The *lrd* Package: An *R* Package and Shiny Application for Processing Lexical Data

Nicholas P. Maxwell1, Mark J. Huff1, and Erin M. Buchanan2

1 The University of Southern Mississippi

2 Harrisburg University of Science and Technology

Author Note

Correspondence concerning this article should be addressed to Nicholas P. Maxwell, School of Psychology, 118 College Dr, Hattiesburg, MS, 39406. E-mail: [nicholas.maxwell@usm.edu](mailto:nicholas.maxwell@usm.edu). The source code for this package as well as all applicable data files have been made available for download at https://osf.io/admyx/. The Shiny application can be accessed at: <https://npm27.shinyapps.io/lrdshiny/> or via www.macapsych.com.

Abstract

Recall testing is a common assessment to gauge memory retrieval. Responses from these tests can be analyzed in several ways; however, the output they generate typically requires manual coding that can be time intensive and error-prone before any analyses can be conducted. To address this issue, this article introduces *lrd* (Lexical Response Data), an open-source tool for quickly and accurately processing lexical response data that can be used either from the *R* command line or through an *R Shiny* graphical user interface. First, we provide an overview of this package and include a step-by-step user guide for processing both cued and free-recall responses. We then validate this program using two methods. First, we use *lrd* to recode output from both cued and free-recall studies with large samples and test whether the results replicate using *lrd* scored data. We then assess the inter-rater reliability and sensitivity and specificity of the scoring algorithm relative to human-coded data. Overall, *lrd* is highly reliable and shows excellent sensitivity and specificity, indicating that recall data processed using this package are remarkably consistent with data processed by a human coder.

Word Count: 184

Keywords: Memory; Cued-Recall; Free-Recall; Lexical retrieval; Recall processing; Recall scoring

The lrd Package: An R Package and Shiny Application for Processing Cued-Recall Data

People are generally able to acquire new knowledge with relative ease. Much of our understanding of how individuals organize and store learned information comes from the use of recall tests (see Polyn, Norman, & Kahana, 2009, a review). These procedures present participants with a set of items to learn within a controlled environment, and following study, participants are asked to recall them on a later test. Recall can either be assessed via free report, in which individuals report information from memory with few if any cues or constraints, or by the presentation of a cue that is used to direct their retrieval. Commonly, cued-recall performance is gauged through the use of cue-target pairs in which participants are required to retrieve a target item at test with the presence of a cue. Recall tests are common in memory research, including studies investigating the effectiveness of different memory strategies (e.g., deep vs. shallow encoding tasks; Craik & Lockhart, 1972), survival processing (e.g., assessing memory for contaminated objects; Gretz & Huff, 2019), and metacognition (e.g., accuracy between judgments of learning and recall; Koriat & Bjork, 2005). Furthermore, because studies often employ words as stimuli (i.e., cue-target pairs), much research has been conducted to explore how the lexical properties of the cue and target can influence later recall (e.g., concreteness, Pavio, Clark, & Khan, 1988; word frequency, Criss, Aue, & Smith, 2011) or how the semantic relationships between pairs affect recall (Nelson, McEvoy, & Schreiber, 2004). Though the research questions differ, recall studies generally employ lexical information in some capacity, either as the stimuli that participants are required to study, the dependent variable of interest, or more commonly, through a combination of the two.

Cued-recall tests are common in psychological research. A cursory search of Google Scholar for the keyword “cued-recall” yields approximately 18,000 publications since 2000, with these results spanning multiple subfields of psychology including neuroscience, psycholinguistics, and cognitive aging. The abundance of these studies can in part be attributed to the rise of the internet and the availability of more powerful computers, which has allowed information about lexical characteristics (such as word length or frequency) to be more efficiently collected and organized. As a result, the past two decades have provided researchers with access a growing number of these normed databases with which to construct lexical stimuli for use within recall studies (e.g., The English Lexicon Project, Balota et al., 2007; The Semantic Priming Project, Hutchison et al., 2013; The Small World of Words Project; De Deyne, Navarro, Perfors, & Brsybaert, 2019). Recently, online tools to aid researchers in selecting stimuli from the appropriate normed database have been made available (e.g., The Linguistic Annotated Bibliography; Buchanan, Valentine, & Maxwell, 2019a) and computer applications such as the *lexOPS* package for *R* ­(Taylor, Beith, & Sereno, 2019) have been developed to automate the stimuli selection process entirely while controlling for several types word properties. Though there has been a proliferation of datasets and tools used to aid researchers with stimuli creation, little attention has been given to developing tools that assist researchers with processing the large amounts of data that are generated from these studies.

Since studies investigating memory typically generate large amounts of lexical data, processing the output obtained from these studies is often a time-consuming and tedious task. Furthermore, the number of participants recruited to take part in these studies has drastically increased within the past decade, partially as a response to the replication crisis (e.g., Maxwell, Lau, & Howard, 2015), which has resulted in an even greater need for efficient methods for processing recall data. As such, the purpose of this paper is to introduce the *lrd* (lexical response data) package, which has been designed to provide researchers with a set of simple and freely accessible tools that can be used to speed up scoring of lexical memory output.

Output from cued-recall tests are generally scored by matching participants’ responses to the various stimuli against a scoring key containing the correct set of responses. Though typed responses are unquestionably easier to process relative to handwritten responses, each response item must still be manually checked against the key to determine accuracy. For large datasets, manually scoring data is arduous, resulting in hours of checking participant responses against an answer key. While such tasks can generally be divided across research assistants in a lab, this may still prove to be a time-consuming endeavor depending on the amount of data requiring processing. Furthermore, this can potentially introduce error in the coded responses, as inconsistencies across raters may arise if not properly controlled for (i.e., addressing participant misspellings, plural vs. singular nouns, alternate tenses, etc.) and scoring discrepancies are not resolved.

To reduce both the overall time spent processing raw output and potential coder inaccuracies, an alternative method is to automate the data coding processes by employing a computer application that can automatically compare participant responses relative to a scoring key. However, simply having a program match responses does not account for participant errors in responses that may still involve a correct memory as these may be scored as incorrect due to misspellings or the addition of an additional character either before or after a memory item such as an extra space. While a human scorer could certainly adjust any minor character additions, omissions, or misspellings to score correctly retrieved memory items, an automated program may not unless a sufficient degree of flexibility is programmed into the scoring package.

The functions comprising the *lrd* package have been specifically designed to accurately score lexical data while granting increased flexibility for minor errors that may be present in recall output. Importantly, this cost-free package has been carefully crafted to require minimal programming experience. The goal of this article is two-fold. First, we provide brief overviews of each function contained in both the *lrd* package in the *R* environment and detail the corresponding *R Shiny* application by providing step-by-step guides on how to implement each of these tools to process both cued-recall and free-recall data. Second, we test the accuracy and reliability of the scoring algorithm by comparing output obtained from this package with human coded data using three large data sets. Specifically, we test this package’s reliability by using its scoring functions to recode cued-recall data derived from two recent cued-recall studies (Maxwell & Buchanan, 2020; Maxwell & Huff, in press) and a study employing a free-recall task (Huff, Yates, & Balota, 2018). We then compare data processed using *lrd* to the findings in the original human coded datasets and test whether the original findings reported in these studies replicate. These studies were selected given that all were memory studies that required participants to complete a recall test, their similarity in design, which allowed for easy comparison between each study, and their use of a wide variety of different items to test the reliability of the *lrd* package.

For the two cued-recall studies, participants studied lists of paired associates and judged either how related the words in each pair were (Maxwell & Buchanan, 2020) or how likely they would remember the second word if cued by the first at test using a Judgment of Learning rating (Maxwell & Huff, in press). Upon conclusion of the study/judgment tasks, participants completed a distractor task followed by a cued-recall task in which the first word in each pair was presented and participants were asked to respond with the item it was originally paired with (e.g., mouse - ?). For the free-recall data taken from Huff et al. (2018), participants studied six word lists in which list items were either semantically related or unrelated. Following study of each list, participants then engaged in a free-recall task. The recall data reported in each of the above studies was initially scored by manually checking responses against a scoring key via human coders. We rescored this output using *lrd* to illustrate that output generated automatically from this package is able to replicate human scored results across both recall paradigms with a high degree of precision.

**Overview of the *lrd* Package**

*lrd* is an open-source package developed for the *R* environment that consists of three basic functions for scoring lexical cued-recall data and assessing the output. This package’s primary goal is to automate the scoring process by matching participant responses to a list of correct responses stored in a key. Critically, this package has been designed to accomplish this task while granting flexibility towards participant response errors (e.g., misspellings or incorrect tenses).

We begin by providing a set of general instructions for downloading and installing the *lrd* package within the *R* environment. Next, we provide a basic overview of the scoring functions for both cued and free recall as well as a set of functions that can be used to compute recall proportions for each test type. Third, we provide a general guide on how to use the package within both the *R* environment and through the use of a graphical user interface (GUI) implemented in *Shiny* (RStudio Inc, 2020). Finally, we conclude by assessing the validity of this package by using the cued and free recall scoring functions to process sets of cued and free recall data that have been scored by human coders.

**Installation and Set Up**

Here we present instructions and setup for users wishing to use *lrd* within the *R* environment. Specific instructions for accessing and implementing the *Shiny* application are provided following our description of the *lrd* *R* package.

The latest version of *lrd* (including all applicable documentation and source code for each function) can be accessed via GitHub (https://github.com/npm27/lrd). While proficiency with *R* is not required to run this package, it is assumed that users will have some familiarity with the *R* environment and/or basic experience with object-oriented programming. Installation is relatively straightforward, but currently requires the use of the *devtools* package (Wickham, Hester, & Chang, 2019) to download and install the files from GitHub. Typing the following command, devtools::install\_github(‘npm27/lrd’) will begin the installation process by downloading and installing the latest version of *lrd* along with all dependency packages from GitHub. Source code has been made available on GitHub (https://github.com/npm27/lrd/), and researchers are able to download and modify functions of this package as needed.

**Function 1: Compute the Percent Match Between Response and Key**

The percent\_match.cr() function allows for the comparison of a participant’s typed response with the correct response stored in a scoring key. At a minimum, this function requires three user inputs that are then arranged in a dataframe object: A list of participant responses (generally typed responses collected through a computer based data collection tool such as *E-Prime*), a scoring key containing the correct response, and a unique identifier for each participant. The percent match function works by treating each word in the dataframe columns as a string object. This function then computes the percentage of shared characters between a participant’s typed response and the corresponding correct response from the key and returns this value as a new column in the dataframe. The percentages derived by this function are computed bidirectionally to account for any differences in length between the response and key that may arise due to participant errors (i.e., the longer item is always the denominator when computing this percentage). For example, if the word *home* is stored in the scoring key, *lrd* will compute a participant response of *hom* as a 75% percent match with the target due to the absence of one of the four letters (i.e., 25%), while a response of *homme* would be computed as a match of 80% (i.e., the four letters in the original word divided by the five letters in the response). Table 1 illustrates how output obtained using percent\_match.cr()is formatted.

Since percent\_match.cr()relies on the length of the two words being compared when computing this percentage, shorter words are more likely to have lower percent matches and are thus at a disadvantage relative to longer words. This is because typos and misspellings will have a greater negative impact on short words when this percentage is calculated. Table 1 provides an example of this. Looking at the percent match column, the participant responses for the items *home* and *windshield* each contain one typo relative to the answer key. However, because *home* is a four-letter word, the negative impact that this misspelling has on the percent match is magnified relative to when the ten-letter word *windshield* is misspelled. To account for this, percent\_match.cr()also includes an optional “weighted” argument, should researchers wish to control for item length when processing responses. By activating this argument (weight.by = TRUE), the user is able to specify a value that is then used to adjust the percent match values. The weighted percent match is then computed using the following formula:

(1)

In Equation 1, *p* represents the percentage of shared characters between the participant response and its corresponding answer key (i.e., the percent match between the two), *v* is a user specified weight value ranging from 0 to .99 that is specified using the weight.by argument, and *c* equals the total number of characters comprising the correct response (as stored in the answer key). Weighted match values computed from Equation 1 are then stored in a separate vector which is appended to the dataframe, rather than overwriting the initial, unweighted percent match values. The inclusion of both columns allows the user to see what improvements in scoring accuracy occur by using the weighed match method relative to using the non-weighted percent match values.

**Function 2: Score Cued-Recall Responses as Correct or Incorrect Based on Percent Match**

The second function comprising *lrd* is the score\_recall.cr() function. This function operates by taking the saved output from percent\_match.cr() and using values stored in the percent match column to determine whether an item was recalled correctly. Using the cutoff argument, the user is able to specify a cutoff value that *p* must eclipse in order for the response to be marked as correct. For example, if the cutoff value is set at 0.80, then responses that are at least an 80% match (i.e., *p* ≥ .80)would be marked as correct. Because the user is able to freely specify a desired cutoff point, this allows the scoring algorithm to be tuned to the dataset being processed.

Using score\_recall.cr() returns a .csv file saved to the working directory. The first three columns of this file contain the three initial inputs used when computing the percent match (i.e., participant number, participant response, and answer key). The remaining columns denote the percentage of characters shared between the response and the key, the weighted version of this percentage (if applicable), whether the item was recalled correctly. Scores are represented as a series of 0’s and 1’s denoting whether the item was correctly recalled (i.e., 0 = incorrect, 1 = correct). Table 1 illustrates the structure of the output file. Because *lrd* does not change the row order of the input file, any additional columns (such as those denoting experimental conditions) can then be appended to the scored output.

**Function 3: Compute Proportion of Correct Cued-Recall Responses**

The final function for processing cued recall data with *lrd* is prop.correct.cr(). This function operates on the scored output and provides a means for quickly assessing each participant’s mean recall score. At minimum, this function requires the user to provide two inputs: A dataframe column consisting of item scores (e.g.., scores generated using the score\_recall.cr() function) and a column containing participant identifiers, which is specified using the id argument. This will return a dataframe object consisting of each participant’s mean recall score collapsed across any experimental conditions and automatically computes *z*-scores (see Table 2 for an illustration of how this output is formatted). If the scored output contains additional columns denoting experimental conditions, then these can be used to group the participant means. Using the group.by argument, the user can specify one column that can be used as a grouping variable. Mean recall scores will then be displayed for each participant at each level of the grouping variable. Finally, if a researcher wishes to compute condition means collapsed across participants, the id argument can be set to a condition column.

**Function 4: Arrange Free-Recall Data for Scoring**

As with cued-recall data, *lrd* requires that free-recall data be arranged in long format such that each individual participant response is a unique string and is stored in its own cell within the dataframe. However, often times participants completing free responses tasks will input all responses as one string (i.e., all responses are stored within one dataframe cell; see Tables 2 and 3 for an illustration). To account for this, the fourth function contained in this package (the arrange\_data()function) can be used to structure free recall data into the correct arrangement used for processing free-recall data in *lrd*. This function requires three user inputs: A column containing participant responses, the character separating each response (denoted using the sep argument) and the column containing participant identifiers. Finally, any additional columns denoting experimental conditions can be included using the other argument. Using arrange\_data() returns a dataframe object with at least three columns: The participant identifier, participant responses, and position column denoting the order in which a word was recalled. Any additional columns are included following the third column. Tables 3A and 3B provide an illustration of both the required input and the output obtained from using this function.

**Function 5: Score Free-Recall Responses**

Once free-recall data has been arranged into the proper format, the score\_recall.f() function can be used to match each participant response to the scoring key and compute whether an item was correctly recalled. This function operates by taking each participant response and matching it to the scoring key. Levenshtein edit distances (Levenshtein, 1966) are then computed between each participant response and each item stored in the scoring key using functions from the *RecordLinkage* package (Sariyar & Borg, 2010). These distance values represent the number of character changes required to transform the first word into the second. For example, two identical words such as *cat* and *cat* would have distance of 0, while *cat* and *bat* would have a distance of 1, and *cat* and *dog* would have a distance of 3. We selected this measure instead of using the percent match algorithm introduced with the cued-recall functions because unlike the percent match, Levenshtein distances are sensitive to changes in character order. Thus, while *bear* and *bare* would be computed as a 100% character match, they would receive a Levenshtein distance score of 3, indicating the changed order of the characters between the two words. As with the cued-recall function, the user is able to freely specify a desired cutoff point used for matching responses, allowing the scoring algorithm to be tuned to the dataset being processed.

After running score\_recall.f() the scored output is saved to a .csv stored in the working directory. This file contains four columns consisting of the participant’s id number, their responses, matching response from the answer key, and participant scores for each item. (Represented as a series of 0’s and 1’s). Table 4 shows the structure of this output file.

**Function 6: Compute Proportion of Correct Free-Recall Responses**

The next function included with *lrd* is prop.correct.f(). This function can be used to compute participant level proportions of correct responses for free recall data and is analogous to prop.correct.cr(). Three inputs are required to run prop.correct.f(). The first two inputs can be obtained from the output of score\_recall.f(). These include the output column containing participant scores and the column containing participant identifiers. Finally, using the key.length argument, the user must specify the number of objects contained in the scoring key. Mean recall scores will then be displayed for each participant, and this output can then be saved to a dataframe.

**Function 7: Compute the Percent Match Between Two Sentences**

The final function included with *lrd* is sentence\_match(). This function computes the percentage of words that are shared between two sentences. Additionally, it returns unique items within each sentence (i.e., items in sentence one that are missing in sentence two or vice versa). Similar to the previous percent match function, sentence\_match()was designed to operate on a dataframe containing participant responses and a scoring key. As such, this function requires three inputs derived from the dataframe: The column containing participant responses, the column containing the scoring key, and the column participant’s unique identifier. Finally, a percent match value for scoring needs to be specified using the cutoff argument. This function then returns a dataframe object containing the three initial input columns, the percent match between the sentences, a column denoting words omitted in the response, and a column denoting extra words in the response that are not in the scoring key.

**Cued-Recall Scoring Functions Example**

In this section, we provide a general guide to using *lrd* to score cued-recall data. This example uses a set of simulated response data that were designed to mimic output that might be obtained from a cued-recall study. While the dataset is smaller than what is typically generated from psychological experiments, we note that they are sufficient for our purpose of illustrating how *lrd* scores participant responses. We begin this section by detailing the creation of this dataset before providing a step-by-step walkthrough of the *lrd* package’s cued-recall scoring functions. Code and data for all examples is available at [LINK].

**Materials and Dataset Creation**

To simulate a set of cued-recall data, forty words were randomly generated using *LexOPS* (Taylor et al., 2019) to serve as target items (i.e., the scoring key containing correct responses). To simplify the stimuli selection process, we followed the general example provided by Taylor et al. by controlling for word prevalence and concreteness when generating this set of items. First, only highly concrete words were included (concreteness ≥ 4; Brysbaert, Warriner, & Kuperman, 2014). Pairs were then evenly split based on word prevalence (e.g., the proportion of individuals who are familiar with a word; Brysbaert, Mandera, McCormick & Keuleers, 2019). Thus, the final stimuli consisted of 20 concrete, high prevalence words (i.e., prevalence ≥ 4) and 20 concrete, low prevalence words (i.e., prevalence ≤ 2).

We next simulated six sets of participant responses to these items. These response simulations varied in their degree of accuracy to cover a broad spectrum of potential participant responses, including no response errors (Participant 1), minor misspellings (Participants 2, 3, and 4), and major response errors (e.g., blank responses, incorrect answers, misspellings of more than two letters; Participants 5 and 6). For Participant 1, all responses matched the key to simulate a situation in which a participant correctly recalls all items. Data for Participants 2 and 3 was manipulated to simulate situations in which participants make minor mistakes at recall that don’t necessarily preclude their responses from being counted as correct (e.g., misspellings where it is evident what the intended word is). These were generated by removing, replacing, or doubling specific letters. As such, the letter “e” was removed from all responses for subject 2 (e.g., “hey” becomes “hy”). For Participant 3, all instances of the letter “t” were doubled (e.g., “edit” becomes “editt”). Next, for Participant 4, all instances of the letter “a” were replaced with an “e”. This allowed us to simulate a range of common participant errors such as omitting a letter, typing the wrong letter, or double pressing a key by mistake. Finally, data for Participants 5 and 6 were manipulated to simulate situations in which participants make major mistakes on recall (e.g., responding with an incorrect word). To simulate this type of response error for Participant 5, five responses from the answer key were randomly changed to a different but conceptually similar word (e.g., *financial* becomes *money*). The simulated data for Participant 6 increased the number of incorrect responses and added three instances of missing data. This sample dataset (test\_data.csv) has been made available for download at https://osf.io/admyx.

**Formatting and Loading the Dataset**

When processing cued-recall data, the *lrd* package requires that the initial input data is formatted as .csv with a header row. This file will need to be arranged in long format and must contain the following columns: A unique identifier for each participant, a participant response column, and a unique identifier for each tested item (e.g., such as a trial number). Additionally, this function requires an answer key containing the correct responses and a unique identifier for each key item; however, these columns can either be stored as part of the input data or may be stored in a separate .csv file of the same length as the file containing participant response. Finally, the upload file may contain additional columns (e.g., columns denoting experimental conditions) that can be used to group the output. Because scoring is case sensitive, the response and answer key columns will need to be checked for case discrepancies. For simplicity, we suggest converting both the answer key and response columns to lowercase before scoring the data. Finally, any missing responses will need to be converted from NAs to blanks.

[R CODE]

##set up

library(lrd)

##load test data

dat = read.csv("test\_data.csv")

##make sure everything is lowercase

dat$Target = tolower(dat$Target)

dat$Answer = tolower(dat$Answer)

##replace NAs with blanks

dat$Answer[is.na(dat$Answer)] = ""

**Scoring Cued-Recall Data**

Scoring cued-recall data with *lrd* is a relatively straightforward process. To begin, run prop\_correct\_cued()and save the output as a new object (see code below for an example). When running this function, you will need to specify the columns containing the participant responses, the answer key, and the subject number. You will then need to specify the Levenshtein distance used for scoring (for this example, we used a cutoff value of 1). To illustrate how changes in the percent match cutoff value influence final scores for various types of participant errors, Table 5 displays a subset of the scored output (one word per participant) at each cutoff value (the full output file generated from this example has been made available on our OSF page). This function provides up to three sets of output, which can then be saved a list object. Outputs can then be accessed using the $ operator (see example code). First, this function provides trial level data showing participant responses to each test item, the corresponding answer key item, and whether the program scored the response as correct (denoted as 1 for correct, 0 for incorrect). Next, this function provides correct recall proportions at the participant level, along with corresponding *z*-scores. Finally, if a grouping condition is selected at scoring, recall proportions collapsed across grouping condition can be accessed.

[R CODE]

scored\_cued = prop\_correct\_cued(dat,

responses = "Answer",

id = "Sub.ID",

id.trial = "Trial\_num",

key = "Target",

key.trial = "Trial\_num",

cutoff = 1)

#View the output

View(scored\_cued$DF\_Scored) #Trial level

View(scored\_cued$DF\_Participant) #Participant proportion correct

**Free-Recall Scoring Functions Example**

The next section provides a general guide for using *lrd* to score free-recall data. For this example, we simulated a set of free-recall responses. The sample data was modeled after output obtained by Gretz & Huff (2019) in which participants watched videos of either healthy or sick individuals interacting with a variety of household objects and were presented with a free recall test. First, we by detail the creation of this dataset. We then provide a detailed walkthrough of the *lrd* package’s free-recall scoring function.

**Materials and Dataset Creation**

To simulate a set of free recall data, a list of 22 common household objects was first generated. This list was based on the “bedroom” list used by Gretz and Huff (2019) and can be viewed at https://osf.io/qbrgm/. Next, to simulate a set of responses, data was generated for six participants. To capture response variability, we varied the number of responses each participant provided, spelling errors of correct items, and inclusion of incorrect items. The full sample dataset and answer key files as well as all code used in the following examples has been made available at [OSF LINK].

**Formatting and Loading the Dataset**

To process free-recall data with *lrd*, two separate upload files are required: The participant response data and the scoring key. Both files will need to be formatted as .csv and contain a header row. For the scoring key, this can be structured as a one column .csv file in which each row contains a correct response. The .csv file containing participant responses will need to be structured using a similar format as the required by the cued-recall functions and include participant identifiers and responses. Furthermore, this file will need to be arranged in long format such that each participant response corresponds to a row in the data. However, because free-recall data is not generally structured in this format, the arrange\_data() function can be used to correctly structure the output for situations in which participant responses are contained within the same dataframe cell.

[R CODE]

# Arrange Free Recall for Scoring

dat = read.csv("free\_example\_data.csv") #Load dataset

arranged = arrange\_data(dat,

responses = "Response", id = "Sub.ID", sep = ", ")

This code returns a dataframe that contains at least three columns: The participant identifier, participant responses, and a position column denoting the order in which a word was recalled. Additionally, any extra columns (such as those denoting experimental conditions) are included following the third column. As with cued-recall scoring, the free-recall functions are case sensitive and cannot process missing responses. As such, we again recommend converting both the answer key and response columns to lowercase before scoring the data and replacing all missing responses (NAs) with blanks.

**Scoring Free-Recall Data**

After correctly formatting the participant responses, the data is now ready to be scored. Free-recall data is scored using the prop.correct.free() function. The scored output can be saved as a new object (see the example code below). To run this function, you will need to specify the columns in the response data that contain the participant responses and the subject number. You will also need to specific the answer key used for scoring. Finally, you will need to specify the cutoff value to use for scoring. As with cued-recall scoring, the cutoff denotes the Levenshtein distance score used to match the participant response to the scoring key. For this example, we use a cutoff value of 1. The output from prop.correct.free()is structured in the same manner as the cued-recall output. Thus, this function provides trial level data showing participant responses to each test item, correct recall proportions at the participant level, and (if specified) recall proportions collapsed across grouping condition can be accessed. To illustrate both how the output file is structured and how changes in scoring criteria can affect accuracy, Table 4 displays a subset of the scored output (participant 1) at matches of 0, 1, and 2 (see our OSF page for the full output file).

[R CODE]

####Score the data####

key = read.csv("free\_example\_key.csv") ##Load the key

scored\_free = prop\_correct\_free(arranged,

responses = "Response",

key = key$Answer\_Key,

id = "Sub.ID",

cutoff = 1)

#View the output

View(scored\_free$DF\_Scored) #Trial level

View(scored\_free$DF\_Participant) #Participant proportion correct

**Sentence Scoring Function Example**

In addition to scoring cued-recall and free-recall responses, *lrd* can also be used to score the match between sentences. In this section, we provide a general overview of using *lrd* to score sentences. This example uses a set of simulated sentence responses generated for six participants. We begin this section by detailing the creation of this dataset. We then provide a detailed walkthrough of the *lrd* package’s sentence processing functionality.

**Materials and Dataset Creation**

To simulate a set of sentence responses, we first generated a list of five simple sentences to serve as the answer key. Next, to simulate a set of responses, sample data was generated for six participants, leading to a total of 30 observations. To capture response variability, we varied the amount of error within each response, such that some sentences included spelling error, inclusion of extra words, omission of words, and/or semantically similar words. The full sample dataset and all code used in the following example has been made available at [OSF LINK].

**Formatting and Loading the Dataset**

When using *lrd* to process sentence responses, the input data will need to be structured as a long-format .csv file containing the following columns: A unique identifier for each participant, a participant response column containing the full sentence typed by the participant, and a unique trial number for each response. Furthermore, prop.correct.sentence() requires an answer key containing the full, correct sentence and a unique identifier for each key item. As with cued-recall scoring, these columns can either be stored as part of the input data or can be uploaded as a separate .csv file of the same length as the file containing participant response. Finally, the upload file may contain additional columns (e.g., columns denoting experimental conditions) that can be used to group the output. Sentence scoring is case sensitive, so the response and answer key columns will need to be checked for case discrepancies prior to scoring. Finally, any missing sentence responses will need to be converted from NA to blanks.

[R CODE]

##set up

library(lrd)

##load test data

dat = read.csv("sentences.csv")

##make sure everything is lowercase

dat$Sentence = tolower(dat$Sentence)

dat$Response = tolower(dat$Response)

##replace NAs with blanks

dat$Response[is.na(dat$Response)] = ""

**Scoring Sentence Data**

To score sentence data with *lrd*, begin by running prop\_correct\_sentence()and save the output as a new object. This function follows the same general format as the cued and free-recall scoring functions. As such, you will need to specify the columns containing the participant responses, the answer key, and the subject number. You will then need to specify the Levenshtein distance used for scoring (for this example, we used a cutoff value of 1). Finally, the token.split argument can be used to specify the character that separates words in each sentence (note that is argument defaults to a single blank space between words). This function provides up to three sets of output, which can then be saved a list object. As with the previous scoring function, up to three sets of scored output are available and can be accessed using the $ operator (see example code).

[R CODE]

####Score the data####

scored\_sentences = prop\_correct\_sentences(dat,

responses = "Response",

key = "Sentence",

key.trial = "Trial.ID",

id = "Sub.ID",

id.trial = "Trial.ID"

token.split = " ",

group.by = "Condition"

cutoff = 1)

#View the output

scored\_sentences$DF\_Scored ##Trial level data

scored\_sentences$DF\_Participant ##Participant level

scored\_sentences$DF\_Group ##Group level

**R Shiny Application**

While *lrd* was initially designed as a package to be used within the *R* command environment, we recognized the need for an easy to access option that can be used independent of *R*. As such, we have also developed a pair of *Shiny* applications that provide researchers with a programming-free alternative to using this tool that can be operated using basic Excel skills. Furthermore, because this application is web based (available at https://npm27.shinyapps.io/lrdshiny) no software downloads are required.

**Cued-Recall**

The cued-recall *lrd Shiny* application consists of four tabs: An instructions tab and three output tabs that display a preview of the scored dataset, tables showing mean correct responses, and plots that can be used for basic data visualization (Figures 1-4 display the various tabs). To begin the scoring process, a .csv file will need to be uploaded to the application. This file will need to contain at least three columns that have been arranged in the following order: A unique participant identifier, a scoring key, and a set of participant responses. The input file may also contain additional columns (e.g., optional columns denoting experimental conditions), however, these columns must be placed starting with the fourth column. To begin the file upload process, the input settings must first be selected (e.g., the type of separator used). Next, the scoring criteria must be specified. The application defaults to using a 75% match, but a slider has been provided to make simple online adjustments to scoring cutoffs. The strictest option available is a 100% match, however, this can be decreased down to a 50% match (all 1% increments between 100% and 50% are available). Changes in scoring that occur due to updating the percentage correct are reflected in the across the three output tabs in real time. Thus, researchers can quickly assess how different scoring criteria affect their data and can fine-tune the scoring algorithm to meet their individual needs. Once all options have been selected, the file can be uploaded (see Figure 1 for an example of how to configure the input settings).

After the input file has been uploaded, a preview of the scored data will appear in the “Scored Output” tab. A download button located at the top of this tab can be used to save the scored data as a .csv file (see Figure 2). Each participant's mean proportion of correct responses and corresponding *z*-score can be viewed using the "Proportion Correct" tab. The output displayed in this tab can be customized based on any of the optional condition columns that are attached to the upload .csv file (data can be split on up to two experimental conditions at a time, which are selected using a set of drop-down menus; see Figure 3), however, *z*-scores will only be displayed when grouping a dataset by one condition (e.g., showing each participant’s mean proportion correct or collapsing across participant to show condition means). As with the scored output, proportion correct values can be saved as a .csv file using the download button located at the top of the tab. Finally, the "Plots" tab can be used to visualize the scored data. As with the proportion correct, these plots can also be customized based on any of the optional condition columns uploaded with the dataset to conditionalize the visual output as a function of item type or other grouping variable (see Figure 4). If no condition columns are included with the upload file, this tab will instead display a histogram showing the distribution of participant responses (this can also be viewed at any time by selecting the participant identifier as the grouping condition).

**Free-Recall**

[WORDS HERE]

**Sentence Processing**

[WORDS HERE]

**Cued-Recall Scoring Functions Validation**

In the next section, we report results from two sets of analyses in which we tested the cued-recall scoring accuracy of *lrd*. Each analysis serves as an additional assessment to ensure *lrd* can consistently produce accurate scoring across different sets of stimuli. First, we use *lrd* to score the datasets used for each set of analyses. These data were derived from two sources: Maxwell and Buchanan (2020) and Maxwell and Huff (in press). We then conducted three sets of analyses to test the reliability of this package. First, we tested whether the results of these studies would significantly differ from the original findings after the raw data was processed and scored using *lrd*, allowing us to test the accuracy of this package at the participant level. We then computed Cohen’s *κ* to assess reliability between the different coding sources.

We begin this section by providing details for each dataset, including participant and stimuli characteristics for each study. We then discuss the selection criteria for the percent match value and detail the results of a set of sensitivity and specificity analyses that were used to test potential cutoff values and provide a step-by-step walkthrough of the scoring process. Finally, we conclude this section by detailing each of the analyses described above.

**Participants and Materials**

Each dataset was collected separately across two different experimental settings. The first set of participants was originally reported in Maxwell and Buchanan (2020; dataset available at https://osf.io/y8h7v/). This dataset consists of 222 participants who were recruited online via Amazon’s Mechanical Turk, a site which allows researchers to access a large pool of participants who complete surveys in exchange for small sums of money (Buhrmester, Kwang, & Gosling, 2011). Next, Maxwell and Huff’s (in press) data consists of 112 undergraduate students who were recruited from The University of Southern Mississippi’s psychology research pool and tested in lab (dataset available at https://osf.io/hvdma/). These participants completed the study in exchange for partial course credit and were recruited to take part in one of four experiments. For purposes of this paper, we collapsed across experiment to include all 112 subjects in one dataset. Combining datasets across both studies resulted in 31,301 recall entries generated from 334 participants.

Datasets were selected due to their similarity in design. Each study presented participants with paired associate study lists and later had them complete cued-recall tasks. Furthermore, each study contained reasonably sized samples (all *n*s > 90) and presented participants with at least 60 item pairs to study, providing us with a sufficient number of observations with which to test the reliability of this package. Each study presented participants with a set of cue-target paired associates (e.g., credit – card). Participants were asked to study each pair before making a judgment of either the pair’s relatedness or their ability to recall the pair at test. After completing the study and judgment tasks, participants then complete a cued-recall test. While participant judgments were collected in each experiment, they are not included in the following analyses as we are only interested in analyzing the accuracy of *lrd* in scoring recall responses.

First, Maxwell and Buchanan (2020) used 63-word pairs that were selected using the Buchanan et al. (2013) semantic feature overlap norms. The stimuli pairs used in this study were selected based on the strength of their semantic relatedness as measured by cosine overlap (See Buchanan, Valentine, and Maxwell (2019b) for a review of cosine overlap) while also controlling for association strength and thematic similarity. Next, the Maxwell and Huff (in press) dataset used 180 study pairs selected from the University of South Florida Free Association norms (USF norms, Nelson et al., 2004). Stimuli pairs used in this study were originally selected based on their levels of forward associative strength (FAS) and backward associative strength (BAS).

Each of these studies assessed participant recall using the same method. After conclusion of the study tasks, participants completed a cued-recall test in which the first item of each study pair was presented with the second item removed (e.g., *mouse - ?*). Participants in each study were informed that they would not be penalized for guessing or incorrect spellings of answers.

**Determining the Optimal Scoring Criterion**

Given the *lrd* package’s scoring functions work by computing the Levenshtein distance between two strings (i.e., the number of character insertions, deletions, or changes required to transform string A into string B), we first needed to determine the optimal distance score that would maximize the number of correct hits (e.g., true positives) while minimizing the number of false positives and false negatives. To determine this value, we conducted a set of sensitivity and specificity analyses for each dataset (see Altman & Bland, 1994, for review). Within the context of this study, sensitivity refers to the proportion of true positives that *lrd* correctly identifies (i.e., a participant correctly responds to the cue item with the correct target word and the program correctly identifies it), while specificity refers to the proportion of true negatives identified by the program (i.e., the program correctly identifies that a participant missed an item at test).

Sensitivity and specificity analyses were computed in *R* using the *caret* package (Kuhn, 2008). Tables 6 and 7 report sensitivity and specificity percentages for each dataset computed across of the five Levenshtein distance cutoff values. Overall, both datasets displayed a consistent pattern of results: Sensitivity and specificity were each maximized when the scoring cutoff used a Levenshtein distance of 1, suggesting that this value allowed the scoring algorithm to achieve maximum accuracy. We therefore suggest that a Levenshtein distance of 1 provides the optimal cutoff value for minimizing false positives and negatives; however, the program allows researchers to increase or decrease the cutoff value as desired.

**Data Processing and Scoring**

To assess the reliability of the cued-recall scoring functions, we next used *lrd*’s cued-recall scoring function to process and score the two cued-recall datasets introduced above. We then compared output obtained through this scoring process to the original, manually coded output that was originally reported in these studies and tested whether the original findings would replicate.

Prior to running the scoring algorithm, .csv files consisting of the participant responses, answer key, and unique identifiers for each participant were created for each of datasets. Data from each study were then scored using the prop\_correct\_cued() function. Scoring was an iterative process which used each of the six Levenshtein distances from the sensitivity and specificity analyses. Thus, each dataset was scored six times (once for each scoring criterion). This allowed us to track how changing the percent match criteria affected scoring accuracy.

**Analyses and Results**

After determining the optimal range of cutoff values to use with the scoring functions, we now turn to a set of analyses that test whether the data scored using *lrd* can successfully reproduce the results from each of the original manually scored datasets. We begin this section by providing descriptive statistics of recall rates for both the original and rescored datasets and then test whether these recall rates differ as a function of coder. Finally, we compute the inter-rater reliability between the human coded and *lrd* scored data. Each dataset was analyzed individually, providing us with two separate tests of the *lrd* package’s scoring accuracy. For all analyses, significance was set at the *p* < .05 level.

**Replication of Cued-Recall Studies**

First, each dataset was scored with *lrd* using six Levenshtein distance cutoff values between 0 and 5 as used in the sensitivity and specificity analyses. Next, two one-way Analysis of Variance (ANOVA) models were used on each dataset to test whether recall rates differed between the 7 scoring types (the 6 *lrd* scoring criteria plus the human coded data). For completeness, means, 95% CI’s and Cohen’s *d* effect size indices for all comparisons are reported in Tables 8 and 9.

Starting with the Maxwell and Buchanan (2020) dataset, a significant effect of scoring type was detected between the human coded data or the *lrd* scored data at any of the percent match cutoff values, *F*(6, 1320) = 558.12, *MSE* = 3115.42, *η*2G = .26. However, post-hoc analyses indicated that this effect was largely driven by differences between the higher Levenshtein distances (i.e., scored using a cutoff of 3 or greater) and the human coded data (54.14, *t*s ≥ 3.19, *d*s ≥ 0.29). Recall rates from the *lrd* scored data did not significantly differ from the human coded data when *lrd* scoring used a Levenshtein distance of 2 (53.37), 1 (52.14), or 0 (50.23; *t*s ≤ 1.23, *p*s ≥ .21).

Next, for the Maxwell and Huff (2020) dataset, a significant effect of scoring type was also detected, *F*(6, 666) = 1433.93, *MSE* = 14.82, *η*2G = .53. Post-hoc analyses again showed that this effect was largely driven by differences in mean recall between the human coded data (43.96) and the higher Levenshtein distances *lrd* scoring conditions, *t*s ≥ 3.86, *d*s ≥ 0.52. Recall rates did not differ between the human coded data and any of the other *lrd* cutoff points, *t*s ≤ 1.60, *p*s ≥ .11. Thus, using *lrd* to score participant responses did not result in significant changes in outcome across any of the experiments, particularly when an optimized scoring criterion as based on the sensitivity and specificity analyses was used. As such, these findings suggest that *lrd* is able to code cued-recall data equivalently to human coders.

**Inter-Rater Reliability**

To test the inter-rater reliability between the original data and the rescored data, we computed *κ* values for all data sets at the individual trial level. These values were computed in *R* using the *psych* package (Revelle, 2019). The *κ* statistic ranges from -1 to 1, and inter-rater reliability is considered strong if *κ* exceeded .80 (Cohen, 1960).

Beginning with the Maxwell and Buchanan (2020) data, a strong agreement was detected between the human coded data and response sets scored using Levenshtein distances of 0, 1, and 2, *κ*s ≥ .96, with this agreement weakening when the data was scored using higher Levenshtein distances ( The Maxwell and Huff (in press) dataset showed a similar pattern of agreement between coding methods, with strong agreement for Levenshtein distances less than 3, *κ*s ≤ .79, and weaker agreement when more liberal scoring criteria were used (*κ*s ≤ .85). Table 10 reports individual *κ* statistics for all comparisons within each dataset. Across datasets, reliability between human and *lrd* scored data was highest when a Levenshtein distance of 1 was used, and lowest when scoring used a Levenshtein distance of 5. As such, these results provide further evidence that using *lrd* to score cued-recall responses results in output that is highly consistent with what is produced by human coders.

**Free Recall Scoring Functions Validation**

We now turn to a set of analyses in which we tested the *lrd* package’s ability to accurately score free-recall data. First, we detail the dataset, including all participant and stimuli characteristics. We follow the same general procedure used to validate the cued-recall functions, including the use of sensitivity and specificity analyses to test potential cutoff values and comparing the *lrd* scored output to the original human coded data as a test of whether the original results can replicate. Finally, we conclude the analyses by computing Cohen’s *κ* to assess reliability between the various coding sources.

**Participants and Materials**

All data used in these analyses was originally published in Experiment 4A of Huff et al. (2018), who recruited 120 to complete the study online via Amazon’s Mechanical Turk. Participants were presented with three types of study lists: Categorical lists in which items were strongly related to one another (e.g., birds, fruits, etc.), ad-hoc lists in which items were weakly e.g., things made of wood, things that are liquids, etc.), and unrelated lists in which items shared no semantic relatedness. Each list contained 20 items and all participants studied six lists. List type was manipulated within subjects such that participants studied 2 of each list type. Thus, each participant always studied 2 categorical lists, 2 ad-hoc lists, and 2 unrelated lists. Following presentation of each list, participants completed a cued-recall task. As such, this provided us with 720 individual free-recall tests (120 participants X 6 list presentations). Because each list contained 20 items, this resulted in 14,400 potentially correct responses.

**Determining the Optimal Scoring Criterion**

Before scoring the data, we again needed to determine the optimal cutoff value for this function that would maximize the number of correct hits (e.g., true positives) while minimizing the number of false positives and false negatives. To do so, we again turn to a series of sensitivity and specificity analyses for each the three datasets. These analyses followed the same design used when validating the cued-recall functions.

Table 11 displays sensitivity and specificity percentages for each dataset computed at each of the selected cutoff values. Each of the three datasets displayed a similar pattern of results. Sensitivity and specificity were maximized when the Levenshtein distance was set at either 2 (ad-hoc and categorical lists) or 3 (unrelated lists). As such, we propose that using a cutoff of 2 at scoring provides the best method to mitigate false positives and negatives. However, prop.correct.free() allows this value to be edited as desired, providing users with maximum control over the scoring process.

**Data Processing and Scoring**

Next, to assess the reliability of the free-recall scoring functions, we used *lrd* to first convert the data into the correct format and to then score participant responses. First, arrange\_data() was used to convert the data into long format. The output data contained participant responses and unique identifiers for each participant. Next, a scoring key was created for each of the 6 lists. Each list was then scored separately using prop.correct.free(). This was an iterative process which used each of the six cutoff values as used in the sensitivity and specificity analyses, allowing us to monitor how changes to the cutoff criteria affected the scored output. The final datasets were created by combining the scored output within each list type at each of the four cutoff values. This resulted in three datasets, each corresponding to one of the three list types (categorical, ad-hoc, or unrelated).

**Analyses and Results**

We next conducted a series of analyses that tested whether free-recall data scored with *lrd* successfully replicates the results from the original human coded dataset. First, we provide descriptive statistics of recall rates for both the original and rescored datasets. Next, we test whether these recall rates differ as a function of coding. Finally, we conclude this section by computing the inter-rater reliability between the human and *lrd* coded datasets.

**Replication of Free Recall Studies**

First, data from each of the three list types were scored with *lrd* using all 6 Levenshtein distance cutoff values between 0 and 5. Next, three one-way ANOVAs were used to test whether recall rates differed between the 6 scoring types (the 5 *lrd* scoring criteria plus the human coded data) for each of the three study list types. For completeness, means, 95% CI’s and Cohen’s *d* effect size indices for all comparisons are reported in Table 12.

Beginning with the categorical list items, no significant differences were detected between the human coded data or the *lrd* scored data at any of the percent match cutoff values, *F*(6, 833) = 1.22, *MSE* = 220.57, *p* = .29. Additionally, this pattern replicated for the ad-hoc lists *F*(6, 833) < 1, *MSE* = 228.51, *p* = .95, and the unrelated lists, *F*(6, 833) < 1, *MSE* = 241.92, *p* = .99. As such, using *lrd* to score free-recall responses did not result in significant changes in outcome across any of the datasets, regardless of whether a strict or lenient scoring criterion was selected. Thus, the results of these analyses suggest that *lrd* is able to code free-recall data equivalently to human coders.

**Inter-Rater Reliability**

Finally, we computed *κ* values for all data sets at the individual trial level as a test of inter-rate reliability. Starting with the categorical list, a strong agreement was detected between the human coded data and the *lrd* scored data when using each of the six scoring conditions, *κ*s ≥ .89. Next, for the ad-hoc dataset, a moderate pattern of agreement was detected when scoring used cutoffs of 0 and 1 (*κ*s = .76), while a strong agreement was detected when scoring used a cutoff of 2 or greater (*κ*s = .92). Finally, the unrelated list exhibited a pattern similar to the categorical lists, with a moderate agreement observed between the human and *lrd* coded data when scored using cutoffs of 0, 1, and 2 (*κ*s = .80) and stronger agreement when using more lenient cutoffs (*κ*s = .93). Table 13 reports individual *κ* statistics for all comparisons within each dataset. Based on the results of these analyses, we again suggest using a Levenshtein cutoff of 3 when scoring free-recall. Taken together, the results of these analyses provide further evidence free-recall data scored with *lrd* to is consistent to what is generated by human coders.

[SENTENCE SCORING VALIDATION?]

**Summary and Conclusion**

Although recall tests are widely used in Psychology, no open access tools currently exist to quickly process the large amounts of lexical data that these studies generate. The *lrd* package addresses this need by providing researchers with a means of automating multiple types of recall scoring as a means to save time and minimize coding errors, while also being able to control for minor errors in participant responses. By using this package to replicate the results of two cued-recall studies and one free-recall experiment, we show that both cued-recall and free-recall data scored using *lrd* accurately reproduces manually coded data. We hope that *lrd* will both drastically reduce the amount of time spent coding lexical data and assist the reproducibility measures being adopted by the field by providing researchers with a standardized, open-source method for processing lexical output across psychological studies.

**Open Practices Statement**

The data for all experiments have been made available at https://osf.io/admyx/ and none of the experiments were preregistered.

References

Altman, D. G., & Bland, J. M. (1994). Diagnostic tests. 1: Sensitivity and specificity. *BMJ (Clinical research ed.)*, *308*(6943), 1552.

Balota, D. A., Yap, M. J., Hutchison, K. A., Cortese, M. J., Kessler, B., Loftis, B., Neely, J. H., Nelson, D. L., Simpson, G. B., & Treiman, R. (2007). The English lexicon project. *Behavior Research Methods 39*, 445-459.

Brysbaert, M., Mandera, P., McCormick, S. F., & Keuleers, E. (2019). Word prevalaence norms for 62,000 English lemmas. *Behavior Research Methods, 51*, 467-479.

Brysbaert, M., Warriner, A. B., & Kuperman, V. (2014). Concreteness ratings for 40 thousand generally known English word lemmas. *Behavior Research Methods, 46*, 904-911.

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*, 746-757.

Buchanan, E. M., Valentine, K. D., & Maxwell, N. P. (2019a) LAB: Linguistic Annotated Bibliography – a searchable portal for normed database information. *Behavior Research Methods, 51*, 1878-1888.

Buchanan, E. M., Valentine, K. D., & Maxwell, N. P. (2019b) English semantic feature production norms: An extended database of 4436 concepts. *Behavior Research Methods. 51*, 1849-1863.

Burhmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon’s Mechanical Turk: A new source of inexpensive, yet high-quality data? *Perspectives on Psychological Science, 6*(1), 3-5.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Education and Psychological Measurement, 20,* 37-46.

Craik, F. I. M. & Lockhart, R. S. (1972). Levels of processing: A framework for memory research. *Journal of Verbal Learning and Verbal Behavior, 11*(6), 671-684.

Criss, A. H., Aue, W. R., & Smith, L. (2011). The effects of word frequency and context variability in cued recall. *Journal of Memory and Language, 64*(2), 119-132.

De Deyne, S., Navarro, D. J., Perfors, A., Brysbaert, M., & Storms, G. (2019). The “Small World of Words” English word association norms for over 12,000 cue words. *Behavior Research Methods, 51*, 987-1006.

Gretz, M. R. & Huff, M. J. (2019). Did you wash your hands? Evaluating memory for objects touched by healthy individuals and individuals with contagious and noncontagious diseases. *Applied Cognitive Psychology, 33*(6), 1271-1278.

Huff, M. J., Yates, T. J., & Balota, D. A. (2018). Evaluating the contributions of task expectancy in the testing and guessing benefits on recognition memory. *Memory, 26*(8), 1065-1083.

Hutchison, K. A., Balota, D. A., Neely, J. H., Cortese, M. J., Cohen-Shikora, E. R., Tse, C.-S., … Buchanan, E. M. (2013). The semantic priming project. *Behavior Research Methods, 45*(4), 1099–1114.

Koriat, A., & Bjork, R. A. (2005). Illusions of competence in monitoring one’s knowledge during study. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 31*(2), 187-194.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software, 28*(5), 1-26.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics, 10,* 707-710.

Maxwell, N. P. & Buchanan, E. M. (2020). Investigating the interaction of direct and indirect relation on memory judgments and retrieval. *Cognitive Processing, 21*, 41-53.

Maxwell, N. P. & Huff, M. J. (2020). The deceptive nature of associative word pairs: The eﬀects of associative direction on judgments of learning. *Psychological Research*, 1-19.

Maxwell, S. E., Lau, M. Y., & Howard, G. S. (2015). Is psychology suffering from a replication crisis? What does “failure to replicate” really mean? *American Psychologist, 70*(6), 487-498.

Nelson, D. L., McEvoy, C. L., & Schreiber, T. A. (2004). The University of South Florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers, 36*, 402-407.

Pavio, A., Clark, J. M., & Khan, M. (1988). Effects of concreteness and semantic relatedness on composite imagery ratings and cued recall. *Memory & Cognition, 16*, 422-430.

Polyn, S. M., Norman, K. A., & Kahana, M. J. (2009). A context maintenance and retrieval model of organizational processes in free recall. *Psychological Review, 116*(1), 129-156.

Revelle, W. (2019). psych: Procedures for Psychological, Psychometric, and Personality Research. Northwestern University, Evanston, Illinois. R package version 1.9.12, Retrieved from https://CRAN.R-project.org/package=psych.

RStudio Inc (2015). Easy web applications in R. [computer software manual]. (http://www.rstudio.com/shiny/).

Sariyar, M. & Borg, A. (2010). The RecordLinkage package: Detecting Errors in Data. *The R Journal 2*(2), 61-67.

Taylor, J. E., Beith, A., & Sereno, S. C. (2019). LexOPS: An R package and user interface for the controlled generation of word stimuli. Retrieved from https://github.com/JackEdTaylor/LexOPS

Wickham, H., Hester, J., & Chang, W. (2019). Devtools: Tools to make developing R packages easier. R package version 2.2.1 Retrieved from https://CRAN.R-project.org/package=devtools

Table 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Participant | Response | Key |  |  |
| 1 | hom | home |  |  |
| 1 | windsheld | windshield |  |  |
| 1 | pepper | pepper |  |  |
| 2 | homme | home |  |  |
| 2 | windsheild | windshield |  |  |
| 2 | pepper | pepper |  |  |

*Sample Output Obtained using prop.correct.cued()*

*Note.*

Table 2

|  |  |
| --- | --- |
| Participant | Response |
| 1 | Rosemary, Onion, Salt |
| 2 | Salt, Garlic |
| 3 | Rosemary |

*Sample Free-Recall Data Before Running* arrange\_data()

*Note.* This data is arranged in wide format with each participant’s response in the same row. Responses share the same cell of the dataframe and are separated by a comma.

Table 3

|  |  |  |
| --- | --- | --- |
| Participant | Response | Position |
| 1 | Rosemary | 1 |
| 1 | Onion | 2 |
| 1 | Salt | 3 |
| 2 | Salt | 1 |
| 2 | Garlic | 2 |
| 3 | Rosemary | 1 |

*Sample Free-Recall Data After Running* arrange\_data()

*Note.* This data is arranged in long format with each response having a unique row and cell in the dataframe.

Table 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cutoff Criteria | Participant | Response | Key | Scored |
| 0 | 1 | basket | basket | 1 |
|  | 1 | flowrs | NA | 0 |
|  | 2 | flwrs | NA | 0 |
|  | 2 | glasses | NA | 0 |
| 1 | 1 | basket | basket | 1 |
|  | 1 | flowrs | flowers | 1 |
|  | 2 | flwrs | NA | 0 |
|  | 2 | glasses | NA | 0 |
| 2 | 1 | basket | basket | 1 |
|  | 1 | flowrs | flowers | 1 |
|  | 2 | flwrs | flowers | 1 |
|  | 2 | glasses | NA | 0 |

*Sample Output Obtained using prop.correct.free()*

*Note.* Columns 2 through 5 illustrate the structure of the output file. Three types of potential responses are included: Correct spellings, misspellings that are scored as correct based on the cutoff, and incorrect responses not in the key (either misspellings or responding with a word not in the key; denoted by the NA in the key column). Responses are scored using Levenshtein distance cutoffs of 0, 1, and 2.

Table 5

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Participant | Response | Key | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
| 1 | sleek | sleek | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | slk | sleek | 0 | 0 | 1 | 1 | 1 | 1 |
| 3 | stteadily | steadily | 0 | 1 | 1 | 1 | 1 | 1 |
| 4 | vibretion | vibration | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | draw | doodle | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | watch | stopwatch | 0 | 0 | 0 | 0 | 1 | 1 |

*Example of Different types of Participant Response Errors in the Cued-Recall Example*

*Note.* Participant 1 provided the correct response; Participant 2 omitted two letters; Participant 3 included one extra letter; Participant 4 mistyped one letter; Participant 5 responded with a synonym; Participant 6 responded with a shortened form of the word. *lrd* columns denote Levenshtein distance used at scoring.

Table 6

|  |  |  |
| --- | --- | --- |
| Scoring Criteria | % Sensitivity | % Specificity |
| *lrd* 0 | 99 | 94 |
| *lrd* 1 | 99 | 96 |
| *lrd* 2 | 97 | 96 |
| *lrd* 3 | 80 | 97 |
| *lrd* 4 | 47 | 98 |
| *lrd* 5 | 24 | 99 |

*Sensitivity and Specificity Results for Maxwell and Buchanan (2020)*

*Note.* Column labels indicate Levenshtein distance used at scoring. Values denote percentages.

Table 7

|  |  |  |
| --- | --- | --- |
| Scoring Criteria | % Sensitivity | % Specificity |
| *lrd* 0 | 99 | 93 |
| *lrd* 1 | 99 | 97 |
| *lrd* 2 | 97 | 98 |
| *lrd* 3 | 87 | 99 |
| *lrd* 4 | 62 | 99 |
| *lrd* 5 | 40 | 99 |

*Sensitivity and Specificity Results for Maxwell and Huff (in press)*

*Note.* Column labels indicate Levenshtein distance used at scoring. Values denote percentages.

Table 8

*Mean Recall Rates as a Function of Human Coded and lrd Scored Data Collapsed Across Item Type in Maxwell and Buchanan (2020).*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 |
| Human Coded | 54.14 (3.47) | -- |  |  |  |  |  |
| *lrd* 0 | 50.72 (3.58) | 0.13 | -- |  |  |  |  |
| *lrd* 1 | 52.14 (3.57) | 0.07 | 0.05 | -- |  |  |  |
| *lrd* 2 | 53.37 (3.53) | 0.03 | 0.10 | 0.05 | -- |  |  |
| *lrd* 3 | 61.30 (2.91) | 0.29\* | 0.43\* | 0.37\* | 0.32\* | -- |  |
| *lrd* 4 | 77.42 (1.86) | 1.10\* | 1.24\* | 1.17\* | 1.12\* | 0.87\* | -- |
| *lrd* 5 | 88.48 (1.09) | 1.76\* | 1.88\* | 1.82\* | 1.77\* | 1.63\* | 0.96\* |

*Note.* Mean recall rates for each scoring condition. *95%* *CI*’s are in parentheses. HC = Human coded data. HC and *lrd* columns indicate Cohen’s *d* effect sizes for post-hoc comparisons, \* = *p* < .05.

Table 9

*Mean Recall rates as a Function of Human Coded and lrd Scored Data Collapsed Across Associative Direction Items in Maxwell and Huff (in press).*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 |
| Human Coded | 43.96 (6.57) | -- |  |  |  |  |  |
| *lrd* 0 | 41.06 (6.59) | 0.21 | -- |  |  |  |  |
| *lrd* 1 | 43.11 (6.59) | 0.06 | 0.15 | -- |  |  |  |
| *lrd* 2 | 44.86 (6.58) | 0.06 | 0.28\* | 0.13 | -- |  |  |
| *lrd* 3 | 50.83 (6.42) | 0.52\* | 0.75\* | 0.59\* | 0.46\* | -- |  |
| *lrd* 4 | 64.84 (5.51) | 1.79\* | 2.08\* | 1.67\* | 1.74\* | 1.33\* | -- |
| *lrd 5* | 77.69 (4.21) | 3.15\* | 3.49\* | 3.15\* | 3.12\* | 2.81\* | 1.76\* |

*Note.* Mean recall rates for each scoring condition. *95% CI*s are in parentheses. HC = Human coded data. HC and *lrd* columns indicate Cohen’s *d* effect sizes for post-hoc comparisons, \* = *p* < .05.

Table 10

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Experiment | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
| MB | .93 | .95 | .94 | .79 | .49 | .24 |
| MH | .94 | .97 | .96 | .85 | .59 | .36 |

*Inter-rater Reliability Statistics (Cohen’s κ) for Maxwell and Buchanan (2020) and Maxwell and Huff (in press)*

*Note.* MB = Maxwell and Buchanan, 2020; MH = Maxwell and Huff (2020). *lrd* columns indicate Levenshtein distance used at scoring*.* All values are Cohen’s *κ* between human scored data and data scored at each *lrd* percentage.

Table 11

|  |  |  |  |
| --- | --- | --- | --- |
| List Type | Scoring Criteria | % Sensitivity | % Specificity |
| Ad-hoc | 0 | 90 | 86 |
|  | 1 | 89 | 87 |
|  | 2 | 98 | 95 |
|  | 3 | 95 | 96 |
|  | 4 | 96 | 96 |
|  | 5 | 96 | 96 |
|  |  |  |  |
| Categorical | 0 | 98 | 90 |
|  | 1  2  3  4 | 98  98  97  95 | 91  91  92  94 |
|  | 5 | 95 | 94 |
|  |  |  |  |
| Unrelated | 0 | 93 | 87 |
|  | 1 | 93 | 87 |
|  | 2 | 92 | 87 |
|  | 3 | 97 | 96 |
|  | 4 | 97 | 96 |
|  | 5 | 97 | 96 |

*Sensitivity and Specificity Results for Huff et al. (2018)*

*Note:* Analyses are split by list type. Scoring criteria indicates Levenshtein distance used when running prop.correct.free().

Table 12

*Mean Correct Free-Recall as a Function of Human Coded and lrd Scored Data for each list type used in Huff et al. (2018).*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| List Type | Group | *M* | HC | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd 4* |
| Ad-hoc | Human Coded | 50.00 (2.78) | -- |  |  |  |  |  |
|  | *lrd* 0 | 48.10 (2.52) | 0.13 | -- |  |  |  |  |
|  | *lrd* 1 | 48.64 (2.51) | 0.09 | 0.04 | -- |  |  |  |
|  | *lrd* 2 | 48.58 (2.73) | 0.09 | 0.03 | 0.01 | -- |  |  |
|  | *lrd* 3 | 49.48 (2.78) | 0.02 | 0.09 | 0.05 | 0.06 | -- |  |
|  | *lrd* 4 | 49.71 (2.80) | 0.01 | 0.11 | 0.07 | 0.07 | 0.01 | -- |
|  | *lrd 5* | 49.71 (2.80) | 0.01 | 0.11 | 0.07 | 0.07 | 0.01 | 0.00 |
|  |  |  |  |  |  |  |  |  |
| Categorical | Human Coded | 47.86 (2.50) | -- |  |  |  |  |  |
|  | *lrd* 0 | 44.13 (2.67) | 0.25 | -- |  |  |  |  |
|  | *lrd* 1 | 44.60 (2.65) | 0.23 | 0.03 | -- |  |  |  |
|  | *lrd* 2 | 44.71 (2.64) | 0.22 | 0.04 | 0.01 | -- |  |  |
|  | *lrd* 3 | 45.56 (2.65) | 0.16 | 0.10 | 0.06 | 0.06 | -- |  |
|  | *lrd* 4 | 47.19 (2.74) | 0.05 | 0.20 | 0.19 | 0.16 | 0.11 | -- |
|  | *lrd* 5 | 47.19 (2.74) | 0.05 | 0.20 | 0.19 | 0.16 | 0.11 | 0.00 |
|  |  |  |  |  |  |  |  |  |
| Unrelated | Human Coded | 37.99 (2.68) | -- |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *lrd* 0 | 37.40 (2.75) | 0.04 | -- |  |  |  |  |
|  | *lrd* 1 | 37.52 (2.75) | 0.03 | 0.01 | -- |  |  |  |
|  | *lrd* 2 | 37.88 (2.74) | 0.01 | 0.03 | 0.02 | -- |  |  |
|  | *lrd* 3 | 38.46 (2.85) | 0.03 | 0.07 | 0.06 | 0.04 | -- |  |
|  | *lrd* 4 | 38.46 (2.85) | 0.03 | 0.07 | 0.06 | 0.04 | 0.00 | -- |
|  | *lrd 5* | 38.46 (2.85) | 0.03 | 0.07 | 0.06 | 0.04 | 0.00 | 0.00 |
|  |  |  |  |  |  |  |  |  |

*Note.* Mean recall rates for each scoring condition. *95%* *CI*’s are in parentheses. HC = Human coded data. *lrd* columns and row labels indicate each of the tested cutoff criteria. HC and percentage columns indicate Cohen’s *d* effect sizes for post-hoc comparisons, \* = *p* < .05.

Table 13

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| List Type | *lrd* 0 | *lrd* 1 | *lrd* 2 | *lrd* 3 | *lrd* 4 | *lrd* 5 |
| Ad-hoc | .76 | .76 | .93 | .92 | .92 | .92 |
| Categorical | .89 | .90 | .90 | .90 | .89 | .89 |
| Unrelated | .80 | .80 | .80 | .93 | .93 | .93 |

*Inter-rater Reliability Statistics (Cohen’s κ) for Huff et al. (2018).*

*Note.* List Type corresponds to the three study lists conditions used in Huff et al. (2018). *lrd* columns indicate each of the tested cutoff criteria All values are Cohen’s *κ* between human scored data and data scored at each *lrd* cutoff.



*Figure 1.* Illustration of the *lrd Shiny* application’s Instructions tab prior to uploading a dataset.



*Figure 2.* Illustration of the *lrd Shiny* application’s Scored Output tab after uploading a dataset. Data in this is example is scored using a 75% cutoff criteria.



*Figure 3*. Illustration of the lrd Shiny application’s Proportion Correct tab. Data is grouped by participant identifier.



*Figure 4*. Illustration of the lrd Shiny application’s Plots tab. Data in this example is grouped by an optional condition column attached to the upload .csv file.