴

¹⁶⁵   The finalized data now represents a a processed set of cue-feature combinations with

¹⁶⁶ their frequencies for analysis. Given the differences in sample size across data collection

¹⁶⁷ points from Buchanan et al. (2019), this information was merged with the sample data.

¹⁶⁸ Table 6 includes descriptive statistics for the processed cue-feature set. First, the number of

¹⁶⁹ cue-feature combinations was calculated by taking the average number of cue-feature listings

¹⁷⁰ for each cue. Therefore, the total number of features listed for *zebra* might be 100, while

¹⁷¹ *apple* might be 45, and these values were averaged.

¹⁷²   More cue-feature combinations are listed for the multi-word approach, due to

¹⁷³ differences in combinations for some overlapping features as shown in Table 4. The large

¹⁷⁴ standard deviation for both approaches indicates that cues have a wide range of possible

¹⁷⁵ features listed. The correlation provided represents the relation between sample size for a

¹⁷⁶ cue and the number of features listed for that cue. These values are high and positive,

¹⁷⁷ indicating that the number of unique features increases with each participant. Potentially,

¹⁷⁸ many of the cue-feature combinations could be considered idiosyncratic. The next row of the

¹⁷⁹ table denotes the average number of cue-feature responses listed by less than 10% of the

¹⁸⁰ participants. This percent of responses is somewhat arbitrary, as each researcher has

¹⁸¹ determined where the optimal criterion should be. For example, McRae et al. (2005) used

¹⁸² 16% or 5/30 participants as a minimum standard, and Buchanan et al. (2019) recently used

¹⁸³ a similar criteria. The average number of cue-features that would be considered low in

¹⁸⁴ proportion is quite large, indicating that these are potentially idiosyncratic or part of long

¹⁸⁵ tailed distribution of feature responses with many low frequency features. The advantage to

¹⁸⁶ the suggested data processing pipeline and code provided here is the ability of each

¹⁸⁷ researcher to determine their own level of response necessary, if desired.

¹⁸⁸   The next two lines of Table 6 indicate cue-feature combination frequencies, such as the

number of times *zebra-stripes* or *apple-red* were listed by participants. The percent of responses is the frequency divided by sample size for each cue, to normalize over different sample sizes present in the data. These average frequency/percent was calculated for each cue, and then averaged over all cues. The correlation represents the average frequency/percent for each cue related to the sample size for that cue. These frequencies are low, matching the results for a large number of idiosyncratic responses. The correlation between frequency of response and sample size is positive, indicating that larger sample sizes produce items with larger frequencies. Additionally, the correlation between percent of response and sample size is negative, suggesting that larger sample sizes are often paired with more items with smaller percent likelihoods. Figure 2 displays the correlations for the average cue-frequency responses and the percent cue-frequency responses by sample size. It appears that the relationship between sample size and percent is likely curvilinear, rather than linear. The size of the points indicates the variability (standard deviation of each cue word's average frequency or percent). Variability appears to increase linearly with sample size for average frequency, however, it is somewhat mixed for average percent.

Internal Comparison of Approach

In this section, we show that the bag of words approach processed completely through code matches a bag of words approach that was hand coded from Buchanan et al. (2019). In Buchanan et al. (2019), the McRae et al. (2005) and Vinson and Vigliocco (2008) datasets were recoded in a bag of words approach, and the comparison between all three is provided below. The multi-word sequence approach would be comparable if one or more datasets used the same structured data collection approach or with considerable hand coded rules for feature combinations. The data from open ended responses, such as the Buchanan et al. (2019), could potentially be compared in the demonstrated multi-word sequence approach, if the raw data from other such projects were available.

214   Cosine is often used as a measure of semantic similarity, indicating the feature overlap

215 between two sets of cue-feature lists. These values can range from 0 (no overlap) to 1

216 (perfect overlap). There are two potential cosine values from the Buchanan et al. (2019): the

217 raw cosine, which included all features as listed without lemmatization or stemming, and the

218 translated cosine, which included hand lemmatization processing. Each cue in the sample

219 data for this project was compared to the corresponding cue in the Buchanan et al. (2019).

220 If data were processed in an identical fashion, the cosine values would be nearly 1 for

221 Buchanan et al. (2019) data or match the cosine values found for McRae et al. (2005) and

222 Vinson and Vigliocco (2008) in the Buchanan et al. (2019) results (original feature cosine =

223 .54-.55, translated features = .66-.67). However, all previous datasets have been reduced by

224 eliminating idiosyncratic features at various points, and therefore, we might expect that

225 noise in this data to reduce the average cosine values. The cosine matches for original

226 features averaged: $M_B = .69$ ($SD = .17$, $N = 226$); $M_M = .38$ ($SD = .18$, $N = 61$); $M_V =$

227 .59 ($SD = .18$, $N = 68$). These values indicate a somewhat comparable set of data, with

228 lower values for McRae et al. (2005) than previous results. The cosine matches for translated

229 features averaged: $M_B = .72$ ($SD = .16$, $N = 226$); $M_M = .58$ ($SD = .14$, $N = 61$); $M_V =$

230 .58 ($SD = .19$, $N = 68$). Again, these values indicate that the data processed entirely in R

231 produces a comparable set of results, albeit with added noise of small frequency features.

232 **External Comparison of Approach**

233   The MEN dataset (Bruni et al., 2014) contains cue-cue pairs of English words rating

234 for similarity by Amazon Mechanical Turk participants. In their rating task, participants

235 were shown two cue-cue pairs and asked to select the more related pair of the two presented.

236 Each pair was rated by 50 participants, and thus, a score of 50 indicates high relatedness,

237 while a score of 0 indicates no relatedness. A range of relatedness values were selected from

238 this dataset with overlapping cues from Buchanan et al. (2019), and these values were

[239] compared to the cosine calculated between cues using the bag of words method. The

[240] correlation between cosine on the processed data and the MEN ratings was $r = .69$, 95% CI

[241] $[.61, .76]$, $N = 179$, indicating considerable agreement between raters and cosine values.

Future Directions

[243]   Generally, coding ontology is cumbersome on the researcher, as it is normally

[244] performed by hand using a coding schema. Wu and Barsalou (2009) developed a hierarchical

[245] taxonomy for coding categories as part of the feature listing task, that has been used in

[246] several projects, notably the McRae et al. (2005). Examples of the categories include

[247] taxonomic (synonyms, subordinates), entity (internal components, behavior, spatial

[248] relations), situation (location, time), and introspective properties (emotion, evaluation).

[249] Coding ontology may be best performed systematically with look-up rules of previously

[250] decided upon factors, however, clustering analyses may provide a potential avenue to explore

[251] categorizing features within the current dataset. One limitation to this method the sheer size

[252] of the idiosyncratic features as mentioned above, and thus, features smaller in number may

[253] be more difficult to group.

<div align="center">

Discussion

</div>

[255] - this sort of thing is great for replication purposes, which is pretty important because of

[256]   the garden of forking paths which applies not just to statistical analyses but also to

[257]   processing.

[258] - we've provided a workflow suggestion that a researcher can use to format their work,

[259]   along with functions that can be detailed to match any hand processing results.

[260] - weave this to match introduction

References

Ashcraft, M. H. (1978). Property norms for typical and atypical items from 17 categories: A description and discussion. *Memory & Cognition*, *6*(3), 227–232. doi:10.3758/BF03197450

Benoit, K., Muhr, D., & Watanabe, K. (2017). stopwords: Multilingual Stopword Lists. Retrieved from https://cran.r-project.org/web/packages/stopwords/index.html

Bruni, E., Tran, N. K., & Baroni, M. (2014). Multimodal Distributional Semantics. *Journal of Artificial Intelligence Research*, *49*, 1–47. doi:10.1613/jair.4135

Buchanan, E. M., Holmes, J. L., Teasley, M. L., & Hutchison, K. A. (2013). English semantic word-pair norms and a searchable Web portal for experimental stimulus creation. *Behavior Research Methods*, *45*(3), 746–757. doi:10.3758/s13428-012-0284-z

Buchanan, E. M., Valentine, K. D., & Maxwell, N. P. (2019). English semantic feature production norms: An extended database of 4436 concepts. *Behavior Research Methods*. doi:10.3758/s13428-019-01243-z

Collins, A. M., & Quillian, M. R. (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, *8*(2), 240–247. doi:10.1016/S0022-5371(69)80069-1

Devereux, B. J., Tyler, L. K., Geertzen, J., & Randall, B. (2014). The Centre for Speech, Language and the Brain (CSLB) concept property norms. *Behavior Research Methods*, *46*(4), 1119–1127. doi:10.3758/s13428-013-0420-4

Feinerer, I., Hornik, K., & Artifex Software, I. (2018). tm: Text Mining Package. Retrieved from https://cran.r-project.org/web/packages/tm/index.html

Gagolewski, M., & Tartanus, B. (2019). stringi: Character String Processing Facilities.
    Retrieved from https://cran.r-project.org/web/packages/stringi/index.html

Kremer, G., & Baroni, M. (2011). A set of semantic norms for German and Italian.
    Behavior Research Methods, *43*(1), 97–109. doi:10.3758/s13428-010-0028-x

Lenci, A., Baroni, M., Cazzolli, G., & Marotta, G. (2013). BLIND: A set of semantic feature
    norms from the congenitally blind. *Behavior Research Methods*, *45*(4), 1218–1233.
    doi:10.3758/s13428-013-0323-4

McRae, K., Cree, G. S., Seidenberg, M. S., & McNorgan, C. (2005). Semantic feature
    production norms for a large set of living and nonliving things. *Behavior Research
    Methods*, *37*(4), 547–559. doi:10.3758/BF03192726

Michalke, M. (2018). koRpus: An R Package for Text Analysis. Retrieved from
    https://cran.r-project.org/web/packages/koRpus/index.html

Montefinese, M., Ambrosini, E., Fairfield, B., & Mammarella, N. (2013). Semantic memory:
    A feature-based analysis and new norms for Italian. *Behavior Research Methods*,
    45(2), 440–461. doi:10.3758/s13428-012-0263-4

Ooms, J. (2018). The hunspell package: High-Performance Stemmer, Tokenizer, and Spell
    Checker for R. Retrieved from https://cran.r-
    project.org/web/packages/hunspell/vignettes/intro.html{\#}setting{_}a{_}language

Perry, P. O. (2017). corpus: Text Corpus Analysis. Retrieved from http://corpustext.com/

Reverberi, C., Capitani, E., & Laiacona, E. (2004). Variabili semantico lessicali relative a
    tutti gli elementi di una categoria semantica: Indagine su soggetti normali italiani per
    la categoria "frutta". *Giornale Italiano Di Psicologia*, *31*, 497–522.

Rosch, E., & Mervis, C. B. (1975). Family resemblances: Studies in the internal structure of

306   categories. *Cognitive Psychology*, *7*(4), 573–605. doi:10.1016/0010-0285(75)90024-9

307   Ruts, W., De Deyne, S., Ameel, E., Vanpaemel, W., Verbeemen, T., & Storms, G. (2004).

308   Dutch norm data for 13 semantic categories and 338 exemplars. *Behavior Research*

309   *Methods, Instruments, & Computers*, *36*(3), 506–515. doi:10.3758/BF03195597

310   Schmid, H. (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees.

311   doi:10.1.1.28.1139

312   Stein, L., & de Azevedo Gomes, C. (2009). Normas Brasileiras para listas de palavras

313   associadas: Associação semântica, concretude, frequência e emocionalidade.

314   *Psicologia: Teoria E Pesquisa*, *25*, 537–546. doi:10.1590/S0102-37722009000400009

315   Toglia, M. P. (2009). Withstanding the test of time: The 1978 semantic word norms.

316   *Behavior Research Methods*, *41*(2), 531–533. doi:10.3758/BRM.41.2.531

317   Toglia, M. P., & Battig, W. F. (1978). *Handbook of semantic word norms.* Hillside, NJ:

318   Earlbaum.

319   Vinson, D. P., & Vigliocco, G. (2008). Semantic feature production norms for a large set of

320   objects and events. *Behavior Research Methods*, *40*(1), 183–190.

321   doi:10.3758/BRM.40.1.183

322   Vivas, J., Vivas, L., Comesaña, A., Coni, A. G., & Vorano, A. (2017). Spanish semantic

323   feature production norms for 400 concrete concepts. *Behavior Research Methods*,

324   *49*(3), 1095–1106. doi:10.3758/s13428-016-0777-2

325   Wickham, H. (2014). Tidy Data. *Journal of Statistical Software*, *59*(10), 1–23.

326   doi:10.18637/jss.v059.i10

327   Wickham, H., Francios, R., Henry, L., Muller, K., & Rstudio. (2019). dplyr: A Grammar of

328   Data Manipulation. Retrieved from

329   https://cloud.r-project.org/web/packages/dplyr/index.html

330   Wu, L.-l., & Barsalou, L. W. (2009). Perceptual simulation in conceptual combination:

331   Evidence from property generation. *Acta Psychologica, 132*(2), 173–189.

332   doi:10.1016/j.actpsy.2009.02.002

Table 1

Example of Data Formatted for Tidy Data

word	answer
airplane	you fly in it its big it is fast they are expensive they are at an airport you have to be trained to fly it there are lots of seats they get very high up
airplane	wings engine pilot cockpit tail
airplane	wings it flys modern technology has passengers requires a pilot can be dangerous runs on gas used for travel
airplane	wings flys pilot cockpit uses gas faster travel
airplane	wings engines passengers pilot(s) vary in size and color
airplane	wings body flies travel

Table 2

Lemma and Part of Speech Information from TreeTagger

token	tag	lemma	lttr	wclass
is	VBZ	be	2	verb
are	VBP	be	3	verb
trained	VBN	train	7	verb
lots	NNS	lot	4	noun
seats	NNS	seat	5	noun
wings	NNS	wing	5	noun

Table 3

Original Data with Lemmatization

word	answer
airplane	you fly in it its big it be fast they be expensive they be at an airport you have to be train to fly it there be lot of seat they get very high up
airplane	wing engine pilot cockpit tail
airplane	wing it fly modern technology have passenger require a pilot can be dangerous run on gas use for travel
airplane	wing fly pilot cockpit use gas fast travel
airplane	wing engine passenger pilot(s) vary in size and color
airplane	wing body fly travel

Table 4

Multi-Word Sequence Examples for Zebra

Word	Feature	Frequency
zebra	be_horse	1
zebra	be_similar_horse	1
zebra	build_horse	1
zebra	horse	22
zebra	horse-like	1
zebra	look_similar_horse	1
zebra	related_horse	1
zebra	resemble_small_horse	1
zebra	run_fast_horse	1
zebra	run_horse	1

Table 5

Bag of Words Examples for Zebra

Word	Feature	Frequency
zebra	stripe	71
zebra	black	63
zebra	white	61
zebra	be	56
zebra	animal	54
zebra	have	54
zebra	a	46
zebra	and	46
zebra	in	41
zebra	horse	32

Table 6

Descriptive Statistics of Text Processing Style

Statistics	Multi-Word Sequences			Bag of Words		
	M	*SD*	*r*	*M*	*SD*	*r*
Number of Cue-Features	191.85	98.19	0.74	171.80	76.96	0.66
Frequency of Idiosyncratic Response	182.57	96.43	0.76	158.85	73.97	0.69
Frequency of Cue-Feature Response	2.14	3.46	0.73	2.73	4.80	0.83
Percent of Cue-Feature Response	3.47	5.14	-0.64	4.34	4.80	-0.62

Note. Correlation represents the relation between the statistic listed for that row and the sample size for the cue.
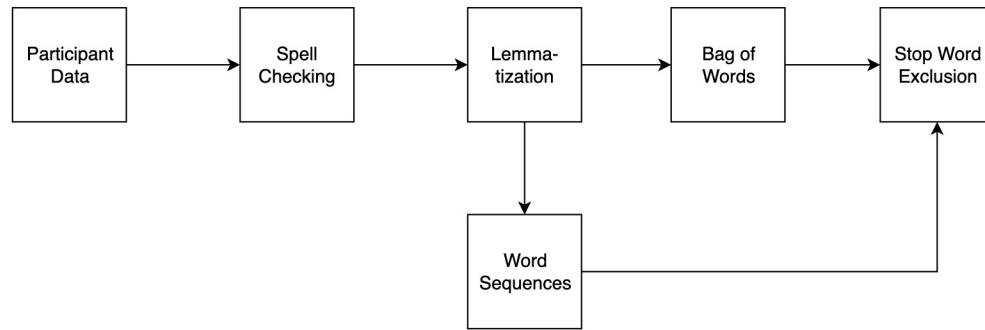
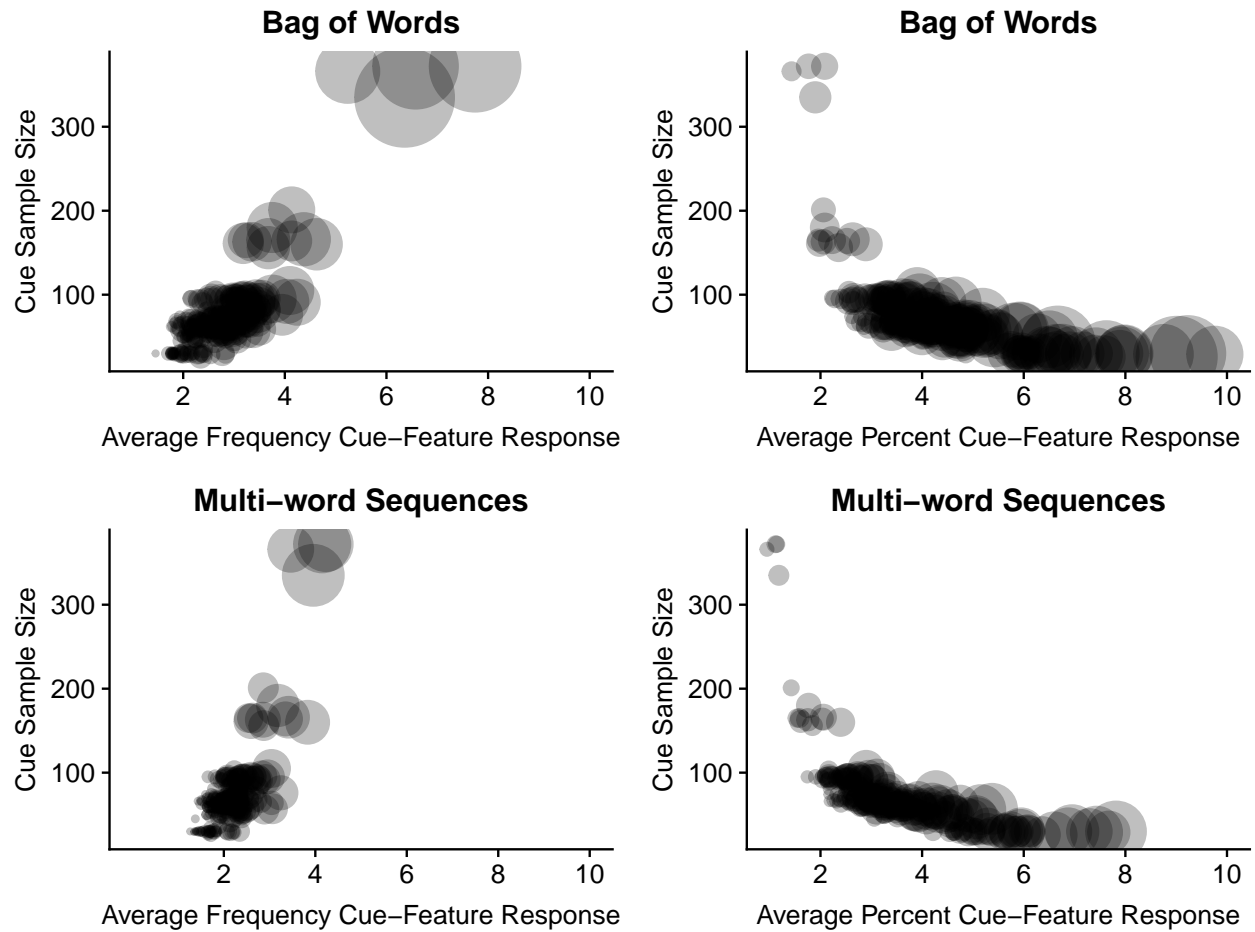*Figure 1*. Flow chart of proposed semantic processing feature steps.

Figure 2. Correlation of sample size with the average cue-feature frequency (left) and percent (right) of response for each cue for both processing approaches. Each point represents a cue word, and the size of the point indicates the variability of the average frequency or percent.